Reference software architecture for improving modifiability of personalised web applications - a controlled experiment

Luz-Viviana Cobaleda, Raúl Mazo, Jorge Luis Risco Becerra, John-Freddy Duitama

Reference software architecture for improving modifiability of personalised web applications – a controlled experiment

Luz-Viviana Cobaleda*

Engineering and Software Research Group,
University of Antioquia,
Calle 70 No. 52-21, Medellín, Colombia
Email: luz.cobaleda@udea.edu.co
*Corresponding author

Raúl Mazo

Centre de Recherche en Informatique (CRI),
Paris 1 Panthéon Sorbonne University,
90 rue de Tolbiac, 75013 Paris, France
Email: raul.mazo@univ-paris1.fr

Jorge Luis Risco Becerra

Laboratory of Software Technology (LTS),
University of São Paulo,
Rua prof. luciano gualberto 158 trav 3 São Paulo, Brazil
Email: jorge.becerra@usp.br

John-Freddy Duitama

Engineering and Software Research Group,
University of Antioquia,
Calle 70 No. 52-21, Medellín, Colombia
Email: john.duitama@udea.edu.co

Abstract: Although web personalisation has been studied for the last two decades, there remains a need to address current challenges: context-awareness and the inclusion in a business environment. The wide variety of mobile devices and their continuous technological evolution demands the permanent development of new personalisation strategies. Additionally, two factors complicate the inclusion of personalised web applications in a business environment: the frequent change of personalisation strategies for each business, and the technical complexity to integrate these strategies in a short time. We propose a reference architecture as a tool to favour their modifiability. Moreover, our proposal facilitates the opportunity for enterprises to adopt web-personalised systems into their business as a strategic tool. A controlled experiment validates our approach; we compare five change scenarios that are implemented under two architectures: experimental and control architecture. We used change scenarios derived from a real Brazilian e-commerce enterprise.

Keywords: personalisation; reference software architecture; web application; modifiability; personalised web applications; software components.

Reference to this paper should be made as follows: Cobaleda, L-V., Mazo, R., Becerra, J.L.R. and Duitama, J-F. (2016) 'Reference software architecture for improving modifiability of personalised web applications – a controlled experiment', *Int. J. Web Engineering and Technology*, Vol. 11, No. 4, pp.351–370.

Biographical notes: Luz-Viviana Cobaleda is a doctoral candidate at University of Antioquia, Colombia. She holds a Master degree in Engineering with emphasis on informatics from the same university and a specialisation degree in software engineering at the EAFIT University, Colombia. She has been an Assistant Lecturer at the University of Antioquia. She has participated in projects about development of personalised mobile applications and personalized embedded software systems in the medical field at the University of Antioquia. Her research interest includes software engineering, specially software design and specification, personalised systems, adaptive systems, and model-driven engineering.

Raúl Mazo received his Master of Science degree and PhD degree in Informatics from the Panthéon Sorbonne University (France). Since 2012, he has been an Associate Professor with that university and researcher with the Centre de Recherche en Informatique (CRI). His research and teaching topics include: software engineering, requirements engineering, eRP and configurable systems, and (dynamic) product line engineering. He has published more than 50 scientific works on these topics and regularly participates in projects, and program commits on these topics. In 2015, the French Ministry of Education and Scientific Research accredits him to review industrial projects.

Jorge Luis Risco Becerra is a PhD from Polytechnic School of the University of São Paulo, Magister from the same University and received a degree in Electronic Engineering from the Universidad Nacional Mayor de San Marcos. Currently, he is a Professor in the Department of Computer Engineering at the Polytechnic School of the University of Sao Paulo (Brazil). He works in the area of software engineering, and its main research lines are software architecture, process architecture (software factory) and automation systems.

John-Freddy Duitama is a Professor at University of Antioquia. He received his PhD degree in Computer Science from National Telecommunication Institute (France), a Master degree in System Engineering from National University (Colombia), and a System Engineering degree from the University of Antioquia (Colombia). His research areas include development of personalized web applications, mobile applications and big data.

1 Introduction

Personalisation can be understood as the process of tailoring the business information and services to needs, interests, preferences, context, behaviour and specific requirements of an individual or community. It provides a customised environment with an increased value to the customer and to the business (Brusilovsky, 2001, 1996; Karat et al., 2003;

Brusilovsky and Nejdl, 2004). Personalisation is intended to increase customer fidelity (Kwon and Kim, 2012), and to filter the information that users need most. Although web personalisation has been studied for the last two decades, there remains a need to address current challenges: context-awareness and the inclusion in a business environment. Context-awareness refers to systems that can both sense and react based on their environment, thus, the web applications are expected to respond appropriately to the context of users. For example, a mobile phone may detect that a user is sitting or walking, and the web application reacts in correspondence with the user state. In these cases, besides the wide variety of devices, their continuous technological evolution demands the permanent development of new personalisation strategies. Although personalisation has demonstrated advantages (Alotaibi, 2013; Kwon and Kim, 2012) in web applications, two factors complicate its inclusion in a business environment: the frequent change of personalisation strategies for each business, and the technical complexity to integrate these strategies in a short time. Personalisation strategies can change continuously as a natural result of getting closer to diverse audiences; and, they can change in response to organisational interests and market evolution. For example, in an e-commerce domain, personalised strategies can provide offers of products in a matter of minutes or just a few days or hours. On the other hand, the personalisation code in web applications is intermingled with the basic functionality (De Virgilio, 2012; Fernández et al., 2010), and the process to add or modify a personalisation strategy in a web application is difficult. Different approaches (De Virgilio, 2012; Fernández et al., 2010) have been explored in order to permit the continuous update of personalisation strategies, and to reduce the complexity in the implementation. These last issues and others like user model reuse and the use of external personalisation components have not been sufficiently studied.

In the area of software engineering, a reference software architecture is a consistent set of architectural best practices, which are designed with the aim of providing a template solution for a particular domain. It gathers the learning experiences gained from past projects, and offers guidance for future developments. It also enables strategic reuse of architectural assets in a particular domain (Kazman and McGregor, 2012; Reed, 2002; Bengtsson et al., 2004; SEI, n.d.). A reference architecture is necessary to support increased complexity, scope and size of software systems; as well as, to support the dynamics of enterprises that need to respond more quickly to market demands.

In this paper, we propose a reference software architecture that has the software modifiability as the main architectural drive, and is based on component weaving process. We adopt the software modifiability definition as "the ease with which it [software system] can be modified to changes in the environment, requirements or functional specification" (Bengtsson et al., 2004). We validate our reference architecture through a controlled experiment in a real business case taken from a Brazilian e-commerce enterprise.

The remainder of this paper is organised as follows: Section 2 presents related works. Section 3 presents the running example that we use throughout the paper to explain the proposed reference architecture for personalised web applications. Section 4 describes the reference architecture, and presents a methodology to apply. Section 5 presents the controlled experiment. Section 6 presents our results. Finally, Section 7 presents our conclusions and future work.

2 Related work

Managing and assessing system changes have been addressed in research for many years. Some of the more well-known modifiability assessment approaches include the software architecture analysis method (SAAM) (Bass et al., 1998) and the Oman taxonomy (Oman and Hagemeister, 1992). The research community has used these assessment approaches to conduct experiments on modifiability analysis of web applications, and has proposed architectures intended to respond to the need of managing the personalisation in web applications. Below, we briefly describe and analyse some of these assessment approaches, experiments and architectures.

2.1 Analysis of software modifiability

Stella et al. (2008) compare the modifiability of a web application implemented from the same requirements, on three platforms [J2EE, .NET and Ruby on Rails (RoR)]. In order to do so, they conducted a change propagation analysis on each implementation of the web application, and used three modifiability metrics (number of modified files, number of modified lines of code, and development time in man-hours to incorporate the change) to compare the extent to which each platform facilitates modifiability. Stella et al. (2008) observed that the web application developed on .NET required more modifications to source code, and more effort for implementing enhancements than the ones implemented with RoR and Spring-Hibernate. These results can be attributed mainly to two reasons: the enhancement of the .NET application, which required hand code mapping from the database to the entity object, whereas J2EE Hibernate and RoR ActiveRecord automated the mapping process. The second reason could be that .NET offers tighter coupling between concerns versus the cleaner separation of concerns in J2EE and RoR.

Other works related with the measurement of modifiability as a mean to compute the maintainability of web applications are the model for assessing the maintainability proposed by Di Lucca et al. (2004) and the taxonomy of metrics presented by Oman and Heigemaster (1992). Both are used to estimate the maintainability of traditional software, and establish other metrics specific to web applications, such as web page data coupling (Di Lucca et al., 2004; Oman and Hagemeister, 1992). Besides metrics presented by Oman and Heigemaster there are several other methods supporting the analysis of software modifiability; such as, the SAAM (Bass et al., 1998), architecture level modifiability analysis (ALMA) (Bengtsson et al., 2004), and aspectual software architecture analysis method (ASAAM) (Tekinerdogan, 2004). SAAM takes several quality attributes as key issues: performance, security, availability, functionality, usability, portability, reusability, testability, integrability, and modifiability. Bass et al. (1998) categorise modifications as follows: extending or changing capabilities, deleting unwanted capabilities, adapting to new operating environments, and restructuring. Based on the quality attributes presented, Bass et al. (1998) propose different architectural styles that then are employed in the SAAM. ALMA is a scenario-based evaluation method focused on modifiability issues that provides quantitative predictions (via metrics and change impact analysis) about the modifiability of a system when it is confronted with different scenarios. ASAAM is an extension and refinement to the SAAM in order to include explicit mechanisms for identifying architectural aspects and components. Similar to SAAM, ASAAM takes as input a problem description, requirements and architecture descriptions for

1 developing a candidate architecture to provide a design that will be analysed with respect to the required quality factors and aspects

2 develop scenarios from different stakeholders.

Stafford and Wolf (2001) proposed an automated technique for architecture dependency analysis that builds graphs of architectural components and captures their static and behavioural relationships. Stafford and Wolf's approach is implemented on top of an architectural description language, and it serves mainly to support architects in the navigation and analysis of the set of components related to a given particular concern.

2.2 Architectures for personalised web applications

This section briefly presents and analyses architectures proposed in response to the need of managing the personalisation in web applications.

Fernández (2008) proposes a three layer architecture to support adaptive web applications (Fernández, 2008). Adaptive web applications creator (AWAC) tool generates applications on personalisation rules modelling language (PRML) language. The three layers are application data, main modules and user interface layers; the personalisation functionalities are concentrated only in the first two layers. The main modules layer comprises the website engine, PRML Manager, and PRML Evaluator modules. The website engine module interacts with the user, gathers the requests and gives back the response. In addition, it loads the user model (from the Database) when a user starts a new session, captures the events performed with his browsing actions and sends them to the PRML Evaluator module (Fernández, 2008). The PRML Evaluator module executes the personalisation rules attached to the events. When a rule is triggered, this module evaluates the rule conditions, and performs the proper actions. The adaptive actions are only performed once during a session to avoid overwhelming the user. The PRML manager module allows reading, and updating rules at runtime. PRML rules are defined in a separate file. The application data layer consists of both a text file containing the set of rules defining personalisation policies on the website, and the application database. Although Fernández' approach considers different modules to tailor the response to the user, the personalisation actions are so fine-grained causing the creation of many rules to achieve a single strategic personalisation goal. As a result, it is hard to manage all of them. Moreover, in time, the user model accumulates valuable information, this approach lacks a clear way to reuse the user model in other applications. Lastly, Fernández' approach does not support the integration of different approaches using techniques such as collaborative-filtering or content-based analysis.

De Virgilio (2012) proposes an architecture implemented with a tool called flexible adaptation of web information systems (FAWIS) that is used for the automatic generation of adaptive websites. FAWIS is based on adaptation modelling language (AML) language. Users can specify declaratively how to build a configuration satisfying the adaptation requirements for a given profile; they use production rules to achieve automatic adaptation of content delivery. The architecture consists of four modules: context manager (CM), adaptation manager (AM), response generator (RG), adaptation designer (AD). The CM is able to capture and classify a description of the client characteristics (the context). The AM takes as input the context of the client, and generates a suitable adaptation configuration. The AM communicates with a repository of

adaptation rules. Three modules, one for each level of the response, compose the RG: presentation, navigation and information recovery. RG generates an appropriate response for the client profile to deliver over the web. The AD communicates with a repository of adaptation rules, and allows the designer to define new adaptations of a previously unpredicted event. In this way, it is possible to extend the functionality of the tool. Although this approach considers different modules for each type of personalisation technique, it lacks ways to communicate with external modules like those provided by recommended systems, and lacks strategies to enable the reuse of context model by other applications.

Ceri et al. (2007) propose a method for designing and implementing data-intensive web applications as well as an associated language called web modelling language (WebML). The WebML language has evolved to the interaction flow modelling language (IFML). IFML allows expressing the content, user interaction and control behaviour of the front-end of software applications. WebRatio platform supports both languages. The personalisation support in WebML consists of the definition of different views according to user profile data, or the browsing device. This process is completed at design time, and is based on the user-group-module pattern. This pattern consists of associating users to groups, and associate groups to modules. It is a way to indicate that a user, who belongs to a group, has access to that specific module. When user login, system enables those modules associated to groups which the user belongs (Fernández et al., 2009; Kęsik and Żyła, 2010; Martinenghi, 2014). The process to fill the personalisation entities is done manually (Kęsik and Żyła, 2010). IFML is dedicated to support data-intensive business applications (Brambilla and Butti, 2014). It lacks concern about modelling display content such as layout, style and look and feel in the application front-end. This approach considers a limited view of the user model using just the user role, or group to deliver different content; and, it is insufficient the consideration of user model reuse and access to external modules for personalising.

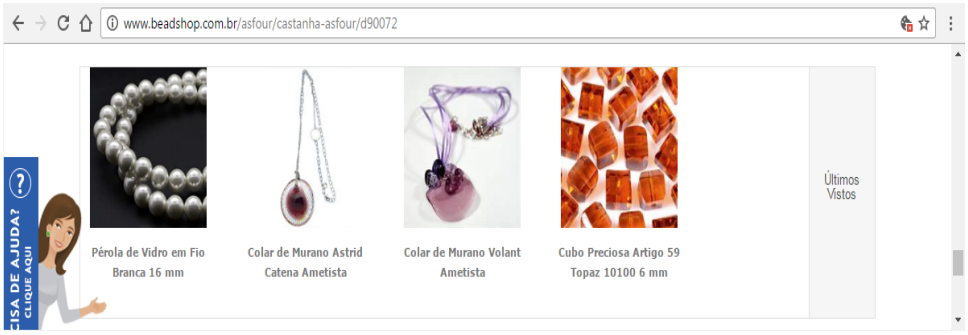
Our approach can be seen as a complement to the application of the aforementioned architectural assessment methods to test offering cases and change scenarios. In the other hand, it can be seen as a complement for the previously described architectures in several senses. The difference between our approach, and the approach of Fernández and Virgilio is that they use a fined-grained personalisation instruction like add/remove links, or show/hide texts, and we conceive components implementing a personalisation goal instead of just focusing on one personalisation technique. Additionally, we consider an external module dedicated to managing the information of users, contexts and groups, which makes possible the reuse of other personalised systems in the same enterprise; additionally, we permit the extension of personalisation strategies by binding external components.

3 An e-commerce system – running example

To illustrate our approach, we provide a real scenario in the e-commerce domain from a Brazilian enterprise called VTEX. This enterprise is a leader in e-commerce technology in Latin America, and is dedicated to the commercialisation of software as a service. VTEX offers solutions to enterprises that have websites in different market segments.

Taking into account the experience and knowledge of the e-commerce business by VTEX, we extracted various scenarios of personalisation to define our running example. The functionalities included are in the categories of product discounts and product recommendation. In an e-commerce domain, it is common to offer product discounts to attract new clients and to gain their loyalty. The discounts category offers customers a specific discount (e.g., 10%) for accumulated purchases greater to a fixed value over a period; and, highlights the discount in the web page. The recommendation category is adopted to increase the conversion rate. The conversion rate is the percentage of website visitors who actually purchase a product on the site. In our scenario, the recommendation functionality consists of several modalities: the history of recent products visited by customers with a link to the detailed product description (Figure 1), and recommend products based on similarity measures between users and/or products. The products recommended to a user are those preferred by similar users, or those similar to the product that the user is searching.

Figure 1 History of recent products visited by customer (see online version for colours)



4 Reference architecture

This section presents a conceptual view of the reference architecture, and makes use of our running example. It also presents a methodology to apply the architecture.

The reference architecture designed to support personalisation on web applications has the software modifiability as the main architectural drive. We presented an example in the e-commerce domain to illustrate the importance of the modifiability feature in personalised web software. To increase the customer's loyalty, an enterprise may want to define continuously new personalisation strategies like different types of product discounts. These discounts may last from a few hours to many days, so it is valuable to permit the change of personalisation strategies in a short period. Therefore, web applications should have the capacity to include, or discard different personalisation strategies in a short period; that is, software should be modifiable. This reference architecture proposes the use of component weaving as an alternative to tackle the challenges of including personalised behaviour.

4.1 Reference architecture description

The standard MVC web application architecture supports the proposal. It separates an application into three main logical layers: the model, the view, and the controller. Objects in the model layer encapsulate the data specific to an application; view objects are the user graphical interfaces; and the objects in the controller layer are intermediaries between view objects and model objects, and coordinate tasks for an application. In this paper, we refer to the model layer as 'persistence'. Our approach manages each personalisation strategy as a specialised component, which can be added or removed from basic application. Additionally, we add three specific modules to facilitate the integration of personalisation strategies as specialised components:

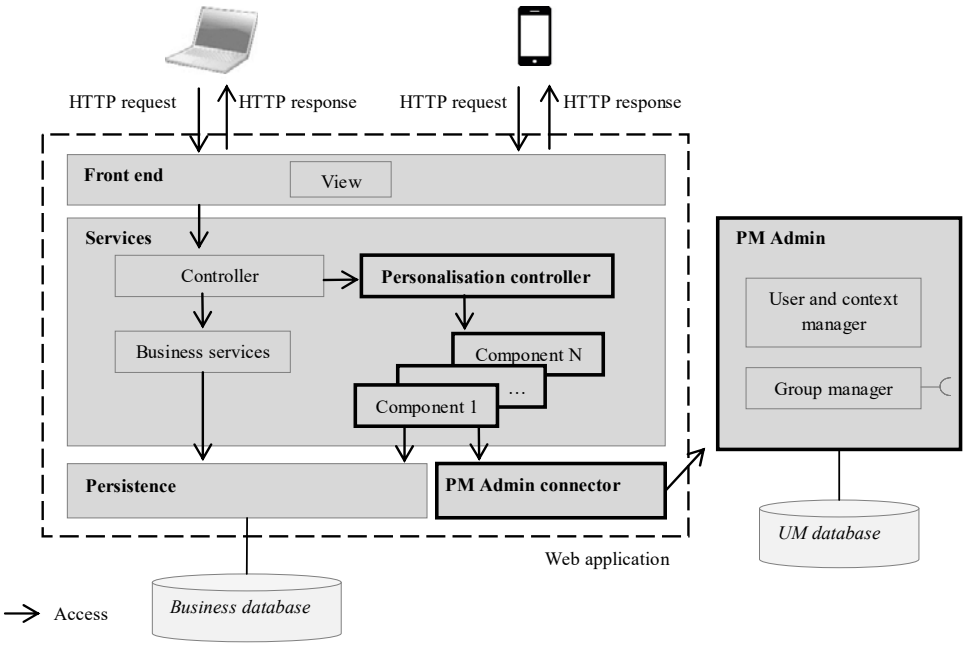
1 the personalisation controller (PC) module

2 the connector to personalisation administrator (PMAdmin connector) module

3 the personalisation model administrator (PMAdmin) module.

Figure 2 shows the reference architecture where bold lines mark the modules to manage personalisation process.

Figure 2 Reference architecture for personalised web applications



Regarding to the set of specialised components (1 … N), each component implements a personalisation goal or strategy, and possibly each component uses different personalisation techniques. For instance, in our running example, the 'discount by accumulated value' component may use techniques like 'adaptive selection' to capture customer purchases in a period and the 'link annotation' technique to emphasise the discount. In the same way, in the recommendation category, the associated component

may use techniques like 'filter collaborative' to find similar products and/or users. Note that each specific personalised component implements a personalisation goal as a key feature instead of just focusing on one personalisation technique.

The PC module is responsible for processing events detected by the web application, and coordinates the personalisation effects by accessing the specialised components. PC module manages the specific functionality of personalisation to avoid mixing the web application's basic functionality with the personalised behaviour; in this way, PC simplifies the modifiability of personalised behaviour. In our running example, as a strategy to offer a personalised discount via accumulated purchases, the PC module identifies the relevant events to achieve this purpose, and interacts with the appropriate component. In this case, PC module identifies when a customer browses a product list, and demands to the 'discount by accumulated value' component for a personalised discount value and the type of visual emphasis. The personalised discount component obtains the value of the customer's accumulated purchases as a means to determine the discount percentage and the highlighting mode.

The *connector to personalisation model administrator* (*PMAdmin connector*) serves as a bridge between specialised components and the external *PMAadmin* module. The specialised components send the data request to the *PMAdmin* module through this connector.

The *personalisation model administrator* (*PMAdmin*) is an independent module placed outside the web application. It manages two types of information: inferred data, and redundant data.

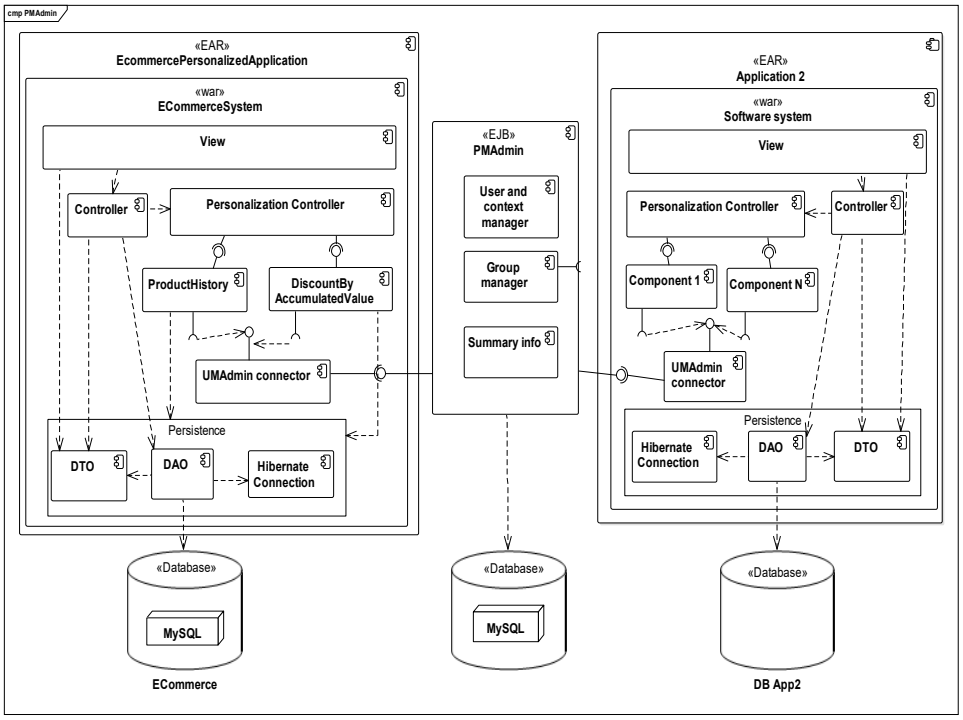
To build the inferred data, the system usually gathers records from diverse sources as transactional databases or unstructured files. Periodically, an extract, transform and load (ETL) process collects and stores these records in a data warehouse; after that, diverse data mining or machine-learning techniques build the relevant information about users, context or items. In our running example, the inferred data could include user profile, product profile, user groups, product groups, similar users, similar products, and prediction models.

The web personalised system requires gathering information while users interact with the system, such as type of user device, geographical position, visited products or the navigation track. This information allows us to establish user behaviour. *PMAdmin* is intended to manage a minimum amount of redundant data in order to establish user behaviour, and to allow their use through a particular personalisation processes. Note that this module excludes transactional information. As *PMAdmin* module uses its own database, at design time, analysts must decide which strategies to utilise as a means to maintain consistency among replicas. This reference architecture separates the transactional process module from the personalisation functionalities. In this way, both personalisation strategies and user data gather techniques run in parallel with respect to transactional operations. This separation facilitates software maintainability tasks, such as the addition of new gathering mechanisms, the addition of new personalisation strategies, and the change prediction models. This facility gains value in changing and dynamic scenarios as e-commerce.

Several software web-personalised applications can share the *PMAdmin* module within the same enterprise (Figure 3); i.e., applications like telesales, call centres, and logistics. A unique *PMAdmin* module reduces the development effort for each personalised application, and improves the collection of more precise information over

time. The more users' information the system has, the greater are the possibilities of offering a personalised experience.

Figure 3 PMAdmin with multiples web applications, represented with UML component diagram



Finally, notice that the persistence layer serves both the web application, and the specialised personalisation components.

4.2 Methodology to apply the reference architecture

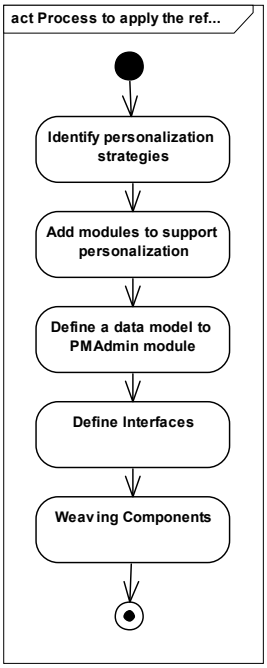
This section establishes a process to guide the developers in the adoption of the reference architecture. The web application can previously exist, or be designed independently; in consequence, the next step is to integrate personalised behaviour. Figure 4 shows the process systematically which is represented using a UML activity diagram.

4.2.1 Identify personalisation strategies

The goal of this step is to identify the personalisation strategies as different portions of functionality, encapsulate them in independent specialised components with defined interfaces, and identify the information sources as a way to support the strategies.

One or several components materialise each strategy. Components may require data from the persistence layer in web application, from the *PMAdmin* module, or from the user interaction.

Figure 4 Process to apply the reference architecture represented with a UML activity diagram



In our running example, the functional requirements correspond to product discounts and product recommendation categories: the *discount by accumulated value* and *product history* components encapsulate this functionality. Note here that the functional decomposition is mainly about the identification of business strategies instead of the selection of a particular personalised technique. Designers can introduce, or remove strategies from global web applications. Additionally, to recommend products based on similarity measures between users, the system may use techniques such as collaborative-filtering or content-based analysis. Furthermore, to offer the discount strategies the system requires accumulated user purchases. Thus, both types of recommendations require information from the *PMAdmin* module.

4.2.2 Add modules to support personalisation

This step consists of adding three modules to enable the ensemble of personalisation strategies. *PC*, and *PMAdmin* connector modules are created inside the web application, whereas *PMAdmin* module is created as an external component.

4.2.3 Define a data model to PMAdmin module

It is required that the designer determines which type of information from *PMAdmin* module is necessary for each specialised component. For example, which user, items and context information will be extracted from the *PMAdmin* module. After, designers define the *PMAdmin* data model, and ways to gather its information; i.e., ETL processes from transactional databases, social networks or another web sources or application. Note that each personalisation strategy can require an additional batch process to provide the

intended functionality, i.e., machine learning algorithms, data mining approaches or another technique could run periodically over the *PMAdmin* module.

Although the goal is to minimise data redundancy, in some cases, the *PMAdmin* module contains redundant data with respect to the storage level on the web personalised system. In these cases, designers must define synchronous or asynchronous replication mechanisms such as online triggers or periodic batch processes.

4.2.4 Define interfaces

This step allows the establishment of components that will interact with other parts of the application. Designers must specify the functionality that each specialised component will provide or require from other modules. Several interfaces allow specifying these interactions: the relation between PC and *specialised components*, the relation of *specialised components* with persistence layer and/or *PMAdmin* connector, the relation between the *PMAdmin* connector and the *PMAdmin* module.

In the running example, one interaction is identified between PC module and two specialised components: *discount by accumulated value* and *product history* components. Thus, it should have an interface that guarantees the input data for first component, the *username*, and the *username* and *customer visited page* for the second one. A second interaction is between the two specialised components and the *PMAdmin* connector. It should guarantee the connection to *PMAdmin* module. A third interaction is between the *PMAdmin* connector and the *PMAdmin* module. The interface in this case should guarantee the retrieval of the *total value of customer purchases* over a period for the first component, and search the *last customer viewed products* for the second one.

4.2.5 Weaving components

The aim of this step is to compose a connected personalised web application. To do that, designers must attach the web application with the specialised components, with PC module and with *PMAdmin connector*.

Thus, it is necessary to add the specialised components previously identified, and adjust the defined interfaces of the *PC* and *PMAdmin connector*. In our running example, it means to add the *discount by accumulated value* and *product history* specialised components to the global proposed web structure and to update the *PC* and *UM Admin connector* architectural modules according to the defined interfaces.

5 Controlled experiment

In this section, we present the designed controlled experiment to evaluate the software modifiability of a proposed reference architecture.

5.1 Design of the experiment

Following the goal-question-metric (GQM) suggested in Wohlin et al. (2012), we stated that the goal of the experiment was to *analyse* the reference architecture *for the purpose* of evaluating it *with respect to* its modifiability *from the point of view of* the software

developer *in the context of* personalisation strategies taken from a Brazilian e-commerce enterprise.

The *context* of the experiment was a test case composed of five change scenarios, and a software application implemented under two architectures: experimental and control architecture. The experimental architecture is proposed in this paper; and control architecture corresponds to a standard model-view-controller (MVC) web application architecture.

The test case comprises five change scenarios described in terms of application changes involving personalisation strategies. These scenarios allow us to evaluate the support of future changes, and consequently, evaluate software modifiability under an architecture. The strategies were taken from a real Brazilian e-commerce enterprise (http://www.vtex.com/). An engineer implemented the same test case in both architectures under the supervision of the first author, resulting in two web applications: experimental and the control web applications, respectively. Later, the first author counted the number of needed changes in both web applications to accomplish the five change scenarios and finally we compared the results. The experiment was performed off-line (not in an industrial software development) and staffed by a software engineer.

With the experiment, we answered the primary research question (RQ):

RQ To what extent is the reference architecture able to allow modifiability?

5.2 Experimental units

5.2.1 Test case

In order to compare architectures, and assess their modifiability, we use a test case as the same point of comparison. The test case consists of a set of representative change scenarios. Here, a change scenario is the representation of the modifiability requirements when a web application needs to include personalisation strategies. The change scenarios should reveal differences in the architecture.

In the definition of scenarios process, first we interviewed two stakeholders at VTEX enterprise: the Strategy Manager and the Project Manager. The idea was to focus on possible and repetitive modifications related to personalisation. Later in the process, we (the researchers) discussed the scenarios and selected the most relevant.

The scenarios we found were the following:

5.2.1.1 Change scenario 1

The broad audience of the system demands a wide visual support for different customers, especially to middle-aged adults that present visual limitations. Thus, the software needs to be modified to adapt to the user's visual limitations. The software should identify if the registered customer has visual limitations and if so, the software should increase size letters and update colours.

5.2.1.2 Change scenario 2

According to ubiquitous web applications paradigm, this kind of application may adapt its services and software structure to user context; including the device, network, time and location context. Thus, the software needs to be changed to adapt its services and

presentation to device context and user location. Specifically, the software should give a discount if the user is located near to a store; also, the site's graphical interface should adjust according to size of the device.

5.2.1.3 Change scenario 3

The personalised recommendation methods are typical strategies in e-shop. Thus, the system shall include a recommendation method showing the last five products visited by user.

5.2.1.4 Change scenario 4

The recommendation strategies need to be more meaningful for customers. The recommendation method needs to be changed to implement techniques like collaborative filtering and content-based filter. The system shall implement two recommendation methods: upselling recommendation and cross-selling recommendation. Upselling recommendation consists of offering the customer additional or complementary products for purchase. The system uses collaborative filtering techniques to find products bought by similar customers but having higher cost. Cross-selling recommendation consists of offering the customer alternative products for purchase. This time, a content-based analysis allows finding similar products to one that the customer is searching.

5.2.1.5 Change scenario 5

The software needs a personalised discount strategy. Thus, the software shall implement a product discount strategy by accumulated value. This kind of discount offers customers a specific discount percentage for accumulated purchases greater to a fixed value over a period.

5.3 Metrics

In order to answer the RQ, we counted the number of changes needed to complete the implementation of the change scenario, in terms of:

- the number of files added or removed
- the number of methods added
- the number of code lines added or modified.

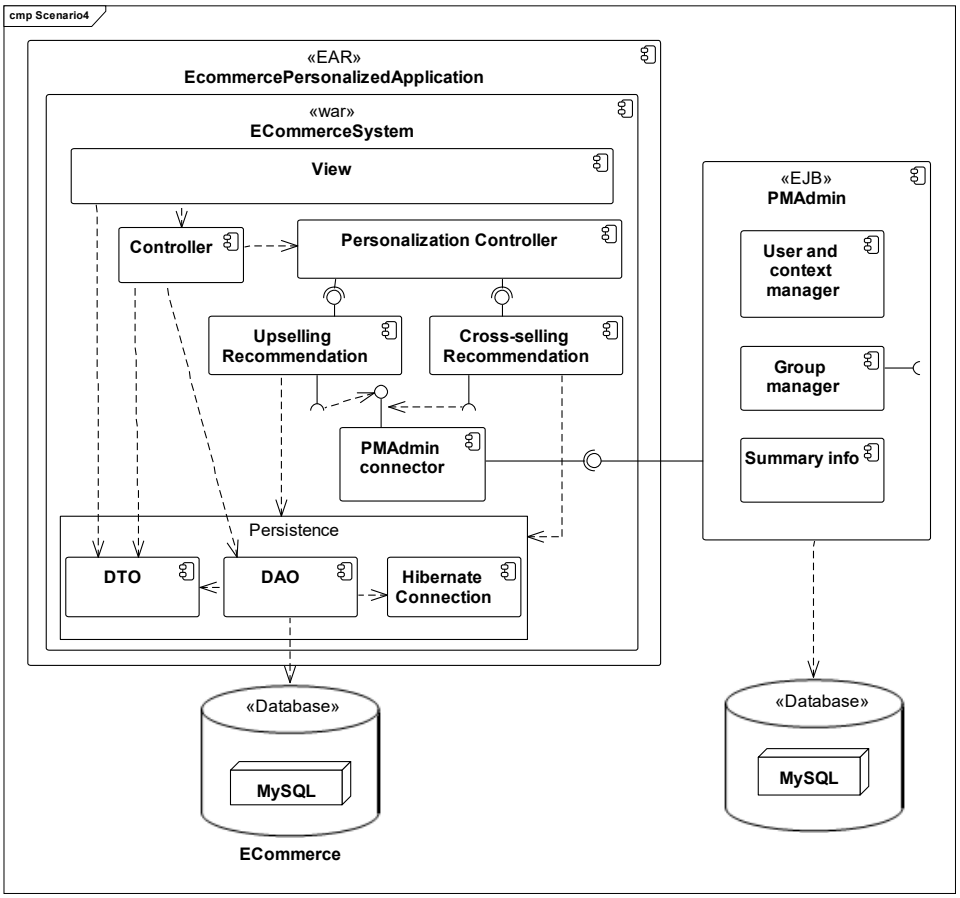
5.4 Prototype implementation

Experimental and control architectures were used to implement all the change cases, resulting in two web applications: the experimental and the control, respectively. Both web applications were implemented using Java programming language and Java Server Pages (JSP) technology, Hibernate framework for managing the data persistence, MySQL 5.6 as a database engine and Wildfly 8.2 as the application server.

In the experimental web application, we implemented the PMAdmin module under Enterprise Java Beans (EJB) technology and used the Java persistence API (JPA) for managing data persistence.

Figure 5 shows the architecture for the implementation of the change scenario 4 according to the reference architecture,[1] where *upselling recommendation* and *cross-selling recommendation* are the specialised components.

Figure 5 Change scenario 4 according to the reference architecture represented with a UML component diagram



5.5 Threats to validity

The main threat to internal validity in this experiment is the subject experience. This threat was alleviated by considering another engineer with good experience on web programming that cross-checked the results. The main threat to external validity of the experiment is the generalisation of the results. This threat was alleviated by selecting representative change scenarios extracted from a real enterprise. However, it is not possible to generalise the results because we only worked in the context of one enterprise, and only performed a comparison against a standard method. The main threat to construct validity is the misunderstanding of the 'modifiability' concept. To alleviate this threat, we defined the metrics based on the definition of modifiability from Bengtsson et al. (2004),

that is a work focused on the modifiability of software architectures. The metrics selected address our RQ in a direct way. Finally, and regard to conclusion validity, we present the results as preliminary validation due to the fact that we do not use statistical validation.

6 Results

Table 1 and Table 2^2 show the result of counting the needed changes to achieve the change scenarios under the implementation of experimental and control architectures respectively. Table 3 shows the changes made in the PMAdmin module corresponding to experimental architecture. We counted the number of files added or removed, differentiating classes (including interfaces) from configuration files; the number of methods added or removed; the number of code lines added or removed differentiating classes (including interfaces) from configuration files.

The fourth change scenario reports two counts because this scenario has two parts: (–) removing an existing recommendation strategy and (+) adding two new different recommendation strategies.

Table 1 Results of the execution of change scenarios under standard MVC Architecture

No. change scenario	*No. of files added or removed*		*No. of added or removed methods*	*No. of code lines added or modified*		*Total*
	Classes	*Configuration files*		*Classes*	*Configuration files*	
1	6	2	3	12	2	*24*
2	6	0	11	136	0	*153*
3	4	1	4	31	1	*41*
4 (+)	4	0	5	56	0	*65*
4 (–)	–4	–1	–4	–31	–1	*–41*
5	3	1	3	12	1	*20*

Note: * Files, methods or lines removed have a negative number.

We create the PMAdmin module before implementing the change scenarios with the experimental architecture. However, this effort was not included in Table 2, because the enterprise makes this task once, and it is not part of the personalisation strategies in a particular web application. Additionally, we add the PC module and PMAdmin connector inside the web application, with its interface and its implementation class. Table 2 excludes these changes, because they only are done once to prepare the web application to accept various personalisation strategies.

We used the data obtained in order to answer our RQ.

RQ To what extent is the reference architecture able to allow modifiability?

Focusing on the column labelled ‘No. of files added or removed’ in Tables 1 and 2, we observed that the number of classes was reduced in the reference architecture. It might be explained because in the control architecture we needed to create files mapping persistence matters; by contrary, in the experimental architecture those functionalities were managed inside the *PMAdmin* module.

Table 2 Results of the execution of change scenarios under reference architecture

No. change scenario	*No. of files added or removed*		*No. of added or removed methods*	*No. of code lines added or modified*		*Total*
	Classes	*Configuration files*		*Classes*	*Configuration files*	
1	3	0	3	13	0	*18*
2	6	0	9	136	0	*151*
3	2	0	6	33	0	*41*
4 (+)	4	0	7	58	0	*69*
4 (–)	–2	0	–6	–33	0	*–41*
5	2	0	3	13	0	*18*

Note: * Files, methods or lines removed have a negative number.

Table 3 Changes on the PMAdmin module in the execution of change scenarios under reference architecture

No. change scenario	*PMAdmin*			
	Number of elements added or modified			
	Classes	*Methods*	*Code lines*	*Total*
1	3	1	1	*5*
2	0	0	0	*0*
3	2	2	2	*6*
4 (+)	2	2	2	*6*
4 (–)	–2	–2	–2	*–6*
5	1	1	1	*3*

Note: * Files, methods or lines removed have a negative number.

In addition, as opposed to control architecture, in the experimental architecture, it was not necessary to add or modify configuration files. It may be explained because in the experimental architecture, those configurations are already in the *PMAdmin* module.

Note that reducing the configuration files changes in the experimental architecture, and transferring them to centralised *PMAdmin* module could bring advantages in the development process, because it may potentially reduce the error introduction that always appears in the modification of software.

Regarding data for the number of added or removed methods, we can observe that the number of changes has been slightly increased in the experimental architecture. This fact occurs because this architecture proposes the *PC* module to filter the personalisation petitions. Thus, the methods augmented.

Concerning changes in code lines and the number of methods, the experimental architecture reports a small increase. The control architecture includes methods for implementing functionalities, and for working on persistence matters. By contrary, the experimental architecture only includes methods for implementing the functionalities because *PMAdmin* module manages the persistence matters. However, the calls to the PC module could explain the increase in code lines in the experimental architecture.

Analysing the overall results, the reduction of configuration files, and the number of classes in the experimental architecture was meaningful. It leads to show advantages in software modifiability. In the control architecture, the code line number was slightly greater but not significant. It could be explained because the experimental architecture demands separate personalisation strategies in different components, and adds new modules that demand the creation, and modification of new files and methods.

Finally, we have tested the experimental architecture against a control architecture, but it is necessary to extend the experiment to contrast the number of changes when the enterprise has more than one application to be adapted with personalisation strategies. We believe that in that scenario, the benefits from *PMAdmin* could be more visible.

7 Conclusions and future work

In this paper, we have proposed a reference architecture for personalised web applications having the software modifiability as the main architectural drive. This reference architecture uses software component weaving as an alternative to tackle the challenges of including personalised behaviour into web applications.

The reference architecture proposes the use of three main modules: a *PC module* designed to diminish the complexity of weaving specialised software components, and to coordinate the personalisation actions to be executed by the system. *PMAdmin connector* serves as single communicator with a third module: an external *PMAdmin* module. It is in charge of administrating personal information, context and user group information. *PMAdmin* is a separate application, in this way, other personalisation systems in the same enterprise could use it; reducing the development time and effort for each new personalised application. A methodology to apply the reference architecture has five steps: identify personalisation strategies; add modules to support personalisation; define a data model to *PMAdmin* module; define interfaces; and weaving components. We executed a controlled experiment to validate the proposal in which we compared five change scenarios implemented under two architectures, experimental and control architecture. The change scenarios were derived from a real Brazilian e-commerce enterprise. The implementation of the change scenarios under two architectures allowed us to identify the benefits in complexity of integration of personalised behaviour in a web personalised application. In spite of that, we cannot guarantee that our reference architecture will always provide similar results on other domains and applications. A complete validation of our approach is part of our future work. In particular, it is necessary to test more scenarios, experiment with different domains such as e-health or e-learning; and integrate model driven development (MDD) approaches and technologies to automatically derive the web applications code. This paper does not address other important issues, such as, the downstream economic benefits of using the reference software architecture for developing personalised web applications. For example, one could raise the question ‘How does fast and personalised web development really benefit software engineering at large?’ ‘How much does it cost to do it early on as compared to later on?’ These complex issues have yet to be investigated.

Acknowledgements

The authors would like to thank VTEX enterprise for providing useful information for this work. This research was supported by University of Antioquia through the committee for the research development – CODI (CODI-PRG13-2-01).

References

Alotaibi, M.B. (2013) 'Adaptable and adaptive e-commerce interfaces: an empirical investigation of user acceptance', *Journal of Computers*, Vol. 8, No. 8, pp.1923–1933.

Bass, L., Clements, P. and Kazman, R. (1998) *Software Architecture in Practice*, S.E. Institute, ed., Addison-Wesley Longman, Reading, MA.

Bengtsson, P. et al. (2004) 'Architecture-level modifiability analysis (ALMA)', *Journal of Systems and Software*, Vol. 69, Nos. 1–2, pp.129–147.

Brambilla, M. and Butti, S. (2014) 'Quince años de Desarrollo Industrial Dirigido por Modelos de aplicaciones Front-End: desde WebML hasta WebRatio e IFML', *Novática: revista de Asociación de Técnicos de Informática.*, p.36.

Brusilovsky, P. (1996) 'Methods and techniques of adaptive hypermedia', *User Modeling and User-Adapted Interaction*, Vol. 6, Nos. 2–3, pp.87–129.

Brusilovsky, P. (2001) 'Adaptive hypermedia', *User Modeling and User-Adapted Interaction*, Vol. 11, Nos. 1–2, pp.87–110.

Brusilovsky, P. and Nejdl, W. (2004) 'Adaptive hypermedia and adaptive web', in Singh, M.P. (Ed.): *Practical Handbook of Internet Computing*, CRC Press.

Ceri, S. et al. (2007) 'Model-driven development of context-aware web applications', *ACM Trans. Internet Technol.*, Vol. 7, No. 1, pp.387–413.

De Virgilio, R. (2012) 'AML: a modeling language for designing adaptive web applications', *Personal Ubiquitous Comput.*, Vol. 16, No. 5, pp.527–541.

Di Lucca, G.A. et al. (2004) 'Towards the definition of a maintainability model for web applications', *Proceedings. Eighth European Conference on Software Maintenance and Reengineering, 2004: CSMR 2004*, pp.279–287.

Fernández, I.,G., Gomez, J. and Houben, G-J. (2010) 'Specification of personalization in web application design', *Information and Software Technology*, Vol. 52, No. 9, pp.991–1010.

Fernández, I.G (2008) *A-OOH: Extending Web Application Design with Dynamic Personalization*, University of Alicante.

Fernández, I.G. et al. (2009) 'Una Aplicación basada en Eclipse para la Personalización de Aplicaciones Web Dirigida por Modelos', *XIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2009)*, 8–11 September 2009, pp.363–366, San Sebastián, Spain.

Karat, C.M. et al. (2003) 'Personalizing the user experience on ibm.com', *IBM Systems Journal*, Vol. 42, No. 4, pp.686–701.

Kazman, R. and McGregor, J., (2012) 'A mashup of techniques to create reference architectures', *SATURN Conference 2012, Architecture and Process*, 7–11 May 2012, St Petersburg, FL, p.22.

Kęsik, J. and Żyła, K. (2010) 'Usability comparison of WebRatio and symfony for educational purposes', *Prace Instytutu Elektrotechniki, Z.*, Vol. 247, No. 1, pp.223–240.

Kwon, K. and Kim, C. (2012) 'How to design personalization in a context of customer retention: who personalizes what and to what extent?', *Electronic Commerce Research and Applications*, Vol. 11, No. 2, pp.101–116.

Martinenghi, D. (2014) 'How to model user and group management', *WebRatio* [online] http://my.webratio.com/learn/learningobject/how-to-model-user-and-group-management-v-72?link=oln72ae.redirect&pcp1x=how-to-model-user-and-group-management-v-72&nav=14&cbck=wrReq58593 (accessed 13 May 2015).

Oman, P. and Hagemeister, J. (1992) 'Metrics for assessing a software system's maintainability', *Proceedings Conference on Software Maintenance 1992*, pp.337–344, IEEE Comput. Soc. Press.

Reed, P. (2002) 'Reference architecture: the best of best practices', *IBM*.

Software Engineering Institute (SEI) (n.d.) 'Glossary' [online] http://www.sei.cmu.edu/architecture/start/glossary/ (accessed 1 September 2015).

Stafford, J.A. and Wolf, A.L. (2001) 'Architecture-level dependence analysis for software systems', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 4, pp.431–451.

Stella, L.F.F., Jarzabek, S. and Wadhwa, B. (2008) 'A comparative study of maintainability of web applications on J2EE, .NET and Ruby on rails', *In Proceedings – 10th IEEE International Symposium on Web Site Evolution, WSE 2008*, pp.93–99.

Tekinerdogan, B. (2004) 'ASAAM: aspectual software architecture analysis method', *Proceedings Fourth Working IEEE/IFIP Conference on Software Architecture, 2004: WICSA 2004*, pp.5–14.

Wohlin, C. et al. (2012) *Experimentation in Software Engineering*, Springer-Verlag, Berlin Heidelberg.

Notes

1 The architecture and the implementation of the remained scenarios are available on http://telesalud.udea.edu.co/Ecommerce/.

2 The table with a detailed description of changes in the experimental and control architectures are available on http://telesalud.udea.edu.co/Ecommerce/.