An end-user tool for supporting frequent multisource web search workflows

Alex Tacuri[1][a], Gonzalo Ripa[1,2], Alejandro Fernandez[1][b], Gustavo Rossi[1,3][c], Juan Cruz Gutierrez Saravia[1] and Sergio Firmenich[1,2][d]

[1]*LIFIA, CIC, Facultad de Informática, UNLP, Argentina*

[2]*CONICET, Argentina*

[3]*Facultad de Tecnologia Informatica, Universidad Abierta Interamericana*

alex.tacuri@lifia.info.unlp.edu.ar

Keywords: web scraping, web search, end-user development, prompt engineering

Abstract: Current web browser support for search tasks often lead to tedious and time-consuming processes especially when information needs span across various domains and when these tasks are done frequently. This paper addresses this challenge by proposing an approach that empowers end users to seamlessly conduct complex web search workflows. By prioritizing end-user programming and controllability, our approach enables users to customize and automate their search processes according to individual preferences and needs. We emphasize the importance of maintaining user control and awareness of information sources. Through our approach, we aim to enhance user experience and efficiency in accessing information across diverse online platforms.

1 Introduction

To initiate a search, users commonly rely on either a crawler-based search service like Google or Bing, or they utilize the search functionality of a particular website when they need to retrieve a domain-specific information objects (Bhavnani, 2002). Both methods are crucial for searching in various contexts, and both may entail conducting ancillary searches (Bosetti et al., 2017) to acquire supplementary information. However, certain web tasks associated with accessing information require conducting searches across multiple websites. When such tasks are performed frequently, the process can become tedious and time-consuming. In a very interesting study about the user behaviour when searching, previous works have classified users between 'navigators' and 'explorers', and although explorers seem to do more branches when looking for information, it is valid for both to say that users utilize more than just one website when searching for information in order to accomplish a task (White and Drucker, 2007).

For example, researchers in any academic field typically have a set of repositories or web applications from which they access relevant information for their work using the search engines provided by those applications. During a typical workday, researchers may need to search for papers on a specific topic on the Springer website. Subsequently, for several of the resulting papers, they might wish to find additional information from other sources. This could involve checking the number of citations on Google Scholar, finding more articles by the same authors on DBLP, or examining scientific ranking services for more details about the journal or conference where the paper was published. This process goes beyond a simple web search; it constitutes a search workflow that demands greater interaction and time. Furthermore, additional effort is required if the researchers aim to consolidate the information scattered across different sources due to the absence of a unified user interface that supports the overall task.

Similarly, various professions, occupations, or activities on the web rely on this kind of workflows using specific websites or web applications. While web browsers, whether mainstream or not, typically offer sufficient support for such search workflows when they are conducted infrequently — often managed through tab management by end users — using them on a daily basis presents challenges. In fact, if multi-

[a] https://orcid.org/0000-0003-3159-5556
[b] https://orcid.org/0000-0002-7968-6871
[c] https://orcid.org/0000-0002-3348-2144
[d] https://orcid.org/0000-0001-9502-2189

ple search workflows are carried out simultaneously, the user experience provided by web browsers is often inadequate.

In the past, various works have addressed this issue through different approaches such as mash-ups (Stolee et al., 2013) and web augmentation tools (Bosetti et al., 2017). Regardless of the final product or outcome, the process of creating these tools has also been the focus of research, resulting in both code-oriented tools and end-user development ones. Despite living in an API-driven world, it remains true that a significant number of websites do not offer APIs, which complicate integration. Furthermore, even if every website were to provide an API, it would still be complex for users to consume them, despite interesting approaches such as ScrAPIr (Alrashed et al., 2020) or the significant advancements in service composition. As we transition towards a world with a growing presence of AI agents, it is unclear which role web searches will play in general information retrieval. It becomes important to consider controllability, a term very well-known in user interface adaptation and recommender systems (Jameson and Schwarzkopf, 2002). In the context of the approach, it means in the best of the cases that users may decide which information sources to use, and in the worst, at least to let them known from where the information is obtained. However, it is certain that various user activities will still require searching information, whether on the internet or on intranets. With the fast advances on AI-based tools and agents, it is clear that web searches and search results consumption surely will suffer profound changes in several dimensions. In this work we focus on how the kind of search workflows described before may be supported, how search results are integrated and how these results are presented to users.

This paper introduces an approach aimed at empowering end users with the ability to seamlessly perform complex web search workflows across diverse online platforms. By placing emphasis on end-user programming and controllability, our approach equips users with the tools to effortlessly customize and automate their search processes according to their unique preferences and requirements. In that process, we strongly believe that generative AI models are useful for different construction steps for a final product, but what would be extremely important is that users still maintain control and knowledge on which information is used and from where that information was obtained.

The article is structured as follows. Section 2 provides an overview of the approach by means of a motivating example. Section 3 explains how search APIs are defined with support from generative AI, whereas Section 4 discusses how search APIs are composed. Section 5 presents the UI for search workflows through an example. Section 6 discusses related works. Section 7 presents conclusions and future works.

2 The approach in a nutshell

This approach focuses on those search workflows that are composed by a sequence of search tasks performed in several websites, combining different information pieces that are provided by each specific website to build more complete search results. To illustrate this concept, we will detail an example on the research domain, where a user needs to perform a very typical and daily research task.

Firstly, the search workflow starts by searching some topic in the Springer repository. From that process, the researcher is interested in several of the resulting articles, and for each one of them requires further information that is not present at Spring. In this regard, the researcher probably opens a new tab to open Google Scholar to search those relevant papers from the first search, in order to look its citations. Also the researcher could be interested in opening another browser tab to search at DBLP for further papers of authors to see if authors have newer publications on the same research line.

This simple search workflow implies to use one main search service (Springer), and for each relevant resulting paper, to open at least two more websites to look further information. It is up to the user to remember the semantic connections among tabs, and to maintain consistency when navigating in any of them.

A workflow like this one could be easily automated by a developer if all of these web applications offered APIs to retrieve information. But in absence of these APIs, and mainly from the point of view of end users without programming skills, a new approach is needed. Such new approach should support end users in defining and triggering complex searches connecting results from multiple sites, and presenting them in a useful and user-friendly mode. Its building blocks are: a) uniform search APIs for any web-site, defined by end users; b) composition of search results from multiple search APIs to obtain navigable data objects; c) straight-forward triggering of searches; d) user-friendly presentation of search results.

In previous work (Bosetti et al., 2022), we proposed a mechanism to aid users in creating uniform search APIs for websites or web applications that offer search functionality. This was done by emulating

users interaction with websites and applying information extractors that were specified by web annotations, which do not require programming skills. Later, we proposed a visual programming tool to combine search APIs and create web scrapers that integrate information spread in several websites (Tacuri et al., 2023). In this article we extend previous visual programming tool to create search workflows that compose search APIs, and by discussing alternative approaches for creating search APIs based on generative AI. In this regard, we designed and developed a web extension as a toolset implementation, whose components overview is shown in Figure 1.

This involves:

- Search API tool: this tool allows end users to create search APIs for those websites that do not provide one, and also define an execution engine of these specifications. In previous works this component was based on an end-user development tool that uses web annotation to define the search API. In this work, we present another approach for search API definitions that is supported in existing generative AI.
- Search API composition tool: it provides a development tool that allows end users without programming skills to define a search API composition using a visual editor. In this way, end users may define the search workflows, i.e. how result's properties values are used as inputs to trigger another search with other search API. This component, similar to the previous one, also provides an execution engine that coordinate the execution of search APIs.
- User Interface: we provide a simple graphical user interface in order to allow users to interact with the results obtained from the execution of a specific search workflow. In this work, this GUI interacts with the Search API composition execution engine to obtain search workflows results.

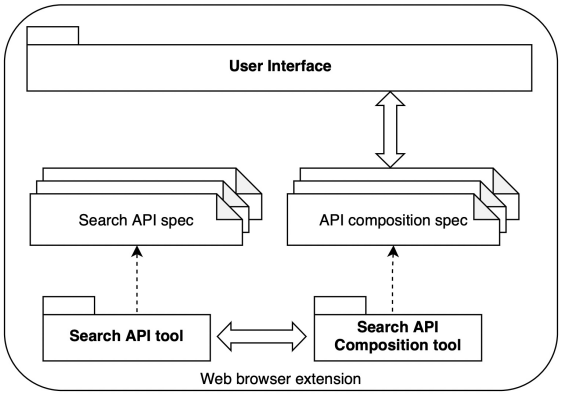

Figure 1: Web extension overview

3 Search APIs

3.1 ANDES Search APIs

This work is based on ANDES' concept of Search APIs, which was published before (Bosetti et al., 2022). ANDES offers tools to annotate the UI search components (search input, trigger button, pagination buttons, etc.), but also is the user who defines the type of the resulting object, its properties, and the mapping from the content of the results web page to each object property values. Calling the search operation on the API triggers the corresponding search and scrapping behavior and returns a collection of (JavaScript) objects. In that work, ANDES was mainly focused on provide better support for ancillary searches. In other works, we also worked on a similar concept, not improving the interaction for ancillary searches, but embedding them into a data service layer that helps web extensions developers to don't have harcoded DOM references in their code, but delegating that online information retrieving to this data service layer (Tacuri et al., 2023). Further details about ANDES can be found in those previous works (Bosetti et al., 2022).

However, in order to reduce the necessary interaction to define a search API, we developed a prototype using existing generative GPT technologies to solve specific steps in the search API definition process. This approach is presented in the following subsection.

3.2 AI-assisted Search APIs creation

In this work we explore the creation of search APIs, APIs similar to the ones proposed in ANDES, by using existing generative AI technologies capabilities. After several tests, we found an efficient way to develop search APIs by dividing the process in small

tasks. For solving each of these tasks we based the new approach on prompt engineering, taking advantage of large language model (LLM) technologies currently available, particularly OpenAI's GPT series of models. We developed a tool that guides end users during this short process, giving proper feedback through messages appearing on top of Web sites. This speeds up considerably the process of creation, eliminating the annotation tasks that end users needed to accomplish to build the Search APIs in ANDES.

3.2.1 Creation

This creation process consist in several steps including:

1. Detects a parameterized URL search expression.
2. Identify a common xPath from the results web page, necessary to access the results objects during real time searches.
3. Build a template of semantic properties that describe a result object.

During this process, our tool coordinates the interaction required by part of users and the calls required to OpenAI's API. Figure 2 depicts this process.

Step 1. To accomplish this step, users need to open a website they are interested in to create its Search API. Our tool detects when users are in a new website and ask them to perform a few searches using the corresponding search input elements (step 1 in Figure 3). The tool also detects the keywords searched by the user giving feedback with proper messages showed on top of the screen during the three searches required (step 2 in Figure 3).

After three searches are completed, the prompt engineering starts taking action, and when it is finished, user gets feedback (step 3 in Figure 3). Behind scenes, our tool communicates with the OpenAI's API, sending a request using the gpt-3.5-turbo-0125 model, with the next prompt:

Prompt 1: *"You are a helpful assistant. Based on these examples: {**setOfUrls**}, give me a JSON with searchToken and url properties."*

where {**setOfUrls**} corresponds to an array containing the search urls detected previously. Then, OpenAI's service will respond with a JSON object containing searchToken and url properties, used later to trigger real searches. An example of a url expression returned is: *"https://listado.mercadolibre.com.ar/{searchToken}"*.

Step 2. After URL is obtained, the second step needs to be done, and our tool will highlight with a red color the sections of the web page where the user is pointing with the cursor (Figure 4). To accomplish 2, the user needs to click on the section that better represents a search result and includes as much information as possible of it. The tool will generate a common xPath expression that will identify all search result elements, necessary at the moment of performing real searches using the search APIs created.

Step 3. Once our tool knows how to obtain DOM elements that represent search results, it is necessary to extract information from them and generate a information object's template. Different to ANDES's approach, where users need to select and configure each property for the selected search result in order to know how to extract information, this new approach communicates again with OpenAI's service, in order to parse the full text content of a particular search result to construct a JSON object, detecting its semantics and properties. In this case, the prompt defined was the following:

Prompt 2: *"You are a helpful assistant. Generate an object JSON-LD using the schema.org context without aggregation and with each property represented as a string, for the next info: {**domElementInnerText**}. Do not nest objects and don't give me a summary. The resulting JSON must have at least a 'name' property, a '@type' property giving a semantic classification, and only have string type properties."*

where {**domElementInnerText**} corresponds to the innerText obtained by the tool when the user clicked on a search result Web page element. The returned JSON will be a template of semantic properties that describe the result object. Finally, the tool will let users customize the final template with the properties they are interested to show in final search results (Figure 5), ending with 3.

3.2.2 Search API execution

To can execute searches using the Search APIs generated through the process described in 3.2.1, the tool will use the common xPath obtained that let to identify each result of the search and get unstructured information through the innerText property of the DOM element, to then communicate with OpenAI's API one more time, sending the next prompt:

Prompt 3: *"You are a helpful assistant. I will provide you with an innerText and you have to generate a JSON-LD object using the schema.org context based on the next JSON template, without aggregation and with each property represented as a string: {**jsonTemplate**}. If some property from the template*

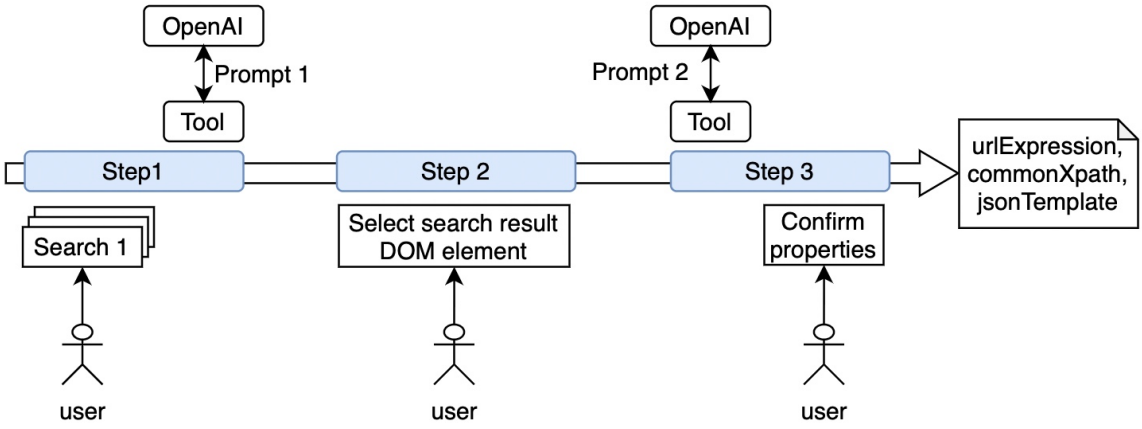

Figure 2: Search API definition process

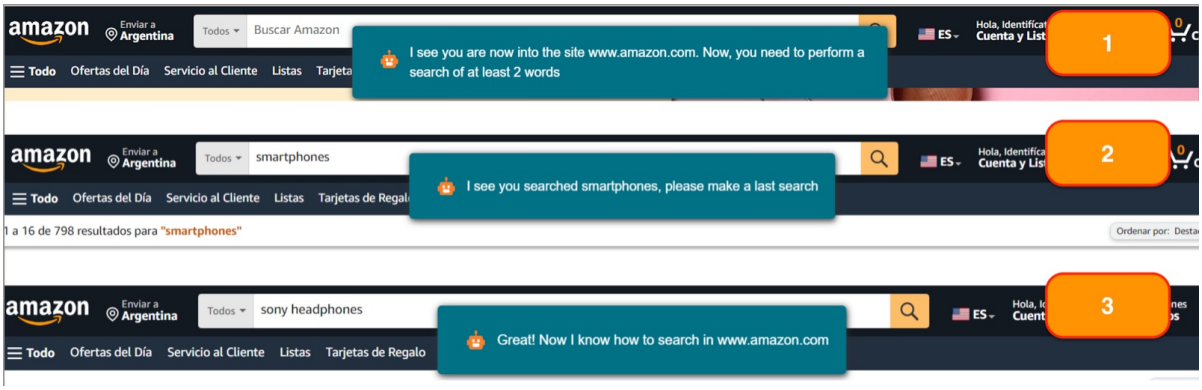

Figure 3: Website's URL search detection

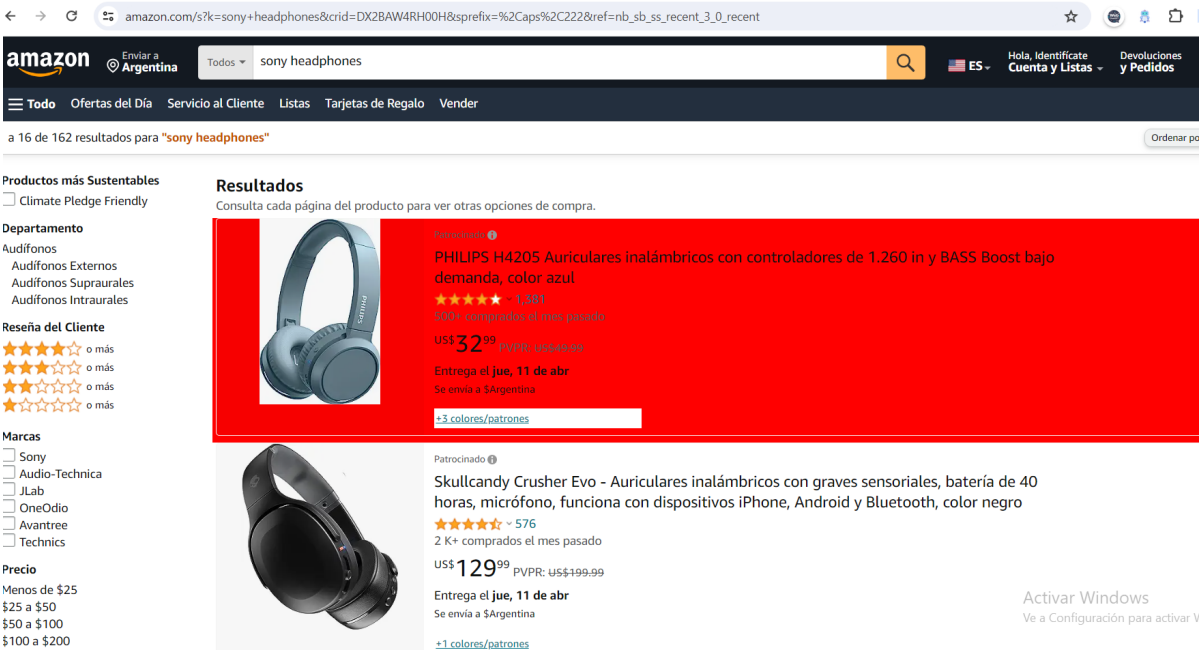

Figure 4: Result element DOM selection

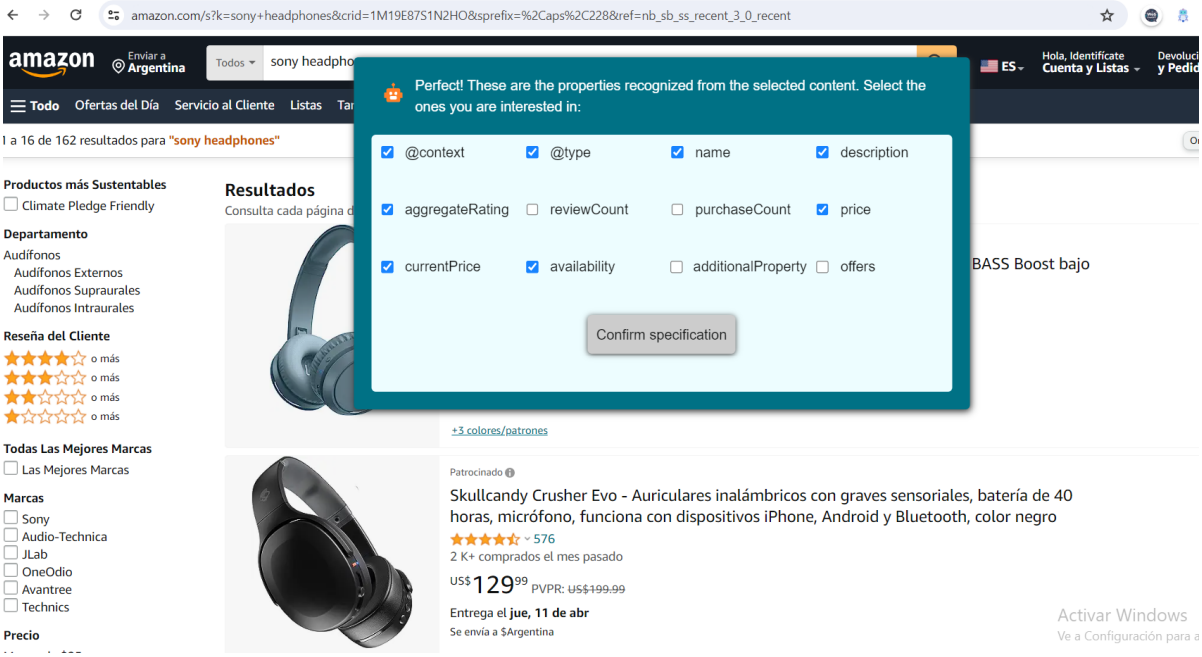

Figure 5: Properties selection

is not found, just complete it with an empty string. Also if you find some new property not present in the template add it to the element. Do not nest objects. The resulting JSON object must have at least a 'name' property, a '@type' property giving a semantic classification and only have string type properties. This is the innerText: {***domElementInnerText***}*."*

where {**jsonTemplate**} corresponds to the template of semantics properties that is part of the specification created, and {**domElementInnerText**} corresponds to the innerText of a search result obtained using the common xPath expression, both generated during Search API creation process described previously. The search execution will return a final JSON with all the results' information exposed as objects with properties that will be based on {**jsonTemplate**}.

3.2.3 Tests

Regarding the development of search APIs involving Prompt 1 and Prompt 2, the results were very promising, demonstrating excellent detection of search URLs, semantic objects, and template structures across 22 websites spanning four distinct domains.

Using the template derived from the search API definition (crafted by selecting all properties detected by OpenAI services), we conducted tests to parse all results for specific searches on each of the 22 websites, essentially employing Prompt 3. The outcomes are presented in Table 1. The column descriptions are outlined below:

- **Website**: The targeted website.
- **Properties**: The number of properties detected during the creation of the search API.
- **Complete**: The number of search results for which the OpenAI service successfully found values for the given template. For example, for MercadoLibre, the value '5/6' indicates that 5 out of 6 retrieved information objects were complete.
- **Blank AVG (%)**: The average percentage of properties for which the value was blank, i.e., where OpenAI could not find a value.
- **Blank STDV (%)**: The standard deviation of the percentage of properties for which the value was blank.
- **With new props**: The number of information objects returned by OpenAI that have additional properties compared to those specified in the template. For example, for MercadoLibre, the value '3/6' indicates that for 3 out of 6 search results, OpenAI service could extract new properties compared to the original template.
- **New props AVG (%)**: The average percentage of new properties.
- **New props STDV (%)**: The standard deviation of the number of new properties.

Overall, the values for all properties of the information objects were accurate, even when the properties were not explicitly specified in the original template. It is noteworthy that in most cases, the completeness of objects was not 100%; however, considering the average of blank properties, it is very low in most instances.

4 Search API composition

To start talking about the composition of search APIs, let's recall an example that we have all experienced: when we search for information, we begin with a search engine and then explore other sites to gain a more comprehensive understanding of the topic. With this premise in mind, let's revisit our motivational example from Section 2. In this example, we start by searching for a specific topic on Springer, and from the results obtained, we look for the number of citations, or link to the document, or a PDF file on Google Scholar, and additionally, we would like to know what other articles the author has written using the DBLP database.

Thanks to the development of our tools, we can now perform the composition or integration of search APIs even without programming knowledge.

The tool enables the creation of various composition models for search APIs, allowing searches to be conducted across each of them. To set up a composition model, it's necessary to first have the search APIs of the relevant websites. For instance, for our motivational example, we need access to the services of Springer, Google Scholar, and DBLP.

Once we have the search APIs ready, we'll create a model in the tool as shown in Figure 6. To do this, the extension provides a graphical environment with drag-and-drop components; one of them represents each of the previously generated APIs. Additionally, the tool displays a canvas where we add the APIs we need and then establish their integration or composition. The composition between search services can be one-to-one, meaning to connect to the first object that meets the search criteria, or one-to-N, meaning to return a collection with all the result.

Returning to our motivational example, we need to add the 3 APIs from Springer, Google Scholar, and DBLP to the canvas. Now, let's proceed to establish the connections between each of them. Firstly, let's analyze the connection between Springer and DBLP. We need that, for each element of Springer, a list of articles from DBLP authored by the first author be obtained. We must assign a name to this link, which in the example is "Articles". We must also select which property of the Springer object will serve as input data for the search in the DBLP search API, which in this case is "Author". Finally, we must indicate the number of resulting elements, which in this case will be all. The next link will be between Springer and Google Scholar. The assigned name is "Citations", the property that will serve as search text will be "Title", and the number of resulting elements will be the first element containing the searched item.

5 Generic Workflow User Interface

The interface we have implemented to showcase this approach displays the results obtained from both the 'Search API Tool' and the 'Search API Composition Tool'. These tools provide us with information in JSON format used to create this interface. The example below shows the data obtained from a search performed in the 'Research' model, which was described in the motivational example.

The interface initially presents a form with a textbox for entering the search text and a menu to select the workflow, as depicted in Figure 7. When a workflow is triggered, and behind the scenes, the tools execute the search APIs composition and integrate their results. For example, we obtain the "title", "abstract", and "authors" of each Springer result, the "number of citations", and the PDF link from Google Scholar. Additionally, clicking on the author of some Springer article, displays a dialog with the search results for the author's articles in DBLP as depicted in Figure 7.

This generic user interface is quite simple. It lists all the information object's properties, and when one of these properties have been used to trigger the execution of another search API that returns a collection of objects, it is automatically transformed to an anchor widget, which offers an interactive element that opens a modal showing that collection. Also, in order to offer users with feedback about the information's source, the UI adds the favicon of the source application.

We strongly believe that this UI could be improved in several ways, however it is clear that if we compare how long it would take to a user to perform this information search using the traditional method, the time and interaction is quite different. For instance, a search for "artificial intelligence" in our tool takes about 10 seconds to perform approximately 60 searches on 3 websites, whereas the traditional method would take much longer to perform 60 searches on the three sites and additionally integrate the results.

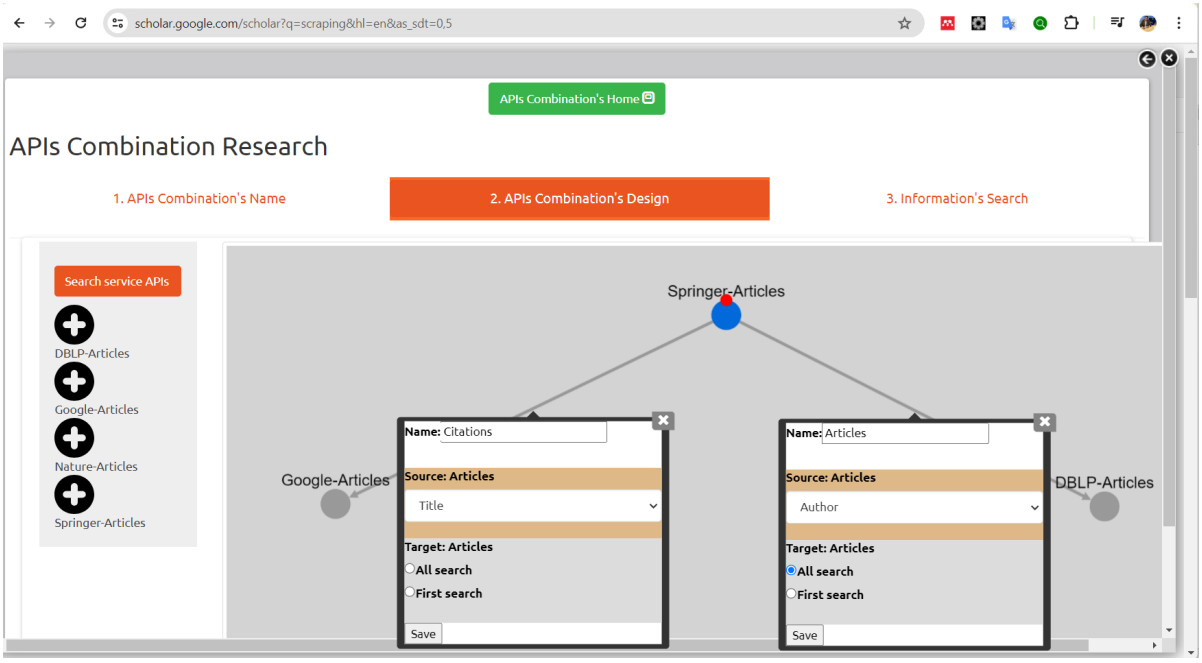

Figure 6: The API composition tool: Integration between Springer and DBLP search APIs

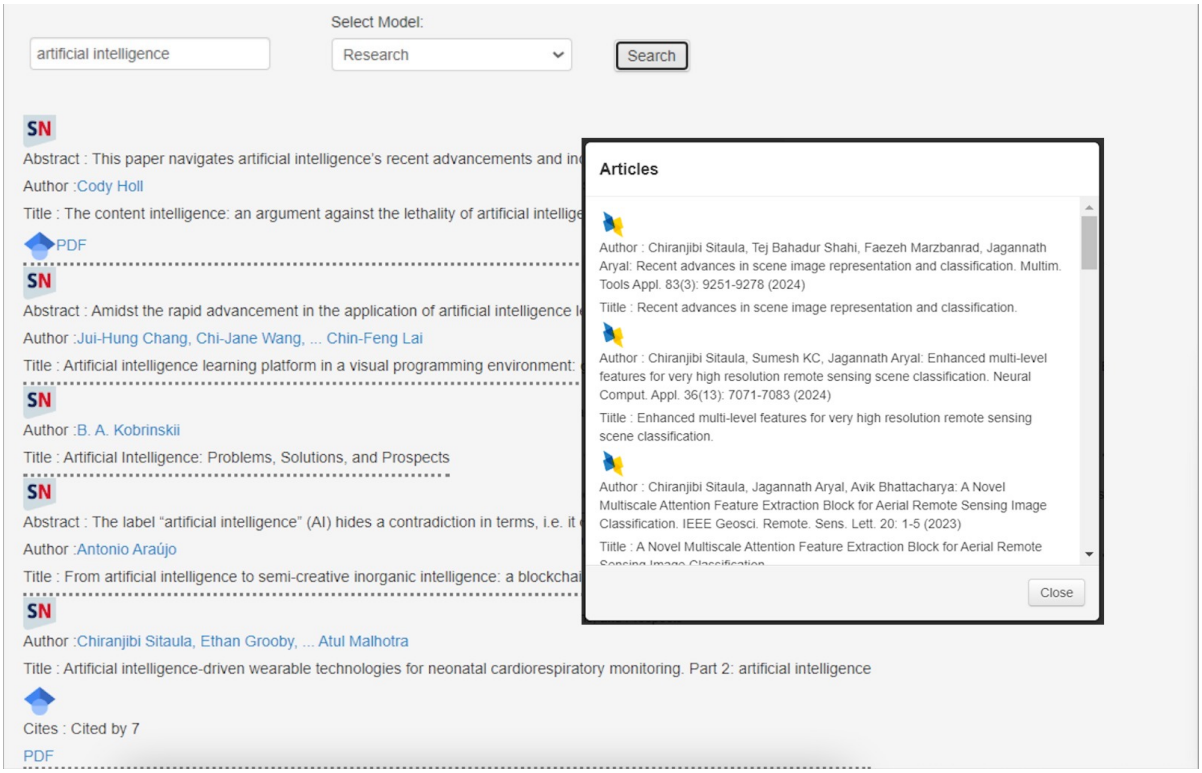

Figure 7: User interface that utilizes the integration of search services from Springer, DBLP, and Google Scholar

6 Related works

There are various dimensions to consider when discussing the approach outlined in this paper within the context of existing literature. To maintain conciseness, we will focus, on one hand, on aspects related to enhancing user experience while navigating the web through external structures (Bouvin, 2019). On the other hand, we will delve into techniques associated with scraping online information to construct structured information objects.

Improving user experience without relying on web application owners has been addressed for over 20 years through techniques like web mashups and web augmentation. The former primarily involves integrating content and services to create new applications, while the latter is more focused on adapting existing web pages that fail to meet user requirements.

In the realm of mashups, notable approaches such as Yahoo Pipes (Stolee et al., 2013) and Marmite (Wong and Hong, 2007) empower end users without programming skills to define workflows combining services and content from multiple sources to produce new applications. The generic user interface proposed in this article for rendering search workflow results shares similarities with a mashup, but with the added capability of generating a search API for web applications lacking one. Although inspired by mashup approaches, our search API composition tool is exclusively tailored for search workflow specification.

Concerning web augmentation (Bouvin, 1999), various approaches aim to integrate information with the primary goal of reducing the interaction needed to access it. Some focus on improving information searches, including methods supporting ancillary searches (Bosetti et al., 2017), and others closer to our approach, like BEAUD (Aldalur, 2023), which enhances web page content with information from other sources. Unlike BEAUD, our approach functions as a search engine, orchestrating searches to present a list of results sourced from multiple platforms.

Web browser extensions are commonly used to enhance browsing experiences, with several works proposing end-user programming tools to specify these extensions. Many such approaches emphasize information extraction from websites. Web scraping, the process of obtaining structured information from non-structured or partially structured website data, is widely utilized in web browser extensions (Tacuri et al., 2023), and numerous scientific works propose web extensions implementing scraping and web semantic technologies (Huynh et al., 2007), similar to our toolset's methodology.

Several approaches enable users to specify existing content structures to manage relevant information objects more effectively. For example, HayStack (Karger et al., 2005) offers an extraction tool for populating a semantic-structured information space. Atomate it! (Kleek et al., 2010) provides a reactive platform to set rules for collected objects, alerting users to noteworthy events. (Van Kleek et al., 2012) facilitates domain-specific application creation working over objects defined in a PIM. Rousillon, an intriguing approach based on end-user programming, allows scraper definition based on hierarchical data (Chasins et al., 2018). Another approach presented in (Katongo et al., 2021) enables web customization through user-defined web scraping without programming skills. Although our previous work, ANDES, for defining search APIs operates similarly to these approaches, this paper introduces structured information object creation by consuming existing generative AI models. Notably, all these semantics extraction-based approaches do not primarily focus on enhancing search processes.

7 Conclusions and future works

In this paper, we have presented an end-user tool designed to enhance the web search experience, particularly for users engaged in frequent multisource search workflows. Our tool addresses several critical aspects of modern web user experience, including the management of searches across multiple web applications and the integration of AI to streamline the search API specifications, reducing the interaction by part of the user in order to specify the semantic structure of the web content.

Through the incorporation of AI technologies, we have aimed to improve the efficiency and effectiveness of scraping specification process and maintaining user's controllability.

Our tests conducted utilizing OpenAI service to create structured information objects from DOM elements' plain text have provided promising results, demonstrating the potential of generative AI in supporting websites scraping. However, we acknowledge that further testing is required to fully explore the capabilities and limitations of generative AI in this context. The emergence of generative AI presents exciting new opportunities for improving information access and search workflows specification. However, it also introduces challenges, particularly regarding the need for structured data and innovative prompt engineering techniques to harness the full potential of AI-based service composition, which is one of the aspects we will study in the future.

REFERENCES

Aldalur, I. (2023). BEAUD: A browser extension to automatize end-user deeds. *Softw. Impacts*, 17:100516.

Alrashed, T., Almahmoud, J., Zhang, A. X., and Karger, D. R. (2020). Scrapir: Making web data apis accessible to end users. In Bernhaupt, R., Mueller, F. F., Verweij, D., Andres, J., McGrenere, J., Cockburn, A., Avellino, I., Goguey, A., Bjøn, P., Zhao, S., Samson, B. P., and Kocielnik, R., editors, *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pages 1–12. ACM.

Bhavnani, S. K. (2002). Domain-specific search strategies for the effective retrieval of healthcare and shopping information. In Terveen, L. G. and Wixon, D. R., editors, *Extended abstracts of the 2002 Conference on Human Factors in Computing Systems, CHI 2002, Minneapolis, Minnesota, USA, April 20-25, 2002*, pages 610–611. ACM.

Bosetti, G., Firmenich, S., Fernández, A., Winckler, M., and Rossi, G. (2017). From search engines to augmented search services: An end-user development approach. In Cabot, J., Virgilio, R. D., and Torlone, R., editors, *Web Engineering - 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings*, volume 10360 of *Lecture Notes in Computer Science*, pages 115–133. 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.

Bouvin, N. O. (1999). Unifying strategies for web augmentation. In Westbomke, J., Wiil, U. K., Leggett, J. J., Tochtermann, K., and Haake, J. M., editors, *HYPERTEXT '99, Proceedings of the 10th ACM Conference on Hypertext and Hypermedia: Returning to Our Diverse Roots, February 21-25, 1999, Darmstadt, Germany*, pages 91–100. ACM.

Bouvin, N. O. (2019). From notecards to notebooks: There and back again. In Atzenbeck, C., Rubart, J., and Millard, D. E., editors, *Proceedings of the 30th ACM Conference on Hypertext and Social Media, HT 2019, Hof, Germany, September 17-20, 2019*, pages 19–28. ACM.

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.

Huynh, D., Mazzocchi, S., and Karger, D. R. (2007). Piggy bank: Experience the semantic web inside your web browser. *J. Web Semant.*, 5(1):16–27.

Jameson, A. and Schwarzkopf, E. (2002). Pros and cons of controllability: An empirical study. In Bra, P. D., Brusilovsky, P., and Conejo, R., editors, *Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002, Malaga, Spain, May 29-31, 2002, Proceedings*, volume 2347 of *Lecture Notes in Computer Science*, pages 193–202. Springer.

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.

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.

Stolee, K. T., Elbaum, S. G., and Sarma, A. (2013). Discovering how end-user programmers and their communities use public repositories: A study on yahoo! pipes. *Inf. Softw. Technol.*, 55(7):1289–1303.

Tacuri, A., Firmenich, S., Rossi, G., and Fernández, A. (2023). A data service layer for web browser extensions. In García-Peñalvo, F. J. and Marchiori, M., editors, *Proceedings of the 19th International Conference on Web Information Systems and Technologies, WEBIST 2023, Rome, Italy, November 15-17, 2023*, pages 49–58. SCITEPRESS.

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

White, R. W. and Drucker, S. M. (2007). Investigating behavioral variability in web search. In Williamson, C. L., Zurko, M. E., Patel-Schneider, P. F., and Shenoy, P. J., editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 21–30. ACM.

Wong, J. and Hong, J. I. (2007). Making mashups with marmite: towards end-user programming for the web. In Rosson, M. B. and Gilmore, D. J., editors, *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*, pages 1435–1444. ACM.

Table 1: OpenAI parsing results

Website	Properties	Complete	Blank AVG (%)	Blank STDV (%)	With new props	New props AVG (%)	New props STDV (%)
Mercadolibre	6	5/6	2,77	6,80	3/6	30,55	35,61
Fravega	5	8/8	0	0	0/8	0	0
Amazon	8	4/9	9,72	12,14	2/9	4,16	8,83
eBay	12	2/9	9,25	6,51	1/9	3,70	11,11
Alibaba	14	1/8	12,5	11,29	0/8	0	0
Biblioteca New York	12	4/8	13,54	16,62	0/8	0	0
Biblioteca UBA	7	7/9	6,34	14,48	0/9	0	0
Biblioteca Australia	7	8/8	0	0	0/8	0	0
Biblioteca España	7	9/9	0	0	0/9	0	0
Biblioteca MIT	13	1/6	29,48	21,43	0/6	0	0
Booking	18	1/9	12,34	7,23	0/9	0	0
Airbnb	6	0/9	24,07	8,78	1/9	1,85	5,55
TripAdvisor	5	4/8	22,5	31,05	2/8	12,5	23,75
IMDb	5	4/4	0	0	0/4	0	0
TheMovieDB	4	9/9	0	0	0/9	0	0
Letterboxd	5	4/4	0	0	0/4	0	0
Metacritic	4	7/9	11,11	22,04	5/9	30,55	32,54
DBLP	9	0/8	11,11	0	0/8	0	0
Google Scholar	10	7/8	1,25	3,53	8/8	10	0
Springer	7	0/8	25	6,61	0/8	0	0
Springer Nature Citations	6	7/9	5,55	11,78	0/9	0	0
ACM	9	8/8	0	0	0/8	0	0