A Model-driven Approach for Empowering Advance Web Augmentation
From Client-side to Server-side Support

Matias Urbieta[1,2], Sergio Firmenich[1,2], Pedro Maglione [1], Gustavo Rossi[1,2]
and Miguel Angel Olivero[3]

[1] *Facultad de Informática, Universidad Nacional de La Plata, calle 50 y 120,1900, La Plata, Buenos Aires, Argentina*

[2] *CONICET, La Plata, Argentina*

[3] *Web Engineering and Early Testing Group, Computer Languages and Systems Department, University of Seville ETSII, Avda. Reina Mercedes S/N, 41012, Seville, Spain*

Keywords: Model-driven Web Engineering, Augmentation, End-user Development, Separation of Concern.

Abstract: Websites augmentations have been adopted as a mean for improving the User Experience of applications that often are not owned by the user. The augmentations alter the page in order to add, modify and even remove its content pursuing the satisfaction of a user's need. However, these augmentations are limited to page modification or transcluding content from another site on Internet. Moreover, advance server-side based augmentations have been released only by developers because of the required technical skill for the task. In this work, we have presented a novel approach for designing Web Augmentation coping client-side and server-side using a Model-Driven Web Engineering approach. The approach rises the abstraction level for server-side developments allowing end-users to design, and even implement the new functionalities. Additionally, the approach uses advance separation of concern principles thus we provide a set of tools for designing the composition of the core application and the augmentation. We show as running example an augmentation that introduces a site community's review support upon an agriculture e-commerce site.

1 INTRODUCTION

Web applications are often designed and developed considering a small subset of stakeholders (managers, internal users, product owners, and developers) without an active listening of the crowd's feedback and awareness of their needs. The emergence of Web personalization allowed introducing improvements to an application that runs as a black box just considering those perceivable behaviors by the end-user. Personalization usually is tied to recommender systems, which basically recommend information items to users according to their user profiles (Brusilovsky et al., 2007). However, beyond the information displayed, other approaches adapt the User Interface (UI), such as the Adaptive Navigation Support defined by Brusilosky (Brusilovsky, 2007). When these mechanisms are chosen directly by users, then some authors called them customizations (Aoki and Nakajima, 1999), that basically are systems allowing users to customize how the application displays contents and functionalities. In spite of which mechanisms (personalization, customization, etc.) that a particular application supports, it is not realistic to determine that the application idealized by a product owner covers every single users need. In this sense, users may have unsatisfied requirements. To tackle this problem a mechanism, very used nowadays, is to alter Web page once these are loaded on the client-side. By manipulating the Web site's Document Object Model (DOM), the user would perceive a variation of the Website that may add, remove or change both contents and functionalities. This technique, called Web Augmentation (Díaz and Arellano, 2015), is very popular and is commonly deployed as Web browser extensions that package software artifacts able to access these DOMs and altering it by using its interface.

For the sake of understanding, consider the example of a simple but real augmentation artefact, called Magic Actions for Youtube[1]. Magic Actions is a browser extension (with more than one hundred thousand users) designed to augment Youtube with functionalities that originally are not supported by this application, such as adaptive layout, volume control by

[1] Magic Actions for Youtube https://goo.gl/AkFnOj

Urbieta, M., Firmenich, S., Maglione, P., Rossi, G. and Olivero, M.
A Model-driven Approach for Empowering Advance Web Augmentation - From Client-side to Server-side Support.
DOI: 10.5220/0006394604440454
In *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*, pages 444-454
ISBN: 978-989-758-246-2

scrolling, etc. Note that, at this point, we are introducing the idea of adapting or customizing third-party Web applications. Web augmentation has an interesting combination of real use from part of the users crowd of Web applications and some research works aiming at conducting this activity promoting good software development practices, such as reuse (Garrido et al., 2013), robustness (Díaz et al., 2010), etc.. Most of these approaches rely on client-side, i.e. without the need of a back-end application for performing the augmentation effect. However, since this architecture limits the power of the augmentation because it does not profit from collaborative features (nowadays a *de facto* requirement) and the limited resources provided by the browser (i.e. processing power, storage alternatives, etc.), other approaches have a traditional client-server architecture, allowing, for instance, the synchronization of devices to support distributed user interfaces (Firmenich et al., 2016c), the use of complex services that cannot be deployed only on client-side (such as the use of a recommender system (Wischenbart et al., 2015)), and social Web content management tools, such as Diigo (Diigo, 2017). The reader must note that all these back-end counterparts are dedicated applications specifically designed and deployed for the particular kind of augmentation, but, to our knowledge, there are not approaches considering both client and server sides in a more generic way. That is, the end-user only contribute designing client-side improvements keeping him-self excluded from contributing complex behavior at server-side because the lack of coding skills that let him face technical challenges (e.g. Database access, complex algorithms design and coding, and User Interfaces component definition).

Current trends in agile software development (Martin, 2003) rely on quick prototyping since it helps to validate easily whether a new requirement meets the users' needs. For example, agile approaches promote the incremental development of Minimum Viable Product (MVP) (Bosch et al., 2013; Torrecilla-Salinas et al., 2015) delivered in a few Sprints considering the value-based contribution to the business. Web augmentation approaches give support partially if a development team wants to obtain a running example of a User Experience improvement or the result of transcluding other website's content. By using Web pages' augmentations it is possible to perform advance A/B testing without the need of branching current version of an application, coding a new feature, and build and release the application. This paper presents a Web Augmentation modeling approach contemplating a client-server application that hides the back-end complexity to users. Existing studies shows that end-users are able to create some simple artifacts using client-side Web technologies (HTML, CSS, and JavaScript) (Scaffidi et al., 2006), but there are not similar reports for managing the logic on the server-side. One manner to reduce the associated complexity to this task is to rise up the level of abstraction required to specify this logic. With this in mind, we propose the use of existing Web modeling languages that may be used in a dedicated server to create Web augmentation back-end counterparts increasing abstraction level and reducing development effort (DDway, 2016). On client-side, we propose a specialized end-user development tool that allows them to recreate an object model of the target application (the one being augmented) by abstracting Web contents. As we will discuss later, applications owners could use our ideas and supporting tools also in order to weave new functionalities without the need of modifying the applications core. In this paper, we present a novel approach to implement complex Web augmentation based on client-side improvements supported by server-side developments. The main contributions of our approach are: (1) a Model-driven approach for modeling augmentation that is not already available on the Internet, (2) a full support for the seamless introduction of augmentation, and (3) a development process that combines WOA tool and WebRatio (WebRatio, 2017) platform for supporting our approach . The paper is organized as follows. Section 2 describes the background. Then, the Section 3 introduces the related works. Section 4 is an overview of the approach. Section 5 shows a comprehensive example. And Section 6 concludes and talks about the future works.

2 BACKGROUND

2.1 Web Application Augmentation

As we mentioned before, Web augmentation is actually used by the users crowd. Besides of Web browser stores, where thousands of extensions for adapting existing Web content may be found, there are some of these tools supported by communities where end-users and other stakeholders with programming skills interact in the creation, sharing and improvement of artifacts. For instance, in the userstyles community (http://userstyles.org) users share artifacts that augment Web sites by adding further CSS designs that may change any aspect of Web sites content presentation. Similarly, in repositories behind userscripts communities (such as greasyfork - https://greasyfork.org/) artifacts written in JavaScript

may be found. In these cases beyond simple content presentation (such as the achieved with userstyles), Web sites may be augmented with new functionalities, given the power of JavaScript. In these communities, in spite of the artifact kind, there is a dependency between users with and without programming skills. Then, some research works proposed End-User Development (EUD) approaches to let users specify their own augmentation artifacts, these are discussed in the related work section.

2.2 Model-driven Web Engineering Approaches

In most mature Web design approaches (Rossi et al., 2008) , such as UWE, WebML, UWA, Hera, OOWS or OOHDM, a Web application is designed with an iterative process comprising at least conceptual and navigational modeling. According to the state-of-the-art of model-driven Web engineering techniques (Aragón et al., 2013; Domínguez-Mayo et al., 2014), these 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.

2.2.1 Conceptual Design

The conceptual model of a Web application (a.k.a., application, domain, or content model) is focused on defining the contents of the application with their attributes and associated behavior. When it is defined using the OOHDM method (or others such as UWE), this model is an object-oriented model described with UML and comprised of classes, with their attributes and methods, and associations between classes.

2.2.2 Navigational Design

The navigational design of a Web application is aimed at defining views, access structures and navigation paths to contents in order to enable the user easily accessing and navigating them. Most of the Web engineering methods base their navigational model on two modeling primitives, namely Node and Link. IFML (Brambilla and Fraternali, 2014) is no exception to this, as it defines Pages as logical views on application model classes and links as the hypermedia realization of application model associations that defines either browsing capabilities or triggering system behavior.

2.2.3 Abstract Interface Design

In OOHDM, the user interface is specified using Abstract Data Views (ADVs) (Vilain et al., 2000) which support an object-oriented model for interface objects. An ADV is defined for each node class, to indicate how each node attribute or sub-node (if it is a composite node) will be presented to the user. An ADV can be seen as an Observer (Gamma et al., 1995) of the node expressing its perception properties, in general, as nested ADVs or primitive types (e.g. buttons). Using a configuration diagram (Vilain et al., 2000) we express how these properties relate with the node attributes and operations.

ADVs are also used to indicate how interaction will proceed and which interface effects take place as the result of user-generated events. These behavioral aspects are specified using ADV-charts (Vilain et al., 2000), a kind of statecharts representing states and states transitions for a given ADV. ADV-charts are useful when we need to model rich interface behaviors such as that of Rich Internet Applications (RIA) (Niederhausen et al., 2009). ADV-Charts are state machines diagrams that allow expressing interface transformations occurring as the result of the user interaction on a given ADV. ADV-Charts describe interface behaviours through Event-Condition-Action rules.

3 RELATED WORKS

3.1 Web Application Augmentation

End-User Development (EUD) was explored in the field of Web Augmentation in order to let users without advanced programming skills to specify their own augmentation artifacts (Díaz and Arellano, 2012; Bosetti et al., 2017). In previous work we presented some WOA (Web Object Ambient) tool that allows users to recreate an object model of Web sites on the client-side (Firmenich et al., 2016d), however, since we are focused on supporting complex augmentation applications that could not be work just on the client-side, at the same time that with some modeling skills, they can create the back-end counterpart of these augmentations. This generic back-end support will empower augmentation artifacts, given that further features could be contemplated such as more complex business logic, storage, social aspects. Although several aspects about augmentation have been addressed through modeling activities, such as requirement specification (Firmenich et al., 2016a), these usually aim to model the presentation layer,

something that is not enough to represent the back-end logic of the application. In this sense, this paper propose an integration between our tool for extracting a model object from existing Web pages (Firmenich et al., 2016d) with an application that is modeled using existing Web modeling languages. Web Augmentation could be considered a foreign client-side application mechanism, i.e. adapting existing and third-party Web sites on the client-side. However, Client-Side adaptation could be also be considered as a core concern during application design. An interesting approach that considers this aspect was presented before [Model-Driven Design of Web Applications with Client-Side Adaptation], focusing the use of this mechanism in an e-learning system. However, although this work also propose a modeling layer for the client-side adaptation layer, it is not used from the augmentation point of view, but that is defined by application owners, and one more time, this could be part of what end-users would like to change.

3.2 Separation of Concern in MDWE

Several existing Model-Driven Web Engineering (MDWE) allows to seamlessly compose Web applications' concerns such as our previous work to tackle Volatile Functionalities in Web Applications (Urbieta et al., 2012b; Frajberg et al., 2016), modelling sepparation of concerns in Workflows (Urbieta et al., 2012a), and supporting separation of Web-GIS concerns in Web applications (Urbieta et al., 2014). Additionally, other approaches support evolutive requirements such as WebComposition Process Model (Gaedke and Gräf, 2001), Distributed Concern Delivery (Cerny et al., 2015) or, more general principles, such as refactoring (Fowler et al., 1999) and patterns (Gamma et al., 1995). However, all of these approaches are focused to compose concerns at server side only when the developer has access to the core application which is not the case of Web augmentation's purpose. Web

4 OUR APPROACH IN A NUTSHELL

Our approach is based on the idea that even the simplest functionality (e.g., a new community comment feature) 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 separated from the host site (from now on core application) and as much as possible decoupled from that of core and stable functionalities which the augmentations can not be introduced because the augmenter analyst is not part of the Application Webmaster team. Building on the above ideas, our approach can be summarized with the following design guidelines, which are shown schematically in Figure 1:

1. We decouple the augmentation from core application by introducing a design layer (called Augmentation Layer), which comprises a conceptual model, a navigational model, and an interface model.
2. We capture the basic conceptual model by tagging data in host application pages using (Firmenich et al., 2016d). The lack of access to the host application's underlying models requires the perceivable conceptual model extraction. In this process, data elements in the page are tagged and grouped into an entity definition by an Augmentation analyst in such a way a simplified conceptual model is obtained. The augmentation analyst is a skilled end-user with advance knowledge of Web Application who has as goal to improve the core application. In further steps, the model instantiation in a particular user session will be used for giving contextual information to the augmentation engine by providing model instances information when triggering the augmentation.
3. Augmentation requirements are modeled using Web engineering notations (e.g., use cases, user interaction diagrams, etc.) and separately mapped onto the following models using the heuristics defined by the design approach (See for example (Popovici et al., 2002)). Notice that, as shown in Figure 1, augmentation requirements are not integrated into the core requirements model, therefore leaving their integration to further design activities.
   (a) New behaviors, i.e. those which belong to the Augmentation layer, are modeled as first class objects in the Augmentation conceptual model. It defines all the objects and behavior corresponding to the new requirements. Additionally, this model may include core application conceptual classes (captured in step 2) perceivable by the end-user allowing defining relationships between augmentation business model and the core application. Notice that this strategy can be applied to any object-oriented method, i.e., any method using a UML-like specification approach.
   (b) Nodes and links belonging to the augmentation navigational model may or may not have links to the core navigational model. The core navigational model is also oblivious to the aug-

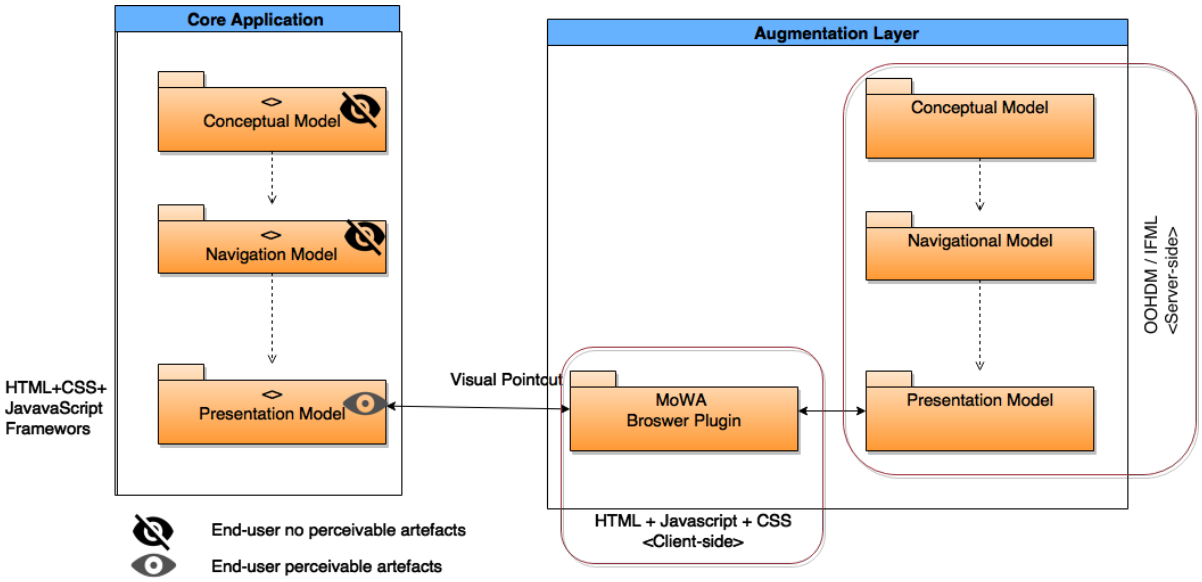

Figure 1: Approach schema.

mentation navigational classes, i.e., there are no links or other references from the core to the augmentation layer. This principle can be applied in any Web design approach.

(c) We use a separate integration specification to specify the connection between core and augmentation nodes. As we show later in the paper, the integration is achieved at run time as part of a client side weaving engine. In other model-driven approaches, the integration can be performed during model transformation by implementing the corresponding transformations.

(d) We design (and implement) the interfaces corresponding to each concern (core and augmentation) separately; the interface design of the core classes (described in OOHDM using Abstract Data Views (ADV) (Vilain et al., 2000)) are oblivious with respect to the interface of augmentation concerns. As in the navigational layer, this principle is independent of the design approach.

4. Core and augmentation interfaces (at the ADV and implementation layers) are woven by executing an integration specification, which is realized using DOM transformations. Again, the idea of model weaving is generic and therefore the same result can be obtained using other technical solution.

Once the augmentation requirements are modeled in Step 3, the augmentation of another site will be quite straightforward requiring to find and map the virtual classes (Step 2) and to specify how to wave the augmentation UI artifacts (Step 4).

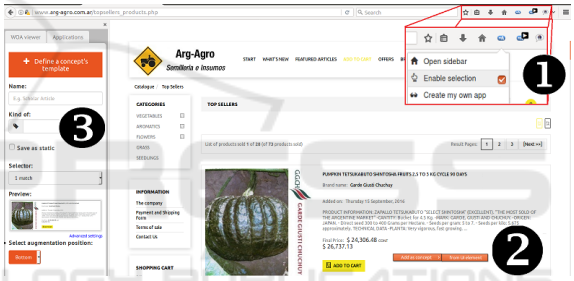

Figure 2: Concept definition.

We next explain how these principles have been put into practice in the OOHDM approach.

4.1 Core application Conceptual Model extraction (Step 2)

Our approach is based on the creation of objects that are specified by abstracting Web pages contents (Firmenich et al., 2016b).

In order to do it, we developed a visual programming tool that allows users to select a DOM element from which the abstraction process starts. For this DOM element, the user must define which is the conceptual class, which are the properties of that class and also to select from which children DOM elements are the values for these properties taken from. This process is shown in Figure 2, our tool adds the necessary controls that let users creating objects, no matter what Web resource has been loaded in the browser.

The first step is enabling the DOM selection (Step 1). By clicking this option, every DOM element is highlighted on a mouse-over event, so the user can

clearly appreciate what is the current target element to collect. Then, as shown in step 2, he can access via a context menu to the options for extracting an element in the current DOM. Once the DOM element is selected, an UI form is opened at the sidebar, which lets the user selecting a name for the concept, a semantic tag, etc. Concerning the different instances that could be extracted from a single Class, a combo is filled with different XPaths applicable to the selected element and allows to unequivocally reference it or to reference a set of similar elements instead. Then, the user may choose one or more elements, according to his needs. Then to select one of the possible selectors in the DOM, so, e.g. he can choose multiple DOM elements by changing the selector. Properties can be added in the same way; the only difference is the addition of a combo for linking such property to an existing concept. The result of this process is the definition of a set of classes specifications which allow obtaining one or multiple instances according to the selector the user has chosen during the authoring process: if it refers to a single element in the DOM or to several of them Once finished the abstraction process, users may see the collected classes and instances viewer panel, from where the use may also export the specifications in JSON format. Further aspects of the abstraction of Web contents as domain object may be found in previous work (Firmenich et al., 2016d).

4.2 Augmentation Layer Modelling

The augmentation request is triggered by a page access of core application. To provide the enhancement, an application server runs application models (Conceptual, Navigational and Interface) in order resolve the request. Next, we introduce the guidelines for designing each model.

4.2.1 Conceptual Model (Step 3a)

Augmentation functionalities like customized recommendation systems or complex concern may involve brand new content classes (e.g., the class modeling a Comment from a user) or the enhancement of existing core application classes and application behaviors. In our approach, a class diagram is designed defining new classes and the enrichments for virtual classes extracted in Step 2. The new classes are new pieces of information and behavior that must be introduced whereas the virtual classes are representations of the information actually perceivable by the end-user which are enhanced with new attributes (e.g. a Product may be enriched with an) or relationships with other virtual classes or augmentation ones. Any new feature of virtual classes will be woven automatically and must be considered as a Decorators (Gamma et al., 1995) because it allows adding new features (properties and behaviors) to an application in a non intrusive way. In our approach, augmentation functionalities might be new behaviors which are added to the conceptual model (and which might encompass many classes) or full-fledged navigation models, containing new nodes, links and even relationships with conceptual classes. Each augmentation functionality is treated as a self-contained sub-system and modeled using the OOHDM method. The notation is similar to symmetric approaches for separation of concerns such as the one described in (Rossi et al., 2008).

4.2.2 Navigational Design (Step 3b)

At the navigational layer, Augmentation and core navigational components are connected using an integration specification which indicates, for example, if the augmentation features are inserted in the core node or if they are connected with a hyperlink. This specification also includes a query indicating which core nodes will contain the extension. Nodes matching the query are affected by (or enhanced with) the augmentation functionality and represent the Affinity of the augmentation functionality. It is possible to define one or more affinities for the same augmentation functionality, i.e., the same functionality might be incorporated in different parts of the application, by following different rules. The affinities of an augmentation functionality are specified with the same query language used in OOHDM to define nodes (Rossi et al., 2006). The language is based on object queries. Using this query language the definition of an affinity assumes the following form:

```
AFFINITY: AffinityName
FROM C1..Ci
WHERE Predicate
INTEGRATION: Extension | Linkage(V1..Vi)
```

In it, AffinityName is the name associated with the affinity, *C1..Ci* indicate core node classes involved in the query, Predicate is a logical expression defined in terms of properties of model objects which determines the instances of the core node classes *C1..Ci* that will be affected by the augmentation functionality, and Extension / Linkage indicate the way the augmentation functionality is integrated into core nodes through the augmentation nodes *V1..Vi*. An extension indicates that the core nodes are enhanced to contain the new functionality information (and operations). In a linkage integration, the core nodes just allow navigation towards the augmentation nodes *V1..Vi* which actually contain the augmentation functionality, and

therefore does not support new behaviors. In the case of linkage integration, we can also specify additional features such as attributes or anchors that have to be added to the extended node (e.g., to make navigation more clear).

4.2.3 Structural Weaving of Augmentation (Step 3d)

As a consequence of inserting augmentation functionalities into the conceptual model or the navigational model, new interface elements must be added into the interface model, therefore introducing new fields with data or control interface objects (anchors, buttons, etc.). Though we described this process in (Ginzburg et al., 2009), we briefly review it here for completeness and readability reasons. Each concern (core and augmentation) will comprise ADVs for its corresponding nodes. During the interface design stage and when a node should exhibit some augmentation functionality, we indicate the look and feel of the final page by specifying how the augmentation interface will be inserted into the core ADV. More specifically, we indicate the relative position of the added interface objects with respect to the core interface objects. To express the integration, we have defined a simple specification language which allows indicating point-cuts and insertions at the abstract interface level, i.e. the position where the augmentation ADV has to be inserted in the core ADV. The specification generalizes the idea of point-cuts in aspect-orientation to the two dimensional space of Web interfaces. A point-cut and the corresponding insertion are specified using the following template:

```
Integration: IntegrationName
Target: ADVTargetName
Add: ADVSourceName | InsertionSpecification
Relative to: ADV name
Position: [above | bottom | left | right]
```

The field Integration is an identification for this specification. It may refer or not to a navigational affinity since the same User Interface (UI) integration specification can be used with many navigational affinities. The field Target indicates the names of the ADVs (one or more) which will host the augmentation interface code. Inner ADVs may be specified using a . notation. As an example, we will write Product.Reviews to indicate that the insertion will take place in the ADV Reviews, which is a part of the Product ADV. The Add field indicates which elements must be inserted in the target, either an ADV or an immediate specification which is used when the inserted field is simple enough to avoid the specification of another (auxiliary) ADV. Finally, we indicate the insertion position by using the Relative and Position fields. It is worth to notice that the specification is still abstract, thus leaving space to fine tuning during implementation.

5 A COMPREHENSIVE EXAMPLE

In this section, we show and describe an augmentation example performing all the steps proposed in the approach. For example, we expect to improve a Website that sells seeds for agriculture proposes. The application lacks an end-user community reviews features that let the end-users share their experience with a given product. In following steps we detail our approach in practice.

5.1 Extracting Conceptual Model from Core Application

The first step in the approach is to capture the underlaying model perceivable by the end-user that will be used later to be enhanced. For instance, in Figure 3 we highlight a particular part of the DOM that contains information about an instance of the class Product. Also, we can see that there other DOM elements presenting other instances of the same class. All of these instances seem to present similar information responding to different properties of the Product virtual class, such as the name, the price and a small description. The main idea is to define the class Product and its properties and how they are matched to existing DOM elements to extract concrete values for concrete instances.

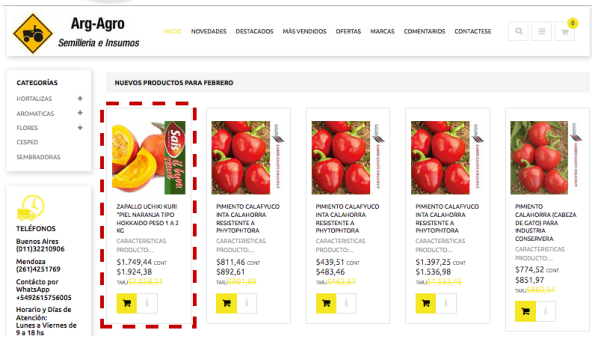

Figure 3: Concept and instance identification in existing Web content.

5.2 Modeling Augmentation Layer

In order to give support to the augmentation aspects that need to be performed on the server, we have designed a Web application using IFML which comprises Conceptual, Navigational and Interface models. The application gives backend support to the whole Augmentation experience.

Once the Virtual Classes are extracted, we design the conceptual model that documents how Augmentation enriches the core application model. In our example, a Product will be enriched with a set of Comments that represent community's reviews. In IFML, the first step is the description of conceptual mode using Entity/Relationship modeling. In Figure 4, we show how a Product will be associated with its comments. The Product virtual class which has doted lines is augmented with a reference multi-valuated to Comment entity. It is noteworthy that Comment entity does not need to state all the information available on the page; indeed, the basic set of information (e.g. any information that identify the object such as an id or a name) that let a basic lookup implementation is required.

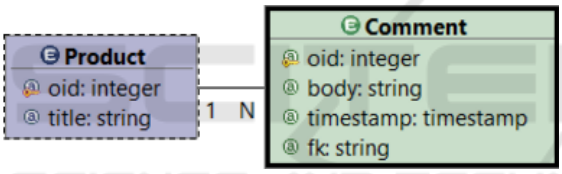

Figure 4: Entity relationship model.

Figure 5: Service represented on Web IFML.

Actions on client-side require, in addition to the conceptual model enhancement, improvements at navigational models to describe application behaviour. This is modeled using units, components, and links as part of an IFML site view. In Figure 5 a Page is defined listing all available Comments for a given product. Furthermore, a product's id is required for resolving the comments by browsing the relationship depicted in Figure 5. At the bottom of Figure 5 there is an URL responsible of triggering the server-side behaviour which sends the required id ˙pankins. Actually, the URL is triggered by the Augmentation plugin running in the browser once the user access the target page. The next step in the augmentation design is the specification of how to achieve the composition the core application navigational model with the augmentation. For such task, we use the affinities (introduced in Section 4.2.2). In the following specification, the affinity points out that the Product-Detail node (the one perceived by the end-user) must include all the behavior required for listing Product's comments:

```
AFFINITY: IncludeProductComment
FROM ProductDetail
WHERE Predicate
INTEGRATION: ProductComments
```

Finally, the designer must study the best experience to the end-user regarding the Comment list's presentation. The visual point-cut is specified using integration rules (presented at Section 4.2.2):

```
Integration: Community review
Target: ProductDetail
Add: ProductComments
Relative to: ProductTitle
Position: bottom
```

This determines the user interface point-cut and corresponding transclusion position to be done. Each time a user access a page, the captured augmentation model instance, in this case a Product, is sent to the server for its processing and output rendering. The outcome will be transcluded into the core application page using the configuration set. As result of augmentation execution on client-side, the web page is modified listing the product's comment as expected. In Figure 6, the reader can appreciate how the the comment block is inserted above product title. The weaving of pages content is achieved by means of the Web Object Ambient (WOA) Web browser extension (Firmenich et al., 2016b) which will be who calls the service and perform the augmentation on the client-side based on users configuration. It is important to highlight that special care must be taken in client side object identification for matching with persisted object, otherwise, you run the risk of not finding any result and augmentation ends without effect.

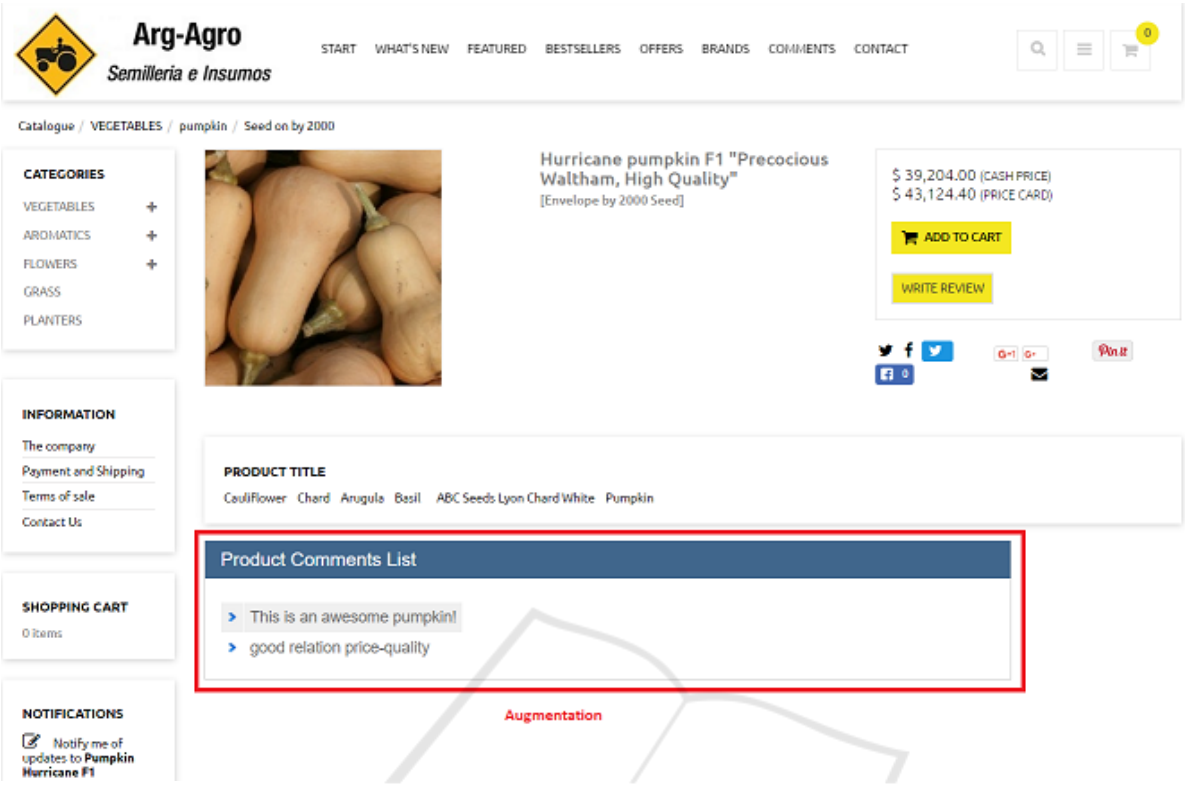

Figure 6: Augmentation result.

5.3 Implementation

The architecture is composed of two parts, on one hand there is a Web service which is intended to be public showing how to interact with it, specifying in and out parameters, and designed by WebRatio IDE to take advantage of the benefits it brings on the ease of its use for end user. This service should be part of the communitys available augmentation options to make it richer and create a collaborative context.

There are some client-side restrictions that have been taken into account. One is Cross-origin resource sharing (CORS)[2] because it can not be assumed that its allowed to make a call to other domain from the web where extension is running on. To resolve that problem, a proxy is used which wraps the call adding headers to enable cross domain calls by the insertion of request headers, that way calls can be made to bring augmentation services data. Furthermore, the script is programmed in basic JavaScript to not have dependencies in any possible client side library, like jQuery[3].

6 CONCLUSIONS

In this work we have presented a novel approach for designing Web Augmentation coping with client-side and server-side behaviors. The augmentations are modeled using IFML but our approach can be instantiated with other approaches (Rossi et al., 2008) such a UWE and OOHDM. The approach relies on the separation of concerns principles thus we provide the composition mechanism for each model (Conceptual, Navigational and User interface). We used as running example the design of a Community review feature instantiated, but not restricted to, at an agriculture e-commerce site. The new feature gives information to the decision making activities. We plan to perform an evaluation for assessing the end-user perception about the augmentation and how it improves his site's experience. On the other hand, we will study how to improve the reuse of the augmentations. We also consider to evaluate the approach based on a quality framework for MDWE approaches

[2]Cross-origin resource sharing is a mechanism that allows restricted resources (e.g. fonts) on a Web page to be requested from another domain outside the domain of which the first resource was served.

[3]jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

(Domínguez-Mayo et al., 2012). This will help to improve our approach's processes that let applying and designing efficiently. Finally, we will study the use of Model-Driven Web Augmentation in diverse business domains such as Financial, Banking, Logistic,etc.. Additionally, we instantiate our approach using OOHDM and UWE among other approaches.

ACKNOWLEDGEMENTS

Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015.

Additionally, this research has been partially supported by the POLOLAS project (code TIN2016-76956-C3-2-R) of the Spanish Ministry of Science and Innovation

REFERENCES

Aoki, Y. and Nakajima, A. (1999). User-Side Web Page Customization. In *Human-Computer Interaction: Ergonomics and User Interfaces, Proceedings of {HCI} International '99 (the 8th International Conference on Human-Computer Interaction), Munich, Germany, August 22-26, 1999, Volume 1*, pages 580–584.

Aragón, G., Escalona, M. J., Lang, M., and Hilera, J. R. (2013). An analysis of model-driven web engineering methodologies.

Bosch, J., Olsson, H. H., Björk, J., and Ljungblad, J. (2013). The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups. *Lean Enterprise Software and Systems*, pages 1–15.

Bosetti, G., Firmenich, S., Gordillo, S. E., Rossi, G., and Winckler, M. (2017). An End User Development Approach for Mobile Web Augmentation. *Mobile Information Systems*, 2017:1–28.

Brambilla, M. and Fraternali, P. (2014). *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann.

Brusilovsky, P. (2007). Adaptive Navigation Support. In *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 263–290.

Brusilovsky, P., Kobsa, A., and Nejdl, W. (2007). *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*. Springer.

Cerny, T., Macik, M., Donahoo, M. J., and Janousek, J. (2015). On distributed concern delivery in user interface design. *Computer Science and Information Systems*, 12(2):655–681.

DDway (2016). Calculation of the Functional Size and Productivity with the IFPUG method (CPM 4 . 3 . 1). The DDway experience with WebRatio, http://www.webratio.com/website/documentation/Case_Study_Productivity_with_WebRatio.pdf. Last accessed March 29, 2017.

Díaz, O. and Arellano, C. (2012). Sticklet: An End-User Client-Side Augmentation-Based Mashup Tool. In *Web Engineering - 12th International Conference, {ICWE} 2012, Berlin, Germany, July 23-27, 2012. Proceedings*, pages 465–468.

Díaz, O. and Arellano, C. (2015). The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding. *TWEB*, 9(2):8.

Díaz, O., Arellano, C., and Iturrioz, J. (2010). Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. In *Web Engineering, 10th International Conference, {ICWE} 2010, Vienna, Austria, July 5-9, 2010. Proceedings*, pages 233–247.

Diigo (2017). Diigo, https://www.diigo.com/. Last accessed March 29, 2017.

Domínguez-Mayo, F., Escalona, M., Mejías, M., Ross, M., and Staples, G. (2012). Quality evaluation for Model-Driven Web Engineering methodologies. *Information and Software Technology*, 54(11):1265–1282.

Domínguez-Mayo, F. J., Escalona, M. J., Mejías, M., Ross, M., and Staples, G. (2014). Towards a Homogeneous Characterization of the Model-driven Web Development Methodologies. *J. Web Eng.*, 13(1&2):129–159.

Firmenich, D., Firmenich, S., Rivero, J. M., Antonelli, L., and Rossi, G. (2016a). CrowdMock: an approach for defining and evolving web augmentation requirements. *Requirements Engineering*, pages 1–29.

Firmenich, D., Firmenich, S., Rivero, J. M., Antonelli, L., and Rossi, G. (2016b). CrowdMock: an approach for defining and evolving web augmentation requirements. *Requirements Engineering*, pages 1–29.

Firmenich, S., Bosetti, G., Rossi, G., and Winckler, M. (2016c). Flexible Distribution of Existing Web Interfaces: An Architecture Involving Developers and End-Users. In *Current Trends in Web Engineering - {ICWE} 2016 International Workshops, DUI, TELERISE, SoWeMine, and Liquid Web, Lugano, Switzerland, June 6-9, 2016, Revised Selected Papers*, pages 200–207.

Firmenich, S., Bosetti, G. A., Rossi, G., Winckler, M., and Barbieri, T. (2016d). Abstracting and Structuring Web Contents for Supporting Personal Web Experiences. In *Web Engineering - 16th International Conference, {ICWE} 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 77–95.

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Frajberg, D., Urbieta, M., Rossi, G., and Schwinger, W. (2016). Volatile Functionality in Action: Methods, Techniques and Assessment. In Bozzon, A., Cudre-Maroux, P., and Pautasso, C., editors, *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 59–76. Springer International Publishing, Cham.

Gaedke, M. and Gräf, G. (2001). Development and Evolution of Web-Applications Using the WebComposition Process Model. In *Web Engineering*, volume 2016, pages 58–76.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*.

Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina-Medina, N., and Harari, I. (2013). Personalized Web Accessibility using Client-Side Refactoring. {*IEEE*} *Internet Computing*, 17(4):58–66.

Ginzburg, J., Distante, D., Rossi, G., and Urbieta, M. (2009). Oblivious integration of volatile functionality in web application interfaces. *Journal of Web Engineering*, 8(1):25–47.

Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

Niederhausen, M., Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., Meißner, K., Van Der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.-J., Meißner, K., and Matthias Niederhausen, Kees Van Der Sluijs, J. H. E. L. G.-j. H. K. M. (2009). Harnessing the power of semantics-based, aspect-oriented adaptation for AMACONT. In Gaedke, M., Grossniklaus, M., and Díaz, O., editors, *Web Engineering*, chapter Harnessing, pages 106–120. Springer, Berlin, Heidelberg.

Popovici, A., Gross, T., and Alonso, G. (2002). Dynamic Weaving for Aspect-oriented Programming. In *Proceedings of the 1st International Conference on Aspect-oriented Software Development*, AOSD '02, pages 141–147, New York, NY, USA. ACM.

Rossi, G., Nieto, A., Mengoni, L., Lofeudo, N., Silva, L. N., and Distante, D. (2006). Model-Based Design of Volatile Functionality in Web Applications. In *2006 Fourth Latin American Web Congress*, pages 179–188.

Rossi, G., Pastor, s., Schwabe, D., and Olsina, L. (2008). *Web Engineering: Modelling and Implementing Web Applications*, volume 12. Springer-Verlag London.

Scaffidi, C., Ko, A., Myers, B., and Shaw, M. (2006). Dimensions characterizing programming feature usage by information workers. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 59–64. IEEE.

Torrecilla-Salinas, C., Sedeño, J., Escalona, M., and Mejías, M. (2015). Estimating, planning and managing Agile Web development projects under a value-based perspective. *Information and Software Technology*, 61:124–144.

Urbieta, M., Oliveira, A., Araújo, J., Rodrigues, A., Moreira, A., Gordillo, S., and Rossi, G. (2014). WebGIS models: accomplishing modularity with aspects. *Innovations in Systems and Software Engineering*, 10(1):59–75.

Urbieta, M., Retschitzegger, W., Rossi, G., Schwinger, W., Gordillo, S., and Luna, E. R. (2012a). Modelling adaptations requirements in web workflows. *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services - IIWAS '12*, page 72.

Urbieta, M., Rossi, G., Distante, D., and Ginzburg, J. (2012b). Modeling, Deploying, and Controlling Volatile Functionalities in Web Applications. *International Journal of Software Engineering and Knowledge Engineering*, 22:129–155.

Vilain, P., Schwabe, D., and de Souza, C. S. (2000). A Diagrammatic Tool for Representing User Interaction in UML. In Evans, A., Kent, S., and Selic, B., editors, *International Conference on the Unified Modeling Language*, volume 1939 of *Lecture Notes in Computer Science*, pages 133–147. Springer.

WebRatio (2017). WebRatio Platform, http://www.webratio.com/site/content/en/web-application-development. Last accessed March 29, 2017.

Wischenbart, M., Firmenich, S., Rossi, G., and Wimmer, M. (2015). Recommender Systems for the People - Enhancing Personalization in Web Augmentation. In *Proceedings of the Joint Workshop on Interfaces and Human Decision Making for Recommender Systems, IntRS 2015, co-located with {ACM} Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 19, 2015.*, pages 53–60.