A data service layer for Web extensions

Alex Tacuri Name[1][a], Sergio Firmenich Name[1][b],Alejandro Fernandez Name[1][c] and Gustavo Rossi Name[d]

[1]*LIFIA, Facultad de Informática, UNLP, Argentina*
{*alex.tacuriu,s_author*}*@info.unlp.edu.ar, t_author@dc.mu.edu*

Keywords: Web scraping Web Browser extensions

Abstract: Nowadays, Web browser extensions are the more convenient way to modify existing Web applications and the browser itself to satisfy non-contemplated requirements. Extensions play an important role because of the relevance that the Web and the Web browser have in our daily life. The functionality offered by Web extensions varies. Many Web extensions are tools that improve the interaction with existing Web sites based on techniques such as mashups and Web augmentation. Accessing and manipulating Web sites' content is a challenge for the development and maintenance of extensions. In this paper, we present a data service layer that allows extension developers to create APIs based on Web annotations to access existing Web content. The data service layer simplifies the retrieval of structured information from Web pages, without the need of DOM manipulation in the extension code. We provide visual programming tools to create these APIs so there is no need for advanced programming skills, making extension development and maintenance easier.

1 INTRODUCTION

The Web browser is no longer a simple client for rendering Web pages. Nowadays, it is also a platform to run rich applications, that may enhance Web applications loaded in the browser, but also improve the browser with new functionalities and applications. Web extensions are a particular kind of software, running on the browser, that augments the browser's capabilities at different layers. Extensions may add new capabilities such as scraping, may improve accessibility, integrate crypto wallets, adapt loaded Web pages, change the default behavior of the new tab page or even serve a specific application that is hosted inside the Web browser.

Many Web extensions consume content available online. Usually, Web browser extensions perform some form of Web scraping to obtain the required information. In other cases, they use APIs to obtain specific information elements. For instance, there are several web extensions to create alerts about the variations of cryptocurrencies. All of them scrap information from online crypto portals, or use APIs to obtain up-to-date information. In many other cases, the information is scraped from Web pages and used as part of a Web augmentation artifact (Díaz and Arellano, 2015) or a mashup application.

Although the availability of APIs has grown steadily in recent years, Web scraping is still a frequently used technique for retrieving Web content, just because a great majority of existing Web sites do not provide an API. In these cases, Web extensions may obtain the HTML document for a specific URL, and manipulate its DOM and obtain specific information items. Web scraping is challenging in several aspects. First, Web extensions are frequently implemented with hard-coded DOM references, making them fragile as DOM references tend to brake when the Web site is modified. Second, scraping becomes more difficult when a scraping pipeline involves several web sites (i.e. the result of scraping connects elements from various web sites). Third, it is challenging (if not impossible) to reuse the scraping code in different Web extensions.

In this paper, we propose a data service layer for Web extensions. The layer encapsulates site-specific, search and scraping logic, and exposes object-oriented search APIs. By using this layer, developers can build Web extensions based on this API, which automates the use of Web site search engines and abstract their object models. In this way, developers are unaware of the complexity of data search,

[a] https://orcid.org/0000-0003-3159-5556
[b] https://orcid.org/0000-0000-0000-0000
[c] https://orcid.org/0000-0000-0000-0000
[d] https://orcid.org/0000-0000-0000-0000

retrieval and scraping. The data service layer includes a visual programming environment for the specification of data search and object model creation, which are exposed then as a programatic API. Users can define a search API that uses an existing Web site's search functionality, and defines how search results are abstracted into objects. In addition, the user of the approach can compose several search API to construct a more complex model object. A model triggers requests to other APIs and defines how the resulting data objects are connected (thus returning a graph of connected objects). This layer acts as a bridge between a Website's DOM, and the implementation of the extensions. When a Web site is changed, only search APIs and their object models require maintenance and Web extensions keep working without changes. We believe that this layer could make it much easier to create Web extensions that require information retrieving. The impact of the proposed data service layer could be significant if we consider the number of Web extensions and user scripts already available in public repositories. Currently, and just for considering extensions for Google Chrome Browser, there are more than 137,345 extensions any another kind of artifact (more related to Web augmentation) such as user-scripts also has huge repositories like greasyfork.org with thousands of artifacts, that could use this service layer.

This paper is structured as follows. In Section 2 we present a running example. Section 3 presents the approach, its architecture, the tool involved to create search APIs and the tool for defining information models. In Section 3.4 we present briefly a usability study. Section 5 introduces related works and finally, in Section 6 we conclude and plan future works.

2 RUNNING EXAMPLE

Figure 1 depicts a mockup of the UI of a Web extension that provides a mashup application for the domain of scientific research. This mashup application allows one to search scientific articles in Springer, and when an article is selected further information is retrieved and shown. Particularly, it displays the number of citations (which is extracted from Google Scholar). It is also possible to select an author of the article in order to obtain and show related articles of that author (extracted from DBLP). As the reader may note, it would require using the search engine from Springer, parse its DOM to obtain articles' information, and with this information performing similar tasks with Google Scholar and DBLP. The Web extension developer must create the scraper for all three Web sites but also program the logic to create the pipeline in which the title of an article obtained in Springer is used to search for it in Google Scholar and, similarly, the logic to obtain author's articles from DBLP.

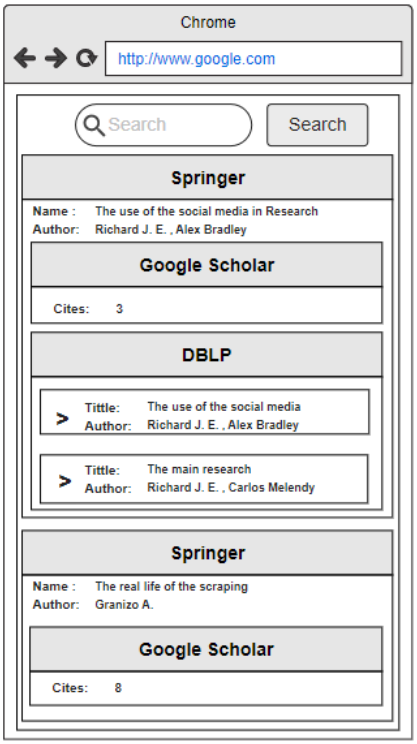


Figure 1: Mockup of the UI of a Web extension that a provides a mashup application

3 THE APPROACH AND TOOLS

In this section, we present our approach and tools. First, we describe the architecture. Next, we explain how search APIs may be defined based on Web content annotation. Third, we present our model editor that allows to combine different search APIs to create more complex information models.

3.1 Architecture

We propose a simple architecture in which Web extensions developers need to have installed our Web extension to use the data service layer. Figure 2 shows an overview of the architecture. In this Figure it can be seen that our data service layer includes two tools (Search API tool and API composition tool), each one generating a kind of artifact that is stored in this layer. Also, it provides an API Interaction service layer that

allows Web extensions to consume those information models created with the API composition tool.

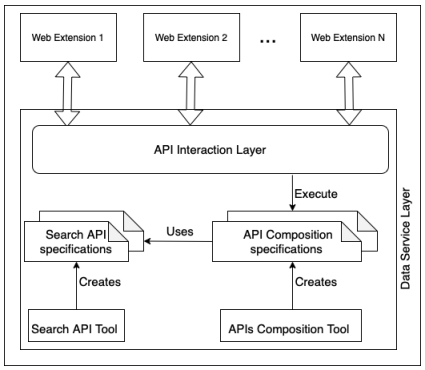


Figure 2: Data service layer architecture

3.2 Search services definition

To create a search service we use the search engine of a specific website and by web annotation and scraping techniques, we extract the results that the search engine returns to create domain objects. The tool allows to annotate the UI search components, and abstract the results as domain objects allowing the user to define different properties for them, all these operations are done by using an end-user programming tool. A detailed explanation of how these search services are defined can be found in previous work (Bosetti et al., 2022).

3.3 Information models through API composition

To show how the data service layer is used to create web extensions, let us first return to the running example presented in Section 2. The main idea is to obtain an information model based on this composition of search APIs for Springer, DBLP and Google Scholar, offering the integration of information from the three Web sites in a single response. Then when the model is executed by searching for something with the Springer search service, the response will include the number of citations obtained from Google Scholar and related articles (articles from the first author) obtained from DBLP.

To create an information model based on search service composition, the user must first have created the search service of the websites that he wishes to integrate, following the process presented in the previous subsection. Then, the user starts the API Composition Tool from the extension's main screen. The first step to create a composed model is to name it (we use the name "Research" for our model). Next, the extension shows an editor where the defined search services are presented in a menu at the left, and the composition canvas is the main component of the UI. The user drag&drop the services he want to compose (i.e., Springer, Google Scholar and DBLP services) into the canvas. Each search service is shown as a node in a graph. Now, in order to integrate the results of the search services, the user needs to establish links between the search services nodes. This is done by relating properties in the data models of each service. In this example, we are going to connect Springer with Google Scholar through the Title property and Springer with DBLP through the Author property. Each connection must have a name and a way to interact with the search service, that is, how many results can it obtain from the search service. It is worth mentioning that if it does not find any matches, there are no results. Figure 3 provides an overview of the resulting composed model. The property Author of results from a search in Springer is used to search in DBLP. The results from DBLP are injected into the "Articles" property of each Springer result. The integration between services can be done as one-to-one (connect to the object corresponding to the first search result) or one-to-N relationships (as in this example).

3.4 Using the search service in a web extension

At the moment we present two use cases that we have given to the composition, the first with a mashup and the other with requirements not contemplated in websites.

3.4.1 Using composite search API on mashup web extension

Now that we have created a composed search API and its data model, let's use it to create the mashup application discussed in Section 2.

The web extension presents a main screen (see the browser window in the middle of Figure 4) with an input text, where the user enters a topic to search for, (in this example it was "scraping"). The search button triggers a search using the composed search API described in the previous section. Results are displayed along the number of cites (which is extracted from Google Scholar), and related articles of the selected author (extracted from DBLP). It was implemented as a browser's web extensions with background and content scripts. Next, we discuss key parts of the source code to show that the developer is unaware of the web requests and scraping tasks.

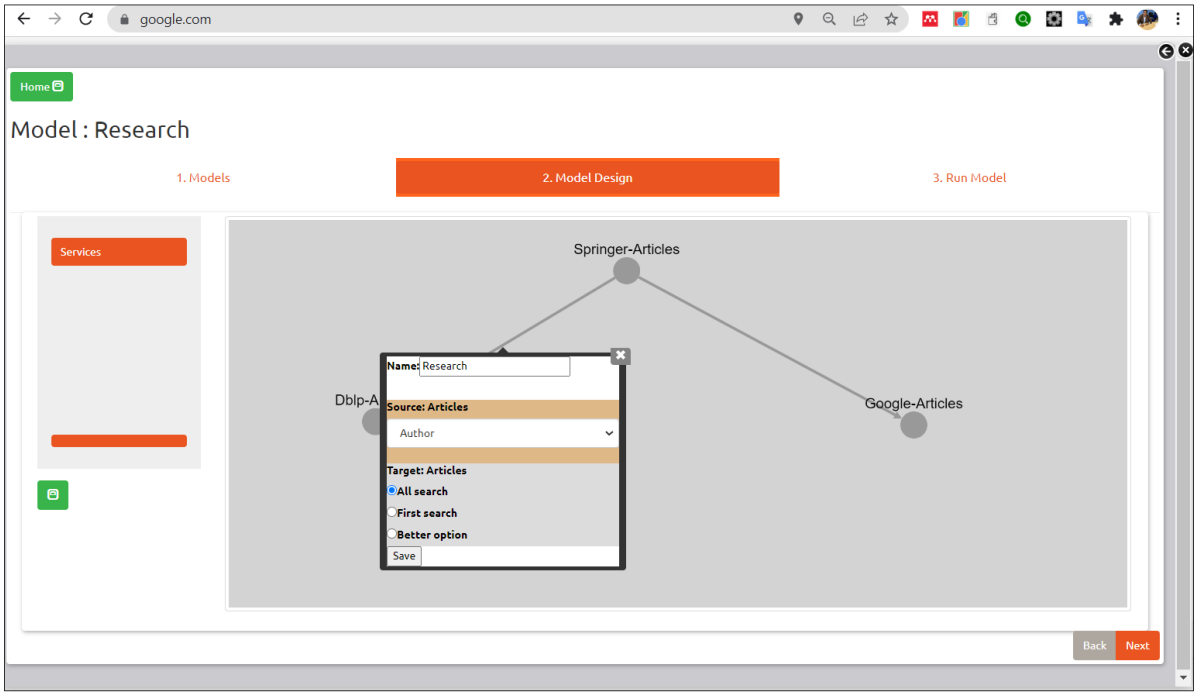


Figure 3: The API composition tool: Integration between Springer and DBLP search APIs

The consumer API has the "search" function in Listing 3.1, which receives four parameters such as: the token or service layer identifier (which is different in each browser), the text to search within in the composed search API, the composed search API's name to which we are going to invoke and the Search API where we are going to start the search for information. When the service layer is invoked through the search function, it executes the model and returns a JSON message, whose structure can be observed in the code 3.2

```
$scope.searchData = function (args) {
    var api=new dataServiceLayer();
    api.search(args.token,args.text,
    args.model,args.searchapi).then(
    papers=>{
        $scope.results=papers
    })
}
```

Listing 3.1: Source code for the service layer invocation

```
{
    "Author":"Yang Lin,      Chun-
    Yaun Yeh,      Shiau-Cheng Shiu",
    "Title":"The design and
    feasibility test of a mobile
    semi-auto scraping system",
    "service":"Springer",
    "cites":[
      {
        "cite":"Cited by 4",
        "title":"The design and
    feasibility test of a mobile
    semi-auto scraping system",
        "service":"Google"
      }],
     "articles":[
      {
          "Author":"Junchao Xiao,
    Lin Yang, Fuli Zhong, Hongbo
    Chen, Xiangxue Li: Robust
    anomaly-based intrusion
    detection system for in-vehicle
    network by graph neural network
    framework. Appl. Intell. 53(3):
    3183-3206 (2023)",
          "Title":"Robust anomaly-
    based intrusion detection system
     for in-vehicle network by graph
     neural network framework.",
          "service":"DBLP"
        },
        ...
      ]
}
```

Listing 3.2: Json of the communication message between the service layer and the mashup.

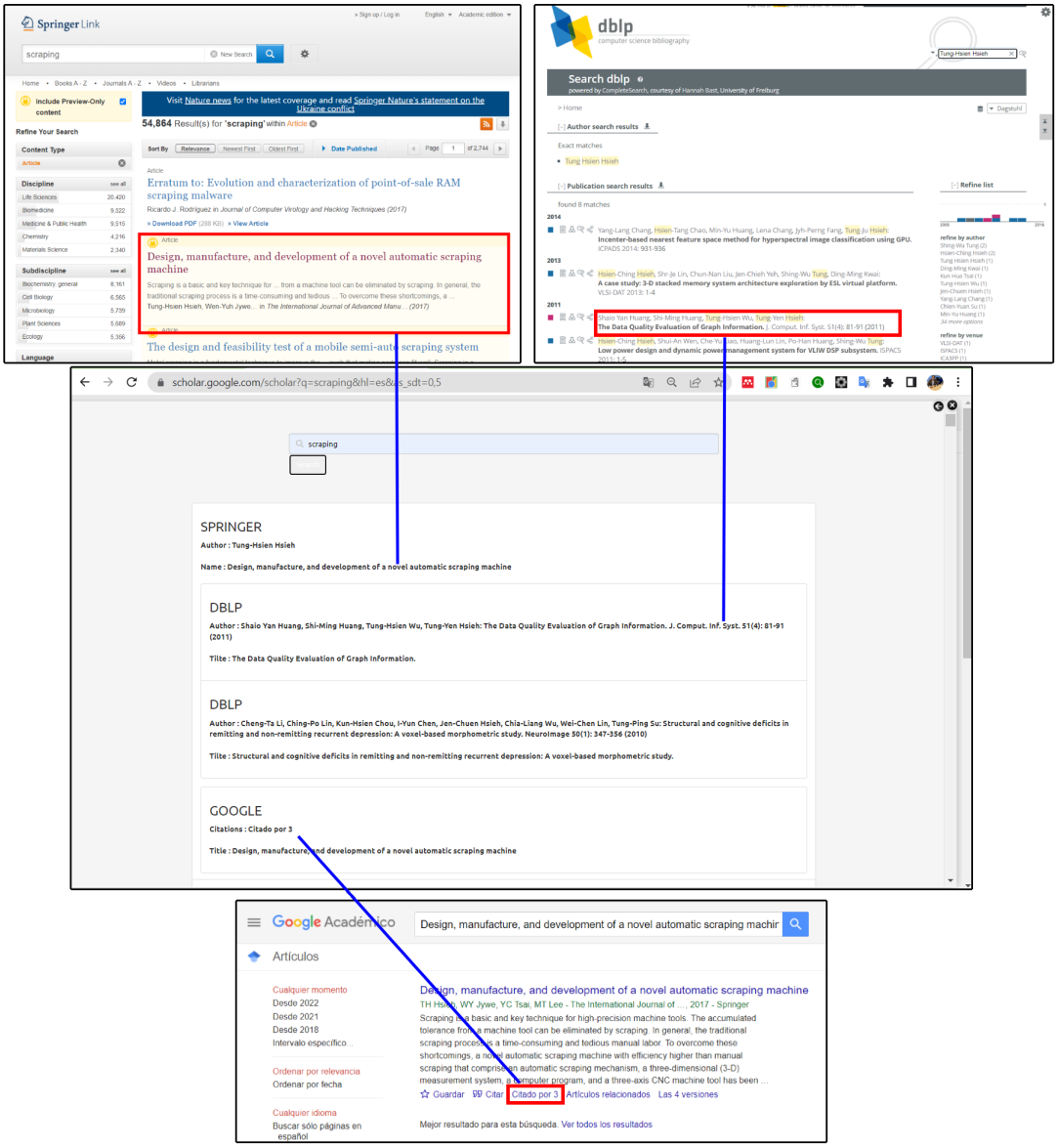
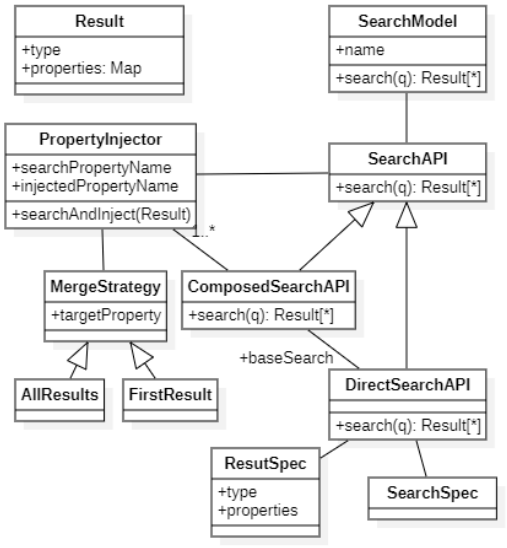


Figure 4: Search results in the model containing Springer, Google Scholar and DBLP

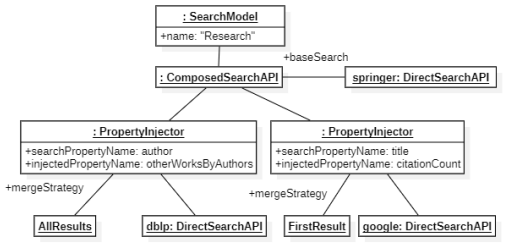
Figure 5: Configuration of the web extension that contains the mashup application

3.4.2 Class, Object and Interaction diagrams

The figure 6 shows the diagram of classes and objects of the services layer at the macro level, the main classes are: "SearchAPI", which defines the abstraction of DOM elements, and "ComposedSearchAPI", which integrates the SearchAPIs to generate more complex structures.



(a) Class diagram



(b) Object diagram

Figure 6: Service layer class and object diagram

In the figure 7 we present the interactions that occur between mashup and the proposed services layer. First, the client web extension sends a message to the service layer, which executes the search in Springer; then, upon receiving the information, for each of the objects received, it searches the different integrations that are part of the model, in this case in Google Scholar and later in DBLP to finally return a message with the information received from the three search engines.

3.4.3 Using composite search API on unsatisfied website requirements

We can also use the composed search API to satisfy requirements and functionalities that websites do not have. An example of this use is shown in Figure 8 where the number of citations that the article has from Google scholar is presented within the Springer site and also the articles that the first author has made in DBLP is shown on the same site. Note that the code the developer must write to conduct searches and obtain the results is the same as in the mashup application previously discussed. The data service layer not only hides the complexity of scraping and data integration but also supports reuse.

4 Preliminary evaluation

We have conducted a usability study with 16 participants in order to measure the usability of the tool and detect usability problems to later carry out a more complete experiment to study in depth the impacts and applicability of the approach. In previous work we have evaluated the creation of Search Services (Bosetti et al., 2022).

In this new study, participants were asked to do 7 tasks related to the creation and edition of API composition models. The model that participants were asked to create was the one of the example of this paper, using Springer, Google Scholar and DBLP. 13 people managed to complete the tasks, 2 people completed 4 tasks and 1 completed 5 tasks. It is important to mention that participants had no programming skills.

At the end of the tasks, the System Usability Scale (SUS) method was used, which as a result presents us with an average of 68.11, which according to the different standards is acceptable.

5 Related works

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 in determining what information to look for and still less about what to do with the abstracted objects. 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

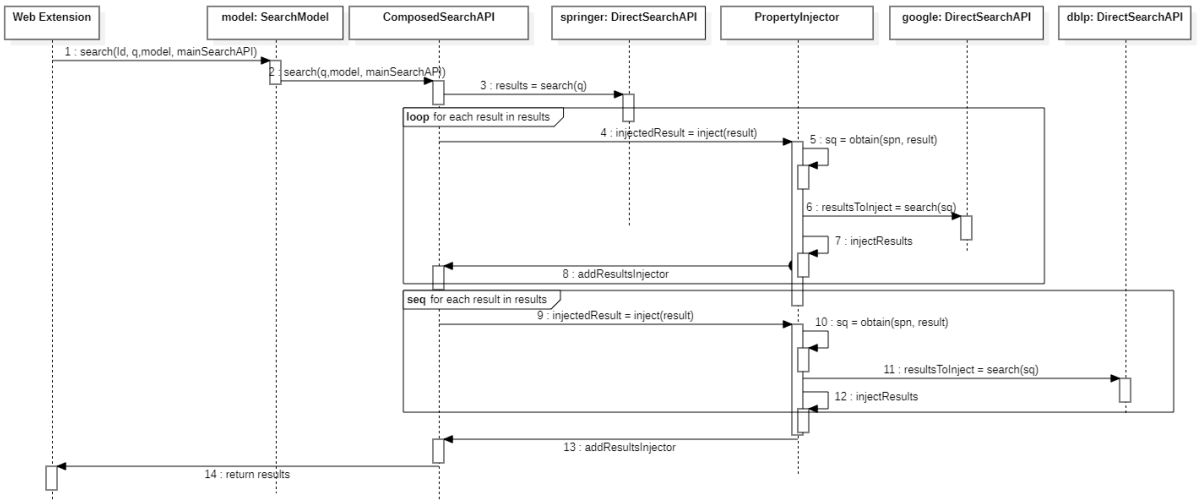


Figure 7: Service layer interaction diagram

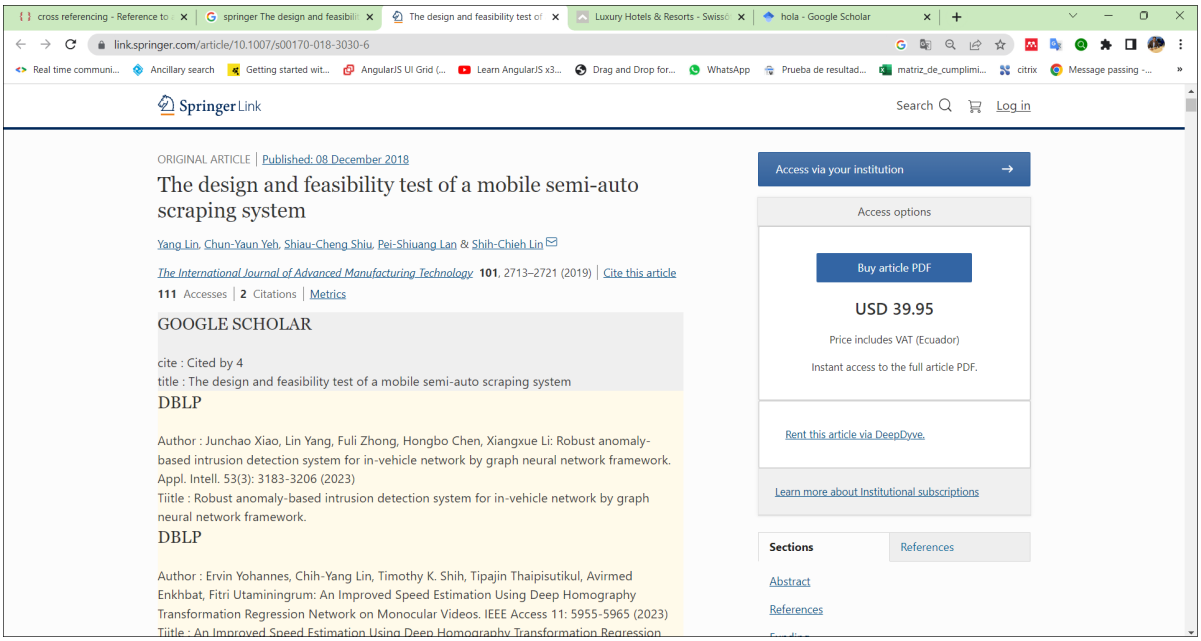


Figure 8: Reusing the search API to augment Springer's web site

structured content. A well-known approach for giving some meaning to Web data is Microformats (Khare and Çelik, 2006). 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. A very similar approach is Microdata. Considering Semantic Web approaches, and an aim similar to our proposal, (Kalou et al., 2013) 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 does not provide this annotation layer. According to (Bizer et al., 2013), only 5,64% among 40.6 million Web sites provide some kind of structured data (Microformats, Microdata, RDFa, etc.). This reality raises the importance of empowering users to add semantic structure when it is not available. Several approaches let users add structure to existing contents to ease the management of relevant information objects. For instance, HayStack (Karger et al., 2005) offers an extraction tool that allows users to populate a semantic-structured information space.

Atomate it! (Kleek et al., 2010) 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. (Van Kleek et al., 2012) allows the creation of domain specific applications that work over the objects defined in a PIM. Rousillon is an interesting approach based on end-user programming for defining scraped based on hierarchical data (Chasins et al., 2018). In (Katongo et al., 2021) another approach is presented to enable web customization through web scraping defined by users without programming skills. However, it is different to our approach because our data service layer could be used for many Web extensions.

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 (Díaz et al., 2014), to manipulate DOM (Document Object Model) objects in order to specify the adaptation. Reuse in Web Augmentation has been tackled. For example, Scripting Interface (Díaz et al., 2010) 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.

6 Conclusions and future works

Web extensions are a very particular kind of software that allows end users to adapt their Web experience. Many of them are based on some kind of scraping of Web sites to obtain the desired information to enrich the user experience. A part of the development of web extensions has proved to be possible without programming skills, using end-user development tools. However, when the extension requires information scraping, the development becomes more complex.

In this paper we presented a data service layer approach, that allows web extension developers to define in a no-code fashion how to scrap web sites based on the abstraction of their search services. Then, developers may delegate in our extension all the scraping required to obtain a JSON object with the results.

We have conducted a usability study showing promising results. Although we have detected some usability problems, most of the participants were able to define the API composition models required in the tasks, even although all participants had no programming skills.

As future works, we plan to conduct a more comprehensive experiment for validating other aspects of our approach.

REFERENCES

Bizer, C., Eckert, K., Meusel, R., Mühleisen, H., Schuhmacher, M., and Völker, J. (2013). Deployment of rdfa, microdata, and microformats on the web - A quantitative analysis. In Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J. X., Aroyo, L., Noy, N. F., Welty, C., and Janowicz, K., editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*, pages 17–32. Springer.

Bosetti, G., Tacuri, A., Gambo, I. P., Firmenich, S., Rossi, G., Winckler, M., and Fernández, A. (2022). ANDES: an approach to embed search services on the web browser. *Comput. Stand. Interfaces*, 82:103633.

Chasins, S. E., Mueller, M., and Bodík, R. (2018). Rousillon: Scraping distributed hierarchical web data. In Baudisch, P., Schmidt, A., and Wilson, A., editors, *The 31st Annual ACM Symposium on User Interface Software and Technology, UIST 2018, Berlin, Germany, October 14-17, 2018*, pages 963–975. ACM.

Díaz, O. and Arellano, C. (2015). The augmented web: Rationales, opportunities, and challenges on browser-side transcoding. *ACM Trans. Web*, 9(2):8:1–8:30.

Díaz, O., Arellano, C., Aldalur, I., Medina, H., and Firmenich, S. (2014). End-user browser-side modification of web pages. In Benatallah, B., Bestavros, A., Manolopoulos, Y., Vakali, A., and Zhang, Y., editors, *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part I*, volume 8786 of *Lecture Notes in Computer Science*, pages 293–307. Springer.

Díaz, O., Arellano, C., and Iturrioz, J. (2010). Interfaces for scripting: Making greasemonkey scripts resilient to website upgrades. In Benatallah, B., Casati, F., Kappel, G., and Rossi, G., editors, *Web Engineering, 10th International Conference, ICWE 2010, Vienna, Austria, July 5-9, 2010. Proceedings*, volume 6189 of *Lecture Notes in Computer Science*, pages 233–247. Springer.

Kalou, A. K., Koutsomitropoulos, D. A., and Papatheodorou, T. S. (2013). Semantic web rules and ontologies for developing personalised mashups. *Int. J. Knowl. Web Intell.*, 4(2/3):142–165.

Karger, D. R., Bakshi, K., Huynh, D., Quan, D., and Sinha, V. (2005). Haystack: A general-purpose information management tool for end users based on semistructured data. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*, pages 13–26. www.cidrdb.org.

Katongo, K., Litt, G., and Jackson, D. (2021). Towards end-user web scraping for customization. In Church, L., Chiba, S., and Boix, E. G., editors, *Programming '21: 5th International Conference on the Art, Science, and Engineering of Programming, Cambridge, United Kingdom, March 22-26, 2021*, pages 49–59. ACM.

Khare, R. and Çelik, T. (2006). Microformats: a pragmatic path to the semantic web. In Carr, L., Roure, D. D., Iyengar, A., Goble, C. A., and Dahlin, M., editors, *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006*, pages 865–866. ACM.

Kleek, M. V., Moore, B., Karger, D. R., André, P., and m. c. schraefel (2010). Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In Rappa, M., Jones, P., Freire, J., and Chakrabarti, S., editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 951–960. ACM.

Van Kleek, M., Smith, D. A., Shadbolt, N., et al. (2012). A decentralized architecture for consolidating personal information ecosystems: The webbox.