A Web-based Model-driven Platform for Web Augmentation

Matias Urbieta[1,2], Franco Mahl[1], Gustavo Rossi[1,2] and Gabriela Bosetti[1]
[1]*Facultad de Informática, Universidad Nacional de La Plata, calle 50 y 120,1900, La Plata, Buenos Aires, Argentina*
[2]*CONICET, La Plata, Argentina*

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

Abstract: 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. In spite of which mechanisms (personalization, customization, etc.) that a particular application supports; it is not realistic to state that any application, being idealized by few people (e.g., its owners), covers every single user's needs. In this sense, users may have unsatisfied requirements. Nowadays available Web augmentations are making full use of server-side capabilities for meeting requirements. We present a Web Augmentation modeling approach contemplating a client-server application that hides the back-end complexity to users. In this work we present a Web CASE tool to model server-side behavior for Web augmentation. This tool provides a full web-based experience for designing and running Web augmentations that requires client and server-side components.

1 INTRODUCTION

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. In spite of which mechanisms (personalization, customization, etc.) that a particular application supports; it is not realistic to state that any application, being idealized by few people (e.g., its owners), covers every single user's needs. In this sense, users may have unsatisfied requirements. To tackle this problem a mechanism, widespread 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, is widely adopted and is commonly deployed as Web browser extensions that package software artifacts able to access these DOMs and altering it by using its interface. Nowadays available Web augmentations are making full use of server-side capabilities for meeting requirements that demand high-performance computing (i.e. Similar Sites[1]), collaborative features –nowadays a *de-facto* requirement– (i.e. Evernote[2]), persistence (i.e. Last Pass[3]), data synchronization between devices. For the purpose of clarification, let's consider again the Grammarly extension; at the server-side end, the sentences are processed using Natural Language Processing which demands CPU resources, and storage for the text under processing. The reader must note that companies are using Web augmentations technique for enhancing their solutions which aim at providing an improved experience to the user and this increasing its adoption among users. 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

[1]Similar Sites, https://chrome.google.com/webstore/detail/similar-sites-discover-re/necpbmbhhdiplmfhmjicabdeighkndkn

[2]Evernote,https://chrome.google.com/webstore/detail/evernote-web-clipper/pioclpoplcdbaefihamjohnefbikjilc?hl=es-419

[3]Last Pass, https://chrome.google.com/webstore/detail/lastpass-free-password-ma/hdokiejnpimakedhajhdlcegeplioahd?hl=es-419

Urbieta, M., Mahl, F., Rossi, G. and Bosetti, G.
A Web-based Model-driven Platform for Web Augmentation.
DOI: 10.5220/0008559304770486
In *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pages 477-486
ISBN: 978-989-758-386-5

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 , the use of complex services that cannot be deployed only on client-side (such as the use of a recommender system, 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).

This paper relies on our previous work (Urbieta et al., 2017) that presents a Web Augmentation modeling approach contemplating a client-server application that hides the back-end complexity to users. 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 present a novel domain specific modeling languages that may be used in a dedicated server to create Web augmentation back-end counterparts extending our previous work (Urbieta et al., 2017). 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 to weave new functionalities without the need of modifying the application's core. In (Urbieta et al., 2017) we relied on IFML for designing server-side artifacts, and Web Object Ambient (WOA) modeling pages enhancements. Regarding the former, WebRatio (WebRatio, 2017) is the official tool for IFML which aids engineers to design and generates the running application meeting the designs. While the WOA tool is a Chrome plugin providing a natural ambient for extending the Web. The combination of tools results in a hybrid web-desktop environment. The experience feels awkward to the engineers as they must jump from one realm (Web) to other (desktop) and the other way around when developing an augmentation. The usage of Desktop-based tools like Webratio introduces additional effort than a Web-based one because it requires local resources to run the platform, and time to install the tool. Finally, IFML presents the benefit of modeling broad domain spaces but it also leaks the specificity for certain domain specific problems. This results in investing time to learn language elements that hardly ever will be used to develop an extension.

In this work we present a Web CASE tool to model server-side behavior for Web augmentation (Urbieta et al., 2017). This tool complements the current WOA tool providing a full web-based experience for designing and running Web augmentations that requires client and server-side components. The tooling will provide the following functionalities:

- Conceptual model design: Definition of the Entities and Relations diagram (domain model).
- Creation of the Database and tables derived from the Entity-Relationship conceptual model where entities are mapped to tables and relations to foreign keys (in the cases of relations *one to one* or *one to many*) or intermediate tables (in the cases of relations *many to many*).
- Navigation model. A domain specific design model to describe who Web augmentation are browsed as well as the actions that the user can perform. It is inspired in mature Web modeling languages such as IFML or OOHDM (Rossi et al., 2008) so the augmentation navigation is specified using Node/Link (frontend).
- The server-side component generation that allows rendering the navigation model by creating the pages with the elements contained in them: Lists, Forms and Scripts that will be described later in this article. The links between the pages will be rendered as hyperlinks between them.
- API REST generation that exposes endpoints dynamically created based on the navigation model. This will also let developers, with basic experience in JavaScript and API REST developments, create their own custom endpoints and thus send and consume server-side information from the client-side.

As described in the previous points, once the diagrams are obtained, it will be possible to create the database derived from the conceptual model, and create and render the modeled pages with their links and detailed functionalities for each one.

The main contributions of this work are: (1) a web based Model-driven platform for modeling augmentation (2) comprehensive examples . 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 future works.

2 BACKGROUND

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.

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), 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.

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., 2016b), 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 counter-part 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., 2016b) 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 (Ceri et al., 2004), focusing the use of this mechanism in an e-learning system. Although this work also proposes 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., 2012; Frajberg et al., 2016). 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.

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. 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.

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., 2016b). 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 the 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. 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. 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 augmentation navigational classes, i.e., there are no links or other references from the core to the augmentation layer. We design (and implement) the interfaces corresponding to each concern (core and augmentation) separately; the interface design of the core classes are oblivious with respect to the interface of augmentation concerns.
4. Core and augmentation interfaces 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).

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

5 INSTRUMENTING WEB AUGMENTATIONS

In this section, we present how to design a Web augmentation comprising the client-side and server-side behavior. As an example, we will use a Web augmentation that improves a website in the domain of agriculture that sells different types of seeds.

5.1 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., 2016a).

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

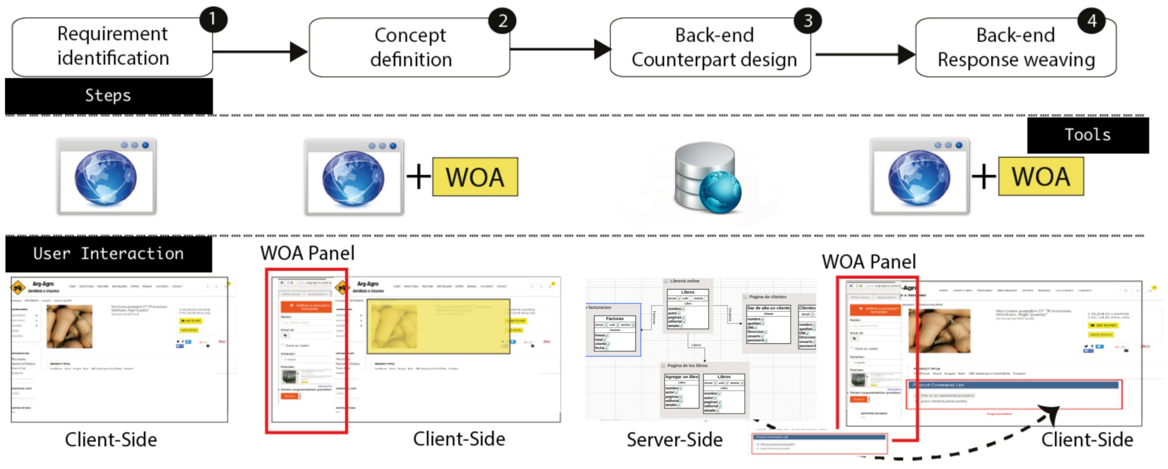


Figure 1: Approach schema.

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.

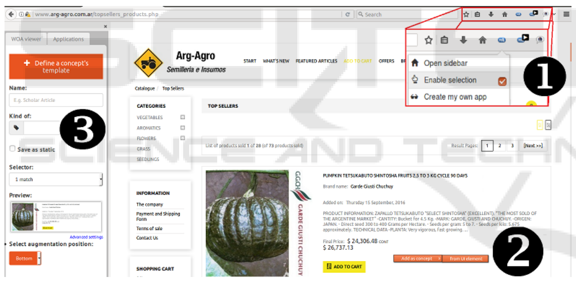


Figure 2: Concept definition.

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 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, a 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 several of them. In the Figure 3 the extraction of the conceptual model is presented which highlights a specific part of the DOM containing information about an instance of the class `Product`. Such class will be augmented allowing to record comments of the users of the site which will require to add a new relationship between `Product` and a new class `Comment`. Once finished the abstraction process, users may see the collected classes and instances viewer panel, from where the user 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., 2016b).

5.2 Modelling the Augmentation Layer using WeModelAug

The WeModelAug tool was designed with the specific objective of providing server-side support for Web Augmentations, i.e. development of conceptual and navigation diagrams. Prior to this tool, developers of enhancements used WebRatio (WebRatio, 2017) that presented several difficulties: it is a desktop tool that requires local resources and software requirements to run (a specific version of Java); it is a general-purpose solution that takes time to learn by software engineers. Finally it is a proprietary solution that does not allow to access the source code and thus it is not possible to customize the transformations.

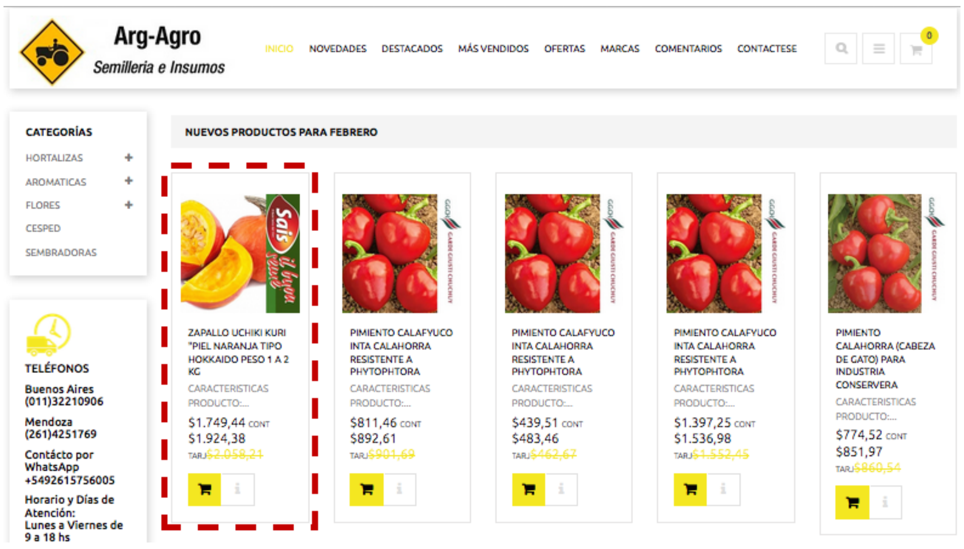

Arg-Agro
Semilleria e Insumos
INICIO
NOVEDADES
DESTACADOS
MÁS VENDIDOS
OFERTAS
MARCAS
COMENTARIOS
CONTACTESE
0
CATEGORÍAS
HORTALIZAS
AROMATICAS
FLORES
CESPED
SEMBRADORAS
TELÉFONOS
Buenos Aires (011)32210906
Mendoza (261)4251769
Contácto por WhatsApp +5492615756005
Horario y Días de Atención: Lunes a Viernes de 9 a 18 hs
NUEVOS PRODUCTOS PARA FEBRERO
ZAPALLO UCHIKI KURI "PIEL NARANJA TIPO HOKKAIDO PESO 1 A 2 KG
CARACTERISTICAS PRODUCTO:...
$1.749,44 CONT
$1.924,38
TARJ ~~$2.058,21~~
PIMIENTO CALAFYUCO INTA CALAHORRA RESISTENTE A PHYTOPHTORA
CARACTERISTICAS PRODUCTO:...
$811,46 CONT
$892,61
TARJ ~~$901,69~~
PIMIENTO CALAFYUCO INTA CALAHORRA RESISTENTE A PHYTOPHTORA
CARACTERISTICAS PRODUCTO:...
$439,51 CONT
$483,46
TARJ ~~$462,67~~
PIMIENTO CALAFYUCO INTA CALAHORRA RESISTENTE A PHYTOPHTORA
CARACTERISTICAS PRODUCTO:...
$1.397,25 CONT
$1.536,98
TARJ ~~$1.552,45~~
PIMIENTO CALAHORRA (CABEZA DE GATO) PARA INDUSTRIA CONSERVERA
CARACTERISTICAS PRODUCTO:...
$774,52 CONT
$851,97
TARJ ~~$860,54~~

Figure 3: Identifying objects in Web pages required for an augmentation.

To counteract these difficulties, WeModelAug was designed over the Web platform. That is to said, the WOA tool is built on top of the web browser and the server-side component designer is a Web app. So the Web Augmentation can be completely designed using a browser. This avoids the need for setting up locally the tools. On the other hand, it presents other benefits: it is inspired on IFML and UML, so it is easy to learn and start using by software engineers; it is also accompanied by a simple and user-friendly interface; it is open source so any developer can clone the repository[4] and create their version of WeModelAug.

To tackle the augmentation design, the web augmentation platform is divided into two sections: conceptual modeling to specify the entities manipulated by the augmentation server-side, and navigational or navigation modeling. In Figure 4 the metamodel for the augmentation specification is shown.

5.3 Business Domain Modelling

The tool assists in designing Entity-Relationship diagram that describes the business domain. If we recall our example, we have extracted the classes `Product` and `Comment` shown in the Figure 5. The tool implements the Object-Relational Mapping (ORM) feature so that the instances of the model are persisted transparently.

An entity attribute has a **type** (numeric, text, date and other types allowed by the underlying Database). The attribute flag **unique** determines that there cannot be two records with the same value in this attribute, and, if it is marked as **required**, a new class instance must provide value to such field. Otherwise, the instance will not be persisted prompting an error message.

In addition to the entities specification, it is possible to model, if necessary, the relations between entities. The relations between two entities can be one to one (1..1 - 1..1), one to many (1..1 - 1..*), or many to many (1..* - 1..*). In case of being optional, it is represented with a 0 instead of 1.

Once the conceptual diagram is finished, the creation of the corresponding database is allowed, where each entity will be mapped to a table and its attributes will be fields in that table. In addition, the relationships between entities will be represented as foreign keys between the tables or, in the case of many to many relationships, as an intermediate table. In our example the tables `Product` and `Comment` will be created where the `Comment` table will have a foreign key of `Product`.

5.4 Navigation Modelling

Once finished the conceptual model design and having specified the entities with their attributes and relations, it is possible to model the navigation of the Web application using pages and Links, the hyperlinks between them.

The pages are individual elements (nodes) containing other elements. A page can contain three different types of elements: **forms**, **lists** and **scripts**, in addition to the **links** to other pages.

[4]https://github.com/francomahl/sswmfa

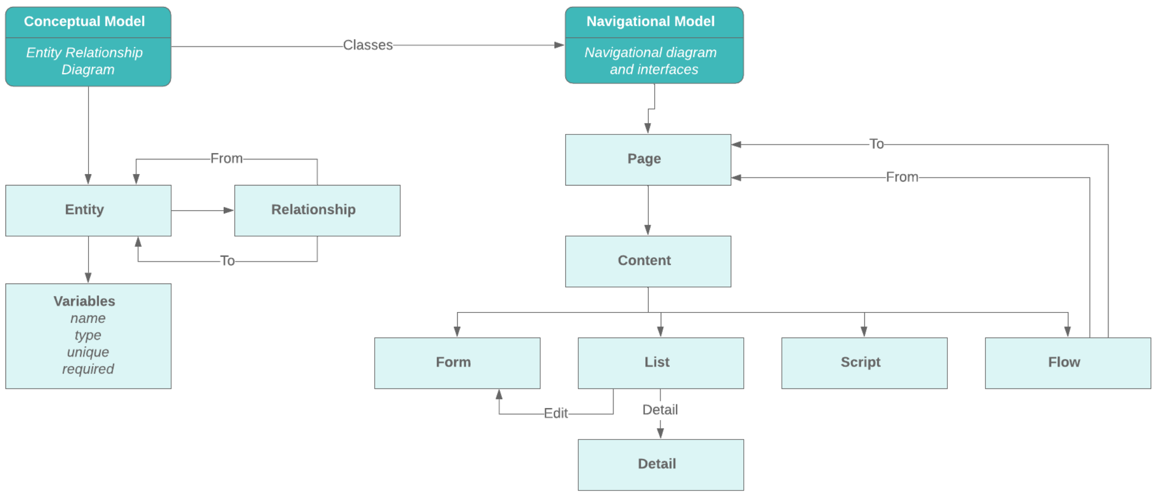


Figure 4: Metamodel for server-side artifacts.

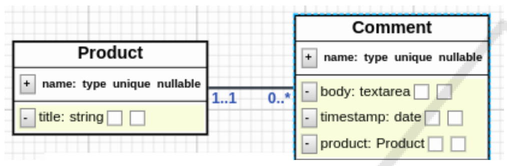


Figure 5: Entity-Relatinship diagram.

5.4.1 Forms

The Forms must be related to an (only one) entity in the conceptual diagram and are used to create or edit records that are persisted in the database.

The **record creation forms** are explicitly shown in the diagram within a page, whereas the **record editing forms** are created as a result of the ability to edit records in lists, as described in the following section. For example, we modeled in Figure 6 a node Page `Comments` that contains a comment form and a node Page `Product` with a product form. .

When using the relationship of the form element with the entity `Comment` of the conceptual model, the attributes of the entity will be automatically listed, allowing you to choose those that will be included as form fields.

Once the page `Comments` is rendered, you will get a page with the form `Add comment` that will be used to create new comments to be shown in the augmented site; the form fields will be the attributes specified in the model. The same will happen with the page `Products` that contains the Form `Add product`. Please note that the fields marked as required in the conceptual model will always be included in the form, otherwise the insertion or update of records in the database will fail. The required fields will have a special style in the view. In the example they appear with a * after the name and they will have the validation in the browser that provides HTML5.

5.4.2 Lists

Like forms, lists should relate to one, and only one, entity of the conceptual diagram by checking the attributes that will be shown as fields in the list. In our example you can see in the Figure 6 that the node Page `Comments` and `Products` have, besides the forms, two elements list `Comments` and `Products` respectively. These elements will be shown as lists containing the objects stored in the database when rendered. `Products` will show the products selected from the site as products to be augmented.

A List element must be placed within a page and has the following functionalities over the records of the related entity:

- List the records in HTML list form
- Delete records (optional). A button is shown for each record in the list that calls the backend by removing it from the database.
- Edit records (optional). This functionality is provided through a link for each record that redirects to a page that contains a record editing form with the current values preloaded in the form fields. This allows you to edit them and save the changes. Additionally when links are clicked, they trigger an action in the backend that updates the record in the database.
- Show record detail (optional). Enabling this feature displays a link for each record that redirects to a detail page listing all record fields with their values. This can be used if only some

fields are shown in the list and the rest of the record information is available in the detail page.

For the example we modeled a page that contains a form (creation) of comments and a list of the comments created, when rendering this page we obtain the form for insertions and the list modeled.

5.4.3 Scripts

The tool will provide the possibility of inserting a Script type element containing JavaScript code which, when rendering the model, will be executed when accessing the page that contains it.

5.4.4 Links

In the diagram, the pages (nodes) can be linked to each other with links (addressed links) specifying the source and destination pages. A link from one page to another is rendered as a link or hyperlink on the source page.

5.5 Rendering

Rendering is the last step in server-side web augmentation modeling. What is shown in Figure 6 will be rendered into HTML content. You can see in the Figure 7 the node page `Comments` rendered to HTML content showing the Form and the List with comments of the product to augment. Then, going back to the augmented site you can see that the comments are in the product.

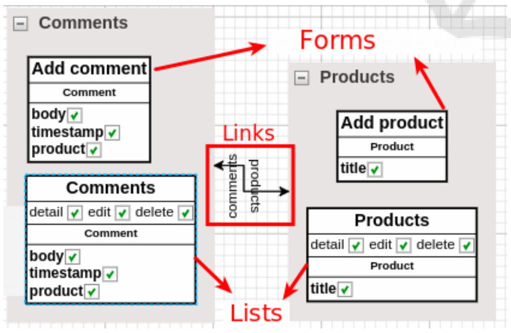


Figure 6: Navigation model.

Once the navigation has been modelled, the web pages can be rendered with the elements described in the previous section. The rendering process transforms models into HTML code of the pages. Having rendered the navigation model, it will be possible to navigate through the pages and interact with the database by creating (persisting), listing, editing and deleting records.

6 TECHNICAL DISCUSSION

At the time of devising a tool to assist Web augmentation design, it was thought as a web-based solution to abstract the developers from the operating system and achieve better portability of the artifacts.

The language chosen to develop WeModelAug was JavaScript with the framework NodeJS[5] and ORM Sequelize[6]. The library chosen for the development of the diagrams was GoJS[7].

JavaScript was chosen as the development language because it has several frameworks and a very large community of developers around the world and a very large documentation.

About the ORM we know that the database must be dynamically created and modified at runtime, so it must be possible to destroy it and generate it again quickly and securely. Moreover. it must also be able to support various database engines such as Mysql, Sqlite or PostgreSQL. **Sequelize** was the ORM chosen for its simplicity, its easy integration into a NodeJS project, dynamic query generation, and for the possibility of generating the Data Definition Language sentences to destroy and generate a database on the fly.

For modeling, there are several open source JavaScript libraries that can be used to make Entity-Relationship and Navigation diagrams .

GoJS is a JavaScript library that allows you to develop interactive diagrams and complex visualizations across different Web browsers and platforms. This library offers many functionalities for user interaction such as drag-and-drop, copy and paste, on-site text editing, tooltips, templates, data related models, palettes, preview, event handlers, the ability to represent diagrams in JSON format, among others.

GoJs is a pure JavaScript library, so users can interact with it without the need for the application to connect to a server or install additional plugins to the browser. It runs completely in the web browser and renders HTML5 Canvas or SVG images without the need for server-side requirements.

The fact that this library does not require a back-end and is easy to implement in any framework (even without one) made it the one chosen to develop WeModelAug.

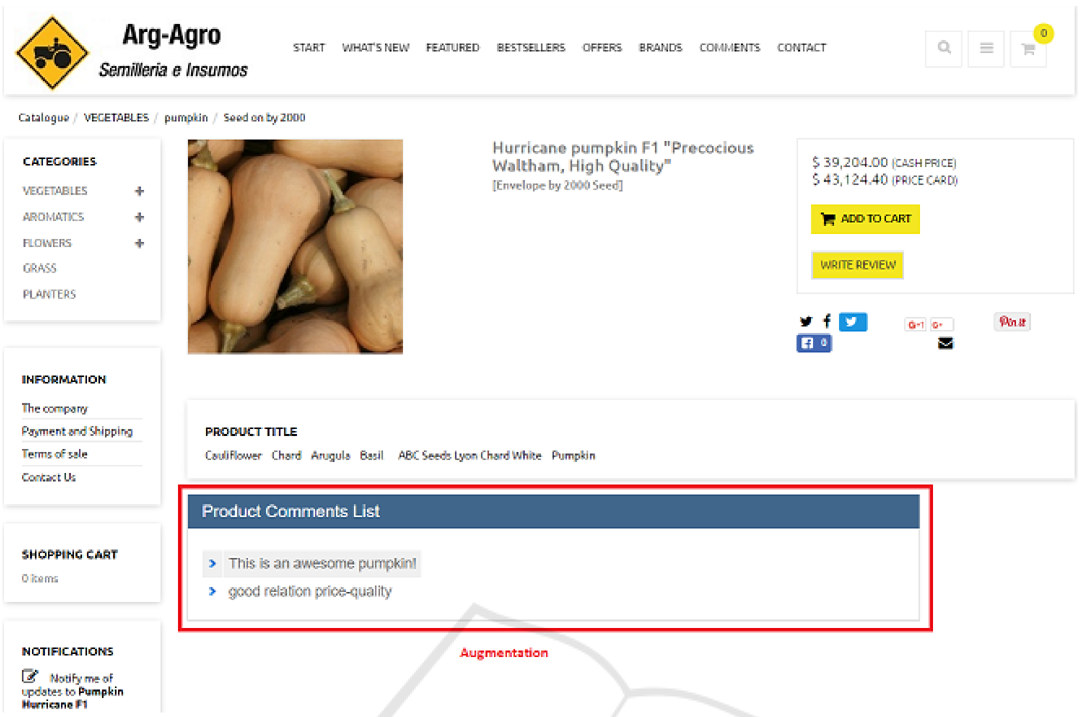
Finally, it was decided to include Docker Compose[8] to achieve a higher level of abstraction for developers. Docker provides the possibility to

[5]https://nodejs.org/es/

[6]http://docs.sequelizejs.com/

[7]https://gojs.net/latest/index.html

[8]https://docs.docker.com/compose/



Hurricane pumpkin F1 "Precocious Waltham, High Quality"
[Envelope by 2000 Seed]

$ 39,204.00 (CASH PRICE)
$ 43,124.40 (PRICE CARD)

Notify me of updates to **Pumpkin Hurricane F1**

- This is an awesome pumpkin!
- good relation price-quality

Figure 7: Site augmented.

run the WeModelAug code locally or in the cloud without the need to have the corresponding versions of JavaScript, Node, and other necessary packages since the required installations are done in a Docker container and the tool code runs there.

Once the development of WeModelAug was completed the code was uploaded to GitHub[9] with instructions on how to create the environment locally and how to run the application.

7 CONCLUSIONS

In this work we have presented a web-based platform supporting approach for designing Web Augmentation coping with client-side and server-side behaviors. The augmentations are modeled using a custom domain-specific language designed to abstract Web augmentations. We used as a comprehensive example instantiated illustrate the approach instantiation and how the tool support the study case. We plan to perform an assessment of the language usability.

[9]https://github.com/francomahl/sswmfa

REFERENCES

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

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.

Ceri, S., Dolog, P., Matera, M., and Nejdl, W. (2004). Model-Driven Design of Web Applications with Client-Side Adaptation. In Koch, N., Fraternali, P., and Wirsing, M., editors, *Web Engineering: 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004. Proceedings*, pages 201–214. Springer Berlin Heidelberg, Berlin, Heidelberg.

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.

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., 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.

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, S., Bosetti, G. A., Rossi, G., Winckler, M., and Barbieri, T. (2016b). 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, Boston, MA, USA.

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*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

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.

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

Urbieta, M., Firmenich, S., Maglione, P., Rossi, G., and Olivero, M. A. (2017). A Model-driven Approach for Empowering Advance Web Augmentation - From Client-side to Server-side Support. In *APMDWE*. INSTICC, ScitePress.

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

WebRatio (2017). WebRatio Platform.