Abstracting and Structuring Web contents for supporting Personal Web Experiences

Sergio Firmenich[1], Gabriela Bosetti[1], Gustavo Rossi[1], Marco Winckler[2], Tomas Barbieri[1]

[1]LIFIA, Facultad de Informática, Universidad Nacional de La Plata and Conicet
{sergio.firmenich, gabriela.bosetti, gustavo}@lifia.info.unlp.edu.ar

[2] ICS-IRIT, University of Toulouse 3, France
winckler@irit.fr

Abstract. This paper presents a novel approach for supporting abstraction and structuring mechanisms of Web contents. The goal of this approach is to enable users to create/extract Web contents in the form of objects that they can manipulate to create Personal Web experiences. We present an architecture that not only allows the user interaction with individual objects but also supports the integration of many objects found in diverse Web sites. We claim that once Web contents have been organized as objects it is possible to create many types of Personal Web interactions. The approach involves end-users and developers and it is fully supported by dedicated tools. We show how end-users can use our tools to identify contents and transform them into objects stored in our platform. We show how developers can use of objects to create Personal Web applications.

Keywords: Personal Web, Web Augmentation, Mashups

1 Introduction

Current Web personalization approaches usually suffer a boundary problem, since most, if not all, work in an individual application basis. When a user needs to deal with two or more applications for performing a particular task, he will face differences in the personalization approach for each of them (if any). Another drawback of personalization mechanisms is that, specified by application's developers, do not necessarily may foreseen the requirements of every single application user. These problems have been the base for the Personal Web, defined in [2] as a "collection of technologies that confer the ability to reorganize, configure and manage online content rather than just viewing it". This generic definition might be realized in different ways such as: (1) PIMs and object manipulation which allow users to collect information objects and make them available for performing operations [16], e.g. to collect scientific work's titles relevant for a researcher to perform further tasks. (2) Mashups, to integrate and combine information objects from different resources into a specialized application [11][12], e.g. to combine multimedia search results from different resources in a single view. (3) Web augmentation, where users are able to enrich information objects in-situ, i.e. in the same Web page they appear [1] e.g. to add information to each movie in the IMDB's Top250 list. (4) Reactive Web which allows users to obtain reactive feedback from information objects under certain events that these objects are able to detect automatically [14], e.g. to inform the user that a new movie was presented. (5) Creation of specific applications: for example, running specific client-side applications that using existing information objects, use them to build a domain specific application, such as a personal agenda, e.g. a personal application for managing scientific literature.

These approaches, altogether, provide users with the possibility of interacting with Web objects (information items from the existing Web) in different ways. However, all the approaches work isolated, with specific and dedicated information models, which makes very complicated to have a complete Personal Web experience supporting arbitrary combinations of these kinds of interactions, given that several different and specialized tools should be developed and maintained which moreover hinders reuse (of contents and behaviors).

In this paper we present a platform for supporting the abstraction of domain objects from Web sites with the goal of creating applications (Mashups, Web augmentation, independent applications, etc.) providing a full interactive Personal Web experience supporting the reuse of structure definition and behavior. The main contributions of our approach are that (1) it supports all the kinds of interactions mentioned above and new ones that could be envisioned in the future; (2) it achieves this goal by using a uniform and rich underlying object-oriented model and (3) the possible combinations of application types (e.g. mashups and augmentations) makes the overall result much richer than the mere sum of these individual approaches.

The paper is organized as follows. Section 2 presents the motivation and an overview of the approach. Section 3 presents the related works. Section 4 introduces our approach. Section 5 describes the tool support and in Section 6 several case studies for illustrating the approach are explained. Finally, Section 7 concludes and presents the future works.

2 Motivation and approach overview

Imagine a journalist who must be informed constantly. With this aim, too many contents might be got from different Web sources, and even different kind of interactions could be needed for achieving a real Personal experience: (I1). *Interact with information objects directly from an object representation space*: if the user needs to store preferred news that he wants to follow, then a PIM with those news could be good for starting any interaction with them. (I2) *Merge objects from different sources into specialized apps*: in this case he would need a mash-up application that integrates the daily news from the preferred media Web sites, allowing him to browse several sources at one time. (*I3) Interact with objects when they are presented in the visited pages*: when visiting a media Web sites, the user could take advantage of Web augmentation capabilities, and then augment the news in specific Web sites with behavior to obtain related news, multimedia resources, look for reactions on social networks, etc. (I4) *Get reactive interaction from the Web*: when there is a hot topic, the user could be interested in some kind of reactive experience, for instance to have immediate notifications informing him the last news. (I5) *Domain Specific application experience:* this user may appreciate a specialized news and journalist tracker application that allows him to follow specific journalists, recommend news, etc. This is similar to mashups, but the underlying application behavior is specific for the news domain.

For providing a full Personal Web experience like this, we must use several applications: a Web augmentation script, a Mashup tool, a PIM, an environment for client-side domain specific applications, etc. In this context, all approaches listed before tackle only a "portion" of the Personal Web. This situation presents some challenges to end-users, who could need to deal with different kind of artifacts at different moments and circumstances. To obtain all these tools for a single domain could be time consuming for end-users, or simply not possible because the required programming skills.

In this paper, we propose a layer for identification and abstraction of information objects; we call these objects *Instance Object (IO)* when they are specific and static instances and *Class Object (CO)* when they refer to the type of those instances and will help to retrieved *IO* dynamically (we explain how both are collected by users in Section 4.1). In Figure 1 we give an overview of the approach showing how *IO* are extracted from Web sites and put on a specific ambient, that we named Web Object Ambient (WOA). Once there, *IO* can be decorated with specific and varied behaviors. These decorations are basically set of messages that the object may response. Some of the messages could be eventually sent by end-users and answers to that messages might have a UI effect correlation. Together with the WOA, we developed a default application (WOA application), that similar to a PIM, presents the *IO* to the user with different menus to send these messages directly to them. This first WOA application allows users to interact directly with the abstracted objects, without the need of visiting the corresponding Web site, as we will show later. Users can determine the relevance of *IO* over the Web and then deal or interact with them in different contexts, but reusing the same model and implemented behavior. This layer is transversal to all the mentioned approaches (mashups, Reactive Web, Web Augmentation, etc.), which impact directly on the maintenance of the client-side artifacts used. Similar to other approaches related to the Personal Web, end-users are responsible of managing the data model (i.e. giving structure to Web objects and collet them into the WOA), while advanced users with some programming skills may create and share complex applications using the data model specifications. The approach offers the WOA functionality through an API over which any kind of application, and even combinations of these (such as Reactive Web Augmentation) may be developed. All these applications are thought to be ran on client-side, but if any communication with a specific server is needed, the approach does not limit it. Supporting different kinds of applications with the same underlying model makes much easier sharing artifacts among users; both *IO* specifications and applications (since they run on top of the same underlying data models). In this approach, we envision two kind of user roles: (a) Developers: they may create specific behavior for *IO* (called decorators) as well as new data collectors, and domain-specific applications. (b) End-Users: they may either consume *IO* or produce *CO*, which consist of specifications for creating *IO,* by using visual tools. When a DOM element is collected and a *CO* is specified, then WOA can instantiate *IO* enhanced with some basic operations, either through the WOA Viewer or in-situ modality. However, end-users may use any product created by developers in order to use *IO* in advanced ways.

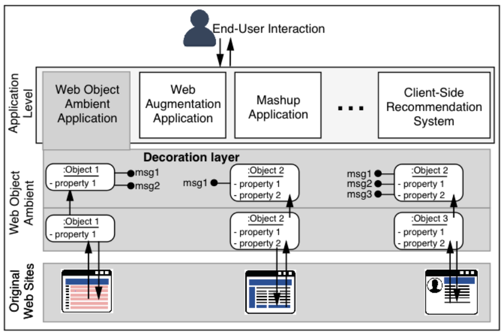


Fig. 1. Web Ambient Object layer

3 Related works

The idea of object extraction is similar to Web Scraping [4]. Web scraping is the process of non-structured (or with some weak structure) data extraction, usually emulating the Web browsing activity. Normally, it is used to automate data extraction in order to obtain more complex information, which means that end-users are not usually involved on determining what information to look for and still less about what to do with the abstracted objects. When Web content has not an underlying structure, Web scraping would be a good option in order to retrieve information from Web sites.

Some Web sites already tag their contents allowing other software artifacts (for instance a Web Browser plugin) to process those annotations and improve interaction with that structured content. A well-known approach for giving some meaning to Web data is Microformats [5]. Some approaches leverage the underlying meaning given by Microformats, detecting those objects present on the Web page and allowing users to interact with them in new ways [6]. A very similar approach is Microdata [7]. Considering Semantic Web approaches, and an aim similar to our proposal, [9] presents an approach for mashups based on semantic information; however, it depends too heavily on the original application owners, something that is not always viable.

However, when analyzing the Web, we see that a huge majority of Web sites do not provide structured data. According to [8], only 5,64% among 40.6 million Web sites provide some kind of structured data (Microformats, Microdata, RDFa [13], etc.). This reality raises the importance of empowering users to add semantic structure when it is not available. Several approaches let users adding structure to existing contents to ease the management of relevant information objects. For instance, HayStack [16] offers an extraction tool that allows users to populate a semantic-structured.

Atomate it! [14] offers a reactive platform that could be set to the collected objects by means of rule definitions. Then the user can be informed when something interesting (such as a new movie, or record) happens. [15] allows the creation of domain specific applications that work over the objects defined in a PIM.

Web augmentation is a popular approach that lets end-users improve Web applications by altering original Web pages with new content or functionality not originally contemplated by developers. Nowadays, users may specify their own augmentations by using end-user programming tools. Very interesting tools have emerged [2], to manipulate DOM (Document Object Model) objects in order to specify the adaptation. However, the costs associated to specifying similar functionality in different Web applications sharing the same underlying domain may be high. Reutilization in Web Augmentation has been confined to reusing scripts. For example, Scripting Interface [10] is oriented to support better reutilization by generating a conceptual layer over the DOM, specifically for GreaseMonkey scripts. Since the specification of a Scripting Interface could be defined in two distinct Web sites, the augmentation artifacts written in terms of that interfaces could be reused.

Another well-known approach for integrating content and services are mashups. Very popular tools such as Yahoo Pipes! [11] allowed users to combine different resources and present a specific result. Yahoo Pipes! is strongly based on the existence of APIs, but other approaches propose in-situ composition, i.e. without generating a new independent application [12]. Although MashMaker allows to abstract widgets with their properties, the way in which the widgets are used is always the same and extending the use implies modifying the application.

It must be noted that if we consider the interactions mentioned before (I1-I5), we can see that they may be supported individually by one of the mentioned approaches. Nevertheless, none approach supports these interactions altogether; therefore, the Personal Web experience might be restricted. Moreover, how future kind of interactions could be contemplated is not taken into account in most of the approaches. The main reason for that, is that the underlying data models seems to be specifically defined for supporting a particular kind of interaction.

4 Our approach

Our approach proposes using a reusable object layer to build any kind of Personal Web application. This is achieved by giving end-users the possibility to structure *CO* from existing content, to import them into the WOA and to interact with their instances either from our WOA viewer, using in-situ Web Augmenters, or in domain-specific applications. Applications and decorators are created by developers, who profit from a reusable layer of specifications of *CO* and their behavior.

4.1 Abstracting and collecting objects

Most Web users' tasks involve looking for information objects (news, papers, movies, hotels, books, etc.). We focus on the problem of identifying and abstracting information objects as *IO* and *CO* that can be used in different contexts. For doing this, our approach first adds a meaning layer to any Web content, similarly to other PIM approaches [16]. For each object type that the user wants to import into the ambient (e.g. books), he should create a *CO*, which is a template that will allow producing instances of such concept. Although this kind of content structuring is not new, our approach is different at the end of the process. The process ends on the object materialization, i.e. generating live objects with internal state and intrinsic behavior that can be used in different contexts. Such process is composed of three steps, as shown in Figure 2: (1) Class Identification, (2) Conceptual abstraction and class structuring, and (3) Instances Extraction and materialization.

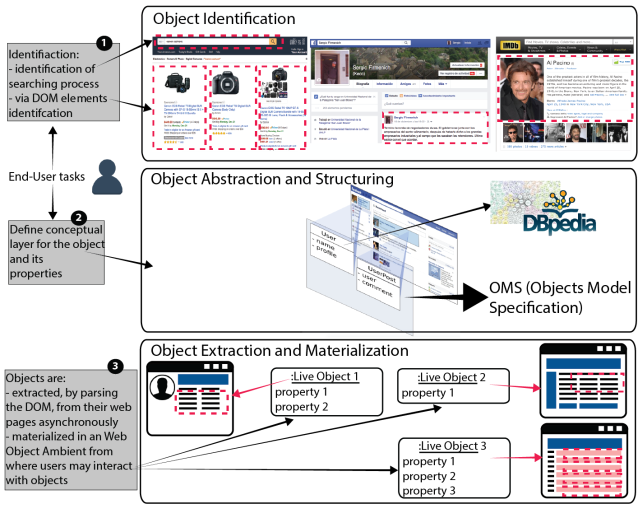
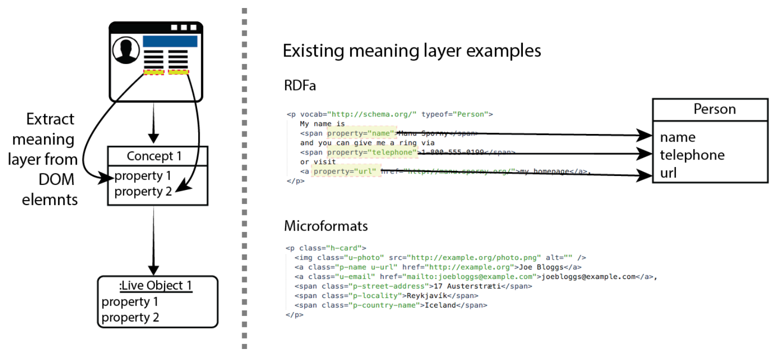

Object Identification
Identifiaction:
- identification of searching process
- via DOM elements identifcation
End-User tasks
Object Abstraction and Structuring
DBpedia
Define conceptual layer for the object and its properties
User name profile
UserPost user comment
OMS (Objects Model Specification)
Object Extraction and Materialization
Objects are:
- extracted, by parsing the DOM, from their web pages asynchronously
- materialized in an Web Object Ambient from where users may interact with objects
:Live Object 1
property 1
property 2
:Live Object 2
property 1
:Live Object 3
property 1
property 2
property 3

Fig. 2. Object Materialization Process

Class identification (1), consists of identifying relevant DOM elements on the context of any Web site, yielding either a single or a list of occurrences of the same element (such as a resulting list of products in Amazon). Implementation details are presented in Section 5, but it is worth mentioning that users are enabled to select DOM elements, and decide between extracting only such element or collecting all similar occurrences. For instance, in Figure 4, although the collection task is made with the Carrie Fisher actor, users may choose to collect all similar detected objects, such as Harrison Ford, etc. Either collecting only one instance or a collection of them, WOA can manage both the Actor *CO* and its individual *IO*.

Concerning *conceptual abstraction and class structuring (2)*, the semantic type and the internal state of a *CO* should be set. The user might provide some required data for defining the *CO*: the generic name, a tag and if it should be statically stored or not. The most outstanding step consists in associating the identified element with a concrete tag, which preferably corresponds with a class in the DBPedia's ontology and will be used for further matching decorators with proper functionality. For example if the user defines a Book, it could be matched and wrapped (in the following step) with behavior that allow looking for the book in multiple stores, looking for movies based on it, etc. Although the user can manually add these data, our tool can auto fill some values at extraction time. For those Web sites that provide some semantic structure via DOM annotation, the tool suggests certain values for tagging the authored *CO*; otherwise, the user should write a tag, which better represents the identified element. Values are also suggested when the user input data at the sidebar, but it is not mandatory to choose one of them. The benefit of using these tags is that our repository contains many decorators associated with the classes of such ontology (the suggested tags). Regarding the extraction techniques, an *IO* can be obtained by parsing one or different DOM elements, including plain text. The component enabling the objects harvesting activity is called *ObjectCollector*. When the identification is based on structured DOM elements, we can create the meaning layer for that object either consuming DOM annotations (e.g. RDFa [13] or Microformats [5] annotations) or asking the user for a tag. We also differentiate –at least– three ways for abstracting and structuring objects, which we describe below. Our goal is to define the same transversal model layer for any of these possibilities, and then to generate instances of that model independently of how the meaning is either added or extracted from DOM elements. Our approach is not tied to any implementation of annotations and new types can be supported by extending the *ObjectCollector*. At the left of Figure 3, we show a generic example of extraction based on an existing semantic layer; by analyzing the DOM, we can detect the concept definition and then extract it for creating the *CO*, which will allow instantiating "live" objects. At the right of Figure 3, we show two different examples. At the top, a RDFa-based example where the concept Person and some of its properties are nested into the HTML. At the bottom, a similar example is presented, based on Microformats.



```
<p vocab="http://schema.org/" typeof="Person">
   My name is
   <span property="name">Manu Sporny</span>
   and you can give me a ring via
   <span property="telephone">1-800-555-0199</span>
   or visit
   <a property="url" href="http://manu.sporny.org/">my homepage</a>.
</p>
```

```
<p class="h-card">
  <img class="u-photo" src="http://example.org/photo.png" alt="" />
  <a class="p-name u-url" href="http://example.org">Joe Bloggs</a>
  <a class="u-email" href="mailto:joebloggs@example.com">joebloggs@example.com</a>,
  <span class="p-street-address">17 Austerstræti</span>
  <span class="p-locality">Reykjavík</span>
  <span class="p-country-name">Iceland</span>
</p>
```

Fig. 3. Examples of existing semantic layers

Regarding the internal structure, a *CO* may be composed of some properties that require similar identification and abstraction steps than concepts, but also implies defining a mapping to its corresponding *CO*. When a concept is abstracted from existing DOM elements, the properties associated to the object may be more than one. For instance, at the right of Figure 4, the concept Actor is defined with two properties (picture and name). In cases where there are many instances available, the properties should be obtained from children elements of each DOM element representing a whole Actor instance. In this way, by defining how to get the properties for a specific instance should be enough for inferring other instances existing in the same Web page. In the center of Figure 4, we show how the same concept could also be abstracted from different Web sites with different strategies. Each particular instance has different properties related to the information available in each Web site. When the object is abstracted from plain text, only one property might be available. For example in the concept Actor (Figure 4), the property "name" acquires the value from the selected text.

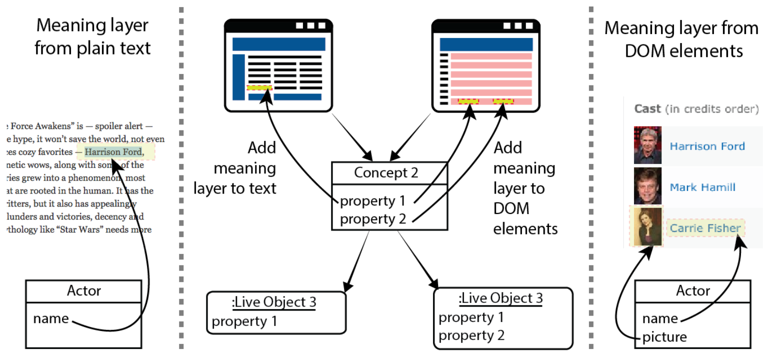


Fig. 4. Examples of semantic layer addition

Finally, all the abstracted objects are stored into the WOA and the *instances extraction and materialization (3)* step takes place, so they can "live" as materialized objects; i.e. besides maintaining their properties in the internal state, they can also respond to messages, as in object-oriented approaches. With the same philosophy, once objects are collected, the WOA may manage both *IO* and their corresponding *CO*, and they may be enhanced via decorators, as we explain later. Summarizing, there are two types of objects available in the WOA: *IO*s which represent a concrete instance of a concept abstracted from a Web site, has the responsibility of maintaining values for its internal state, and respond to messages, and *CO*s which serve for letting end-users to manage all the corresponding *IO* altogether. A *CO* has the responsibility of being aware of all its instances, and when possible, to provide some mechanism for retrieving instances that are not already collected. This is achieved by defining an Object Search Engine for those sites where there are instances of the concept; this is explained in detail in section 5.

Based on the generated *CO* specifications, extraction is the process where the concrete information about the specified DOM elements is obtained. The *CO* may contain the specification for extracting a single object from a Web page (for instance the main news from a media portal), or all the news from the same site. For each object to be extracted, the *CO* contains –at least– the corresponding URL and XPath. In this way, the extraction step includes the task of obtaining a DOM (from that URL), parsing it and getting each information piece to extract all the required for setting the instances internal state (e.g. the title of the last news). Regarding materialization, it implies creating an *IO*; setting its internal state and wrapping it with some behavior.

4.2 Enhancing objects

In the WOA, users may deal with *CO* or *IO*. Both of them has some basic behavior, which is automatically inherited. For example any *CO* responds to messages such as *getInstances()*, *removeInstance()*, etc. An *IO* inherits automatically some behavior such as *showInContext()*, *getDOMImage()*, *getPropertyByName()*, etc. Besides this default behavior, an object can be enhanced either with behavior for the specific object type or with behavior that can be applied over any kind of object (i.e. behavior independent of the application domain). These enhancements are called decorators, inspired in the Decorator design pattern [3] and are developed by advanced users. For instance, if a journalist has collected News objects in the WOA, then an instance decorator could add *getRelatedMultimedia()*, *getRelatedTweets()*, etc. Regarding to the News *CO*, a domain-specific class decorator could add *getCurrentEconomyNews()*, etc. A decorator adds new messages that can be sent to the object from different contexts (from a WOA viewer, augmentation scripts, etc.). Decorators may be generic or even domain specific when these are specifically defined for a type of object from an ontology in DBpedia. When a new *IO* is obtained, then available decorators may be automatically applied. Since decorators specify meta-information related to the type of objects over which it can be applied and also related to the needed properties to work properly, the WOA may discard those decorators that do not fit with an OMS.

End-users may add existing decorators in their browsers and then decide which decorators to apply over the WOA objects (See Section 5). Decorating an object requires identifying the desired decorator and choose the target objects. This can be done from the WOA Viewer, which helps end-users in this task by filtering decorators and *CO* analyzing their compatibility. Decorators must specify (a) the needed object structure: to which kind of objects the decorator may be applied. When the decorator is domain-specific, the target objects may be a particular *CO* or its *IO*. When the decorator is generic, then the target objects may be any *CO* or any *IO*. (b) The messages with which target objects will be enhanced: decorators must be able to define which are the messages for enhancing objects, which also includes if the messages have or not a UI effect that the end-user may perceive.

Decorators may use (a) WOA objects (*CO* and *IO*): although the behavior is going to be added to particular objects, decorators may consume any other objects existing into the WOA for accomplishing that behavior; (b) Any Web content: decorators may need to consume other content besides WOA objects. This can be done in two ways. First, decorators may consume any Web content via the use of APIs or ad-hoc DOM parsing. However, decorators may also reuse other OMS and obtain objects from different Web sites, without the need that these objects already exist in the WOA. For instance the *getRelatedNews()* decorator could parse a media Web site by applying (on the fly) an OMS to GoogleNews, etc., in order to obtain other objects.

Section 4 presents further technical details, but it is important to note at this point that the fact of separating decorator development from the underlying object in which it is going to be applied, implies that these behaviors are intrinsically reusable among Web sites sharing the same domain model in different contexts or applications.

4.3 Interacting with objects

The reason for collecting information objects into the WOA, is that end-users may interact with them in different ways and contexts in order to obtain personal experiences. Such interaction may be performed in two basic ways, which are illustrated in Figure 5: (1) by interacting in a Web page context or (2) in the WOA viewer or any other application. In the first case, it is achieved through Web Augmentation decorators, and the context could be the Web page from which the object was extracted, or any other where the object was added. In the second case, the Viewer (which is the default WOA application), lets end-users to directly send messages to the *IO* and obtaining a visual feedback in response. For interacting from both, the WOA viewer or any other WOA application, the reader should note that the Web site from where these objects are extracted is retrieved transparently by WOA, based on the corresponding *CO*, which does not necessary implies the user opening that site. For instance, if the user has defined the object DRNews (a news *CO* collected from an online media called DiarioRegistrado.com – DR –), and he wants to interact with that object for obtaining the titles of the last news, it is not necessary to open the Web site.

By interacting directly with objects, end-users may send messages (provided by their chosen decorators) to the objects. For instance, a journalist may send the *get-RelatedMultimedia()* message when he wants; this message may return a list of Yooutube Videos and Google Images, while other similar decorators may consume content form other sources. All the messages shown in the menu are dynamic, because this behavior is implemented by decorators, as explained in Section 4.4.

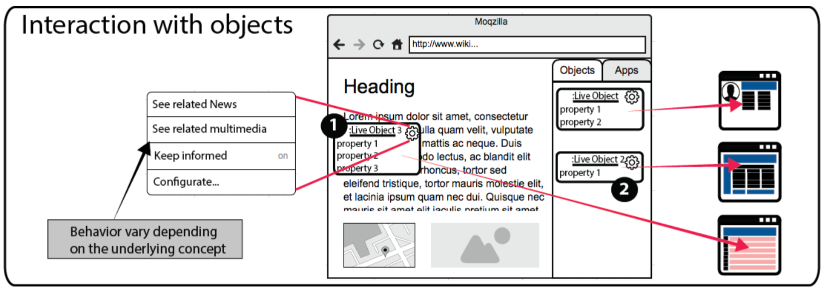


Fig. 5. Interaction patterns with Materialized Objects

Besides interacting directly with objects, end-users may install further WOA applications (created by developers), which might provide different ways to interact with objects. For instance, if the journalist wants to be informed about news related to a particular topic (economy, sports, etc.), he could use a WOA application for Reactive News, which alerts the user when a news appears. Other kind of applications such as one for integrating news related to a map could be also possible, as Section 6 shows.

4.4 Programming WOA objects and applications

A decorator is a JavaScript object developed by extending an existing class, *AbstractDecorator*; see line 5 of the code in Figure 6. Several behaviors are inherited from AbstractDecorator, and two abstract methods must be implemented in order to allow WOA to manage decorators. One, called *getDisplayName*, to return the Decorator's name in order to be visualized by users. A second one, called *getMessages*, returns a collection of associative arrays describing each of the messages (each new behavior) that the decorator adds to the target object, as the simple example from Figure 6 shows. For each of these associative arrays, four properties are needed. The first describes the name of the corresponding decorator's method that will be executed when messages are sent; the second one that is the display name of the message, the third one indicates if that message returns a UI feedback to recognize which are the messages that can be sent under demand directly by end-users. The last attribute lists the properties that the object must have in order to be compatible with the message.

```
1   function GenericDecorator(concept){
2       AbstractDecorator.call(this, concept);
3       this.tags.push('*');
4       this.getSelectedMessages = function(){}
5       this.highlight = function(){ ... }
6       this.orderByName = function(){  ... }
7       ...
8   }
9   GenericDecorator.prototype = new AbstractDecorator();
10  GenericDecorator.getDisplayName = function(){ return "Generic"; }
11  GenericDecorator.getMessages = function(){
12      return [
13          {id:"highlight", name:"Highlight in Web page", showInUI:true },
14          {id:"orderByName", name:"Order by name", showInUI:true, properties:[{"id": "name", "name": "Name"}]},
15          ...
16      ];
17  }
```

Fig. 6. Decorator implementation

```
{
    "id": "dash@lifia.info.unlp...",
    "name": "DR News Dashboard",
    "version": "1.0.3",
    "description": "A dashboard ...",
    "author": "LIFIA",
    "homepage": "http://www.lifia...",
    "license": "MPL 2.0",
    "index": "index.html"
}
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>DR News Dashboard</title>
        <meta charset="UTF-8"/>
        <!-- ... -->
        <script type="text/javascript">
            window.addEventListener("DOMContentLoaded", function(){

                this.initWOAscript = function(){
                    WOA.initSubset();
                    WOA.filterByTag('news');
                    WOA.filterByOrigin('http://www.diarioregistrado.com/*');
                    WOA.filterByAttribute('title', 'ARSAT');
                    WOA.decorate('News');
                    var news = WOA.getInstances();

                    for (var i = 0; i < news.length; i++) {
                        drawInDashboard(news[i].getRepresentation());
                    };

                    WOA.clearFilters();
                    ...
                }
            });
        </script>
    </head>
    <body onload=""> <!-- ... --> </body>
</html>
```

Fig. 7. Using WOA objects from WOPs

WOA applications meanwhile are Web pages with embedded JavaScript code. Basically a WOA application has an associated UI layout (implemented with HTML), and from the JavaScript code it is possible to interact with the WOA, since in the context of a WOA application a WOA library is available allowing to make queries to the materialized objects. The use of this library is shown in Figure 7. At the left, we show the specification of a WOA application in a special package.json file needed for importing the application. The required fields are an ID for resolving the file in the system, a name for displaying at management time, and the name of the entry point file. At the right we can see the pure JavaScript code defined under a script tag, using the WOA library (this code could be included in both decorators and WOA applications). Note that besides the use of our library, the layout of the application is also defined using HTML. With this approach, domain-specific application such as a News dashboard can be implemented in a straightforward way. Besides, this allows working with several kinds of objects in the same application, allowing powerful relationships among different WOA objects.

5 WOA supporting tools

The complete tool is deployed as a Firefox browser extension, including the WOA, the WOA application runner, and the Object Collectors. More Object Collectors, WOA applications and decorators may be added in a plug-in-like style.

5.1 Tool support for collecting and structuring objects

As shown in Figure 8, our tool adds the necessary controls that let users creating objects, no matter what Web resource has been loaded in the browser.

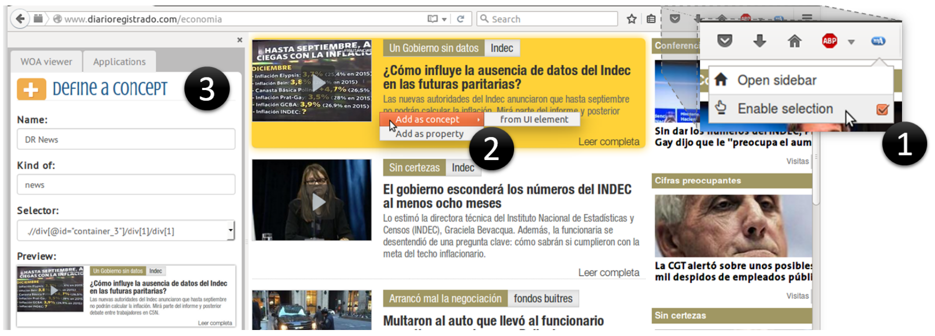


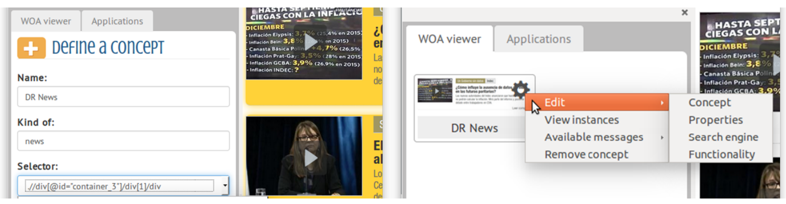
Fig. 8: Identifying and abstracting concepts

First, we added a toolbar button with two options: opening WOA, and enabling the concept selection. Clicking the second option (step 1 in the picture), 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. Options are dynamically loaded according to the selected target element. This behavior is provided by a set of *ObjectCollector*s explained later. Once one of the options is clicked, a sidebar is opened for completing the remaining data required for the abstraction and structuring stage. The contextual menu is populated with those *ObjectCollectors* that match with the selected element. This is carried out by asking the set of collectors to analyze the target DOM element, and rendering just the ones that accomplish the required characteristics for being created with such extraction technique. Our tool currently supports collecting elements from Microformats, DOM element selection and text highlighting. New collectors can be incorporated by extending the framework. Each collector must be capable of analyzing a target HTML element and, if applicable, rendering a context-menu item with their description and associating some behavior to it, in order to return the created object.

Back to the materialization process, once the DOM element is selected, a UI form is opened at the sidebar, which lets the user selecting a name for the *CO*, a semantic tag, the saving method and the number of *IO* in the original Web page. The saving method determines if the extracted element should be stored as a static definition or to be retrieved from its context every time an *IO* is created. Concerning the occurrences, a combo is filled with different XPaths applicable to such element, and allow to univocally reference it or to reference a set of elements instead. For example, it is possible to identify an element by its type of node at certain level of the document tree, or by the CSS class it has applied. This allows the user to choose one or more elements, according to his needs. As shown at left in Figure 9, he is asked to name the *CO* and tag it, 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 *CO* specifications which allow to obtain one or multiple *IO* 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. As WOA is a browser extension, it counts on the necessary privileges to retrieve external and even third party content, to manipulate it and then use it into the required context (e.g. in the sidebar, WOA application or augmented Web page).

5.2 WOA viewer

Once saved into the WOA, users may see the *CO* and *IO* in the WOA viewer, as shown at the right of Figure 9. We show the view of a *CO*, whose contextual menu allows to manage the properties, edit the *CO*, wrap it with some behavior and define an Object Search Engine for retrieving *IO* that may not be present as a result in the current DOM. If there are class decorators enabled, then the messages that can be sent directly by the user are shown under the submenu “Available Messages”.


Define a concept

Name: DR News

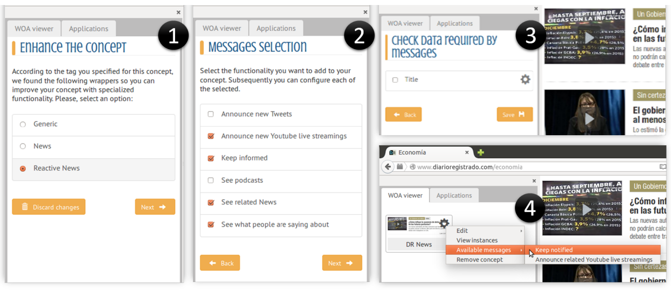
Kind of: news

Selector: .//div[@id="container_3"]/div[1]/div

Fig. 9: Structuring a concept with WOA

5.3 Decorating objects

In Figure 10, we show how the user associates the DR News concept with the *ReactiveNews* decorator (1). He selects the messages he wants to use for enhancing such concept instances (2), link the required decorator's messages parameters with the properties of the created concept (3). He can return to the WOA viewer for interacting with the functionality provided by the decorator (4).



According to the tag you specified for this concept, we found the following wrappers so you can improve your concept with specialized functionality. Please, select an option:

Select the functionality you want to add to your concept. Subsequently you can configure each of the selected.

- [ ] Announce new Tweets
- [✓] Announce new Youtube live streamings
- [✓] Keep informed
- [ ] See podcasts
- [✓] See related News
- [✓] See what people are saying about

Fig. 10: Wrapping an object with specialized behavior

5.4 Object Search Engines

To support different ways of searching objects, we take into advantage original Web applications engines, allowing end-users to abstract that searching engine UI similarly to the way in which they can abstract content into objects. These ObjectSearchEngine are search APIs, each of them containing the searching URL, the form where the user would enter the text to search, and the button for performing the action. Also searching modifiers (such as filter or ordering options) and pagination managers are supported. Then, for example, a decorator may easily search for news in Google News given a particular news title from an object extracted from DiarioRegistrado.com and materialized into the WOA, assuming that an Object Search Engine for Google News was defined. Finally, a *CO* that was added into the WOA may have associated several ObjectSearchEngine defined in different Web sites. For the sake of space we omit further explanation on creating custom search engines, which can be found in an online documentation site (see footnote on page 17th).

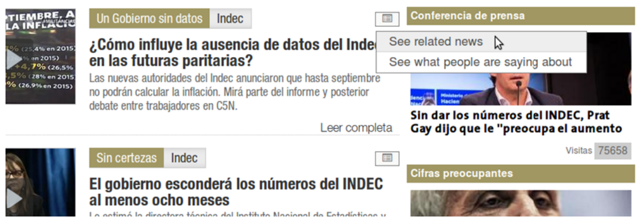
6 Case studies

In this section we present some case studies demonstrating the power of the approach. Here several examples show how CO and IO materialized into the WOA may be enriched with decorators and then used in different contexts.

6.1 A Web augmentation approach based on domain-specific models

Another possible scenario for using directly materialized concepts is in-situ Web Augmentation. When the concept has been wrapped with a decorator with Web Augmentation capabilities, every DOM element related to an *IO* is enhanced with a floating-menu in its original context. Such menu is placed at the top-right corner of the element and makes it possible to interact with the decorator messages, in the original context of the structured data.

Consider a scenario in which a radio journalist frequently accesses a set of Web sites that allow him to be informed about what is happening outside while he is producing his live broadcast, and also to provide their listeners with additional updated information. He uses a news portal as his main source of information. Once he has read the last entries in the portal, he navigates to other sites looking for the concrete topics he read about. As his program concerns political and society issues, he also uses to read people opinions on social networks. In this sense, he has a particular interest in Twitter, because of the public visibility of its messages. It would be highly desirable for this user to create a custom WA application, that takes the news portal as background information and that add functionality for easily retrieving each portal's entry with a set of related news and tweets. For achieving such goal by using WOA, the user should identify the "News" concept, then abstract and structure *a CO* by using any available WOA collector. During the structuring process, he finds a matching tagged decorator available for the "News", but it is required that the user defines, at least, the "Title" property. Once done, this decorator augments the news portal site with a floating menu at the right-top corner of the main HTMLElement related to each *IO*. When the user clicks on it, he is offered with the configured messages for interacting with such *IO*. Figure 11 shows a screenshot of a WOA application satisfying the journalist's needs.


Un Gobierno sin datos Indec

¿Cómo influye la ausencia de datos del Indec en las futuras paritarias?

Las nuevas autoridades del Indec anunciaron que hasta septiembre no podrán calcular la inflación. Mirá parte del informe y posterior debate entre trabajadores en C5N.

Leer completa

Conferencia de prensa

Sin dar los números del INDEC, Prat Gay dijo que le "preocupa el aumento

Visitas 75658

Sin certezas Indec

El gobierno esconderá los números del INDEC al menos ocho meses

Cifras preocupantes

Fig. 11. Web Augmentation Decorators enhancing entries of a News portal

Until this point, creating a personal solution does not require any programming skills. However, if the needed functionality is not being contemplated by any of the existing decorators, a developer should implement it. Developers can create not only decorators but also applications. In both cases, the WOA library is accessible for also querying instances of existing templates, concepts and decorators.

6.2 A personal dashboard based on composition of abstracted objects

When the journalist needs to be informed with the news from several portals at the same time, he can use an application that uses the news defined for both sites and have an integrated experience, as shown in Figure 12. This application has an HTML page for the application's content structure and the *initWoaScript()* function to initialize after the WOA library was loaded, so the application can start making requests about the *IO* available for each portal and Twitter. He can ask the decorator the representation to display of every *IO*; and wrap them to present them with a unified style. As we can see, the main advantage of implementing this dashboard instead of creating a mashup is that it allows to embed its own domain-specific behavior. Finally, the user should specify the application data in package.json file, place in the root directory of the application and import it with WOA.

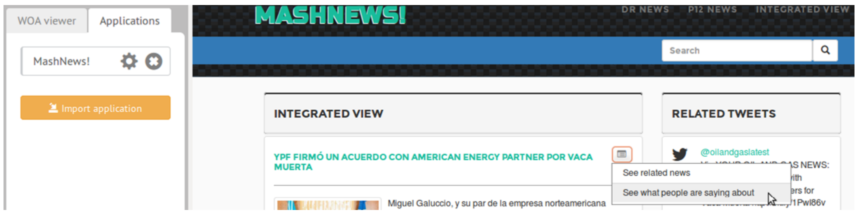


Fig. 12. A Dashboard application integrating entries from two news portals

6.3 Using decorators with Reactive Web capabilities from WOPs

Finally, consider the fact that the consumed news of the previous example were retrieved from certain subsection of both Web portals –e.g. economics–. Both portals have other sections that, under certain circumstances may have news of interest for the journalist, either because the subject is directly related with his interests or because they have reached high level of popularity. Generally, the main entry of news portals usually owns such qualities, so a considerable feature for the journalist's application could be tracking changes of such main entries. This is possible to implement through reactive programming, making elements capable of propagating their changes. As WOA decorators are instantiated in a high privileged context (our browser extension's main code), it is possible to retrieve and manipulate external documents for achieving this goal.

Back to the example, the developer should define such "Main News" as a new *CO*, because even they are also news from the same portals, these are in different contexts and have a different structure, which are unknown for WOA. Then, he can use a *ReactiveNews* decorator for wrapping such *CO*. This decorator allows retrieving the concept's owner document and tracking changes on its proper elements. As a result, notifications can be shown in any context, by defining a target place where the notification should be displayed as a property in the *CO* definition. At the left of Figure 13, you can see a notification as the result of integrating the "Main News" decorated-concept in the Personal Dashboard example. At the right, you can see the same decorator displaying a notification in the Google News site, under the in-situ modality.

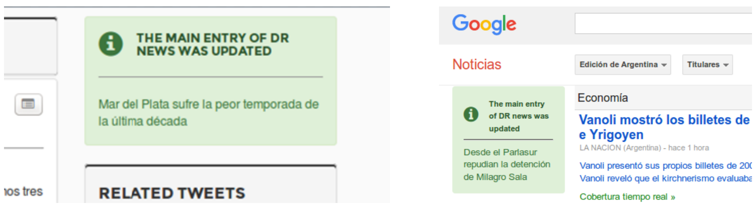

THE MAIN ENTRY OF DR NEWS WAS UPDATED
Mar del Plata sufre la peor temporada de la última década
RELATED TWEETS
Google
Noticias
Edición de Argentina
Titulares
The main entry of DR news was updated
Desde el Parlasur repudian la detención de Milagro Sala
Economía
Vanoli mostró los billetes de e Yrigoyen
LA NACION (Argentina) - hace 1 hora
Vanoli presentó sus propios billetes de 200
Vanoli reveló que el kirchnerismo evaluaba
Cobertura tiempo real »

Fig. 13. A notification as result of the ReactiveNews decorator's message

7 Evaluation

We have performed an expert-oriented evaluation, to measure te power of the approach. Based on the motivational examples presented in Section 2, we identified the dimensions or aspects that an approach must support for letting users obtain such Personal Web experience. We found more than 10 dimensions of interest in the evaluation namely Consumes static data, Consumes dynamic data (Web services or extractors), Consumes structured data, Consumes unstructured data, No technical skills needed, Content authoring, Reusable information objects, Individual information objects shareability, Tracks changes in the original Web content, Allows augmenting existing Web content, Integrates content from multiple sources, Integrates and displays services from multiple sources, Objects can live in background.

We used these dimensions for comparing how each type (e.g mashup) and individual approach (for instance Marmite) support personal experiences. For reasons of space, we cannot include the full comparison table here, but it can be read in the WOA documentation Web site1. As a result, we found that none of these approaches supports all experience at the same time. In some cases, the problem is data structuring. In other cases, the changes in Web pages (where an object was abstracted) are not tracked (and consequently some interactions such as reactive ones are not possible). Others do not support the enhancement of objects when they are visualized in their corresponding Web page. Our approach, in contrast, supports altogether the interaction kinds listed in Section 2 (and further ones, such as client-side recommender systems) since its underlying object-oriented data model is, in our opinion, the best way to implement such a layer, given its intrinsic properties such as reuse and extensibility makes the approach application-agnostic. Over these models, applications may be run in different scopes but always using the same client-side web technologies.

8 Conclusions and Future works

The constant evolution of Web and their users have shown the need of more personal Web applications. Web Mashups, Web Augmentation and other approaches have emerged to reach this goal; however these approaches are usually not integrated and underlying domain models are not easy to reuse. We believe that, for reaching a more Personal Web, the kinds of interaction experiences supported by these approaches should be composable, in such a way that information object models and their behavior could be reused. In this paper we presented an approach for adding an object-oriented layer over Web contents, that serves as a platform for the development of third-party software. Solutions can be created from existing contents, and focused on existing content and decorators –therefore behavior– reusability. We presented our tools and several case studies that demonstrate the power of the approach. We are

[1] WOA Website Comparison table: https://sites.google.com/site/webobjectambient/comparison

currently developing a WOA application and Decorators repository. In this way, we are increasingly covering functionality needs of diverse end-users in the process of decorating the objects they materialize. The same repository is being designed to support collaboration in the creation of OMS and also as a communication platform for sharing them. We are also developing an end-user tool for creating WOA applications, such as the dashboard presented in the case studies section. Finally, we plan to perform experiments with end-users for further validating our approach.

References

1. Díaz, O.; Arellano, C. The Augmented Web: Rationales, Opportunities, and Challenges on Browser-Side Transcoding. *ACM Transactions on the Web*, 2015, vol. 9, no 2, p. 8.
2. Díaz, O., Arellano, C., Aldalur, I., Medina, H., & Firmenich, S. (2014). End-User Browser-Side Modification of Web Pages. In *Web Information Systems Engineering–WISE 2014* (pp. 293-307). Springer International Publishing.
3. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.
4. Ferrara, E., De Meo, P., Fiumara, G., & Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, *70*, 301-323.
5. Khare, R., & Çelik, T. (2006, May). Microformats: a pragmatic path to the semantic web. In *Proceedings of the 15th international conference on WWW* (pp. 865-866). ACM.
6. Operator Firefox Extension, https://addons.mozilla.org/es/firefox/addon/operator/?src=search.
7. Microdata, http://www.w3.org/TR/microdata/
8. Bizer, C., Eckert, K., Meusel, R., Mühleisen, H., Schuhmacher, M., & Völker, J. (2013). Deployment of rdfa, microdata, and microformats on the web–a quantitative analysis. In *The Semantic Web–ISWC 2013* (pp. 17-32). Springer Berlin Heidelberg.
9. Kalou, A. K., Koutsomitropoulos, D. A., & Papatheodorou, T. S. (2013). Semantic web rules and ontologies for developing personalised mashups. *International Journal of Knowledge and Web Intelligence*, *4*(2-3), 142-165.
10. Díaz, O., Arellano, C., & Iturrioz, J. (2010). *Interfaces for scripting: making Greasemonkey scripts resilient to website upgrades* (pp. 233-247). Springer Berlin Heidelberg.
11. Pruett, M. (2007). *Yahoo! pipes*. O'Reilly.
12. Ennals, R., Garofalakis, M. Mashmaker : Mashups for the masses (demo paper). In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD'2007)*.
13. RDFa, https://rdfa.info
14. Van Kleek, M., Moore, B., Karger, D. R., & André, P. (2010, April). Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In *Proceedings of the 19th international conference on World wide web* (pp. 951-960). ACM.
15. Van Kleek, M., Smith, D. A., & Shadbolt, N. (2012). A decentralized architecture for consolidating personal information ecosystems: The WebBox.
16. Karger, D. R., Bakshi, K., Huynh, D., Quan, D., & Sinha, V. (2005, January). Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *Proc. of the CIDR Conf.*