



V4Design

Visual and textual content re-purposing FOR(4) architecture, Design and virtual reality games

H2020-779962

D2.2

Domain specific search and social media crawling tools

| | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Dissemination level: | Public |
| Contractual date of delivery: | Month 18, 30/06/2019 |
| Actual date of delivery: | Month 18, 30/06/2019 |
| Workpackage: | WP2 Multimedia Data Crawling for Reuse and Repurpose |
| Task: | T2.1 Web crawling and retrieval of textual and multimedia data |
| Type: | Report |
| Approval Status: | Approved |
| Version: | 1.1 |
| Number of pages: | 54 |
| Filename: | d2.2_v4design_Domain specific search and social media crawling tools_v1.1.pdf |
| Abstract This deliverable will describe the creation of the multimedia and textual datasets based on Web and social media crawling, as well as for the collection of art-related and architecture-related content. | |
| The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information | |

at its sole risk and liability.



co-funded by the European Union

History

| Version | Date | Reason | Revised by |
|---------|------------|------------------------------------------------------------------------------|------------|
| 0.1 | 08/04/2019 | Table of content created & content defined | CERTH |
| 0.2 | 15/05/2019 | Initial Content added | CERTH |
| 0.3 | 24/06/2019 | Content refinement, Abstract, Executive Summary and Conclusions added | CERTH |
| 1.0 | 26/06/2019 | Content added by content providers | DW, EF |
| 1.1 | 28/06/2019 | Address comments after internal review prepare deliverable for submission | CERTH |

a

| Organization | Name | Contact Information |
|--------------|--------------------------|--------------------------------------------------------------------------------------------|
| CERTH | Konstantinos Avgerinakis | koafgeri@iti.gr |
| CERTH | Stefanos Vrochidis | stefanos@iti.gr |
| CERTH | Vivi Ntrigkogia | vividrig@iti.gr |
| DW | Eva Lopez | eva.lopez@dw.com |
| DW | Stepahn Gensch | stephan.gensch@dw.com |
| EF | Hugo Manguinhas | hugo.manguinhas@europeana.eu |
| EF | Nienke van Schaverbeke | nienke.vanschaverbeke@europeana.eu |
| EF | Liam Wyatt | liam.wyatt@europeana.eu |

Executive Summary

The proliferation of information and the profusion of noise demand for intelligent crawling algorithms and implementations. In V4Design we develop a crawler that utilizes filters to collect only useful subsets of interest of the relevant information. This deliverable describes the creation of the multimedia and textual datasets based on Web and social media crawling, as well as for the collection of art-related and architecture related content. For each module an overview of related work and a comparison to other approaches is included as well as a description of the implementation.

Specifically the following modules are presented: a) the web crawling module, b) the content scraping module, c) the query expansion module, d) the web & social media search module. In addition, the current deliverable documents the SIMMO data model and its adaptations as well as the datasets created using the V4Design Crawler application. The collected resources address the data collection needs defined from the start of the project until this reporting period (M1-M18).

The rest of this report is organized as follows. After a short introduction, we start in Section 2 with a summary of the requirements from the crawler as were defined in deliverable D7.1. Initial use case scenarios and user requirements” of WP7 followed by a description of the framework in Section 3. Then, in Section 4 we describe the focused crawler for web data and in Section 5, we present the entire web and social media searching tools that were developed for various platforms that contain heterogeneous content and provide different ways to access their data. Web searching tools also describe the APIs that the Deutsche Welle and Europeana content providers distribute for accessing their data, along with a dataset Europeana created early in the project using its API. Section 6 describes the query expansion methodologies that we leverage to optimize the results and improve retrieval performance. Section 7 analyses the SIMMO data model and adaptations to it. Then in Section 8 we give an overview of the datasets created from web resources (through crawling), namely Wikipedia, Deutsche Welle "Nico's Weg" language course exercises, Twitter and Flickr. In Section 9 we show the demonstrator and we finish in Section 10 with a summary and conclusions.

Abbreviations and Acronyms

| | |
|--------------|---------------------------------------------------------------------|
| ASR | Automatic Speech Recognition |
| API | Application Programming Interface |
| CHO | Cultural Heritage Object |
| EDM | Europeana Data Model |
| EXIF | Exchangeable Image File |
| FTP | File Transfer Protocol |
| HTML | HyperText Markup Language |
| JSON | JavaScript Object Notation |
| NLTK | Natural Language Toolkit |
| OCR | Optical Character Recognition |
| PDF | Portable Document Format |
| PUC | Pilot Use Case |
| SIMMO | Socially interconnected/interlinked and multimedia-enriched objects |
| STBOL | Spatio-Temporal Building and Object Localization |
| UI | User Interface |
| URL | Uniform Resource Locator |
| VR | Virtual Reality |
| XML | eXtensible Markup Language |

Table of Contents

| | | |
|------------|--------------------------------------------|-----------|
| 1 | INTRODUCTION..... | 8 |
| 2 | RELATION TO USER REQUIREMENTS..... | 10 |
| 3 | FRAMEWORK | 13 |
| 4 | WEB CRAWLING & SCRAPING..... | 15 |
| 4.1 | Crawling | 15 |
| 4.1.1 | Related work | 16 |
| 4.1.2 | Approach and implementation | 17 |
| 4.2 | Scraping | 18 |
| 4.2.1 | Related work | 18 |
| 4.2.2 | Approach and implementation | 20 |
| 5 | WEB & SOCIAL MEDIA SEARCH | 21 |
| 5.1 | Flickr | 21 |
| 5.2 | Search Engines | 22 |
| 5.2.1 | Google | 23 |
| 5.2.2 | DuckDuckGo | 23 |
| 5.3 | Data from content providers | 24 |
| 5.3.1 | DW API | 24 |
| 5.3.2 | EF API..... | 30 |
| 5.4 | Twitter | 35 |
| 6 | QUERY EXPANSION | 36 |
| 6.1 | Neural Word Embeddings..... | 37 |
| 6.2 | Implementation | 38 |
| 7 | DATA MODEL | 41 |
| 7.1 | The SIMMO data model..... | 41 |
| 7.2 | Adaptations..... | 43 |
| 8 | DATASETS CREATED | 46 |
| 8.1 | Wikipedia dataset | 46 |

| | | |
|-----|--------------------------------------------------|----|
| 8.2 | Deutsche Welle Nico's Weg exercises dataset..... | 46 |
| 8.3 | Twitter dataset..... | 47 |
| 8.4 | Flickr dataset..... | 48 |
| 9 | DEMONSTRATOR | 50 |
| 10 | CONCLUSIONS AND NEXT STEPS..... | 52 |
| | REFERENCES | 53 |

1 INTRODUCTION

The prodigious amount of data, structured and unstructured, that individuals, businesses and governments continue to generate at an unprecedented rate and the usability problems that result from attempting to store and manage that data are rendering analysis and indexing of data coming from social networking sites and websites a very challenging task. The proliferation of information and the profusion of noise demand for intelligent crawling algorithms and implementations. With ever expanding data divided in different formats, multiple codes and languages and various categories, interconnected in no particular order, an intelligent web crawler tailored to project and user requirements is necessary to tackle the afore mentioned challenges.

To address this issue, in V4Design we develop a crawler that utilizes filters to collect only subsets of interest of the relevant information. Such filters are related to specific multimedia items i.e. images of buildings or artworks or topics identified in the context of WP4. Figure 1 depicts the V4Design architecture. This document pertains to the red part the data collection of visual and technical data for reuse.

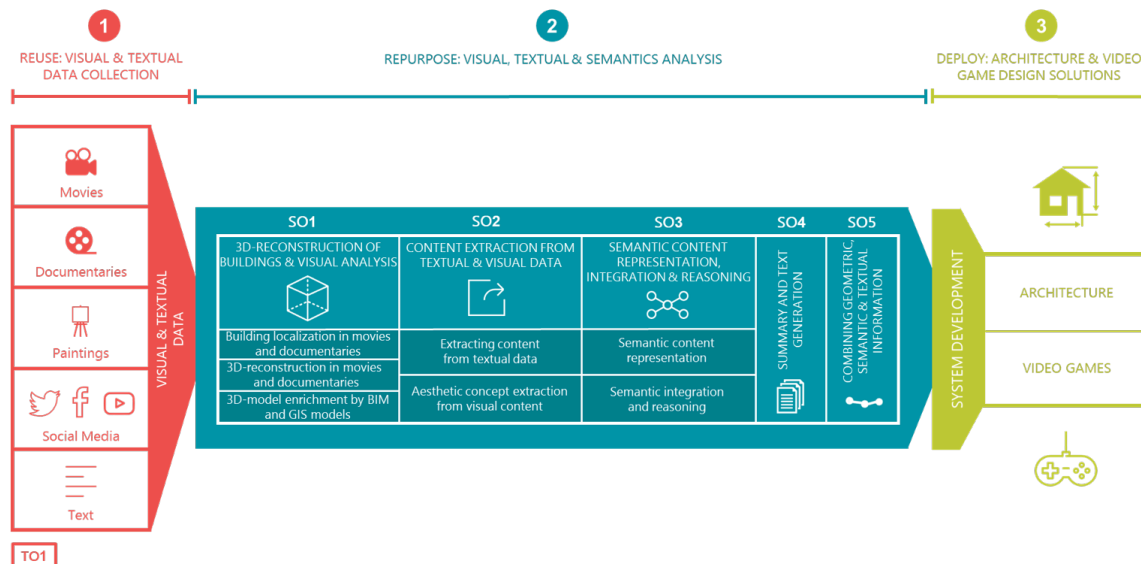


Figure 1: WP2 in V4Design architecture

Three sources of data are crawled to collect, reuse and repurpose content in V4Design: status updates from social media and web pages, movie and documentary data collections from the content providers of the project and artwork data collection from the Europeana Foundation repository. In Section 3, we demonstrate an architecture that crawls both types of sources and puts all data into a unified store (the SIMMO DB) for further processing.

The V4Design Web and Social Media Crawling tool extracts freely available textual and visual artwork content from open web resources. The tool is based on existing open source crawlers and scrapers to collect content based on the automatic formulation of appropriate queries. More specifically, the tool is based on open source Web crawlers (e.g., Apache Nutch, scrapy, crawler4j), search engines and social media querying APIs/libraries (e.g. Bing search API, twitter4j, hbc). In addition, the tool integrates well-known open source scrapers (e.g. Jsoup and Boilerpipe) in order to extract meaningful text elements out of the discovered Web pages.

For webpages, scraping is performed to get only the meaningful multimedia content out of the abundance of information that can be found online. It is also important to underline that in the system that we have developed in the context of the project, we may select to directly extract the content of a specified web resource, without having to perform any kind of crawling or searching.

As far as the movie and documentary data are concerned, services and wrappers are developed upon the provided data from all content providers, apart from Deutsche Welle (DW). All content providers deliver an API for accessing their data. In the case of DW, its API is searched similarly to the other web domains. Last, the Europeana API is used for collecting and accessing artwork from their repository.

The collected content is utilized by the rest of the technical components of the project. More specifically, the textual content is leveraged for text analysis tasks (WP3,WP5), while the multimedia content is used for visual analysis tasks such as the aesthetics extraction (WP3), the spatio-temporal building and object localization (WP4) and the 3D reconstruction (WP4). The Web and Social Media Crawling tool is the starting point of a preprocessing pipeline that populates a knowledge base (WP5) and delivers content to a VR game authoring tool and an application intended for architects and designers (WP6).

2 RELATION TO USER REQUIREMENTS

We list here the requirements for the crawler as defined in "D6.2. Technical requirements pertaining to the crawling and collection of content and accompanying metadata from publicly available online sources and social media platforms". Each requirement focuses on a specific type of information/content to be collected.

| # | SHORT NAME | CODE | Related URs. |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|---------|-------------------------------------------------|
| 1 | Web Crawling | TR_CR_1 | UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56 |
| Description: Using a set of URLs as web entry points, collect all the hyperlinked URLs, up to a predefined depth. Discovers nodes to scrape. | | | |
| Comments: Web content should be collected from trustworthy sources provided by user partners of V4Design. For instance, we collected content from "Nicos Weg" German courses in collaboration with DW. For the architecture-related user cases, we expect similar sources but with a stronger focus on buildings, landscapes and historic spatial elements. Links to such content were provided by user partners. | | | |

| # | SHORT NAME | CODE | Related URs. |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|---------|-------------------------------------------------|
| 2 | Query Expansion | TR_CR_2 | UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56 |
| Description: Discovery of extra keywords relevant to the input query. Add more keywords to refine the search operations. | | | |
| Comments: Users usually provide vague input queries. Our objective with query expansion is to match a larger set of relevant results. We broaden the query by introducing additional tokens or phrases. For example, the query " <i>Eiffel tower</i> " becomes " <i>Eiffel tower in Paris</i> " and therefore brings more relevant images of the Eiffel tower and other buildings in Paris that might be of interest to architects. | | | |

| # | SHORT NAME | CODE | Related URs. |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------|-------------------------------------------------|
| 3 | Web Search | TR_CR_3 | UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56 |
| Description: With the help of API, search a web application (e.g. Flickr) using textual queries. Depending on the available APIs, scraping may also be performed. | | | |
| Comments: We collect the multimedia items (images, videos) pointed to by the shared links. We leverage Flickr for targeted content to be used in 3D reconstruction. | | | |

| # | Related URs. | SHORT NAME | CODE | Related URs. |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|------------|---------|-------------------------------------------------|
| 4 | Web Scraping | | TR_CR_4 | UR_10, UR_11, UR_16, UR_21, UR_22, UR_55, UR_56 |
| Description: Extracts content from web pages | | | | |
| Comments: Web scraping, web crawling, and any other form of web data extraction can be complicated. Between obtaining the correct page source, to parsing the source correctly, rendering JavaScript, and obtaining data in a usable form, there's a lot of work to be done. Different users have very different needs. Our technique allows users to scrape behind login forms, fill in forms, input search terms, navigation bars and advertisements and only keep useful content. | | | | |

| # | SHORT NAME | CODE | Related URs. |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|---------|-----------------------------------|
| 5 | Social media crawling & scraping | TR_CR_5 | UR_10, UR_16, UR_21, UR_24, UR_55 |
| Description: Search and collect social media posts relevant to a keyword or a user account. | | | |
| Comments: Not all social networks are equally important for the use cases. For V4Design we have placed particular emphasis on Twitter. Twitter is known to be the one that provides good and filtered content when relevant Twitter tags are followed. It also tends to point (through tiny URLs being mentioned in the twitter message) to the more relevant items in YouTube. | | | |

| # | SHORT NAME | CODE | Related URs. |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|---------|-----------------------------------|
| 6 | FTP Crawling | TR_CR_6 | UR_10, UR_16, UR_21, UR_24, UR_55 |
| Description: Looks at the V4Design FTP server folders of a content provider of the V4Design project to see if any new content has been added, and if so extracts it to add to data storage | | | |
| Comments: No further comments. | | | |

| # | SHORT NAME | CODE | Related URs. |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|---------|-----------------------------------|
| 7 | Data model mapping | TR_CR_7 | UR_10, UR_16, UR_21, UR_24, UR_55 |
| Description: The function maps incoming data from the incoming data model to SIMMO JSON. Based on an EDM file or a generic JSON file, check if this JSON is SIMMO compliant. If not, use predefined maps to make this JSON file SIMMO compliant. Send to data storage. | | | |
| Comments: Early in the project we agreed that we need a unified data storage model in order to there is a unique representation of data so that other WPs shall not have to build | | | |

different codes for each content provider.

| # | SHORT NAME | CODE | Related URs. |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|---------|-----------------------------------|
| 8 | Resource filtering | TR_CR_8 | UR_10, UR_16, UR_21, UR_24, UR_55 |
| Description: Application of classifiers that categorize the resources as appropriate or not for our purposes. | | | |
| Comments: For example, in the framework of the object localization task, the STBOL module, will filter content that does not pertain to buildings or interior objects as there are prerequisites to perform 3D modeling. | | | |

3 FRAMEWORK

The goal of the crawling, scraping & search system (for convenience reasons we are going to name it as V4Design Crawler from now on) we developed for the project is to collect useful multimedia content that can be then processed by the WP3 and WP4 modules. It is the first step of a pre-processing pipeline that generates content to be utilized by the VR authoring tool and the Rhino plugin, the two tools that are going to be developed in the framework of WP6 and will exploit the generated content. The exact modules that are executed in this framework are depicted in Figure 2.

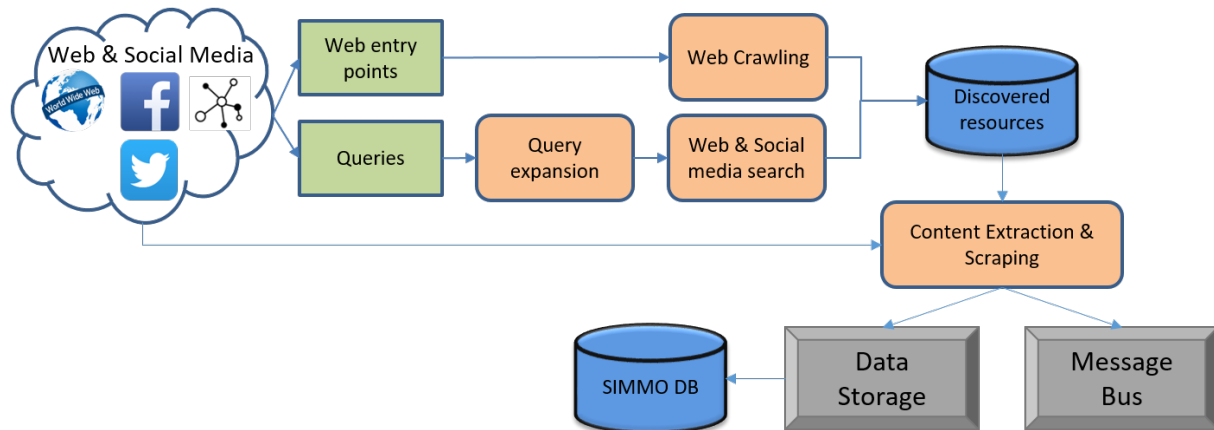


Figure 2: The V4Design Crawler

Based on the content existing on the web and the social media, we first define web entry points and search queries. The web entry points are URL addresses of web domains for which we are interested in collecting their content. Each domain is crawled (TR_CR_1) and as a result we get a set of webpages that belong to this domain. The search queries consist of textual terms that are fed into search engines and APIs (web or social media) with the aim of returning the most relevant results (TR_CR_3, TR_CR_5). Optionally, to optimize their performance we expand them by enriching it with additional terms using the query expansion module (TR_CR_2). The results in the searching scenario may differ depending on the website we are performing search. They may comprise webpages, images, videos or social media posts.

In both crawling and searching scenarios, the content of the discovered resources is extracted in an automatic fashion. For webpages, scraping (TR_CR_4) is employed to get only the meaningful multimedia content out of it. Note that in this system, we may select to directly extract the content of a specified web resource, without having to perform any kind of crawling or searching. For example, if we only want to scrape the Delphi Wikipedia webpage, while we are not interested in any hyperlinked webpage, we feed this webpage directly to the scraping module, bypassing the crawling process. The extracted content is sent to the Data Storage system (described in “D6.3. Operational prototypes and user interfaces for architecture and VR game design application”), which in turn forwards it in a database. The items saved to the database comply with an extended version of the SIMMO data model. Additionally, in order to have the system integrated with the project’s platform, each time a resource is saved to the Data Storage, a message is sent to the message bus (described in “D6.3. Operational prototypes and user interfaces for architecture and VR

game design application”) to notify the directly dependent components that there is something new to process.

Any system of the project that has to process data from the V4Design Crawler must first subscribe to the relevant message bus topic and listen to it all the time. Then, the messages that are published contain all the information needed to retrieve the collected resources from the Data Storage system.

The V4Design Crawler development involves the following tasks:

- i. *T2.1: Web crawling and retrieval of textual and multimedia data.* This task is the one having the biggest involvement in the implementation of the system as it contributes to all the modules that constitute the V4Design Crawler. The web and the social media are the main sources, from which we are going to collect, reuse and repurpose content.
- ii. *T2.2: Movie and documentary data collection.* In the framework of this task, services and wrappers are developed upon the provided data. This applies to all the content providers, apart from Deutsche Welle (DW) that provide an API for accessing their data. In the case of DW, its API is searched similarly to the other web domains that provide an API.
- iii. *T2.3: Artwork data collection and retrieval.* In this task, artwork is collected from the Europeana Foundation repository. Access of their data is done by searching the existing Europeana API.

We can notice that the development of a module of the V4Design Crawler can extend to more than one task. For example, the web search module concerns three different tasks i.e. T2.1, T2.2 and T2.3, as we are interested in searching content providers’ data as well.

This deliverable describes the initial implementation of the aforementioned framework. The final (and extended) one will be detailed in “D2.4. Final web crawling techniques and annotated corpus”.

4 WEB CRAWLING & SCRAPING

The web contains a huge amount of multimedia data that includes buildings, interior objects and artwork that can be re-used and re-purposed in the framework of the project. These data can be found in several types of web resources including standard web pages, pdf files or videos posted on the YouTube platform. The aforementioned objects are of high interest to architects and game designers, as in a typical 3D modeling software usage scenario they are either created from scratch or they are used as a source of inspiration to manually create similar models. At the same time, online discussion mediums, such as forums, provide a large pool of reviews that can be leveraged by the text analysis tasks in order to generate content accompanying the re-used objects.

To the end of automating processes, such as the 3D model creation, that are otherwise done manually and consume a significant amount of the users' time, this wealth of publicly available multimedia data coming from web resources and social media can support the V4Design use cases and feed repositories with relevant content that can be later used by the other components of the preprocessing pipeline. This pipeline includes the visual and textual analysis tasks and the Knowledge Base integration (WP3, WP4, WP5). In order to accomplish exploitation of the available online content, appropriate crawling and scraping tools need to be developed. These run over a list of web domains that have been defined by the user partners as appropriate for generating 3D models and acquire metadata useful to have in their tools for game and architecture design.

In the following sections, the V4Design crawling and scraping components are further described.

4.1 Crawling

In V4Design, there is a need for tracking web data related to the uses cases so as to feed such data in the V4Design Data Storage module. Considering the relevant web resources, we distinguish two main types of data sources with respect to the data discovery functionality: i) data coming from search engines, social media, and other websites that provide an API for accessing their data and ii) unstructured, heterogeneous data coming from web pages, pdf files and forums.

For the first category, searching modules have to be developed that will be described in Section 5. For the unstructured web-based sources, a crawler component has been developed that uses as input seed list the URLs matching the web domains of interest. Specifically, the starting nodes are the URLs of the web resources resulted from the empirical study that has been conducted by the user partners, while the set of neighbouring nodes are restricted only on the web pages deriving from the same domain, in order to perform exhaustive and deep crawling of the specific web domains.

As the crawling is restricted on a specific number of web domains, the chance that those domains can generate a large amount of data is moderated and therefore, a conventional design infrastructure can handle the performance demands.

4.1.1 Related work

Web crawlers, also known as "spiders", "bots" or "wanderers", are software programs that automatically go through the Web in a methodical, automated manner for discovering web resources.

Web crawlers view the Web as a directed graph, where the nodes represent unique web pages (based on their URL) and each directed edge represents a unique hyperlink between two pages. Crawlers are capable of traversing the Web graph in several modes, such as in breadth-first or depth-first manner given the order of hyperlinks within the web pages, or in best-first fashion based on some ranking, e.g., the number of their incoming links (Olston & Najork, 2010).

Web crawlers start with a predefined list of seed URLs, fetch and parse each of them, extract the hyperlinks that these pages contain, place the extracted hyperlinks on a queue and, systematically, fetch and parse URLs associated with these hyperlinks from the queue, and so on. This process is iteratively repeated to an arbitrary depth, depending on the targeted objective that can be until a sufficient number of pages are fetched or a limit on the crawling depth is reached (i.e. the maximum distance allowed between the current and seed pages). During their activity, they store information about the visited pages, such as the web page, the hyperlinks, the content, the metadata, etc. depending on the approach.

The behaviour of a Web crawler is bound by a combination of several policies:

- the *selection* policy that states which pages to download;
- the *re-visit* policy that determines how often a page should be checked for changes in its content;
- the *duplication* policy that examines the Web page content similarity issues;
- the *politeness* policy that provides the rules for avoiding overloading a website or webserver and
- the *parallelisation* policy that specifies how to coordinate distributed Web crawlers.

Additionally, a web crawler has to respect the instructions provided by the web domain owner in the robots.txt file. Robots.txt is a file placed in the root of the website hierarchy (e.g. <https://www.example.com/robots.txt>) hosting instructions for the crawler e.g., which areas of the website should not be processed or scanned.

Today, there exist several well-established web crawlers. Among them, we distinguish Heritrix¹, Apache Nutch² and crawler4j³ as the most widely-used, open-source software programs. Heritrix is the Internet Archive's open-source, extensible, web scale project that is supported by a big community of users. Its well-documented code and the easy to use UI render it more like a light solution for a generic use. Apache Nutch is a more robust, big scale and Google-bot comparable crawler that is highly extensible and open source. Crawler4j is an open source web crawler written in Java which provides a straightforward interface for crawling the Web. By using it, one can quickly setup a web crawler with multiple threads.

¹ <https://webarchive.jira.com/wiki/display/Heritrix/Heritrix>

² <http://nutch.apache.org/>

³ <https://github.com/yasserg/crawler4j>

As an alternative option for crawling, one can choose to use web applications such as Octoparse⁴ and import.io⁵. They have the advantage that they do not require any programming skills as their functionalities are supported via a web user interface. However, so as not to depend on third-party web-based services, we opted to use an open-source programming library.

4.1.2 Approach and implementation

The V4Design crawling component is responsible for periodically running and extracting the hyperlinks identified in the fetched web pages, pdf files and forums. The unique URLs associated with these hyperlinks are then stored in a database. Once the crawling process has been completed, the discovered URLs are forwarded to the scraping component for extracting the content that these pages contain. Figure 3 provides an abstract view of the crawling & scraping architecture, as a subset of the general architecture presented in the entire system framework (Section 3) isolating the functionalities presented in this section.

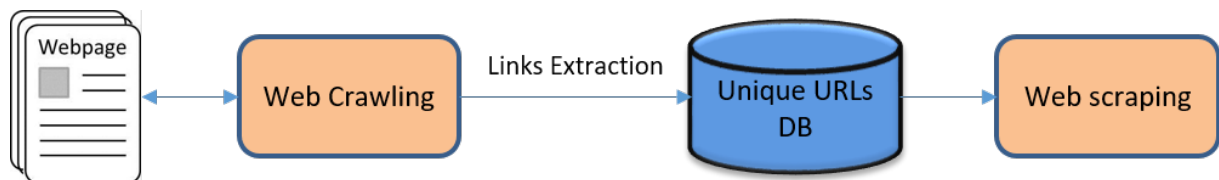


Figure 3: Crawling and scraping modules

Initially, the list of related web domains defined by the user partners is used as seed URLs. The crawling process starts from the home page of each web domain. The crawling component obtains the home page and saves it. Then, it parses the document, extracts all hyperlinks enclosed to the home page, and then extracts the URLs associated with these hyperlinks. Next, the crawler filters these URLs in a way that only URLs pointing to a webpage in the same domain or PDF files pass the filter. The filtered URLs are stored in a temporary repository. The same process is repeated for every unvisited URL in the temporary repository. Assuming that the starting webpage is the parent and the unvisited linked webpages are the children, the temporary repository can be represented as a tree. The crawler is using the *breadth-first* algorithm to traverse this tree. This iteration is over when there is no unvisited URL in the temporary repository or when the number of iterations has reached the predefined crawling depth. The depth is variable depending on the structure of the crawled website. Last, the unique URLs from the temporary repository are stored in a database. The aforementioned process is executed whenever there is a need to crawl new web domains and the database is updated with new URLs (incremental crawling), while URLs already stored in the database are ignored.

Considering the crawling of forums, we have observed that most of them are open access and thus, the crawling component does not require additional configurations to parse their content. Therefore, the crawler already developed for webpages has been also used for forums assuming that a forum is a webpage that hosts forum threads and discussions. Links to PDF files are stored in the database as well.

⁴ <https://www.octoparse.com/>

⁵ <https://www.import.io/>

The resulting database contains a list of URLs of the discovered resources. It has to be noted that at this stage only address links are stored. Extracting and saving the actual content of the discovered webpages is out of the scope of this module as this procedure is executed by the scraper. This comprises the main difference between web crawling and web scraping. Crawling is conducted to discover a list of links, while the responsibility of the scraper is to extract meaningful content out of these links.

For this project, we used Crawler4J as it is able to support our initial crawling needs and due to the fact that it provides capabilities of setting up customized crawlers in JAVA programming language, the main one used in the V4Design Crawler application. Based on that, we configured the default Crawler4J implementation in order to crawl the user defined domains and save the collected URLs in a MongoDB database along with a timestamp that is utilized to identify a unique crawling operation. It does not send the output to the centralized Data Storage component as it constitutes an intermediate and incomplete form that is not meaningful without being processed by the Scraping module. It runs on a Windows 10 64-bit machine, and from the entire list of the URLs, it keeps only the ones that are associated with a webpage or a PDF file.

4.2 Scraping

Scraping as a term refers to the process of automatically extracting information out of any human-readable output. Examples of such output are example PDF files and images. In this project we are mainly interested in web scraping, which is a specialized instance of scraping that is performed only on websites and its goal is to filter out redundant features such as navigation bars and advertisements. Web scraping is also related to indexing, focusing on the transformation of unstructured data on the web, typically in HTML format, into structured data that can be stored and analysed in a central local database. Usually, the scraping techniques depend on the type of the website (i.e., static or dynamic). Additionally, the scraping techniques may depend on the way in which the information is structured or presented and scrapers have to be configured based on that structure. For example, a web scraper extracting content from a forum needs to be configured differently than the scraper extracting content from articles. Also, the same scraper may need different configuration when extracting 2 different domains from the same type of website (e.g., 2 different forums).

In V4Design, there is a need for extracting and aggregating the content from several heterogeneous sources in a central repository using a common format. Once the collected data is stored, it is then consumed by the rest pipeline components, such as the Text Analysis and the Aesthetics Extraction ones.

4.2.1 Related work

In scraping frameworks the selection of elements on a web page is defined as selection function while the acceptance or rejection of a selection result from a web page is defined as validation function. The selection function is responsible for choosing a piece of information (i.e., an HTML element, a list of HTML elements, tuples of text, etc.) from a given web page and delivering the selection results in a suitable format. XPath expressions were specifically invented in order to designate elements in an XML tree structure. Also, regular expressions

and context free parsers have been widely used to locate specific elements by their content and structure.

A validation function utilizes textual dimensions of both the selection result along with the original web page. However, when the element to be selected dynamically changes, other dimensions might be more effective, such as the context (e.g., the tree structure from the grand parent of the selected element) or the appearance of a website.

Today, several scrapers exist supporting a wide range of features and functions. Example open-source libraries that are capable of traversing and manipulating HTML documents are JSoup⁶ and HTMLUnit⁷.

Import.io⁸ is a web-based scraping tool. By following an easy step-by-step plan and without writing any code, someone can select the data to scrape and the tool does the rest. Also, it supports scraping of multiple URLs at once.

DEiXTo⁹ (or ΔEiXTo) is a web data extraction tool that is based on the W3C Document Object Model (DOM). It allows users to create highly accurate “extraction rules” (wrappers) that describe what pieces of data to scrape from a website.

The main idea of Kimono Labs¹⁰ service is to create APIs for websites which don’t have one; another term would be web scraping. It enables not only to generate raw data (in JSON, CSV or RSS format) from a web page, but even instantly create a web application to make those data available on the web.

ScraperWiki¹¹ is a web-based platform for collaboratively building programs to extract and analyse online data, in a wiki-like fashion. This tool is appropriate for massive-scale applications and compared to other tools, it is the most advanced one.

easy Information Extraction¹² (easIE) is a framework for generating Web information extractors and wrappers. easIE offers a set of wrappers for obtaining content from static and dynamic HTML pages by pointing to the html elements using css selectors. An additional functionality is the definition of a configuration file that allows automatically extracting content of a page.

The boilerpipe library¹³ provides algorithms to detect and remove the surplus "clutter" (boilerplate, templates) around the main textual content of a web page. It provides specific strategies for common tasks (e.g., news article extraction) and may also be easily extended for individual problem settings.

⁶ <https://jsoup.org/>

⁷ <http://htmlunit.sourceforge.net/>

⁸ <https://import.io/advanced-data-platform>

⁹ <http://deixto.com/>

¹⁰ <https://www.kimonolabs.com/>

¹¹ <https://scraperwiki.com/>

¹² <https://github.com/MKLab-ITI/easIE>

¹³ <https://code.google.com/archive/p/boilerpipe/>

4.2.2 Approach and implementation

The V4Design scraping module takes two different types of input:

- The URL database that results from the crawling module
- Standalone webpages that need to be directly scraped (without executing any kind of crawling on them)

The extracted information is converted into SIMMO format (described in Section 7) and sent to the Data Storage module that saves it into a MongoDB database.

We have implemented a content scraping component that extracts their content according to the type of web resource. In every update of the unique URLs and whenever new standalone webpages to be scraped are introduced, the component incrementally extracts new content that has not been listed in the respective DB until then. This module not only executes web scraping, but also extracts content from PDF files as both webpages and PDF files are considered as useful for our purposes.

For webpage content extraction, we made use of the Jsoup JAVA library. For the initial version of the module, we target the exact elements of the HTML structure that contain the information we are interested in for each website that is scraped. Their definition is made manually by constructing CSS rules based on element classes and ids. These rules can be directly applied with the help of the used JAVA library to extract the final content to be stored in the Data Storage module. For each webpage, we save their text in both raw and HTML format as well as the multimedia content they contain. This approach produces acceptable results as long as we have a limited website collection. Nevertheless, in order to develop a scraping module that is scalable to the introduction of new and significantly different websites in terms of their HTML structure, in the final version we are planning to set up a different approach that is more flexible and that can be applied to any website.

For PDF text extraction, we have used Apache PDFBox. A new PDF object is added to the initial SIMMO model at the same level of Webpage and Post objects and instantiates the Document object. The purpose of this modification was to enable storing a PDF object that contains a Text object.

5 WEB & SOCIAL MEDIA SEARCH

In the previous section, the aim is to collect information based on the assumption that any webpage that belongs to a specified website includes data that can be utilized for the project's purposes. However, it can be easily observed that not all websites fall into this category. There exist websites where only a small percentage of their webpages, which do not necessarily fall into a unique subdomain, can be exploited. Furthermore, there are cases where we are not able to define specific websites but we are interested in collecting content based on some search criteria. In different scenarios, we wish to collect information from social media instead of websites. Social media platforms enable anyone to circulate relevant to our project posts along with multimedia. In this section, we present all the web and social media searching tools that were developed for various platforms that contain heterogeneous content and provide different ways to access their data (the most common one being by means of a search API).

5.1 Flickr

Flickr is a website created in 2004 that enables users to upload multimedia content, either photos or videos. Any user can access its contents for free in a user-friendly interface (see Figure 4), but the upload feature is activated only for registered accounts.

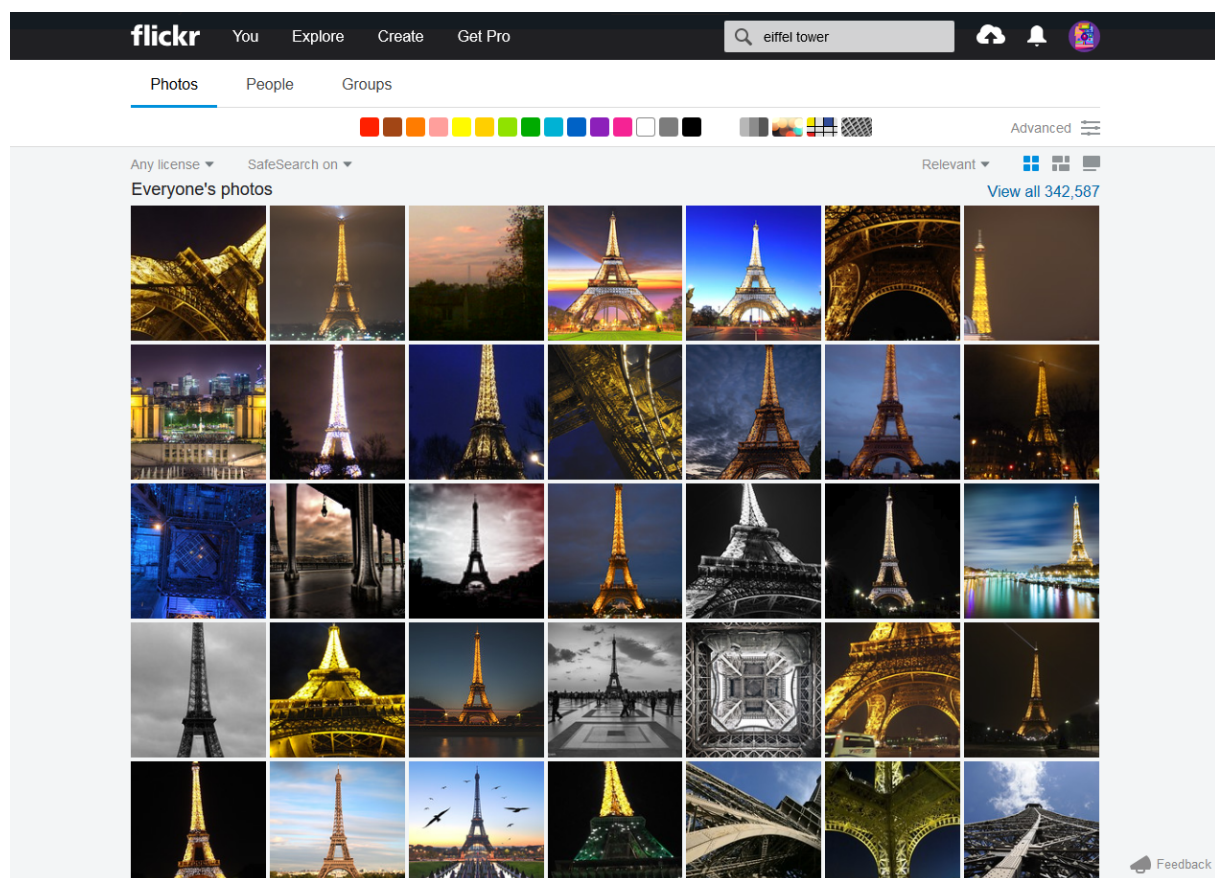


Figure 4: The Flickr website

It became so popular that now it is visited by over 90 million users every month¹⁴ and millions of new images are uploaded on a daily basis. In a very-high traffic day, this number reaches up to 25 million photos¹⁵. In total, Flickr hosts more than 10 billion images¹⁶. We can realize from that fact that Flickr is more than a suitable source for gathering multimedia resources of famous buildings, interior objects and so forth.

In terms of accessing Flickr's content programmatically in the same way users accessing by typing queries in the web interface, the website is accompanied with an API that exposes its data. The Flickr API consists of a set of callable methods, and some API endpoints. From this set, we made use of the following methods:

- *flickr.photos.search*: It is the most important of the methods, as it simulates a search function. It returns a photos list that is relevant to the input user query. As much information accompanying the returned results is not provided by this method, other methods of the API have to be called to retrieve more details.
- *flickr.photos.licenses.getInfo*: Returns the identifiers of the licenses existing in the photos. It is used in order to extract only the freely available content. These ids are given as an additional parameter in the "flickr.photos.search" methods.
- *flickr.photos.getExif*: Returns the EXIF tags for a photo with a specified id. It is used to enrich the resource saved in the Data Storage with more metadata.
- *flickr.photos.comments.getList*: Returns the comments for a photo, given its id as input. Useful to get reviews of a specified multimedia resource.

For V4Design, we used a JAVA wrapper of the Flickr API, named flickr4java¹⁷ to search and retrieve multimedia items and save them as SIMMOs in the Data Storage. As all the needed information can be drawn from the aforementioned methods, no scraping has to be performed.

5.2 Search Engines

Another approach for acquiring data is by querying search engines. General-purpose search engines are extensively used to search for web content relevant to an input text query or an image, with the most widely known one being Google. According to eBizMBA's ranking¹⁸ for January 2019 (see Table 1), the top three most visited search engines are Google, Bing, and Yahoo!, with Google having expectedly and indisputably the largest number of visitors.

For the framework of our project, search engines can be utilized in two ways: a) to use the search results as entry points for a crawling process start, b) to simply scrape the resulting webpages. Next, we describe the implementations made for two search engines, Google and DuckDuckGo. DuckDuckGo was utilised because it doesn't store, collect or share personal

¹⁴ <https://www.flickr.com/jobs/>

¹⁵ <https://code.flickr.net/2017/01/05/a-year-without-a-byte/>

¹⁶ <http://blog.flickr.net/en/2015/05/07/flickr-unified-search/>

¹⁷ <https://github.com/boncey/Flickr4Java>

¹⁸ <http://www.ebizmba.com/articles/search-engines>

information. As V4Design takes seriously into consideration GDPR and takes actions related to the protection of personal data and the preservation of privacy we opted to use DuckDuckGo since it is also free of charge.

Table 1: Top 10 search providers ranked by estimated unique monthly visitors (January 2019)

| Provider | Estimated Unique Monthly Visitors |
|---------------|-----------------------------------|
| Google | 1,800,000,000 |
| Bing | 500,000,000 |
| Yahoo! Search | 490,000,000 |
| Baidu | 480,000,000 |
| Ask | 300,000,000 |
| Aol Search | 200,000,000 |
| DuckDuckGo | 150,000,000 |
| WolframAlpha | 35,000,000 |
| Yandex | 30,000,000 |
| WebCrawler | 25,000,000 |

5.2.1 Google

For accessing functionalities of the most used search engine in the World Wide Web, Google provides the Custom Search JSON API¹⁹, a RESTful API that supports requests for searching webpages and images. As its name implies, results are given in JSON format. Its free version supports up to 100 queries daily and for each query a maximum of 10 results can be returned. Even if the supported number of results is small and restrictive, in most cases it is difficult to find relevant content beyond the top 10 results. We can also infer that, if we think of a web search from a user perspective. In a typical search scenario, there are few cases where the required content can be found after the first page of results. In the V4Design implementation, a simple HTTP query is done to the API and the results are saved into the same unique URL database that is also used for the crawling module.

5.2.2 DuckDuckGo

DuckDuckGo is a search engine created by Gabriel Weinberg on 2008. Parts of its code are open-source, however its core still remains closed-source. The noteworthy difference that distinguishes DuckDuckGo from the other popular search engines is that it protects users' privacy by not storing and using any personal data.

DuckDuckGo, unlike other search engines that provide an API, such as Google and Bing, does not have an equivalent way for accessing its search results. Instead, in this webpage there is a non-Javascript HTML version that makes it very straightforward to detect and scrape its search results. Therefore, for the purpose of the V4Design implementation, given the query

¹⁹ <https://developers.google.com/custom-search/v1/overview>

terms, we formulate an HTTP GET request to its HTML version, retrieve its results page HTML code using JSoup (the library we also use for the scraping module) and finally extract the URL addresses from the tree structure of the webpage by pointing to predefined elements that exist in the DuckDuckGo webpage.

5.3 Data from content providers

5.3.1 DW API

Usage in V4Design

The DW API provides numerous types of data for V4Design purposes in an easily accessible and processable manner. Besides providing media files, like images, videos, text, and audio, the API also provides rich metadata for contextualisation. In short, the DW API supports V4Design through provision of

- images and videos that can be used for 3D reconstruction (WP4) and aesthetics extraction (WP3)
- text and audio that can be used for textual analysis and summarisation (WP5)
- query functions for recent, relevant or related items through search terms to populate the knowledge base (WP6)
- articles that add semantic value to media items for defining ontologies and linked data (WP5)
- rich technical and non-technical metadata (WP5/WP6).

Since DW content is very diverse in terms of regionality, topic, as well as language and it already covers a long time range, it is ideally suited to cover the needs of creative designers, architects, game creators and other potential V4Design target groups.

For pilot use case 3, the development of a VR learning experience based on the videonovela "Nicos Weg," the API makes all videos published easily accessible. They can be retrieved using the global search resource of the API.

<http://api.dw.com/api/search/global?terms=Nicos+Weg&languageId=1>

It returns an article and a video result item. Both have additional text information within a `teaserText` property that can be used for semantic analysis and contextualisation. Using this analysis, it can be tagged and made accessible in the V4Design knowledge base in a more convenient way. Images and videos in this use case mostly show people in an urban environment. Thus, it is less suited for 3D reconstruction through photo- or videogrammetry, since it is often missing baseline movement of the camera and has people moving in the foreground. However, the results from object location algorithms (STBOL) can be very helpful in providing game creators with a list of objects that appear in the scenes and potentially offering already pre-fabricated 3D models. Latest approaches of generative adversarial networks may even help produce 3D models from single frame images for known masks that are detected through object location.

For the pilot use case 4, numerous drone videos of Bauhaus architecture can be used within V4Design that are offered through the DW API. The approach for 3D reconstruction by WP4 is twofold. For once, since the DW DailyDrone videos use graphical overlays for additional information, they are less suitable for 3D reconstruction. So, additional raw footage is being

provided, but needs to be put into context. This is where the DW API can be really helpful, since it provides a very convenient way to retrieve all necessary metadata from the article pages and teaser sections of a video. This can then be used to complete the V4Design knowledge base entries.

The rest of this subsection describes access to and use of the key functionalities of the DW REST **API Version 1.0.4**.

Resource description

The DW REST API is primarily used for the DW mobile app: DW – Breaking World News. It is not a publicly advocated API, but no access rules are enforced as of this writing. The API closely mirrors content available on the DW website. The DW API provides access to articles, video, audio and image galleries. Results are returned in JSON. The base API URL is <https://api.dw.com>. It features several resources that group various endpoints to facilitate structured queries to the DW API. Some resources may have nested resources. They are described within their parent resource.

The starting point for all reference is the configuration resource available at <https://api.dw.com/api/config>.

API Key & Tokens

Currently, no API key or oauth access token are needed to make requests to the DW API. However, the API is not generally public and this documentation is for partners within projects, only.

Search

The Search API allows you to perform the same search that is available via the advanced search on the DW website. This means that you can use modifiers such as `startDate:Date`, `endDate:Date`, or `contentTypes:contentType` within the query to modify the results you get back.

Global search pattern

This global search is looking for keywords in articles and media metadata and returns all objects, where the search term can be found.

`/api/search/global?`

Relevance search pattern

The relevance search returns all results that have a semantic relevance to the search term.

`/api/search/relevant?`

Autocomplete search pattern

The autocomplete search returns a list of suggested terms that can have a defined maximum of suggestions returned.

`/api/search/autocomplete?`

Related search pattern

The related search patterns returns results that have a semantic relation to the search term.

`/api/search/related?`

Resource details

Configuration Feed

The configuration feed is the main entry point and schema document for the DW API. It can be called at <https://api.dw.com/api/config/init>. The returned JSON document is structured as follows ({...} mark omissions for brevity):

```
{
  "apiVersion" : "1.0.4",
  "supportedLanguages" : [ {
    ...
  }, {
    {
      "id" : 2,
      "languageCode" : "en",
      "regionCode" : "GB",
      "rtl" : false,
      "displayNameEnglish" : "English",
      "displayNameLocalized" : "English",
      "defaultChannel" : 1,
      "dataPrivacyPolicyUrl" : "https://api.dw.com/api/detail/article/18265246"
    }, {
      ...
    }
  ],
  "trackingConfig" : { ... }
  "epgConfig" : { ... }
  "urlConfig" : {
    "baseApiUrl" : "https://api.dw.com",
    "globalSearchUrlPattern" : "/api/search/global?...",
    "relevanceSearchUrlPattern" : "/api/search/relevant?...",
    "autoCompleteUrlPattern" : "/api/search/autocomplete?...",
    "mainNavigationUrlPattern" : "/api/navigation/{locale}",
    "indexingDataUrlPattern" : "/api/indexing/{languageId}",
    "mostRecentArticlesLanguageUrlPattern" :
"/api/list/article/recent/{languageId}?...",
    "mostRecentVideosLanguageUrlPattern" :
"/api/list/video/recent/{languageId}?...",
    "mostRecentVideosProgramUrlPattern" :
"/api/list/video/recent/{languageId}/program/{programId}?...",
    "mostRecentVideosSeriesUrlPattern" :
"/api/list/videos/recent/{languageId}/series/{seriesId}?...",
    "mostRecentVideosThematicFocusUrlPattern" :
"/api/list/video/recent/{languageId}/thematicfocus/{thematicFocusId}?...",
    "mostWatchedVideosLanguageUrlPattern" :
"/api/list/video/mostwatched/{languageId}?pageIndex={pageIndex}",
    "offlineData" :
"/api/offline/{languageId}?fromStructurepage={fromStructurepage}",
    "programListUrlPattern" : "/api/epg/list/program/{languageId}",
    "programGroupsUrlPattern" :
"/api/epg/programgroups/topics/{languageId}",
    "thematicFocusListUrlPattern" : "/api/epg/thematicfocus/{languageId}",
    "epgUrlPattern" :
"/api/epg/{channelId}?languageId={languageId}&days={days}&hours={hours}",
    "relatedSearchUrlPattern" : "/api/search/related?...",
    "eds" : "https://buwa.dw.com/eds/germanElection/2017",
    "dvapp" : "https://commons.dw.com/buwa"
  },
}
```

```
"appUpdate" : {...}
}
```

Global search pattern

```
/api/search/global?\\
terms={terms}\\
&languageId={languageId}\\
&contentTypes={contentTypes}\\
&startDate={startDate}\\
&endDate={endDate}\\
&sortByDate={sortByDate}\\
&pageIndex={pageIndex}\\
&asTeaser={asTeaser}\\
&programs={programs}\\
&themes={themes}\\
&ids={ids}
```

Global search example

<https://api.dw.com/api/search/global?terms=delphi&languageId=2>

Global search response

See link above.

Relevance search pattern

```
/api/search/relevant?\\
terms={terms}\\
&junctionMode={junctionMode}\\
&languageId={languageId}\\
&contentTypes={contentTypes}\\
&startDate={startDate}\\
&endDate={endDate}\\
&sortByDate={sortByDate}\\
&pageIndex={pageIndex}\\
&asTeaser={asTeaser}\\
&programs={programs}\\
&themes={themes}\\
&ids={ids}
```

Relevance search example

<https://api.dw.com/api/search/relevant?terms=Bali+Agung&languageId=2>

Relevance search response

See link above.

Autocomplete search pattern

```
/api/search/autocomplete?\\
prefix={prefix}\\
&languageId={languageId}\\
&maxHits={maxHits}
```

Autocomplete search query example

<http://api.dw.com/api/search/autocomplete?prefix=Fede&languageId=2&maxHits=10>

Autocomplete search response example

```
{
  "prefix" : "Fede",
  "resultCount" : 10,
  "results" : [
    "federal reserve",
    "federica mogherini",
    "federal",
    "federal court of justice",
    "federer",
    "federal statistics office",
    "federal constitutional court",
    "federal election",
    "federation",
    "federal police"
  ]
}
```

Related search pattern

The related search patterns returns results that have a semantic relation to the search term.

/api/search/related?

Related search example

<https://api.dw.com/api/search/relevant?terms=Gaugin&languageId=2>

Related search response

See link above.

Search query Parameters

| Parameter | Description | Example | Mandatory or Optional |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|-----------------------|
| Terms | A comma-separated list of strings | terms=Bali%2CAgung | Mandatory |
| languageId | An integer value that represents a language Id. See → https://api.dw.com/api/config/init → supportedLanguages | languageId=2 (English) | Mandatory |

| | | | |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|----------|
| contentTypes | <p>A comma-separated list of content types that are being searched for and returned from a set of</p> <ul style="list-style-type: none"> Article Audio Video ImageGallery | contentTypes=Article%2CAudio | Optional |
| startDate | <p>A date string that marks the start of the search period in the format of YYYY-MM-DD. The default value is 2 years before current day.</p> | startDate=2017-05-27 | Optional |
| endDate | <p>A date string that marks the end of the search period in the format of YYYY-MM-DD. The default value is the current day.</p> | endDate=2019-05-27 | Optional |
| sortByDate | <p>A boolean value that is set true will sort the results in descending order from most recent to oldest. The default value is false.</p> | sortByDate=true | Optional |
| pageIndex | <p>An integer value of the current page index requested. When results are provided over multiple pages, the paginationInfo field contains the nextPageUrl that references the pageIndex of the page following the current, if it exists.</p> | pageIndex=2 | Optional |
| asTeaser | <p>A boolean value to indicate requesting only the teaser headers instead of the full article. The default value is false.</p> | asTeaser=true | Optional |

| | | | |
|----------|-------------------------------------------------------------------------------------------------|----------|----------|
| programs | A comma separated list of integer values of a program ID. Returned as an array of programIds. | 262267 | Optional |
| themes | A comma separated list of integer values of a category ID. Returned as an array of categoryIds. | | Optional |
| ids | A comma separated list of integer values of a content ID. Returned as an array of contentIds. | 45761438 | Optional |

5.3.2 EF API

About Europeana Collections

Europeana brings together the collections from over 3500 heritage institutions from across Europe. Through www.europeana.eu one can access over 58 million objects from music to books, from archaeological findings to paintings. V4Design aims to enable the re-use and re-purpose of data of interest to architects and video game designers. The Europeana corpus is broad and representative of what one may find in the collections of Europe's libraries, archives and museums. Europeana works to ensure accurate rights statements to inform the user what they can or cannot do with the objects.

In the context of V4Design, Europeana is a treasure trove with a continuous flow of new or improved digital cultural heritage data being added to the corpus. Relevant material for V4Design are all openly licensed content depicting buildings in artworks or drawings, publications, maps of cities and places, videos of buildings (very limited), archival footage. It should be noted that contemporary material is mostly restricted and is not suitable for inclusion in V4 Design.

About the Europeana REST APIs

The Europeana Foundation offers several APIs under the umbrella of the Europeana REST API, that allows users to build tools and applications that use the wealth of our collections drawn from the major museums, libraries, archives and galleries across Europe. The Europeana collections contains over 50 million cultural heritage objects, from books and paintings to 3D objects and audiovisual material, that celebrate over 3,500 cultural institutions across Europe.

Over the past couple of years, the Europeana REST API has grown into a wide range of specialized APIs. It offers several APIs that can be used not only get the most out of Europeana but also to contribute back. If you want to search Europeana in a simple way (for instance “give me all results for the word cat”), you can then use the Search API. But if you are looking for a way to delve into the structured metadata of Europeana (e.g. to ask the question “what are all the French 18th-century painters with at least five artworks available through Europeana”) then the SPARQL service is more appropriate. On the other, if you

want to get all the metadata associated with a single item, then you can use the Record API. It is also possible to obtain a larger amount of metadata and ultimately harvest the complete Europeana repository by using the OAI-PMH Service. Regarding contextual information that is associated to items, Europeana also offers an Entity API that gives you access to information such as Topics, Persons and Places. Lastly, if you want to contribute information about the items that are available on Europeana, you can do it via the Annotations API.

The Europeana REST API is available free of charge and only requires an access key which needs to be requested prior to its use. More information and technical documentation about each API, as well as the request form for an access key and the Terms of Use, can be found at the dedicated API documentation homepage²⁰.

Using the Europeana REST APIs to collect sample data

As was described in “D2.1. Initial visual and textual dataset creation and legal and ethical requirements”, in the first phase of the project, both the Search API and Record API were used to extract the metadata necessary to build the sample dataset that was used for several tasks in the V4Design consortium. This dataset contained metadata for approximately 23.400 objects formatted in JSON from which 23.400 images (linked from the metadata) were downloaded from the data providers. Note that this dataset was created early in the project and without the usage of the main V4Design Crawler application. The dataset was done using a Python 3 script written for this purpose which performed the following two step requests to the Europeana APIs:

Step 1: Search for all items that matched the V4Design selection criteria

The Search API was chosen to perform the selection on Europeana content, given its ability to fulfill the selection criteria that was needed and the one that offers the best performance.

This API is available via the URL presented below and can be further tailored by an extensive list of parameters as defined in the public documentation²¹.

<https://www.europeana.eu/api/v2/search.json>

For the purpose of the V4Design project, only a limited number of parameters were used as defined in the following template request:

```
GET
https://www.europeana.eu/api/v2/search.json?query=QUERY&rows=ROWS&cursor=CURSOR&reusability=REUSABILITY&wskey=APIKEY
```

Where the variables in uppercase are used in the following way:

| Variable | Description |
|----------|-------------|
|----------|-------------|

²⁰ <https://pro.europeana.eu/resources/apis>

²¹ <https://pro.europeana.eu/resources/apis/search>

| name | |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QUERY | A search query that combines using OR clauses (see syntax section on the Search API documentation ²²) the keywords of the PUCs, which can be found in Appendix 3 of D2.1. |
| ROWS | The number of items that are retrieved in each request. |
| CURSOR | A cursor mark from where to start the search result set when using deep pagination. The value must be set to * to start cursor-based pagination and following requests must use the value that is returned in the response within “nextCursor” field. |
| REUSABILITY | Filters by copyright status. Acceptable values are open, restricted or permission. In the scope of V4Design only “open” was used. |
| APIKEY | The access key obtained after registration. |

Given that a large number of objects is expected to be retrieved upon a query to the service, the cursor based pagination was used to be able to scroll along the complete result list while preserving a constant time performance upon each page request. This means that multiple requests need to be made varying only on the value of the “CURSOR” until it either reaches the total number of items or the client is satisfied with the number of items obtained until then.

Each response, contains a subset of the metadata for a limited number of items. Given the fact that the metadata present in the response is not sufficient to meet the requirements for the SIMMO model, a second API (Record API) was used by following the URL present in the “link” field in the JSON response, as can be seen in the Example below:

```
{
  "apikey": "api2demo",
  "success": true,
  "requestNumber": 999,
  "itemsCount": 1,
  "totalResults": 1,
  "items": [
    {
      "id": "/11621/_NHMUK_PAL_PV_M_26793",
      ...
      "link":
        "https://api.europeana.eu/api/v2/record/11621/_NHMUK_PAL_PV_M_26793.json?w
        skey=APIKEY"
    },
    ... more items ...
  ]
}
```

Step 2: Retrieve complete data for an item

²² <https://pro.europeana.eu/resources/apis/search#syntax>

The Record API was used to obtain a more complete set of metadata fields that describe the item. This API is available via the URL presented below and can be further tailored by an extensive list of parameters as defined in the public documentation²³.

```
https://www.europeana.eu/api/v2/record/ITEM_ID.FORMAT
```

Where the variables in uppercase are used in the following way:

| Variable name | Description |
|---------------|-------------------------------------------------------------------------------------------------|
| ITEM_ID | The identifier of the item, e.g. <code>"/11621/_NHMUK_PAL_PV_M_26793"</code> . |
| FORMAT | The output format, such as: <code>"json"</code> , <code>"rdf"</code> , <code>"json-ld"</code> . |

The JSON output format was used for mapping, from which the following EDM classes and properties were considered:

| Class: edm:Proxy | |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| Property | Description |
| <code>rdf:about</code> | The identifier of the Item. |
| <code>dc:title</code> | A name given to the resource. Typically, a Title will be a name by which the resource is formally known. |
| <code>dc:description</code> | A description of the resource. |
| <code>dc:language</code> | A language of the resource. |
| <code>dcterms:issued</code> | Date of formal issuance (e.g., publication) of the resource. |
| <code>dc:creator</code> | An entity primarily responsible for making the resource. This may be a person, organisation or a service. |
| <code>edm:hasMet</code> | edm:hasMet relates a resource with the objects or phenomena that have happened to or have happened together with the resource under consideration. |
| <code>dc:subject</code> | The topic of the resource. |
| <code>dc:type</code> | The nature or genre of the resource. Type includes terms describing general categories, functions, genres, or aggregation levels for content. |

²³ <https://pro.europeana.eu/resources/apis/record>

| | |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dc:format | The file format, physical medium or dimensions of the resource. |
| dcterms:medium | The material or physical carrier of the resource. |
| dcterms:temporal | Temporal characteristics of the resource. |
| dcterms:spatial | Spatial characteristics of the resource. |
| dc:coverage | The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant. |
| Class: ore:Aggregation | |
| Property | Description |
| edm:dataProvider | The name of the data provider of the object (i.e. the organisation providing data to an aggregator). |
| edm:isShownBy | The URL of a web view of the object. |
| edm:hasView | This property relates an ORE aggregation about a CHO with a web resource providing a view of that CHO. Examples of view are: a thumbnail, a textual abstract and a table of contents. |
| Class: edm:WebResource | |
| Property | Description |
| dc:description | An account or description of this digital representation. |
| edm:rights | The value in this element will indicate the usage and access rights that apply to this digital representation. The rights statement specified at the level of the web resource will "override" the statement specified at the level of the Aggregation. The value in this element is a URI taken from the set of those defined for use in Europeana. A list of these can be found at http://pro.europeana.eu/web/available-rights-statements |
| ebucore:width | The width of a media file in pixels. |
| ebucore:height | The height of a media file in pixels. |
| ebucore:fileByteSize | The size of a media file in bytes. |
| ebucore:duration | The duration of a media file in ms. |
| ebucore:frameRate | The frame rate of the video signal in frames per second. |
| edm:codecName | The name of a device or a computer program capable of encoding |

| | |
|--|-----------------------------------------------------------|
| | or decoding a digital data stream or signal, e.g. "h264". |
|--|-----------------------------------------------------------|

Live integration with Europeana REST APIs

For the future live addition of Europeana data in the V4Design Data Storage and Retrieval module, we plan to integrate the Search and Records APIs as an additional submodule in the V4Design Crawler. However, when considering that data can change on Europeana as a result of an update from data providers, synchronization mechanisms will need to be implemented on the V4Design side that may impact the way that the APIs will be used which will be further investigated in the next phase of the project.

5.4 Twitter

To address and support searching in social media platforms, we chose to collect content from Twitter. Compared to other social media platforms, Twitter provides vast amounts of publicly available data via its APIs. Two of its APIs stand out for this purpose: (i) the general Twitter API²⁴ and (ii) the Twitter Streaming API²⁵.

The general Twitter API allows for running specific platform queries for many of the Twitter's data models. For example, it allows for requesting information about a given Twitter user profile including the list of their followers and friends, and their posts (i.e. tweets). Generally, the largest part of the user activity in Twitter is available through the general API, with the exception of the private messages (direct messages) and the private profiles. However, it is uncommon for Twitter users to make their profiles private, contrary to other popular social media platforms. The Twitter API provides both application-level and user-level access: the former provides general access to the data available on the API, whereas the latter requires user authentication and provides data related to the authenticated user.

On the contrary, the Twitter Streaming API provides access to similar data models with the general Twitter API, but in an event-driven way. This means that the Streaming API allows for creating a persistent connection to Twitter and monitor new tweets in real-time for a given set of profiles and/or a given set of keywords. Additionally the event-driven connection receives events, such as the deletion of a tweet.

Lots of official and unofficial libraries that act as a wrapper for the Twitter APIs have been created in many different programming languages (e.g. Python, C++, PHP, Ruby). Example libraries for JAVA are hbc²⁶ and Twitter4J²⁷. The former one is built by Twitter and consumes the Streaming API. For V4Design initial implementation, we selected the latter one, which is an open-source unofficial Java library for the general Twitter API, for the reason that the current needs of the project did not require streaming real-time data. Therefore, we used Twitter4J's functionalities to collect twitter posts from specified user accounts and send them to the Data Storage module. Similar to searching through other APIs, scraping is not needed here as well.

²⁴ <https://developer.twitter.com/en/docs/tweets/search/overview>

²⁵ <https://developer.twitter.com/en/docs/tweets/filter-realtime/overview>

²⁶ <https://github.com/twitter/hbc>

²⁷ <http://twitter4j.org>

6 QUERY EXPANSION

The retrieval of content using the web & social media search module requires that proper queries are determined, as they are indispensable as an input. These queries can be created in three different ways: a) manually, b) semi-automatically and c) automatically.

Manual is the most straightforward way of feeding queries to the search module, as it is similar to a typical usage of a search engine in the web. In our project, the queries are produced by the user partners, based on their informational needs for the final tools to be developed.

To tackle queries that are vague and difficult to handle by search engines, they can be transformed by adding and/or removing terms. When these modifications are performed automatically by the system, while the initial query creation still remains manual, the query creation method is characterized as *semi-automatic*. In such scenarios, the changes to the query can be either applied directly by the system, without asking for confirmation by a user, or can be provided as suggestions for a user who is the person that makes the final decision for the search action.

The approach that is exclusively *automatic* does not involve the manual formulation of a query; it simply considers a domain of interest as this is exemplified by a set of documents (e.g., Web pages and/or social media posts) relevant to the domain. Machine learning and text mining techniques can then be applied so as to extract the most important domain concepts, corresponding either to lists of terms or (Boolean) expressions that can be used as automatically formulated queries for discovering further relevant information. This approach is outside the scope of the current project as we consider that the human intervention is essential for the data collection process. Thus, we follow the first two of the three mentioned approaches.

Query reformulation is a general term that encapsulates a wide range of query transformation methods. The most representative ones are query *expansion*, query *substitution* and query *reduction*. A query can be reformulated by making use of one or more of these methods simultaneously.

Query expansion has been successfully used in the past for information retrieval, where it has been proved to optimize performance in various applications (Carpineto and Romano, 2012). Its main strength is that helps relevant documents to be placed higher in the ranking even if they do not literally contain the initial query terms. For instance, if the query “Acropolis” is expanded by adding the terms “parthenon” and/or “Athens” and/or “Greece” and/or “Ancient Greece”, this new query does not only retrieve the documents that contain the original term (Acropolis) but also documents that use relevant and similar words as well as documents that do not directly contain it.

The most common data sources for generating new terms include: (i) large-scale external corpora that can be considered to reflect the overall term distribution in a given language, such as Wikipedia titles and/or articles in a given language (Balog et al., 2008) or even query logs (Wang and Zhai, 2008) (ii) the document collection being searched in the current setting (Xu and Croft, 1996) that can be viewed as modelling term distribution in a particular domain, and (iii) documents relevant to the submitted query which are identified either interactively by the user or automatically by the system; in the former case, i.e., in a so-

called *relevance feedback* cycle, the user pro-actively provides guidance in the form of relevant reference documents (Efthimiadis, 2000), while in the latter case, referred to as *pseudo-relevance feedback*, the top retrieved documents are assumed to be relevant (Xu and Croft, 1996). Query expansion using multiple data sources has been shown to generate complementary results and hence has the potential to further increase the search effectiveness. Nevertheless, its usage must be cautious as adding erroneous expansion terms to the query may degrade its performance.

Query substitution aims to completely change a user’s original search query and introduce a new, superior one in terms of retrieval performance (Jones et al., 2006). It shares similarities with the *query expansion* since the first step involves the *query expansion* process. However, instead of using the produced terms to expand the original query by simply adding them, *query substitution* totally changes it by replacing the original query terms with the automatically extracted ones. Similarly to *query expansion*, the original query can be transformed either by the search system (Craswell et al., 2013) and/or with the help of the end users (Belkin et al., 2001).

Query reduction is the process of removing one or more terms from the original query which leads to the creation of one or more subqueries that aim to increase search efficiency, by dropping, for instance, too specific, unnecessary or ambiguous terms. As search engines often rank documents in response to queries based on weak conjunctions or multiplicative combinations of query terms (Belkin et al., 2001), documents containing such specific, unnecessary or ambiguous terms may end up being highly ranked and as result cause topic drift. Query reduction helps to support documents containing the most essential query terms, particularly in the context of queries that spread to a large number of words, and at the same time to reduce the retrieval score of documents that contain terms that are liable to cause topic drift.

In some occasions, the union of the retrieval results of the original query with the results of the transformed query/queries, by applying one or more types of transformations (expansion, substitution, reduction), is considered (Balog et al., 2008). Retrieving optimal results by taking into account only the transformed versions of a query is a challenging task as these versions have to perfectly capture the informational need. To tackle with this challenge, user interactions can be employed as a way for selecting the best query transformations (Kumaran and Allan, 2008).

Since most of the queries defined for searching for web resources are short (their length is less than 5 words), we are mostly interested in expanding the query to discover terms that are initially omitted by the user as self-evident ones. The *query expansion* method we implemented involves the usage of neural word embeddings which are detailed in the next subsection.

6.1 Neural Word Embeddings

Word embeddings are a popular type of word representation that encodes words in a specified vocabulary as low-dimensional vectors. *Neural word embeddings* are word embeddings that are trained using deep learning neural networks.

The most well-known neural word embedding models are word2vec (Mikolov et al., 2013) and Global Vector for Word Representation (GloVe) (Pennington et al., 2014). In word2vec,

two different architectures are proposed, namely Continuous Bag-of-Words (CBOW) and Skip-gram. Both are trained by taking into account the words being in a small distance in a sentence. The distance considered is configured by a parameter whose name is window size. In CBOW, the vector of a word is predicted by the context i.e. the surrounding words, whereas in the Skip-gram model the exact opposite function is applied, that is given a word its context is predicted. In GloVe, a different approach is proposed where training of the word vectors is made by leveraging co-occurrence statistic of pairs of words given a corpus. Both word2vec and GloVe aim to model the embeddings in a manner that semantically similar words have vectors that are close to each other.

6.2 Implementation

For V4Design, apart from submitting the initial queries provided by the user partners, which falls to the manual query formulation scenario, we also developed a mechanism for extending the query with more relevant terms. As the query expansion mechanism is automatic (while the initial definition still remains manual), the implemented formulation approach is considered semi-automatic.

The initial version of the query expansion module makes use of pre-trained word vectors that were trained on large corpora. Such vectors are trained on loads of text that address numerous different topics and hence they are suitable for reformulating the queries we are interested in. As they cover a broad context, they are named as universal embeddings.

A popular published set of pre-trained word embeddings is published by Mikolov et al. (Mikolov et al., 2013), as part of their work on word2vec. They were trained on part of the Google (English) News dataset, a rather large corpus that contains approximately 100 billion words. The model file provides 300-dimensional vectors for 3 million words and phrases.

Moreover, Pennington et al. (Pennington et al., 2014), in the framework of the Glove model introduction, provided a set of different pre-trained word embeddings. This set includes embeddings of many different dimensionalities that were trained on Wikipedia and Gigaword articles²⁸, Twitter posts, and the Common Crawl dataset²⁹. Table 2 shows some useful information about these datasets.

Table 2: Word embedding pre-trained models

| Corpus | Tokens | Vocabulary Words | Capitalization | Dimensions |
|--------------------------------|-------------|------------------|----------------|------------------|
| Wikipedia 2014 + Gigaword 5 | 6 billion | 400 thousand | No | {50,100,200,300} |
| Common Crawl | 42 billion | 1.9 million | No | 300 |
| Common Crawl | 840 billion | 2.2 million | Yes | 300 |
| Twitter | 2 billion | 1.2 million | No | {25,50,100,200} |

²⁸ <https://catalog.ldc.upenn.edu/LDC2011T07>

²⁹ <http://commoncrawl.org/>

To extract the most relevant terms to a query, we make use of the built-in method of the `gensim`³⁰ python library that calculates the cosine similarity of the query with each one of the vocabulary terms. From the terms list of each model, the ones that do not belong to the NLTK³¹ Python library English words list and the ones that are considered by NLTK as English stopwords are not considered as candidates for the expansion process. Also, we made use of only the models that have vectors with dimensionality higher or equal than 200 because we expect that the quality of vectors is downgraded should we keep few dimensions. The top-5 ranked terms for each model and for indicative queries that are relevant to the project's use cases are presented in Table 3.

Table 3: Top-5 relevant terms to search queries proposed by word embedding models

| Query | GloVe | | | | | Google News |
|------------------|---------------------------------------------------------------|----------------------------------------------------------|-------------------------------------------------------|---------------------------------------------------------------|-------------------------------------------------------|-------------------------------------------------------------|
| | 6B.200d | 6B.300d | 42B.300d | 840B.300d | twitter.27B.200d | |
| Eiffel Tower | skyscraper spire building facade staircase | skyscraper spire building facade edifice | louvre paris building skyscraper bridge | skyscraper Louvre Paris Basilica Manhattan | paris bridge square hotel building | skyscraper Gothic beefeater campanile brutalist |
| Acropolis | parthenon athens colosseum capitoline agora | parthenon athens marbles colosseum monument | parthenon athens agora syntagma athenian | Parthenon Athens Colosseum Athenian Louvre | athens parthenon rally syntagma hermitage | Parthenon Colosseum Panathenaic Lycian Athens |
| Brandenburg Gate | saxony entrance sluice rhine berlin | entrance saxony castle berlin gateway | berlin entrance palace bridge park | Brandenburger Saxony Vienna Waterloo Hohenzollern | bridge station berlin airport rail | Oldenburg Saxony Werner Hohenzollern Wolf |
| Notre Dame | villanova hunchback seton auburn mater | villanova auburn hunchback seton mater | villanova auburn football duke university | Villanova Duke Ole Football Wesleyan | alabama salle football nous duke | hiver Marie salle elle ecole |
| Colosseum | rome crucifixion procession basilica amphitheater | crucifixion rome basilica obelisk procession | pantheon coliseum rome basilica piazza | Coliseum Basilica Flavian Pompeii Rome | sphinx rome pantheon ballroom belvedere | piazza cathedral Pompeii Rome tomb |

From this Table we can observe that the embedding models are capable of detecting relevant terms. For example, the location of the Acropolis and Colosseum monuments is detected in all the models. For Eiffel Tower and Brandenburg Gate, their location is properly retrieved in at least some of the embedding models. Additionally, more relevant terms are accurately proposed such as “Parthenon” for the “Acropolis” query, and “Coliseum” as a

³⁰ <https://radimrehurek.com/gensim/index.html>

³¹ <http://www.nltk.org/>

different formulation of “Colosseum”. However, there are also a lot of noise terms in the top-ranked results, especially in the Notre Dame query where the topic of the query is completely deviated from what is expected by the proposed terms. Instead of fetching terms relevant to the famous cathedral, the models tend to provide noise terms e.g. the “Villanova” one that refers to an academy carrying the Notre Dame name. Therefore, the claim that the expansion models should be used with caution is more than valid. In the updated version of the query expansion module, such issues will be addressed so as to improve the effectiveness of the models in proposing relevant terms and accordingly in improving the final search results of the query.

7 DATA MODEL

7.1 The SIMMO data model

This section presents the proposed framework for the unified representation of Socially Interconnected MultiMedia-enriched Objects (SIMMO) available in web environments (Tsikrika et al., 2015). While similar entities are also encountered, at least in part, in other models (e.g., (S.-F Chang et al, 2001), (Daras et al. 2011), (Bojars et al., 2008)) that have also formed part of our inspiration it is the interconnections among SIMMO elements and the novel approach of bridging the gap between multimedia and social features that make SIMMO unique in its ability to support a wide range of applications. Figure 5 presents the conceptual model of SIMMO with the following core entities and their sub-classes:

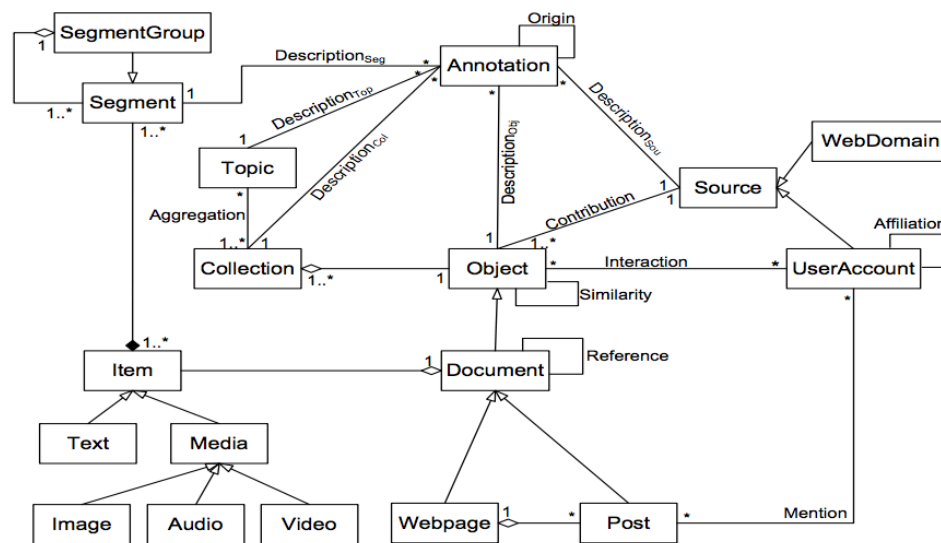


Figure 5: SIMMO conceptual model presenting its elements and their relations. For simplicity, association relations that have attributes are depicted as simple associations.

– **Object** is a generic entity representing media content ranging from monomodal Items to multimedia Documents. Each Item represents the actual media content consisting of a single modality, such as Text, Image, Video, or Audio, whereas Documents may be viewed as container objects consisting of potentially multiple such Items, and thus modalities. The most common instantiations of Web Documents are Webpages (e.g., pages in news sites, in entertainment portals, etc.) or Posts in media sharing platforms with social characteristics (e.g., Facebook posts, tweets, etc.). There are also cases of Webpages consisting of Posts; a forum page, for instance, can be viewed as a container object consisting of posts on the same topic. The Media entity is introduced as an abstraction of Image, Video, and Audio so as to represent their common characteristics, such as the fact that they all may be associated with a Text item modelling the text associated with them (e.g., a caption) or extracted from them through e.g., ASR (Automatic Speech Recognition) for Video and Audio, and OCR (Optical Character Recognition) for Image and Video. Finally, further media (e.g., 3D objects) may be added as Item instantiations depending on the requirements of the particular application.

-
- **Source** is a generic entity representing media content contributors. This includes UserAccounts representing users generating content, mainly posts in social media sharing platforms where they hold accounts, and WebDomains representing the Web sites hosting media content generated by their contributors. WebDomains are viewed as content contributors, even though they do not correspond to the actual person who contributed the content, given that in many cases the information regarding such people may not be available, or may be of much lesser importance in this specific context.
 - **Segment** locates the media content of Items at a finer level of granularity (e.g., a passage in text, a region in an image, or a portion of a video) by including positional information as attributes. Instantiations of Segments (not depicted in Figure 5) include LinearSegments (e.g., with start/end positions as attributes for referring to text parts), SpatialSegments (e.g., with (x, y) pairs as attributes for referring to image regions), TemporalSegments (e.g., with start/end times as attributes for referring to video frames/shots/scenes), and SpatioTemporalSegments. A SegmentGroup represents a collection of Segments; it is also modelled as a sub-class of Segment, thus allowing it to contain both Segments and other SegmentGroups.
 - **Collection** models aggregates of Objects (i.e., higher levels of granularity), such as corpora of Web documents, sets of tweets, and image collections.
 - **Annotation** is a generic entity representing together with its sub-classes (not depicted in Figure 5) a wide range of descriptions extracted from media content. These include annotations typically extracted from text (e.g., keywords, named entities, summaries, categories, etc.), media content features (e.g., low level descriptors, concepts and events), affective descriptions (e.g., sentiment and polarity), veracity scores reflecting the reliability of information and thus the trust that should be placed on it, and many others.
 - **Topic** refers to any subject of interest in the context of information processing, analysis, or access applications that users would like to keep track of. Its explicit representation allows supporting a broad range of tasks, such as information filtering, topic tracking, and classification. The main relations between these SIMMO elements, excluding the already discussed generalisation and aggregation/composition relations, are:
 - The generation of media objects is modelled through a Contribution association between Source and Object.
 - Explicit relations between Documents are modelled as Reference associations, with attributes such as the type of the relation. By considering that a Document may Reference another Document, we also consider (through inheritance) that a Webpage may Reference another Webpage (e.g., link to it) and a Post may Reference another Post (e.g., reply to it or comment on it). We consider that this association is also able to model the References to Webpages from Posts (e.g., the Web links embedded in tweets) and to Posts from Webpages (e.g., to the comments dynamically posted on a Webpage).
 - Objects may also be implicitly related to other Objects, e.g., through a computation of their similarity. Such Similarity relations are modelled as recursive associations between Objects, with attributes such as the type of the relation and the similarity score. This is useful in several applications and tasks, including clustering and verification of content provenance.
-

- A User Account may be involved in several relations, e.g., (i) be mentioned in a Post, (ii) be affiliated with (be friends with, follow etc.) another UserAccount, or (iii) interact with an Object (through likes, shares, views, etc.); the latter is more common for Posts, but users also interact with (e.g., like) whole Webpages. These three relations are modelled through the Mention, Affiliation, and Interaction associations, respectively, with attributes, such as the type of relation and the date it was established. Commenting is not modelled as a relation between Documents and UserAccounts, but rather as a Reference between two Documents (e.g., two Posts).
- All types of entities (i.e., Objects, Segments, Collections, Sources, and Topics) and their sub-classes may be associated with Annotations that are used for describing them. Such Description relations represent, for instance, the annotation of an Image with the SIFT features extracted from it, a TemporalSegment of a Video (such as a shot) with Concepts, or a UserAccount with Keywords reflecting the users' profile. Furthermore, links between different annotations (e.g., low-level descriptors and the concepts obtained from them) are modelled through the reflexive relation Origin between Annotations to denote the provenance of one with respect to the other.
- Each Topic is associated with a Collection of Objects on the particular subject of interest and may also be annotated itself. For instance, the Topic "Tour de France 2014" bicycle race would be associated with a Collection of Documents, such as Webpages and tweets on the subject, and could be annotated with the concepts "cycling" and "yellow jersey", the entity "Union Cycliste Internationale", and extracted locations, such as "Grenoble, France".

SIMMO elements and their relations also have several attributes representing their properties. For example, each Object is associated with a URI, creation date, and crawl date. Text is described by its format (e.g., HTML), an Image by its size, EXIF data, and associated thumbnail, a Video by its duration, number of frames, and associated thumbnail, and an Audio by its duration. Documents also have attributes related to the statistics regarding their social interactions, e.g., numbers of likes, comments, views, etc. The properties of a UserAccount include a stream ID denoting the platform hosting the account, the user's name, and the number of followers/following/friends.

Implementation: We have implemented the SIMMO framework in Java 1.7. We used Maven for controlling the project's build process, unit testing, and documentation creation, and the Google GSON library for converting Java objects into their JSON representation. This does by no means constrain the user from choosing another JSON library or another serialisation method. The SIMMO framework is open-source, released under the Apache License v2, and available at: <https://github.com/MKLab-ITI/simmo>.

7.2 Adaptations

The SIMMO model can cover a wide range of data representation requirements, however, there are a lot entities and metadata that have to be stored for this project and cannot be mapped using the published SIMMO version. Therefore, adjustments had to be made to the model in order to keep a unified way of storing and managing the crawled data. The only alternative option was to employ different data models for the content that could not fit into the SIMMO model, nevertheless such an action would complicate the entire system

architecture as the components connected to the crawling module would have to deal with diverse data that are stored in completely different formats.

In the current version of the modified SIMMO model, three new entities have been introduced as dedicated JAVA classes. The first one covers PDF document files as a special case of Document objects. In the original SIMMO model, the only Document instances are either “Webpage” or “Post” (social media posts). Neither of these two entities could include PDF files, so a dedicated one is created for this purpose. Another entity that was created is a subclass of the “Annotation” class and its usage is to hold metadata annotations, in form of key-value pairs. The last entity was created in order to be able to link multimedia items to their originating resources. For example if we retrieved a video that belongs to a specific webpage, we have to capture a webpage identifier in the video representation in order to access the object containing that multimedia item. That would significantly help in cases when we wish to directly retrieve videos, without having to go through the webpage list first. To this end, we created a “Parent” class containing an identifier and the type of the resource associated with this identifier.

Apart from creating new entities, new fields have been added to the already existing entities. For all SIMMOs, the license, an alternative URL address and a list of comments can be stored. Alternative URLs are mainly added to include links to the data storage module. For each “Item” entity, a list of the newly defined “Parent” objects is added to specify the originating resources. Additionally, changes have been made to the multimedia object entities. More specifically, in image objects an additional URL address can be stored to retrieve their thumbnails and in video objects, saving the video format and the bit rate is now supported.

Last, new behaviors are integrated into the “Document” instances that return if they contain texts, images or videos. These functions support the enrichment of the messages sent to the Message Bus with information that help the other components of the preprocessing pipeline determine if they should process a collected document. For example, the Aesthetics Extraction module (T3.5) cannot process documents containing only text, so it should focus on documents that include at least an image or a video. The enhancements made to the SIMMO model are summarized in Tables 4, 5, and 6.

Table 4: New SIMMO entities

| Entity | Subclass of | Description |
|----------|-------------|------------------------------------------|
| PDF | Document | A PDF file representation |
| Metadata | Annotation | Key-value pairs of metadata annotations |
| Parent | - | The document an item is originating from |

Table 5: New SIMMO fields

| Field | Entity (Class) | JAVA Type | Description |
|----------------------|----------------|--------------|----------------------------------------------------------|
| License | Object | String | The license of the stored resource. |
| alternativeUrl | Object | String | An alternative address to retrieve the resource. |
| comments | Object | Set<String> | A list of comments that have been made for the resource. |
| Parents | Item | List<Parent> | The originating resources. |
| alternativeThumbnail | Image | String | An alternative URL for the thumbnail of an image. |
| Format | Video | String | The video format. |
| Bitrate | Video | String | The bitrate of the video. |

Table 6: New SIMMO behaviour

| Behavior (Function name) | Entity (Class) | Return type | Description |
|--------------------------|----------------|-------------|-----------------------------------------------|
| hasText() | Document | Boolean | Returns True if a Document has “Text” items. |
| hasImage() | Document | Boolean | Returns True if a Document has “Image” items. |
| hasVideo() | Document | Boolean | Returns True if a Document has “Video” items. |

8 DATASETS CREATED

In this section we present the datasets created using the V4Design Crawler application. The collected resources address the data collection needs defined from the start of the project until this reporting period.

8.1 Wikipedia dataset

The Wikipedia website is the largest and most popular multilingual online encyclopaedia in the World Wide Web. The English version contains almost 6 million articles, while in total the entire website has more than 40 million articles in 301 different languages. All the content published there is free and the website covers a wide range of topics, so it is more than appropriate for our data collection tasks. It encompasses both textual content that can be used by the V4Design text analysis tasks such as the concept extraction and visual content that is important for tasks like spatio-temporal object and building localization and 3D reconstruction.

A suitable topic that fits most of the V4Design tasks is “castles”. Consequently, we scraped and added 311 Wikipedia Web pages that described a castle in this dataset. The collected text comprised of the main content of the webpage as well as metadata placed in the infobox that exists at the majority of the Wikipedia pages. Images existing in the webpages are also collected along with their captions. Apart from these 311 webpages that were related to castles, we enriched this dataset with three more webpages that were relevant to a different topic. The reason was that we needed to scrape and test resources that were addressing PUC4 (Design of virtual environments, related to actual news for VR reliving the date). Each one of these 314 resources contained one textual instance and zero, one or many visual instances. The total number of visual instances in this dataset is 661, all of them being images. An example page that was crawled is the Gendarmenmarkt, which describes one of the main squares in Berlin. An image existing in this Wikipedia page is depicted in Figure 6.



Figure 6: 2008 Panorama of the Gendarmenmarkt

8.2 Deutsche Welle Nico’s Weg exercises dataset

“Nico’s Weg” is an online German course provided by Deutsche Welle in the form of a telenovela that is available as a web and a mobile application. It integrates video footage as well as textual content and interactive exercises to practice and improve German language skills in many different levels. The V4Design crawler was used to crawl and scrape the textual content from the domain of the courses that target beginners learning the A1 level of the

language. Pages presenting grammar, vocabulary and culture/society details were scraped and integrated into this dataset. An example of such webpage is the one that explains the usage of the pronoun 'es' and its extracted content is shown in Figure 7. In total, 440 webpages are scraped for this dataset.

Grammar

Pronouns: es

es used as a personal pronoun
The pronoun es is usually used to replace a neuter noun:
Wo ist das Buch? – Es ist in der Tasche. (es = das Buch)

es as the formal subject with impersonal verbs

There are other functions that es can have. Sometimes es does not stand in for a definite noun but takes the role of the subject without having a meaning of its own. This construction is often used in impersonal phrases in which there is no other subject.

Examples of this are sentences or clauses that describe weather phenomena:

Es regnet.
Es schneit.
Es ist sonnig.
Es ist bewölkt.
Es ist neblig.

Here, es stands for a situation or state to which no concrete subject can be attributed.

Figure 7: Scraped text from <https://learngerman.dw.com/en/pronouns-es/l-37653197/gr-38306282>

8.3 Twitter dataset

For addressing the social media content extraction needs, we used the implemented infrastructure to collect textual content from Twitter posts in order to support the text analysis tasks. For this dataset, the main topic considered is Cultural Heritage, so we targeted on tweets published by a specific set of users. The social media search module was applied on 21 selected relevant users which included: a) Twitter bots that continuously tweet out objects from digital archives like the Metropolitan Museum, b) official Twitter accounts of Cultural Heritage institutions, c) other accounts related to Cultural Heritage not falling into the previous two categories. The final outcome consists of texts from about 40 thousand tweets. All contents are stored in MongoDB and they were delivered in JSON format. A typical instance of such Twitter post is located at

<https://twitter.com/MuseumBot/status/892811000994557953>. That post was published from a bot tweeting images from the Metropolitan Museum of Art.

8.4 Flickr dataset

Data collection for the Flickr website was done in two different phases. In the first phase, when an initial implementation of the Flickr search mechanism was complete, we opted to search and collect images from famous building as well as cars. We formulated 13 queries and collected a total of 6209 images. Example queries are the Eiffel tower, the Delphi temple and the Volkswagen Beetle. This collection was made to evaluate the suitability of the image content for the 3D reconstruction tasks, thus a small set of metadata was stored there.

However, to address the rest of the project tasks we updated the Flickr search module in the second phase to capture all the required fields that are provided by the platform. Additionally, based on the user's input we enhanced the query set. 137 queries were defined that were relevant to well-known buildings all around the world. The resulting new dataset consists of 53740 images stored in the SIMMO format. In all the queries for both phases, we set a maximum results limit per query to 1000 and we searched only images that are freely available for reusing and repurposing by setting a filter on the licenses list that are available on Flickr. An example picture returned from the "Hagia Sophia" query along with the stored SIMMO record (its most important fields) in the database is shown in the Figures 8 and 9.



Figure 8: Picture in the Flickr website showing the Hagia Sophia church


```
{
  "_id" : "64ce2353-c8ee-4466-994a-58e87ee76c37",
  ...
  "thumbnail" : "https://farm1.staticflickr.com/148/358429847_3c82e8630e_t.jpg",
  "exif" : {
    "ExifVersion" : "0221",
    ...
    "LightSource" : "Unknown"
  },
  "location" : {
    "coordinates" : [ 41.0061645507813, 28.9773979187012 ],
    "radius" : 0.0, "inferred" : false
  },
  "webPageUrl" : "https://flickr.com/photos/97189870@N00/358429847",
  "source" : "Flickr",
  ...
  "url" : "https://farm1.staticflickr.com/148/358429847_3c82e8630e_o.jpg",
  "title" : "Hagia Sophia (2006-10-087)",
  "description" : "Hagia Sophia, (the Church of) Holy Wisdom, now known as the
Ayasofya Museum, is a former Eastern Orthodox church converted to a mosque in 1453
by the Turks, and converted into a museum in 1935. It is located in Istanbul, Turkey. It is
traditionally considered one of the great buildings in history. Its conquest by the
Ottomans at the fall of Constantinople is considered one of the great tragedies of
Christianity by the Greek Orthodox faithful.",
  "tags" : ["turkey", "istanbul", "architecture"],
  "searchQuery" : "Hagia Sophia"
}
```

Figure 9: A SIMMO record associating to a Flickr search result

9 DEMONSTRATOR

For visualizing and evaluating the data collection results, a website was created that showed the collected textual and visual content of the discovered resources through an easy-to-use user interface. The home page is illustrated in Figure 10. The user can go through three options: a) see the textual content of a scraped webpage, b) see all the scraped images for the Wikipedia dataset, along with their captions and c) see the returned images from a Flickr query. Examples for each one of these three cases are shown in Figures 11,12 and Figure 13.

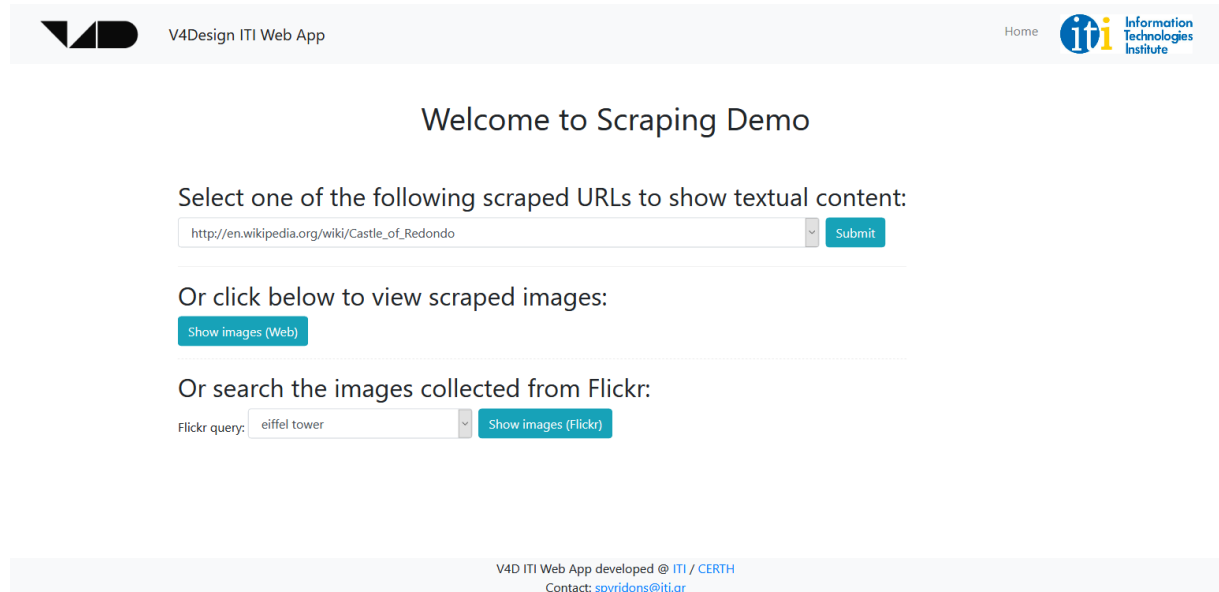
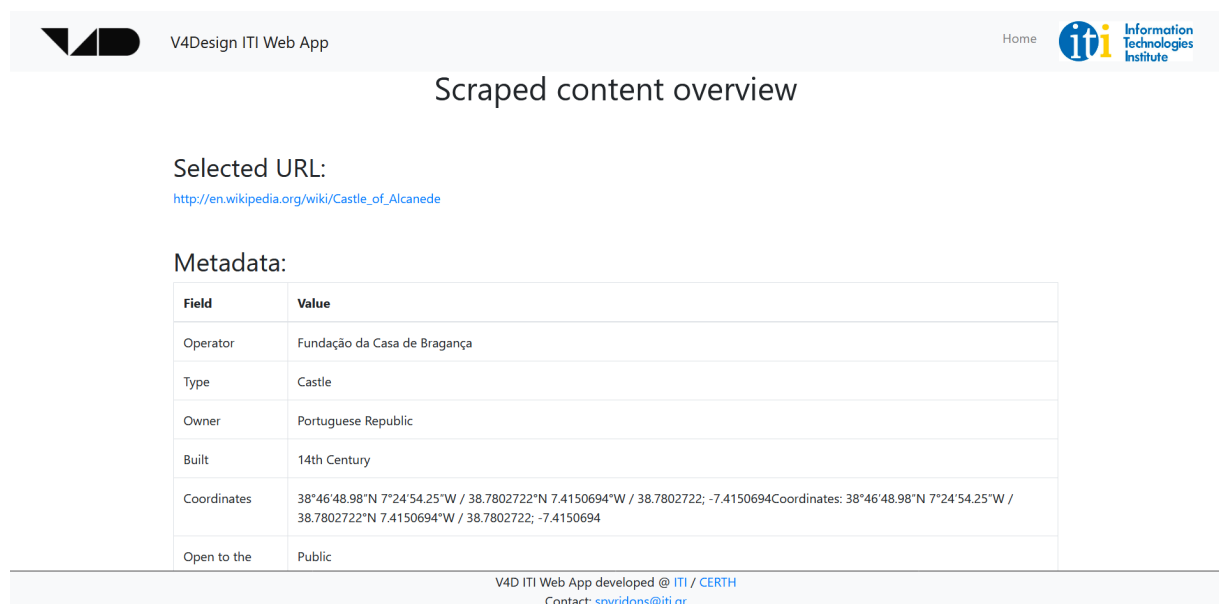


Figure 10: V4Design Crawler demo








| Field | Value |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Operator | Fundação da Casa de Bragança |
| Type | Castle |
| Owner | Portuguese Republic |
| Built | 14th Century |
| Coordinates | 38°46'48.98"N 7°24'54.25"W / 38.7802722°N 7.4150694°W / 38.7802722; -7.4150694Coordinates: 38°46'48.98"N 7°24'54.25"W / 38.7802722°N 7.4150694°W / 38.7802722; -7.4150694 |
| Open to the | Public |

Figure 11: Scraped metadata of a Wikipedia page

V4Design ITI Web App


Home


 Information Technologies Institute

| URL | Source webpage | Caption |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
|  | https://en.wikipedia.org/wiki/Delphi | The Delphic Tholos, seen from above. |
|  | https://en.wikipedia.org/wiki/Delphi | Ruins of the ancient Temple of Apollo at Delphi, overlooking the valley of Phocis. |
|  | https://en.wikipedia.org/wiki/Delphi | The gymnasium |
|  | https://en.wikipedia.org/wiki/Delphi | View of the Athenian Treasury; the Stoa of the Athenians on the Right. |

V4D ITI Web App developed @ ITI / CERTH
Contact: spyridons@iti.gr

Figure 12: Images from the Delphi Wikipedia page

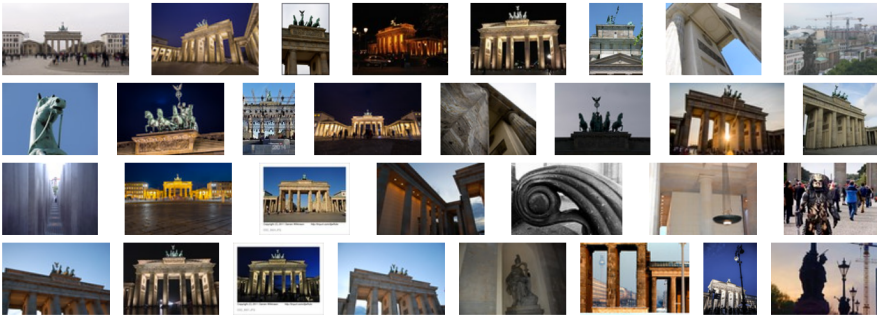

V4Design ITI Web App

Home

Information Technologies Institute

[Back to Demo](#)

Flickr images

Total images: 999



V4D ITI Web App developed @ ITI / CERTH
Contact: spyridons@iti.gr

Figure 13: Flickr search results for the “Brandenburg Gate” query

Both the demonstrator and the included data were evaluated by the user partners. Their general impression was that the purpose of the application becomes clear very quickly and the main functions can be learned with ease. The collected data were assessed as relevant to the project, however, it was not fully clear, how the results are going to be used. That can be attributed to the application being a demonstrator of the V4Design Crawler which produces raw data as output. These raw data themselves cannot comprise meaningful information for the end users without being processed by the rest of the technical components of the V4Design platform.

10 CONCLUSIONS AND NEXT STEPS

In this deliverable, we have presented the initial version of methods that we have developed for the V4Design Crawler, an application that integrates and connects a set of different modules such as the content scraping module, and the query expansion module. The goal for which V4Design Crawler was developed for the project is to collect useful multimedia content that can be then processed by the WP3 and WP4 modules. It is the first step of a pre-processing pipeline that generates content to be utilized by the VR authoring tool and the Rhino plugin (WP6). In addition, the current deliverable has documented the conceptual architecture of the domain-specific query formulation and the query formulation based on user interaction.

Also, we achieved to develop an innovative application that integrates a variety of web data collection functionalities from heterogeneous resources, in contrast with other applications in the market that focus only on one aspect of data collection e.g. web crawling and scraping. Apart from that, to our knowledge no software exists that integrates the functionality of automatically adding more terms to a user search query with the purpose of helping the search engines produce more appropriate results. Finally, we expanded an existing data representation model, the SIMMO one that has been successfully used in previous projects, to support more informational needs and fulfil the additional data storage requirements that emerged specifically in this project.

We also present the datasets created using the V4Design Crawler application. The collected resources address the data collection needs defined from the start of the project until this reporting period. The sources leveraged pertain to Wikipedia, DW Nico's Weg, Flickr, Twitter. For visualizing and evaluating the data collection results, a website was created that showed the collected textual and visual content of the discovered resources through an easy-to-use user interface. The relevance of the results for the project's purposes was confirmed, but it was still difficult to understand how they are used due to the rawness of the output.

As regards our future plans to expand and enrich the current version of developed modules, these are outlined in the following paragraphs.

As for the content scraping module, we plan to update the scraping process to extract more multimedia information. Furthermore, the updated scraping module will be more flexible as an approach that will be identical for all the web domains is going to be implemented.

As future plans for the query expansion are concerned, we plan to introduce more advanced techniques that will better exploit word embedding models to refine the query and, as a result, improve the web and social media searching outcome. More word embedding models will be created and tested for their suitability in the project's searching needs.

Furthermore, we are going to reconsider the data collection needs with the aim of extending the V4D Crawler with more web and social media searching functionalities (e.g. by integrating more APIs into the tool).

Last but not least, we are going to use the current version of the V4Design Crawler (and its extensions) to collect more data relevant to the project's requirements and thus, introduce additional enriched datasets.

REFERENCES

- Balog, K., Weerkamp, W., & De Rijke, M. (2008, July). A few examples go a long way: constructing query models from elaborate query formulations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 371-378). ACM.
- Belkin, N. J., Cool, C., Kelly, D., Lin, S. J., Park, S. Y., Perez-Carballo, J., & Sikora, C. (2001). Iterative exploration, design and evaluation of support for query reformulation in interactive information retrieval. *Information Processing & Management*, 37(3), 403-434.
- Bojārs, U., Breslin, J. G., Peristeras, V., Tummarello, G., & Decker, S. (2008). Interlinking the social web with semantics. *IEEE Intelligent Systems*, 23(3), 29-40.
- Carpineto, C., & Romano, G. (2012). A survey of automatic query expansion in information retrieval. *ACM Computing Surveys (CSUR)*, 44(1), 1.
- Chang, S. F., Sikora, T., & Purl, A. (2001). Overview of the MPEG-7 standard. *IEEE Transactions on circuits and systems for video technology*, 11(6), 688-695.
- Craswell, N., Billerbeck, B., Fetterly, D., & Najork, M. (2013, February). Robust query rewriting using anchor data. In *Proceedings of the sixth ACM international conference on Web search and data mining* (pp. 335-344). ACM.
- Daras, P., Axenopoulos, A., Darlagiannis, V., Tzovaras, D., Le Bourdon, X., Joyeux, L., & Camurri, A. (2011). Introducing a unified framework for content object description. *International Journal of Multimedia Intelligence and Security*, 2(3-4), 351-375.
- Efthimiadis, E. N. (2000). Interactive query expansion: A user-based evaluation in a relevance feedback environment. *Journal of the Association for Information Science and Technology*, 51(11), 989-1003.
- Jones, R., Rey, B., Madani, O., & Greiner, W. (2006, May). Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web* (pp. 387-396). ACM.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
- Olston, C., & Najork, M. (2010). Web crawling. *Foundations and Trends® in Information Retrieval*, 4(3), 175-246.
- Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).
- Tsikrika, T., Andreadou, K., Moumtzidou, A., Schinas, E., Papadopoulos, S., Vrochidis, S., & Kompatsiaris, I. (2015, January). A unified model for socially interconnected multimedia-enriched objects. In *International Conference on Multimedia Modeling* (pp. 372-384). Springer, Cham.
- Wang, X., & Zhai, C. (2008, October). Mining term association patterns from search logs for effective query reformulation. In *Proceedings of the 17th ACM conference on Information and knowledge management* (pp. 479-488). ACM.

Xu, J., & Croft, W. B. (1996, August). Query expansion using local and global document analysis. In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 4-11). ACM.