To cite this version:

Bosetti, Gabriela and Firmenich, Sergio and Fernandes, Alejandro and Alba Winckler, Marco Antonio and Rossi, Gustavo *From Search Engines to Augmented Search Services: An End-User Development Approach.* (2017) In: 17th International Conference on Web Engineering (ICWE 2017), 5 June 2017 - 8 June 2017 (Roma, Italy).

From Search Engines to Augmented Search Services: An End-User Development Approach

Gabriela Bosetti[1(✉)] , Sergio Firmenich[1,2] ,
Alejandro Fernandez[1] , Marco Winckler[3] ,
and Gustavo Rossi[1,2]

[1] LIFIA, CIC, Facultad de Informática, Universidad Nacional de La Plata,
La Plata, Argentina
{gabriela.bosetti, sergio.firmenich,
alejandro.fernandez, gustavo}@lifia.info.unlp.edu.ar
[2] CONICET, La Plata, Argentina
[3] ICS-IRIT, Université Toulouse III, Toulouse, France
winckler@irit.fr

Abstract. The World Wide Web is a vast and continuously changing source of information where searching is a frequent, and sometimes critical, user task. Searching is not always the user's primary goal but an ancillary task that is performed to find complementary information allowing to complete another task. In this paper, we explore primary and/or ancillary search tasks and propose an approach for simplifying the user interaction during search tasks. Rather than focusing on dedicated search engines, our approach allows the user to abstract search engines already provided by Web applications into pervasive search services that will be available for performing searches from any other Web site. We also propose allowing users to manage the way in which the search results are presented and some possible interactions. In order to illustrate the feasibility of this approach, we have built a support tool based on a plug-in architecture that allows users to integrate new search services (created by themselves by means of visual tools) and execute them in the context of both kinds of searches. A case study illustrates the use of such tool. We also present the results of two evaluations that demonstrate the feasibility of the approach and the benefits in its use.

Keywords: Web search · Web augmentation · Client-side adaptation

1   Introduction

Searching is one of most frequent user task over the Web; popular rankings like Alexa's[1] can be used for supporting this claim. We can observe that most popular Web sites mainly support this task: Google, YouTube, Baidu, Yahoo, Wikipedia, Amazon are part of the leading group at the top 10 of the ranking, dismissing Facebook and QQ, and repetitions of Google under a different top-level domain.

---

[1] http://www.alexa.com/topsites/global; 0 accessed on January 17, 2017.

From the perspective of standard information retrieval [3, 14] a search is often treated as a task, driven by an information need that is formulated by a user as a query. The query is processed by a dedicated system able to select the documents that (according to certain rules) better match the user's query; a refinement process might be used to create new queries or to improve the obtained results. Nonetheless, users play a major role in the search process, therefore the understanding of user activities is essential to improve the tools supporting the search tasks [11, 14].

Web browsers and search engines are the primary tools people use to access the vast quantity of information available online [16]. A careful analysis of user activity over Web browsers shows that, quite often, users have to combine search engines of Web sites and browsing activities to find information that fulfils their needs [3]. It is also quite well known from empirical studies [1] that users maintain several tabs or windows open in the browser to track the context of multiple searches for information that interest them. It is interesting to notice that these activities might serve to accomplish two types of user goals [10, 20]: to start an investigation about a particular information (primary search task) and/or to find complementary information providing details about the information currently displayed on a Web site (secondary or ancillary search task). While opening new windows or tabs in the browser might help users to run multiple primary searches in parallel, the same actions create an articulatory distance between the main user's focus of interest [6], as illustrated in Fig. 1. The latter shows a task model matching two primary searches for finding the Web site of ICWE2016 and WWW2016, respectively, from the Google's search engine. Figure 2, in turn, shows the recursive pattern of the ancillary search tasks. There, a user visiting the ICWE2016 Web site performs an ancillary search on DBLP for getting a list of related articles for a concrete author, and navigates to Google to search for the online file. Both sites offered complimentary information that allows him to understand better the program of the ICWE2016 Web site and decide to attend or not.
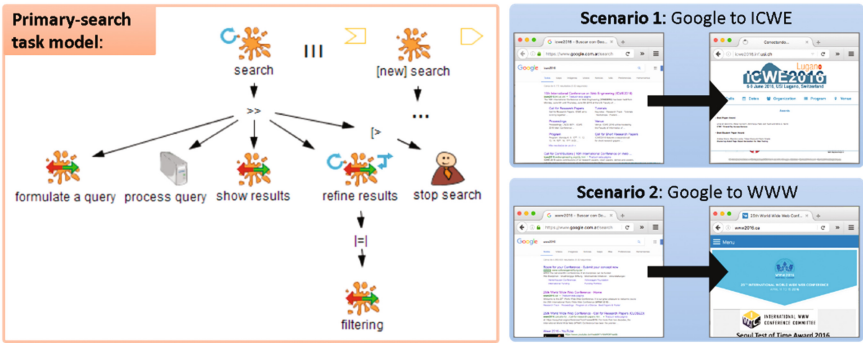

Fig. 1. Primary search task model and two possible scenarios

Today, users can type their search queries directly on the browser's interface components, without having to visit the Web site of a search engine. Safari lets users specify the search engine the user wants to use, like Google, Yahoo! or Bing. Mozilla Firefox also allows users to choose a particular search provider when the browser detects that the current Web page provides one. However, such feature works only for well-known Web sites, as IMDB. In all the cases, search results started from the browser's UI are still displayed in a new tab or window.
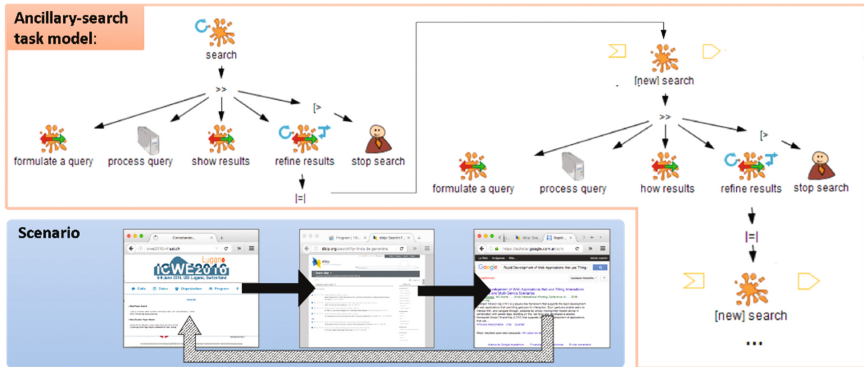

Fig. 2. Ancillary search task model and scenario

There are generic search engines based on Web content scrapers, as Google, and domain-specific ones, as IMDB. The first ones cover a broad scope and their goal is to provide entries to data sources/Web sites containing the information that users are looking for. In contrast, domain-specific search engines are focused in data with a narrow scope but more accurate if users want specialized details about objects in a concrete domain [15]. Indeed, most of ancillary search tasks are triggered because users need complementary information that is frequently domain-specific. A taxonomy of online page-views according to different categories is presented in [12]. The paper reports that users navigate among different kind of pages and use to revisit pages of the same kind and structure. This fact may also suggest that structured objects may take advantage of the already existing content and behavior on the Web, since they share a same domain, and that was the purpose of [8].

In this paper, we propose an innovative approach for dealing with ancillary search tasks while navigating the Web by means of Web Augmentation [17]. The approach is aimed at allowing users to: (i) create custom queries to generic and/or domain-specific search engines; (ii) trigger queries as ancillary search tasks for Web sites they are visiting; and (iii) display the results of the search in the context of the user task. For that, custom user searches are deployed as search services that can be accessed from the Web browser whenever the user is navigating the Web. An overview of this idea is shown in Fig. 3. In this case, we show how our approach would better support the scenario presented in Fig. 2. Instead of requiring the user to move from the current Web page he is visiting, we propose to obtain search results transparently and show

Fig. 3. A scenario for the search approach

them in the same context. This will make the user's task easier and faster than repeating the same search process with similar searches (e.g. other authors at IMDB) and enable the possibility of comparing their results without moving among tabs or windows. The Search Services are specified in terms of a model, and this specification can be shared among users. A support tool, successfully used in a subset of popular Web sites, demonstrates the feasibility of the approach.

The remaining of the paper is organized as follows. Related works are discussed in Sect. 2, and our approach is presented in Sect. 3. The support tool is described through a case study in Sect. 4, and the feasibility on the abstraction of Search Services is demonstrated in Sect. 5 along with the results of a quantitative assessment. Finally, Sect. 6 discusses and mentions some final remarks and the future work.

2 Related Works

Web search has been a target of research from the beginnings of the Web. Several works have contributed in various aspects, as scraping, semantic search or collaborative search. Although these research lines are very active, the design of user interfaces for searching was also studied. For instance, concerning information presentation, focus + context visualizations [4] proposed to maintain the object of primary interest presented in detail, meanwhile, other contextual information may be available. This is clearly related to the idea of ancillary searches; a deep analysis of this kind of search was recently presented in [20]. An inside-in approach for complementary information is presented in [18], conceiving its presentation in the light of the principle of "Overview first, zoom and filter, then details-on-demand". The paper also argues that information may require being presented in different ways, through visualizations that effectively cover different *task-domain information actions*. In the following sections, we propose an approach in which end-users are the ones managing how resulting information may be visualized. To achieve this, it is necessary to analyze search results.

Web search engines return a result page with a list of Web documents (URIs) matching the search criteria. Results are usually presented as a list of page titles and one or two sentences taken from the content. Recent advances in Web search interfaces provide, for a predefined set of element types, such as movies, recipes or addresses, rich snippets that help the user in recognizing relevant features of each result element (e.g., movies playing soon or reviews for a given movie). In some cases, these rich snippets include the data the user is looking for. It is possible because Web site creators include structured data in Web pages that computers can recognize and interpret,

and that can be used to create applications. Viewing the Web as a repository of structured, interconnected data that can be queried is the ultimate goal of the Semantic Web [2]. However, end users do not always have the means to exploit or add information to the Semantic Web. The tools presented here let them extract and use structured data from any Web site, with no need of training on Web development.

There are also models describing how people deal with information and their informational needs [19] and, consequently, providing context to our approach. Eight characteristics (features) adequately describe the information behavior in various disciplines [7]. Such characteristics are: starting (begin the information seeking), chaining (following connections to related content), browsing (semi-structure browsing), differentiating (to filter information), monitoring (keeping up-to-date), extracting (selectively identifying relevant material in an information source), verifying (checking for accuracy), and ending (final searches and tidying up). The definition of Search Services allows the extraction of relevant features in Web content, putting the user in control of what are the relevant properties of information objects, to use them later for differentiation. Moreover, when users select which information repositories will be defined as search services, they conduct an initial quality filter (verification by provenance).

Being able to mouse-select elements in the primary window to launch in-situ searches is a means to support search start and information chaining activities. When the results are displayed in overlay mode, these connections become explicit and persistent, as pointed in [20]. It presents an approach for providing end users with the possibility of executing ancillary searches. However, while in [20] developers need to implement a *broker* at server-side, in charge of retrieving external and already existing services, we provide EUD support at client-side, with no need for server-side components nor textual programming skills for users. Additionally, we allow defining the semantic and structure of the results to be retrieved, without restraints on what kind of element to consider. This suggests a same structure of information for all end users using our tool, which can be later used for querying the common repository or using similar visualizations for the objects sharing a same class.

Finally, research on information seeking behavior has focused mostly on how individuals seek information. However, in many contexts, this process involves collaboration [9]. The Search Service definition tool we propose stores service definitions in a local storage. Users can export and import definitions, thus supporting simple sharing via e-mail and instant messaging. MacLean and colleagues found this simple form of collaborative tailoring via customization files and email sharing an effective mechanism to foster a culture of end-user tailoring [13].

3   End-User Driven Search Services

A vast number of searching scenarios may allow appreciating that the integration support offered by Web applications and Web browsers, in combination, may jeopardize the user experience, since the user may need to perform repeatedly extra operations to obtain the desired information (such as open a new tab, enter the URL of the target Web searcher, etc.). Furthermore, if the user is performing an ancillary search, he might refine the query or go back to the original information context to do it.

This paper presents an approach based on a flexible architecture that allows end users to customize the way they perform Web searches. It follows an inside-in approach, which enables users to perform ancillary searches without leaving their current context. While other searches are performed in relation to a specific goal in mind, the ancillary search is nested among other tasks and is aimed at providing complementary information to the user's current context. The solution we propose allows dealing with ancillary search tasks while navigating the Web by means of Web Augmentation [17], reducing the user's efforts and, therefore, the gulf of execution and the evaluation gulf, as explained in [20].

The searching process we target entails the following steps: (1) define a query; (2) select a search engine; (3) enter the query and trigger the search; and (4) inspect and interact with the results. Our goal is to improve the user experience with search tasks, particularly with the steps 2–4. For the first step, we comply with the query language imposed by the underlying search engine. To better support the aforementioned steps, our approach propose a tool allowing to:

- Trigger searches from the current Web page for reducing the interaction required to perform a search in any foreign search engine.
- Transform search results (DOM elements) into domain objects with specific semantic and structure.
- Integrate the resulting domain instances in the current Web site for further visualization and interaction.

In order to achieve these objectives, we propose:

- Allowing users to encapsulate existing Web applications' search engines into pervasive Search Services. Given that not all the Web applications supporting searches provide an API, we propose to reproduce automatically the UI interaction required to perform a search. It implies that users must select the UI search engine components to create Search Services.
- Integrating the new Search Services with the Web browser search mechanism for ancillary searches. Users should be able to use the created Search Services from any other existing Web site.
- Displaying results in the context of the current Web site, to enable different ways of visualizations supporting primary and ancillary searches. It is done by parsing the DOM, extracting the search results from it, and creating domain object instances.

A simple scenario is presented in Fig. 3, where a user is navigating the accepted articles on the ICWE2016's Web site. At some point, he requires seeing other publications of a particular author. Certainly, this secondary information requirement would be better satisfied by using domain specific Web applications (such as DBLP or Google Scholar) instead of a generic searcher (such as Google or Bing). In this setting and using our tool, the user would be able to trigger the search from the current Web site by highlighting the author's name and selecting the desired Search Service (e.g. DBLP). At the right of the same figure, results are listed in an overlay popup whose content follows a table structure. It is built automatically given the domain object specification that was made when the user created the Search Service. There is also a toolbar where options for filtering and ordering results are available (if these were defined for the
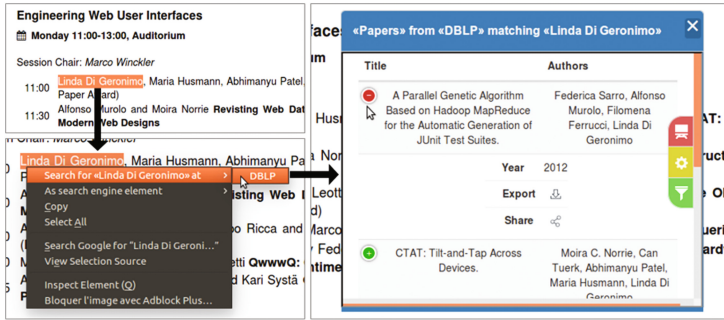
Fig. 4. Example of use of a Search Service

triggered Search Service). Furthermore, a third option allows the user to change the visualization, in case that a table structure does not fit the user's needs.

3.1 The Approach in a Nutshell

We have designed and developed an architecture for supporting the creation of Search Services and the integration tool needed for letting end-users to use these new services from any Web site they are visiting. The architecture has three layers: (i) end-user support tool, (ii) current search results, (iii) model layer, as it shown in Fig. 5.
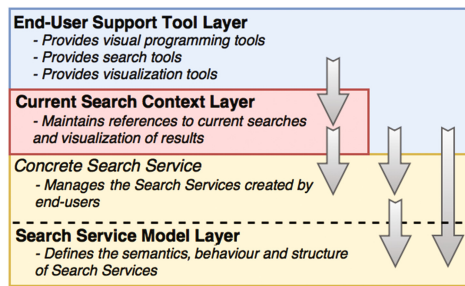
Fig. 5. Search service architecture overview

This diagram refines the relationships between layers' components:

1. *The Search Service Model*. This layer supports the creation of Search Services, which are based on the search engines that Web applications already provide. Note that this is focused mainly on the creation of Search Services for those Web applications that, despite providing a search engine, do not provide an API. In this sense, our approach enables the creation of a service (API alike) based on how users would use these search engines. A Web search engine's interface is usually composed of an input and submit button, filtering and ordering options, and some

mechanism for paginating the results. We propose to abstract all these UI components (DOM elements), wrapping them with objects that conform a Search Service. Then, these Search Services can emulate the user behavior and retrieve the corresponding results given a particular query specification. To provide an API-alike mechanism, results are not interpreted just as DOM elements but also as abstractions of the underlying domain objects. E.g. if the search engine being abstracted is DBLP, results may be wrapped by the *Paper* domain object, which could be populated with domain properties such as *title* or *authors*. Even more, this object may also have properties whose values are taken from another DOM (obtained from another URL), such as a *bibtext* property; we explain this in Sect. 3.3. All these concepts are materialized at the bottom of Fig. 6. As we will show later, it is convenient to provide a semantic layer on the top of the search results because it allows the creation of visualizers that go beyond presenting the raw results.

Fig. 6. End-User driven search service architecture

2. *End-User Tool Support.* Search Services can be incrementally specified by using our visual tool, selecting the key DOM elements related to a Web search engine. The tool is part of the End-User Support Tool layer, where other two components coexist. One is the *SearchTool*, which creates the corresponding menus for the existing *SearchService* objects and allow users to perform searches from the current Web site. The *VisualizationTool* takes the search results and allows users to interact with them in different ways by selecting a particular *Visualizer*.

3. *Current Search Context.* The middle layer represents the current *Query* and its results. When the user wants to perform a search task, a *SearchQuery* is created (there are basically two strategies for doing that: text selection and text input). When the search results are retrieved, these are materialized as instances of *SearchResult*; the Visualizer presents them considering the corresponding *SearchResultSpec;* this specification describes results by means of properties whose values are also obtained from the DOM where actually the results are, or from the DOM obtained from the *targetURL*, which is basically the real domain object's Web page.

3.2 Search Service Architecture: Flexibility, Compliance and Extensibility

In this subsection, we explain two important aspects of the architecture. Flexibility and compliance are related to how the Search Service may be mapped to different search engines provided by Web applications. Extensibility is related to how the whole toolkit may be extended with further components for searching and visualizing results.

Flexibility and Search Engine Coverage

There are some components presented in Fig. 6 that deserves a bit more explanation:

– *Properties*: these are defined for the *SearchResultSpec*. Consider a Search Service is being defined in the e-commerce domain, and properties such as *price* and *availability* could be defined. However, although there is some information obtainable from the DOM presenting the results, more information could be available in the site; as the *technical description* or *shipping costs.* Both kind of properties are extracted from different DOMs, and the behavior for retrieving both DOMs is different. This is the reason why we separate two kind of properties. On the one hand, *InResultPageProperty* allows users to define properties whose values will be extracted from the search result's page. In the other hand, *InTargetPageProperty* let users to extract values for further properties. Our toolkit knows how to reach the actual result's Web page because the attribute *targetURL* is defined, mandatorily, in *SearchResultSpec*. With this in mind, we can note that when using our approach for an ancillary search, as in Fig. 4, the listed properties could be obtained from different Web pages, but this will be transparent from the users' point of view.
– *Ordering and Filtering*: To better reproduce the power of original Web search engines, we have contemplated ordering and filtering features. Some Web sites, such as DBLP, let the users filter results according to some criteria, such as "Journal only", which is applied by clicking an anchor on the search page. Our model contemplates this kind of filters through the *ConditionManager* and *Condition* classes. Ordering and filtering functionality, with roots in the original Web application, will be available in the menu displayed at the right of Fig. 4. Most of the search engines we analyzed offer ordering by clicking anchors or buttons, and these options are available from the same main menu in our Visualizers.
– *Search Execution Strategies:* Search engines spread on the Web have different configurations concerning the involved UI components and the interaction design for

executing a search. Most of them have an input element to write the query but not all of them have a trigger (e.g. anchor, button) and that makes it necessary to have different strategies for carrying it out. We have implemented three alternatives, but they can be extended and integrated in the tool. We covered sites requiring to write a query into an input and handling the query execution by one of these alternatives: (1) clicking on a trigger to load the page with the results; (2) clicking a DOM element but loading the results through ajax-call; (3) listening to some input-related event (e.g. keypress, blur, change) and loading the results by ajax-call. Subclasses implementing this behavior are *WriteAndClickToReload*, *WriteAndClickForAjaxCall* and *WriteForAjaxCall* respectively. The strategy is automatically assigned when the user creates a search service, and each strategy can know if it is applicable based on the components the user has defined and the success on retrieving new results, so this is not a concern for the end user.

Extensibility

As shown in Sect. 5, the Search Service Model is compliant with most of the Web sites' search engines that we analyzed. However, beyond this model and the Search Service specification tool, our approach is extensible in two ways. On the one hand, the model is extensible by means of the creation of services based on the existence of application APIs. On the other hand, the end-user support tool is extensible by the creation of new Visualizers. We next explain both APIBased and Visualizer extension points.

– *APIBased Search Services*: Some Web applications offer APIs for retrieving information from their databases (Twitter, Facebook, etc.). In these cases, it is very common that APIs expressivity goes much further than what is possible to do with our UI-based Search Service approach. With this extension point, developers could create new Search Services (using application's API) to be integrated later by end-users.
– *Visualizers*: as shown in Fig. 4, search results become domain objects whose properties are listed in the default Visualizer (*TableOfProperties*), which create a table where columns represent each property and rows each object. However, further visualizers could be developed and integrated into our toolkit. E.g., a new Visualizer (*GroupByPropertyValue*) could allow users to select a property and group the already obtained results according to the value of this property. Consider the example of Fig. 4; it would be possible to group the author's articles within *journal* or *conference* boxes. Beyond this, other visualizers could be focused on calculating information and visualizing more processed information. For instance, the users could be interested in seeing quickly the evolution of the author production in a chart showing the amount of articles per venue, instead of just displaying each article.

It is important to note that these two extension points require advance JavaScript programming skills to be developed, but once created, they can be installed and used by end-users, who can configure the parameters of such new visualizers according to specific properties defined for the *SearchResultSpec* of a given *SearchService*.

4 Tool Support and Case Studies

In order to support our approach, we implemented our tool as a Firefox extension. It allows both the specification and execution of Search Services. The use of this tool for the first purpose is illustrated in Figs. 6 and 7, and for the second one in Fig. 8.

End users can define *UI-based Search Services* through our tool. A Search Service of such nature should be capable of automatically emulating a search that otherwise the user has to do manually (e.g. opening a new tab, navigating to the search engine of a Web site, typing a text, triggering a button, etc.). Once such Search Service is defined, the user can use it for performing ancillary searches by highlighting a text in any Web page he is navigating and choosing a service from which he wants to retrieve results. In this sense, the selected *UI-based Search Service* must know: in which *input control* the text should be entered, which *button* to trigger to perform the search, how to obtain more results, and how to interpret them. Filters and sorting mechanisms can also be defined, but they are not mandatory.

Consider Amaru, she is always surfing the Web and she uses to look for related books when she finds something (a topic, an author, etc.) of her interest. She is an active user of GoodReads and every time she finds some term of her interest, she copies it, opens a new tab in her browser, accesses GoodReads and performs a search with the copied text as keywords. In this setting, it would be very convenient for her to be able to carry out such searches from the same context in which she is reading the comments.
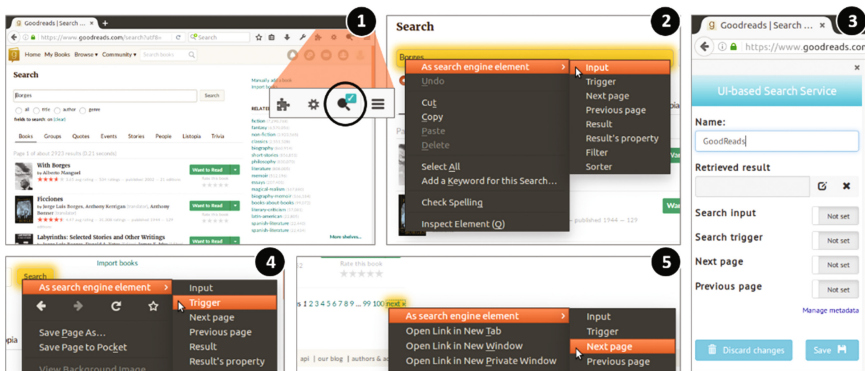

Fig. 7. Defining the input, trigger and pagination elements for a Search Service

Figure 7 shows how Amaru is starting to create a *UI-based Search Service* by selecting DOM elements from the Web site of *GoodReads*, concretely from its search engine[2]. To do so, she navigates to the Web page where the search engine is and enables the «Search Service definition mode» by clicking the highlighted button in step 1. In this concrete case, she should select, at least, the input (step 2) and trigger (step 4)

---

[2] https://www.goodreads.com/search?q=Borges.

controls, and also the one retrieving more results (step 5). The DOM elements defined as the UI-Search-Service controls are selected by right-clicking them.

As it is the first time she defines a Search Service and she has no other Services available in her personal account, the tool asks her to give it a name through a form opened in the sidebar (step 3). Otherwise, the tool should ask the user to select an existing UI-based Search Services for which it is starting to define the controls.

Then, she specifies the kind of results the Search Service will retrieve, as shown in Fig. 8. First, she selects an element in the DOM which represents the main container of the element he is expecting to have as a result. Such element is the highlighted one in step 6. When she chooses to define it as a "result" through the context menu options, a form is loaded in the sidebar (step 7), where she must complete some required data. In this case, she names this kind of results as *Book rating* and selects one of the available selectors generated in relation to the available XPaths for the selected DOM element. This will allow her to choose more than one instance of *Book rating* in the same context (as shown in steps 6 and 8, there are other instances in the background). In a similar way, she should define the properties of the results that are of her interest, so these will be displayed when an ancillary search is performed. For instance, in steps 8 and 9, Amaru is defining the *Title* property for a *Book rating* result of the *GoodReads* Search Service. She repeats the last two steps for defining also a *Rating* property.
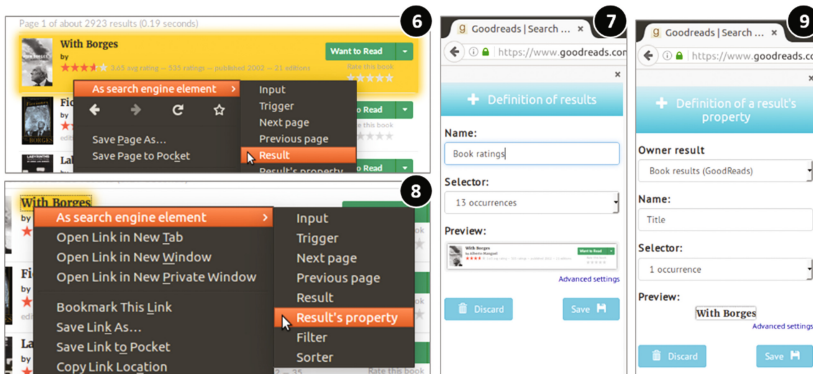

Fig. 8. Defining the expected results for a UI-based Search Service

After the mandatory elements of a *UI-based Search Service* have been defined (input, more-results trigger and the expected structure of the results), the Search Services becomes available in the browser's context menu whether the user has highlighted any text in any Web page. In step 1 of Fig. 9, Amaru has highlighted some text of a Wikipedia article and she is performing an ancillary search using it as keywords. At such point, she can also use other previously defined services, for instance, *Amazon* or *Google Books*. When she clicks one of the menu items, let us say *Amazon*, a draggable panel appears in the middle of the screen, presenting the results of the search for the highlighted keywords. Now, as shown in step 2, she can access related books to *Julio Cortázar* and see their *Title* and *Authors*. However, she can also access the
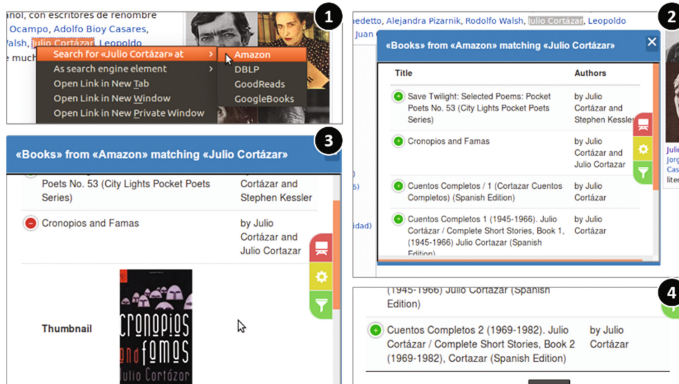
Fig. 9. Visualizing the results of ancillary searches

remaining properties (in this case, the *Thumbnail*) by clicking the «+» button at the left of each row, which will display a section with the data hidden due to the lack of space. There are three fixed buttons at the right of the panel that allow her to: (1) change the kind of visualization – she is using the default one –; (2) to configure some parameters of the selected visualization, as the order or the priority of the columns in the responsive layout; and (3) to apply filters if any was defined. At the bottom of the visualization, as shown in step 4, there are navigation buttons so she can get more results.

Note that the use of the Search Service is not exclusive for the Wikipedia Web site; it is always available in the context-menu of the browser, no matter the Web site the user is navigating. Multiple ancillary searches with the same or diverse Search Services can be performed in the same context, using different keywords, as shown in steps 5 and 6 of Fig. 10. This way, Amaru can search for a second time, by selecting *Rayuela*, one of the books of *Julio Cortázar* listed in the first ancillary search' results popup. This time she is using the Search Service she defined for *GoodReads*, and she is accessing information that was not present in the results of the *Amazon* Search API.

Fig. 10. Performing an ancillary search over the results of a previous one

5 Evaluation

In this section, we present two evaluations of our approach. Section 5.1 proposes a validation by software construction, whose main objectives are to stress our Search Service model to know how it covers the existing search functionality provided by Web applications and to measure the time consumed in real uses of these services. Section 5.2 presents a quantitative assessment to provide some understanding about how our approach influences the user interaction.

5.1 Validation by Construction

The instantiation of Search Services brought different challenges, especially considering that they are built by different UI components and consequently require a different kind of interactions for executing their associated behaviors. For instance, consider the 20 first sites in the top 500 by Alexa[3] meeting the following conditions: (1) the interface is not Chinese or Russian by default; (2) sites of with the same domain but different top-level domain are considered just once (e.g. Google.com and Google.co.in); (3) only consider one instance using the same search engine (e.g. Msn redirects to Bing); (4) do not consider the one with no search engine (t.co). The 20 sites are Google, Youtube, Facebook, Yahoo, Wikipedia, Amazon, Live, Vk, Twitter, Instagram, Reddit, Linkedin, Ebay, Bing, Tumblr, Netflix, Wordpress, Microsoft, Aliexpress and Blogspot. In this list, all sites have a search engine but they are executable by different means.

By analyzing the 20 sites, we can see that the involved UI controls differ in kind and quantity. There is no variation on the kind of DOM element used for entering the search's keywords, it is an input field, but it is different for the remaining controls involved in the search process. Moreover, there is a site hiding the input until the user clicks on a concrete element of the DOM (Live.com), and the search execution strategies (see Sect. 3.3.1) are not always the same: 11 sites use the *WriteAndClickToReload*; 5 the *WriteAndClickForAjaxCall*; 2 the *WriteForAjaxCall*. We successfully defined the service for those 18 of the list of 20 sites with such strategies; but the remaining 2 (Instagram and Live) required different ones that we are currently working on.

Back to the UI components, 17 of the full list of sites have a trigger element, but they differ in kind of control: they were 10 buttons, 6 inputs and 1 anchor. Concerning pagination, 14 sites have a control for retrieving the following elements, but just 12 of them have a control for the previous ones. This is due to the way they handle results in the presentation layer; Instagram and Wordpress have a single DOM element that attaches more results in the results area, expanding its height. Facebook, Live, Vk, Twitter and Tumblr automatically retrieve more results when the user scrolls down to the bottom of the page. The remaining site, Microsoft, have a DOM element with this purpose but clicking on it redirects to a specialized form for searching a concrete kind of

---

[3] http://www.alexa.com/topsites/global; 1.

results, where the results of the first page were included. However, this specialized searcher does not allow changing the search keywords; if you do that, you are redirected to the first results' page. Of the 14 sites with clickable paging elements, 8 of them cause the page to be reloaded while another 6 apply the changes via ajax-call. In this matter, sorting elements are present in 8 sites; 5 of them reload the document and 3 of them use ajax. Filter elements are defined for 16 sites, of which 11 reload and 5 use ajax.

The domain-specific abstraction of search results is another issue to face. Most sites in the list can retrieve more than one kind of result: news, images, videos, channels, playlists, people, pages, places, groups, applications, events, emails, etc. This is not just a problem for naming the kind of results a Search Service retrieves, but for choosing a selector (an XPath) to retrieve a concrete DOM element with a specific structure in the DOM (or all of them) whenever possible, since search engines often present different kinds of results with a different structure/style.

Regarding search performance, we logged the times for the 18 search engines in a 15-inch notebook, with a resolution of 1366×768 pixels. We cannot say the results are the same for other resolutions, since their UI elements may vary. The purpose of this analysis was to demonstrate that our tool allows the instantiation of the Search Services. We successfully defined a Search Service for each of the search engines of the aforementioned 20 sites, and we report below the times it took for executing an ancillary search. For each of them, we performed a search: (1) from exactly the same Web context[4], (2) searching for the same keywords (*Borges*) and (3) expecting to have results with just two textual properties: name and description. There were only two cases in which *Borges* did not produce any results, so the word was replaced by one of the same length: *Ubuntu*. For each test, we cleared the cached and offline Web content, the offline content and user data. We also reloaded the target Web page to augment, for making it sure that no class was already instantiated and giving the entire Search Services the same conditions before executing the search. The full search process took 7 s with a standard deviation of 5 s. The differences depended on the strategy and the response time of the search engines. E.g., there were 3 cases in which it took between 14 and 18 s, while in other 3 sites it took between 1 and 2 s. For a full list of logged times and a video, please visit the project's Web site[5].

This evaluation demonstrated that our approach is feasible, covering 18 of 20 analyzed sites by using our support tool and the 3 implemented strategies to date. The sites for the test were not conveniently chosen, but from a public Web traffic report. Other search engines using the same interaction strategies can be visually specified as Search Service by end users, and such strategies can be also extended by developers to cover more search engines. We also reported the average time in order to compare it against the estimation of times in the following subsection, regarding the time it takes a task to be performed manually or with the support of our tool.

---

[4] https://en.wikipedia.org/wiki/Argentine_literature.

[5] https://sites.google.com/site/webancillarysearches/testing-in-top-sites.

5.2 Quantitative Assessment Based on GOMS-KLM

We present in this section a quantitative assessment of search tasks based on the GOMS-Keystroke (KLM) model [5]. This formal model is used to evaluate the efficiency of interaction with a given software for a specific, very detailed, scenario. The resulting time is calculated by the summation of each user action, whose time are already known in this model. For example, the average time to perform the action *reach* with the mouse is 0.40 s or *click on a button* is 0.20 s. Thus, by providing a detailed scenario of user actions with the Web browser and Web applications, it is possible to use GOMS-KLM to predict performance.

We focus this quantitative assessment for the case of ancillary searches. By using GOMS-KLM, we have specified the traditional scenario for performing ancillary searches, shown in Fig. 2, in order to compare it against our approach, introduced in Fig. 3. We have split the whole scenario into six tasks: *(i)* Visit ICWE2016's Web site, *(ii)* select an author, *(iii)* search for that author in DBLP, *(iv)* select a resulting paper, *(v)* search for that paper in Google Scholar, *(vi)* point the mouse over the article's title. The results are shown in Table 1. Note that tasks *i, ii, iv* and *vi* are equivalents in both scenarios. However, the time required for tasks *iii* and *v*, the ones actually requiring to use the search engine provided by other Web sites, are quite faster using our approach. The whole scenario (involving two ancillary searches) takes 46.6 s without our approach. In contrast, by using Search Services, it takes just 18 s. It is interesting to note that performing a search using our approach and visualizing the results from any Web site would take only 1.5 s (this time would be always the same, regardless both the current Web site and the target Search Service).

Table 1. GOMS-KLM results for both scenarios

Task	Time (without using SS)	Time (using SS)
1. Go to the ICWE web site	8.7 s	8.7 s
2. Select the target author	2.6 s	2.6 s
3. Perform the first ancillary Search (DBLP)	15.9 s	1.5 s
4. Select an article	2.6 s	2.6 s
5. Perform a second ancillary Search (Google Scholar)	15.7 s	1.5 s
6. Point the title of a target paper (Google Scholar)	1.1 s	1.1 s
Total	**46.6 s**	**18 s**
Define both Search Services	-	*39.2 s*

However, for using the Search Services of DBLP and Google Scholar, it is required that the user has previously defined them. With this in mind, we have defined also the GOMS-KLM scenarios related to the creation of both DBLP and Google Scholar Search Services. They were defined considering: *(i)* input search form, *(ii)* trigger search UI component, *(iii)* abstract the search result into the concept "*Paper*" with the property *title*, *(iv)* give a name to the Search Service and save it. All the tasks take 19.6 s.

The specification is the simpler, without filtering nor ordering options, since such features were not required for the contemplated scenarios. This means that the first time the user experiences the scenario in Fig. 3, he needs an extra time of almost 40 s. However, the definition is required just once and it can be avoided by installing an existing Search Service specification.

As a final comment, it is interesting to note that in the context of ancillary searches, the user would go back to the primary context, in this case, the ICWE's Web site. With our approach, it would not require further interaction because the user already is in ICWE's Web site; meanwhile in a traditional scenario it will require further interaction.

6 Conclusions and Future Works

In this paper, we presented an end-user driven approach for the customization of Web search tasks. The main objective is to get better support for ancillary searches, although the approach also reaches primary searches. Several contributions are made in this context. First, we propose an end-user support tool for the creation of Search Services based on the automatic execution of UI interaction required to perform searches in existing and third-party search engines. Second, we propose to transform search results into domain objects and take advantage of the semantic information as in [8]. Both together achieve a third contribution, which is the creation of new ways of interaction and visualization of results in-situ.

We fully supported our approach with already working tools, used in 18 existing Web applications as a way of validating our aims by software construction. Besides, we have shown that our approach is very convenient in terms of performance when users require complementary information for accomplishing their tasks.

Our approach still lacks a full end-user evaluation to measure the potential of adoption and the usefulness of in-situ visualizations of results. Another evaluation with end-users is necessary to demonstrate that the specification of Search Service is clearly doable without programming skills. However, beyond some usability issues, the abstraction of Search Engine components may not be a limitation because of the seamless observed in existing search engines and their common use nowadays.

Beyond further evaluations and the improvement of our tools (e.g. to support more strategies to cover more search engines), some other works are planned. First, although we foresee the usefulness of defining metadata for Search Services, we have not exploited it yet. For instance, this kind of information could be used for automatically perform searches in parallel given a particular context of information. Collaboration is another aspect to be addressed. So far, we allow users to share service specifications by sending their corresponding files, but we want to analyze how to reach this goal through collaborative techniques better. Finally, domain-specific visualizers could improve how end-users interact with the information obtained by our search services.

References

1. Aula, A., Jhaveri, N., Käki, M.: Information search and re-access strategies of experienced web users. In: WWW 2005 Proceedings of the 14th International Conference on World Wide Web, pp. 583–592 (2005)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. **284**, 34–43 (2001). doi:10.1038/scientificamerican0501-34
3. Broder, A.: A taxonomy of web search. SIGIR Forum **36**, 3–10 (2002). doi:10.1145/792550.792552
4. Card, S.K., Mackinlay, J.D., Shneiderman, B.: Focus + context. In: Readings in Information Visualization, pp. 306–309 (1999)
5. Card, S.K., Moran, T.P., Newell, A.: The Psychology of Human-Computer Interaction. L. Erlbaum Associates, Hillsdale (1983)
6. Cava, R., Freitas, C.M.D.S., Barboni, E., Palanque, P., Winckler, M.: Inside-in search: an alternative for performing ancillary search tasks on the web. In: 2014 9th Latin America Web Congress IEEE, pp. 91–99 (2014)
7. Ellis, D., Haugan, M.: Modelling the information seeking patterns of engineers and research scientists in an industrial environment. J. Doc. **53**, 384–403 (1997). doi:10.1108/EUM0000000007204
8. Firmenich, S., Bosetti, G., Rossi, G., Winckler, M., Barbieri, T.: Abstracting and structuring web contents for supporting personal web experiences. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 77–95. Springer, Cham (2016). doi:10.1007/978-3-319-38791-8_5
9. Golovchinsky, G., Qvarfordt, P., Pickens, J.: Collaborative information seeking. Computer (Long Beach Calif.) **42**, 47–51 (2009). doi:10.1109/MC.2009.73
10. Hearst, M.A.: User interfaces for search. In: Modern Information Retrieval the Concepts and Technology Behind Search Engines, pp. 21–56 (2010)
11. Hearst, M.A.: "Natural" search user interfaces. Commun. ACM **54**, 60–67 (2011). doi:10.1145/2018396.2018414
12. Kumar, R., Tomkins, A.: A characterization of online search behavior. IEEE Data Eng. Bull. **32**, 1–9 (2009). doi:10.1145/1772690.1772748
13. MacLean, A., Carter, K., Lövstrand, L., Moran, T.: User-tailorable systems: pressing the issues with buttons. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Empower people - CHI 1990, pp. 175–182 (1990). doi:10.1145/97243.97271
14. Marchionini, G.: Exploratory search: from finding to understanding. Commun. ACM **49**, 41 (2006). doi:10.1145/1121949.1121979
15. McCallum, A., Nigam, K., Rennie, J., Seymore, K.: A machine learning approach to building domain-specific search engines. In: Proceedings of the Sixth International Joint Conference on Artificial Intelligence, IJCAI 1999, pp. 662–667 (1999)
16. Morris, D., Morris, M.R., Venolia, G.: SearchBar: a search-centric web history for task resumption and information re-finding. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems – CHI 2008, pp. 1207–1216 (2008)
17. Obal, C., Diaz, O.: The augmented web: rationales, opportunities and challenges on browser-side transcoding. ACM Trans. Web **9**, 1–30 (2015). doi:10.1145/2735633
18. Shneiderman, B.: The Eyes have it: a task by data type taxonomy for information visualizations. In: Proceedings of 1996 IEEE Symposium on Visual Language, pp. 336–343 (1996)

19. Wilson, T.D.: Models in information behaviour research. J. Doc. **55**, 249–270 (1999). doi:10.1108/EUM0000000007145
20. Winckler, M., Cava, R., Barboni, E., Palanque, P., Freitas, C.: Usability aspects of the inside-in approach for ancillary search tasks on the web. In: Abascal, J., Barbosa, S., Fetter, M., Gross, T., Palanque, P., Winckler, M. (eds.) INTERACT 2015. LNCS, vol. 9297, pp. 211–230. Springer, Cham (2015). doi:10.1007/978-3-319-22668-2_18