Scraping Scientific Web Repositories: Challenges and Solutions for Automated Content Extraction

Article *in* D-Lib Magazine · September 2016

DOI: 10.1045/september2016-meschenmoser

D-Lib Magazine

September/October 2016
Volume 22, Number 9/10
Table of Contents

Scraping Scientific Web Repositories: Challenges and Solutions for Automated Content Extraction

Philipp Meschenmoser, Norman Meuschke, Manuel Hotz, Bela Gipp
Department of Computer and Information Science, University of Konstanz, Germany
{philipp.meschenmoser, norman.meuschke, manuel.hotz, bela.gipp}@uni-konstanz.de

DOI: 10.1045/september2016-meschenmoser

Printer-friendly Version

Abstract

Aside from improving the visibility and accessibility of scientific publications, many scientific Web repositories also assess researchers' quantitative and qualitative publication performance, e.g., by displaying metrics such as the h-index. These metrics have become important for research institutions and other stakeholders to support impactful decision making processes such as hiring or funding decisions. However, scientific Web repositories typically offer only simple performance metrics and limited analysis options. Moreover, the data and algorithms to compute performance metrics are usually not published. Hence, it is not transparent or verifiable which publications the systems include in the computation and how the systems rank the results. Many researchers are interested in accessing the underlying scientometric raw data to increase the transparency of these systems. In this paper, we discuss the challenges and present strategies to programmatically access such data in scientific Web repositories. We demonstrate the strategies as part of an open source tool (MIT license) that allows research performance comparisons based on Google Scholar data. We would like to emphasize that the scraper included in the tool should only be used if consent was given by the operator of a repository. In our experience, consent is often given if the research goals are clearly explained and the project is of a non-commercial nature.

Keywords: Web Crawling; Web Scraping; Crawler Detection; Open Access

1 Introduction

In today's competitive international research environment, scientific Web repositories have become increasingly important to various stakeholders. Researchers have a high incentive to make their publications available in scientific Web repositories to increase the visibility of their research, to build up their personal reputation and improve their career options. Research institutions and funding agencies frequently rely on data from scientific Web repositories, e.g., the publication and citation counts of a researcher, to make hiring, promotion, and funding decisions.

Given that scientific Web repositories contribute to decisions that have far-reaching consequences for individuals and society, we argue that the employed data and metrics should be transparent and independently verifiable. For most scientific Web repositories today this is not the case. The procedures for selecting and ranking search results are just as opaque as the data and the metrics used to measure research performance.

Our past and current research aims at increasing the transparency of research performance metrics and enabling advanced analyses of a researcher's accomplishments. In prior studies [2, 3, 4, 5], we analyzed the ranking algorithms of Google Scholar with explicit consent given by Google Inc. For this purpose, we systematically issued queries to the service, scraped the returned result pages, and investigated which factors affect the ranking of the result set.

For the project that inspired the paper at hand, we use Google Scholar data to provide customizable analysis options of individual publication performance. The goal is to support stakeholders such as search committees tasked with finding the most suitable candidate for a position. For this purpose, we provide a tool to compare the publication performance for a set of researchers who are active in the same research field. The tool consists of a scraper to obtain the necessary data and a web-based user interface. Figure 1 gives an overview of the tool. The code is available on Bitbucket.

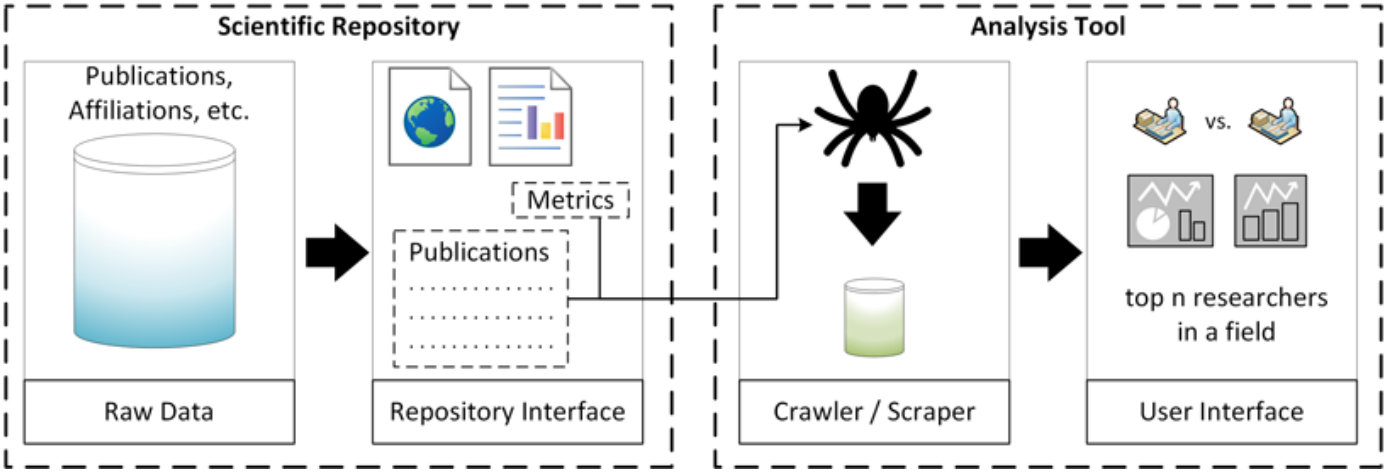

Figure 1: Overview of the analysis tool

The remainder of this paper summarizes the typical challenges to scraping data from scientific Web repositories that we encountered during our research. We also discuss strategies and provide exemplary code to address some of these challenges as part of our analysis tool.

We emphasize that we respect the need and the legal right of repository owners to protect their content from undue use. Therefore, we advise anyone to refrain from extracting content without the consent of the repository owner. In our experience, consent is often given if the research goals are clearly explained and the project is of non-commercial nature.

Even when permission for data use has been given, scraping can still be necessary. For example, before conducting our studies on Google Scholar, we discussed our objectives with the Google Scholar team and asked for permission to access the service or its data at scale. Google Scholar supported our research, but granting access to the data, or turning off the access barriers built into the service to facilitate scraping, would have required too much development and organizational effort to be viable. Although Google Scholar could not actively support our studies, the responsible manager indicated that they would not mind if we accessed the service at scale as long as we use the data for research purposes only, do not interfere with the quality of the service, and share our results with the Google Scholar team.

2 Related Work

Web Scraping, i.e. the automated and targeted extraction of data, is a traditional technique to retrieve Web content at scale. A multitude of frameworks and Application Programming Interfaces to develop customized scrapers, as well as configurable ready-to-use scraping tools exist. Glez-Peña *et al.* [9] and Haddaway [11] present comprehensive overviews of frameworks and tools for different extraction tasks.

However, there are fewer targeted scrapers for mining scientific publications and bibliographic data. The *ContentMine* framework [14] and the *AMiner* platform [15] are two prominent projects that address this and related use cases. *ContentMine* is a framework that allows developing customized scrapers and other content mining components. *AMiner* is a comprehensive social network for researchers that gathers and integrates heterogeneous data from multiple Web sources. While *AMiner* employs sophisticated mining functionality and offers valuable analysis features, the application is monolithic from the user's perspective. The platform provides no support for individually customized content mining.

The goal of our paper is to summarize typical obstacles to scraping content from scientific Web repositories and present possible solutions. The paper will provide developers with a concise overview of common barriers for large-scale access of scientific Web repositories and how they might be addressed. We used Python's *Scrapy* library to develop the scrapers for our demonstration tool.

3 Challenges for Scraping

We distinguish three categories of obstacles that we commonly encountered when scraping scientific Web repositories: *size-limitations of the result sets*, *dynamic contents*, and *access barriers*. We briefly characterize the three obstacle categories in the following subsections before we present strategies to address the obstacles in Section 4.

3.1 Size Limitations of the Result Sets

3.1.1 Static Caps

The search functionality of scientific Web repositories often returns a static maximum number of search results per query. This static cap is appropriate for visitors performing interactive ranked item retrieval, since an effective ranked retrieval system should typically return relevant results within the first 5-10 ranks, otherwise the user will likely refine the query.

3.1.2 Pagination

Aside from static caps for result sets, scientific Web repositories typically use pagination for listing the items in the result set. When employing classic pagination, the interface divides the list of items into multiple pages, each showing a fixed number of items.

The choice of the pagination parameters can strongly influence a scraper's efficiency. Some repositories use pagination parameters that enable Web crawlers to easily access their content. For example, if pages identified by increasing and consecutive integer values are used to paginate the result set, a Web crawler can easily iterate over the result set. If item sub-lists are defined by additional integer parameters, such as "start", "end" and "limit", Web crawlers can even retrieve items at arbitrary positions by sending a single request.

3.2 Dynamic Contents

Scientific Web repositories, and Web pages in general, increasingly rely on content that is dynamically loaded using JavaScript. In this approach, JavaScript is employed to manipulate the currently loaded page at runtime instead of loading a different page.

For example, in contrast to the classic approach described in Section 3.1.2, pagination can also be realized by employing JavaScript. Instead of using multiple pages, which can be accessed by sending distinct HTTP requests, the interface could dynamically update a single content container with additional content items as the user scrolls to the end of the container. Dynamically loaded content can represent a major challenge to scrapers both in terms of complexity and runtime performance.

3.3 Access Barriers

3.3.1 Obfuscated URL Parameters

The use of non-sequential URL parameters, e.g., for handling pagination, is a typical measure scientific Web repositories employ to impede content mining. Page identifiers consisting of randomly selected characters are one example of such obfuscated parameters. Using identifiers that consist of 12 characters, which is a typical length we observed, and allowing lower case letters, upper case letters, digits, underscore, and dash would result in approximately 39×10^{20} possible pages. Even for the largest scientific Web repositories, the vast majority of these identifiers would not link to actual pages in that case. Hence, crawling pages in such a scenario would require finding alternative ways to obtain the identifiers.

Another approach to impede the deduction of succeeding pages in a paginated interface is to introduce a dependency between the URL of the succeeding page to the content of the current page. One example we observed is the use of "after_item" parameters as part of the URLs of paginated views. The "after_item" parameter typically corresponds to the identifier of a content element in the current page and has to be specified to access the succeeding page. In such a case, the content of each page must be parsed to deduce the following page. This requirement can significantly decrease the efficiency of a scraper. Jumping to arbitrary sub-lists using one request becomes impossible.

3.3.2 Robot Detection and Reverse Turing Tests

Many scientific Web repositories implement methods to detect and stop robots. Significant research effort has been invested in developing such detection approaches, which commonly rely on machine learning. For example, Balla *et al.* applied decision trees based on ID3 [1], Bomhardt *et al.* employed neural networks [6], and Lu *et al.* investigated Hidden Markov Models for this purpose [12]. Training sets commonly rely on vectors, the dimensions of which typically represent requested resource types, resource proportions, time intervals, and time deviations between requests [8]. Dikaiakos *et al.* survey additional possibilities, such as resource-size distributions and indices that quantify the popularity of a resource to detect abnormal automated access attempts [7].

If the detection methods suspect automated access attempts, they may trigger reverse Turing tests, which demand specific user interactions to re-enable access to the Web resource. A popular form of reverse Turing tests are CAPTCHAs. Static text CAPTCHAs, which show obscured and distorted textual content, are widely used across the Web. Audio or video CAPTCHAs are less frequently employed, but represent a larger obstacle for automated access than static CAPTCHAs [13]. Image classification or sequencing tasks are additional possibilities for designing reverse Turing tests. The tests can also be as simple as Google's "reCAPTCHA" [10], which only requires clicking onto a checkbox.

4 Scraping Strategies

In the following, we describe possible strategies to address the challenges to scraping presented in Section 3. Our demonstration tool includes exemplary implementations of individual strategies in Python. For technical details, please consult the documentation, which is included in the code repository. We would like to emphasize again that the described measures should only be taken if the content owner has given consent for automated access.

4.1 Dealing with Fixed-Size Result Sets

Designing an approach to effectively traverse a scientific Web repository if fixed result set sizes are imposed requires a customized, in-depth analysis of the repository. Before writing any code, we recommend inspecting all functions offered to retrieve any data items from the repository. This includes observing upper boundaries for item counts and investigating all URL parameters one can specify as part of requests. We recommend investigating whether the imposed restrictions vary for different item types. For example, the

maximum number of results for an author search could be lower than the maximum number of results in a search for publications. Developers need to find the best setting for the specific scraping scenario.

In our experience, developing multiple scrapers for different item types and using the results of one scraper as input to another scraper can be a successful approach. For example, one could scrape a list of frequently occurring last names from a suitable Web resource and use the names on the list to perform author searches in a scientific Web repository. From the result sets of the author searches one might be able to extract the affiliation, co-authors, or fields of research for individual authors. If supported by the repository, searches using this information may yield additional authors that the initial name-based scraper could not retrieve.

4.2 Accessing Dynamic Content

Automated browsing is often an effective, although not very efficient, approach to accessing dynamic Web contents. Tools and frameworks for automated browsing simulate a regular page visit. A popular browser automation framework that supports all modern browsers is Selenium. The WebDrivers package, which is part of the framework, enables browser automation for many programming languages, including Python, C#, and Java. However, simulating page visits decreases a crawler's performance and might not be fully reliable in each scenario.

To develop an alternative approach for accessing dynamic content, the developer needs to investigate how content is added to the current page. Two major methods exist to append content to an HTML page. The first method uses hidden containers, which are revealed using JavaScript if certain frontend events have been triggered. In such cases, the HTML code typically contains the full content after the page has loaded. Therefore, this type of dynamic content is no obstacle for scrapers. The second approach dynamically requests content from the server's backend if certain events are triggered in the frontend. In this scenario, the content is not immediately available to a scraper. To determine which of the two methods is used, a developer can analyze the JavaScript that is executed after certain actions are performed in the frontend.

In our experience, analyzing network logs, e.g., available in the consoles of Google Chrome or Mozilla Firefox, can be an effective approach to accessing dynamic content. In the logs, the developer may be able to identify URLs and parameters that are processed during POST- and GET-Requests. Depending on the server's configuration, a Web crawler can call the identified URLs and process the answer. When investigating this approach, one should keep potential cross-origin exceptions in mind and reflect on ways to solve this issue (e.g. JSONP, middle layers, etc.).

4.3 Avoiding Accessing Barriers

To impede being classified as a robot and subjected to access restrictions, a scraper can implement the following strategies.

Altering the content of request headers: A scraper could pretend to use different user agents for different requests. For instance, user agent settings can be randomly chosen from a pool of settings.

Choosing suitable cookie settings: Depending on the scraping scenario, a developer may want to enable or disable cookies for the scraper. Enabling cookies can be beneficial in letting a scraper appear more human-like. However, developers may want to disable cookies if they suspect that cookies are used by robot detection methods. The decision whether to enable or disable cookies inherently requires a careful investigation of the targeted platform.

Varying the sequence of requested URLs: To avoid unnatural request patterns, such as consecutive iteration over 100 paginated views, the URLs of pages to be scraped can be pooled and then randomly taken from the pool for processing. If possible, URLs for different resource types should be mixed. As an alternative to randomly selecting URLs, following the out-links of pages to a predefined depth can aid in letting the scraper appear more human-like. If the scenario warrants the effort, machine learning can be employed to learn and reproduce typical navigation patterns of human Web surfers.

Varying the time intervals between requests: In addition to the sequence of requested URLs, the time interval between sending requests can be varied. The time intervals could be randomly chosen or be derived from statistical data of user interactions. Developers need to balance the tradeoff between delaying requests and a scraper's efficiency.

Changing the IP address: A developer may change the IP address used to send requests, e.g., by using services such as Tor. The change of the IP address can take place routinely, e.g., at constant or randomly chosen time intervals, or when the queried server returns errors, which typically signal that the scraper was blocked.

As part of our demonstration tool, we vary user agent settings, randomize the entities to scrape, disable cookies and change IPs. We do not vary time intervals between requests, since our scraper is sufficiently effective for our use case and delaying requests would increase processing time.

4.4 Breaking Accessing Barriers

For completeness, we like to mention that approaches to break reverse Turing tests exist. Approaches to solve static text CAPTCHAs commonly segment the CAPTCHA and then apply machine learning to recognize tokens within the individual segments. Figure 2 illustrates this generic process for solving static text CAPTCHAs. Furthermore, several crowdsourcing platforms exist that employ human intelligence to solve image classification or sequencing tasks, audio and video CAPTCHAs, or "reCAPTCHAs". However, we do not advocate the use of such solutions, since their very application indicates that operations are performed for which no consent has been given. Therefore, we do not cover these technologies in detail or employ them in our demonstration tool.

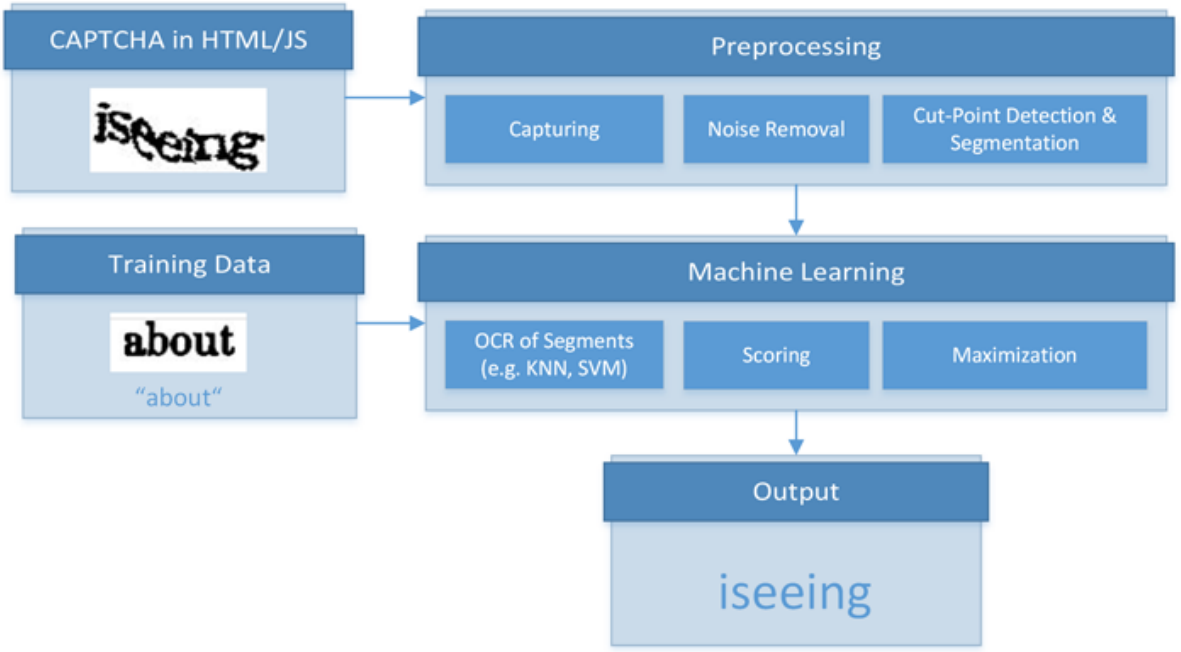

Figure 2: Generic process for solving static text CAPTCHAs

5 Conclusion

We presented our motivation for the automated extraction of information from scientific Web repositories, described typical challenges developers face when scraping such repositories, and discussed possible strategies to address these challenges.

We distinguished three categories of challenges to scraping: size limitations of the result sets, dynamic content, and access barriers. Size limitations of the result sets can often be addressed by carefully analyzing the website's structure and techniques employed to reduce server load. Accessing dynamic content commonly requires the use of headless Web browsers running full JavaScript engines, which imposes a large performance overhead onto the system and drastically reduces the crawling speed. Avoiding access barriers is of variable difficulty. Switching IP addresses is an easy solution but usually involves using Tor. This solution puts a significant load on the Tor network and causes many harmless nodes to be flagged as bots, which impacts private browsing for human users.

We demonstrated some of the strategies as part of an open source tool (MIT license) that allows research performance comparisons based on Google Scholar data. The code and the documentation is available on Bitbucket.

References

[1] A. Balla, A. Stassopoulou, and M. D. Dikaiakos. Real-time Web Crawler Detection. In *Proc. 18th Int. Conf. on Telecommunications (ICT)*, pages 428-432, 2011. http://doi.org/10.1109/CTS.2011.5898963

[2] J. Beel and B. Gipp. Google Scholar's Ranking Algorithm: An Introductory Overview. In *Proc. 12th Int. Conf. on Scientometrics and Informetrics (ISSI)*, volume 1, pages 230-241, 2009. http://doi.org/10.1109/RCIS.2009.5089308

[3] J. Beel and B. Gipp. Google Scholar's Ranking Algorithm: The Impact of Articles' Age (An Empirical Study). In *Proc. 6th Int. Conf. on Information Technology: New Generations (ITNG)*, pages 160-164, 2009. http://doi.org/10.1109/ITNG.2009.317

[4] J. Beel and B. Gipp. Google Scholar's Ranking Algorithm: The Impact of Citation Counts (An Empirical Study). In *Proc. 3rd IEEE Int. Conf. on Research Challenges in Information Science (RCIS)*, pages 439-446, 2009. http://doi.org/10.1109/RCIS.2009.5089308

[5] J. Beel, B. Gipp, and E. Wilde. Academic Search Engine Optimization (ASEO): Optimizing Scholarly Literature for Google Scholar and Co. *J. of Scholarly Publishing*, 41 (2): 176-190, 2010. http://doi.org/10.3138/jsp.41.2.176

[6] C. Bomhardt, W. Gaul, and L. Schmidt Thieme. Web Robot Detection — Preprocessing Web Logfiles for Robot Detection. In *New Developments in Classification and Data Analysis: Proc. of the Meeting of the Classification and Data Analysis Group of the Italian Statistical Society*, pages 113-124. Springer, 2005. http://doi.org/10.1007/3-540-27373-5_14

[7] M. Dikaiakos, A. Stassopoulou, and L. Papageorgiou. *Characterizing Crawler Behavior from Web Server Access Logs*, pages 369-378. Springer, 2003. http://doi.org/10.1007/978-3-540-45229-4_36

[8] D. Doran and S. S. Gokhale. Web Robot Detection Techniques: Overview and Limitations. *Data Mining and Knowledge*

Discovery, 22 (1): 183-210, 2011. http://doi.org/10.1007/s10618-010-0180-z

[9] D. Glez-Peña, A. Lourenço, H. López Fernández, M. Reboiro Jato, and F. Fdez Riverola. Web Scraping Technologies in an API World. *Briefings in Bioinformatics*, 15 (5): 788-797, 2014. http://doi.org/10.1093/bib/bbt026

[10] Google Inc. Google reCAPTCHA: Easy on Humans, Hard on Bots, 2016.

[11] N. R. Haddaway. The Use of Web-scraping Software in Searching for Grey Literature. *Grey Journal (TGJ)*, 11 (3), 2015.

[12] W. z. Lu and S. z. Yu. Web Robot Detection Based on Hidden Markov Model. In *Proc. Int. Conf. on Communications, Circuits and Systems*, volume 3, pages 1806-1810, 2006. http://doi.org/10.1109/ICCCAS.2006.285024

[13] A. Raj, A. Jain, T. Pahwa, and A. Jain. Picture Captchas with Sequencing: Their Types and Analysis. *Int. J. of Digital Society*, 1 (3): 208-220, 2010. http://doi.org/10.20533/ijds.2040.2570.2010.0026

[14] R. Smith Unna and P. Murray Rust. The ContentMine Scraping Stack: Literature-scale Content Mining with Community-maintained Collections of Declarative Scrapers. *D-Lib Magazine*, 20 (11): 12, 2014. http://doi.org/10.1045/november14-smith-unna

[15] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. ArnetMiner: Extraction and Mining of Academic Social Networks. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 990-998, 2008. http://doi.org/10.1145/1401890.1402008

About the Authors

Philipp Meschenmoser is currently completing a master's degree in Computer and Information Science at the University of Konstanz. His research interests lie in natural language processing and information visualization. His main interest is analyzing large-scale information spaces to develop visual representations of those spaces that are both effective and efficient for a user. Supervised by Prof. Daniel A. Keim, he has worked on projects related to visualizing textual content, movement data, and traffic data.

Norman Meuschke is a doctoral researcher in Computer and Information Science at the University of Konstanz, where he is conducting research on methods for semantic similarity analysis and information retrieval in large-scale datasets. Beyond his core research, he is interested in applied data science and knowledge management challenges. Before joining the University of Konstanz, he completed research stays at the University of California, Berkeley and the National Institute of Informatics in Tokyo. More information can be found on his web page.

Manuel Hotz is currently completing a master's degree in Computer and Information Science at the University of Konstanz. His research interests include efficient data storage and processing, information retrieval of structured and unstructured data, and information visualization techniques. Aside from the framework presented in this paper, he contributed to several other open source projects, e.g., the relational database management system for research and teaching Minibase for Java and the streaming framework Niagarino.

Bela Gipp serves as Professor of Information Science at the University of Konstanz. Previously, he held post-doctoral research assignments at the University of California, Berkeley and the National Institute of Informatics in Tokyo. His research interests lie in data science, information retrieval and information visualization, knowledge management systems and web technologies. His recent work focuses on developing semantic methods for analyzing links, citations and other language independent characteristics to improve recommender systems. He is also researching the use of cryptocurrencies for non-payment use cases, such as trusted timestamping, and the use of mind maps for various knowledge management tasks. More information can be found on his web page.