A Pipeline-oriented Processing Approach to Continuous and Long-term Web Scraping

Stefan Huber[a], Fabio Knoll[b] and Mario Döller[c]
University of Applied Sciences Kufstein, Andreas Hofer-Straße 7, 6330 Kufstein, Austria

Keywords: Web Scraping, Data Pipelines, Fault-tolerant Execution.

Abstract: Web scraping is a widely-used technique to extract unstructured data from different websites and transform it into a unified and structured form. Due to the nature of the WWW, long-term and continuous web scraping is a volatile and error-prone endeavor. The setup of a reliable extraction procedure comes along with various challenges. In this paper, a system design and implementation for a pipeline-oriented approach to web scraping is proposed. The main goal of the proposal is to establish a fault-tolerant execution of web scraping tasks with proper error handling strategies set in place. As errors are prevalent in web scraping, logging and error replication procedures are part of the processing pipeline. These mechanisms allow for effectively adapting web scraper implementations to evolving website targets. An implementation of the system was evaluated in a real-world case study, where thousands of web pages were scraped and processed on a daily basis. The results indicated that the system allows for effectively operating reliable and long-term web scraping endeavors.

1 INTRODUCTION

Knowledge workers typically spend a substantial part of their working time with searching and processing information published on the WWW. As new information is published frequently, clerks have to regularly monitor all relevant sources and extract important information as daily routines. These routines are cumbersome, time-consuming and potentially decrease work productivity. Furthermore, the increasing amount of data is overwhelming for manual processing and important information might be missed.

A widely-used technique to automatically extract unstructured data from the WWW and transform it into a unified and structured form is web scraping (Saurkar et al., 2018). A web scraper is a program, which is specifically targeted to extract data from certain web pages, by making use of the HTML-structure, CSS-selectors, XPATH-expressions and regular expressions (Khder, 2021). Web scrapers are commonly built by employing specific tools, such as Scrapy or BeatifulSoup (Mitchell, 2018). The drawback of these approaches is, its static behavior in case of changes in the structure of targeted web pages.

As websites are not primarily built to being scraped, implementing a web scraper comes along with several challenges and is considered an iterative and lengthy process (Landers et al., 2016). Typically, aspects such as pagination, dynamic content or access barriers are common obstacles (Meschenmoser et al., 2016) which need to be handled. Especially, due to networking issues or adaptations in the website structure, failures in the extraction process, should be expected in continuous and long-term web scraping.

State of the art pipeline-oriented data processing architectures, such as stream processing (Noghabi et al., 2017; Lin et al., 2016) or ETL-pipelines (Simitsis et al., 2010) offer desirable execution attributes, which would enhance the reliability of web scraping tasks. Inspired by these findings, in this paper, a pipeline-oriented system design for web scraping is presented and also evaluated on a real-world use case.

The paper start with an overview of challenges in web scraping in Section 2. Section 3 derives requirements for a reliable web scraping system. Section 4 describes the proposed software design and Section 5 introduces a case study for evaluating the proposed architecture in a real-world use case. In Section 6 the approach is reflected in the form of a discussion. The paper is ended by an overview of related work in Section 7 and conclusions in Section 8.

[a] https://orcid.org/0000-0001-7229-9740
[b] https://orcid.org/0000-0002-9368-8056
[c] https://orcid.org/0000-0002-9716-564X

Huber, S., Knoll, F. and Döller, M.
A Pipeline-oriented Processing Approach to Continuous and Long-term Web Scraping.
DOI: 10.5220/0011275100003266
In *Proceedings of the 17th International Conference on Software Technologies (ICSOFT 2022)*, pages 441-448
ISBN: 978-989-758-588-3; ISSN: 2184-2833

2 CHALLENGES FOR RELIABLE WEB SCRAPING

A web scraper is typically targeted to extract data from a specific web page structure (Dongo et al., 2020). Therefore, a sample set of web pages, from a distinct source, are selected to form a test dataset (Landers et al., 2016) for the implementation. For the extraction procedure it is required, that prospective web pages, generally conform to the structure of the web pages in the test dataset.

Studies on the evolution of the WWW have shown that from a content perspective web pages undergo changes at a rather rapid turnover rate (Ntoulas et al., 2004). Additionally, from a structural perspective, the rate of change, when considering the markup (Fetterly et al., 2004; Adar et al., 2009), is also rather frequent. Therefore, when considering long-term and continuous web scraping endeavors, the real or current state of a web page may not reflect the structure in the used testbed dataset.

Deviations of the current web page structure with respect to the testbed structure could lead to failures in the extraction process. These type of failures we termed *representational issues*. In addition, as web scrapers are operating over the Internet, failures due to *networking issues*, need to be considered for a reliable extraction procedure. Table 1 summarizes the types of issues and the corresponding mitigation strategies. The rest of this section describes the issues in more detail.

We classified structural deviations from the expectation as either major or minor. A major structural change would typically result after a redesign of the website. Such changes are rather seldom and most likely would require a complete rewrite of the web scraper. Minor structural changes on the other hand could happen more often, especially on popular websites. These changes result from adding new features (e.g., user comments) to a website or by restructured parts of a website (e.g., based on the results of A/B testing). As the basic structure of the website stays the same, such changes would only require an adaptation of the web scraper.

The information presented inside web pages is typically created by humans. Depending on the validation capabilities of the input systems, some entry fields might be omitted (e.g., a missing e-mail address) or provided with idiosyncratic formatting (e.g., date or phone number formatting) by content editors. This may lead to missing or ill-formatted data. A web scraper could integrate a certain tolerance for parsing ill-formatted inputs. For missing data, also default values could be used. As unpredictable formatting issues could arise over time, a continuous adaptation of the scraper might be necessary. In case of a lower importance of a specific data input, an ill-formatted value might even be ignored by the web scraper.

Finally, as web scrapers are operating over the Internet various network issues may occur. For instance, these can be internal or external network outages, temporary unavailability, SSL connection errors, blockages of various causes, website maintenance or network timeouts due to congestion. A website could even be moved to another location or introduce an authentication mechanism. On an abstract level, we distinguished networking issues into temporary and non-temporary. The former type could potentially be resolved either by a delayed retry or by a high timeout threshold (Liu and Menczer, 2011). The later type would require adaptation of the web scraper implementation.

3 REQUIREMENTS FOR RELIABLE WEB SCRAPING

In consideration of the challenges introduced in the previous section, several requirements have been derived, which must be fulfilled in order to achieve a reliable web scraping system.

3.1 Fault-tolerance

The system should be tolerant to errors, which occur within the execution of a web scraping task. An error must not lead to the termination of the entire program flow, but only skip the web scraping task in which the error occurred. For instance, a formatting issue on a single web page that leads to a parsing error should not affect the scraping procedure of the whole website.

3.2 Error-handling Strategies

For a web scraping system, which is daily operating on thousands of web pages, errors should be expected. The system should be able to identify and classify errors. Based on the error type, different error-handling or mitigation strategies could be executed. E.g., a temporary networking issue, due to a maintenance downtime of a website, should be detectable as a type of error by the system. A correctly configured webserver returns an HTTP 503 status code in such a situation. An error-handling strategy, in such a case, could be a retry of the web scraping task after a certain timeout.

Table 1: Types of web scraping issues and mitigation strategies.

web scraping issue		**mitigation strategy**
representational issue	major structural issue	web scraper rewrite
	minor structural issue	web scraper adaptation
	ill-formatting issue	tolerance, ignore or adapt
networking issue	temporal	delayed retry or timeout threshold adaptation
	non-temporal	web scraper adaptation

3.3 Monitoring and Logging

The system should include an effective monitoring procedure, which would on the one hand allow to check the successfully executed web scraping tasks and on the other hand logs tasks which resulted in an error. Moreover, in the case of an error a corresponding error context should be stored. This data would consist of the web page, if it was accessible, and corresponding contextual meta data, such as the access time, HTTP headers and status codes. The error context, which produced the error, should be stored in a form that can be used as input to a test case for the specific web scraper implementation. This would allow the maintainers of the system to replicate the error and adjust the web scraping task. Also, an analysis of the error could result in adapting the error-handling strategies of the system. A new type of error could result in the identification of a new mitigation strategy. As a whole, the monitoring data should be usable to continuously improve the system and adapt to changes in the web scraping targets.

4 SYSTEM DESIGN

The system design was derived from the previously introduced requirements and is depicted in Fig. 1. This section describes design principles and components of the system.

4.1 Job Granularity

A web scraping task for an entire website is divided into several fine-granular job implementations. The granularity of a job implementation is guided by the ability of error detection and maintainability. An analogy could be derived from end-to-end testing of web-applications. The page object design pattern (Leotta et al., 2013) is an empirically validated concept for maintainable end-to-end test suites, as such test suites face similar issues as web scrapers regarding structural changes of websites.

A popular and frequently used design pattern in web development is the master-detail pattern (Molina et al., 2002). For a website that implements this pattern, one job could be developed for scraping the links to the detail pages from the master page and another job could be developed for scraping the data from the individual detail pages. Additional complexities, such as downloading and extracting data from linked and related documents, could be also implemented by encapsulating each sub-task in its own corresponding job implementation.

A fine-granular division of an entire web scraping task into jobs has several advantages. It ensures precise root cause identification of an error, since it is evident in which part of the task the error has occurred. It enables the desired property of fault-tolerance. In the case of a scraping error on a single web page (e.g., due to a format parsing issue), only a single job fails and not the entire scraping task for the website. Also, in consideration of the development workflow of a large web scraping endeavor, each job can be developed and tested in isolation.

4.2 Event-driven Workflow

Jobs are implemented loosely coupled with no direct dependencies to other jobs. A job triggers a generic event after its execution. The corresponding processing result of the job is attached as payload to the event and processed by the Pipeline Flow Manager component of the system.

This components couples the different jobs as an event-driven workflow. Based on the job events, corresponding successor jobs are selected and started. The selection of successor jobs is based on the payload of the preceding job.

The flow logic of the jobs is defined based on the event-condition-action (ECA) metaphor (Kappel et al., 2000). This approach allows for adapting and extending the execution pipeline without modifying existing job implementations.

4.3 Parallelization

The job execution is handled in the form of a FIFO job queue. Worker processes on the corresponding machine take open jobs from the queue for process-

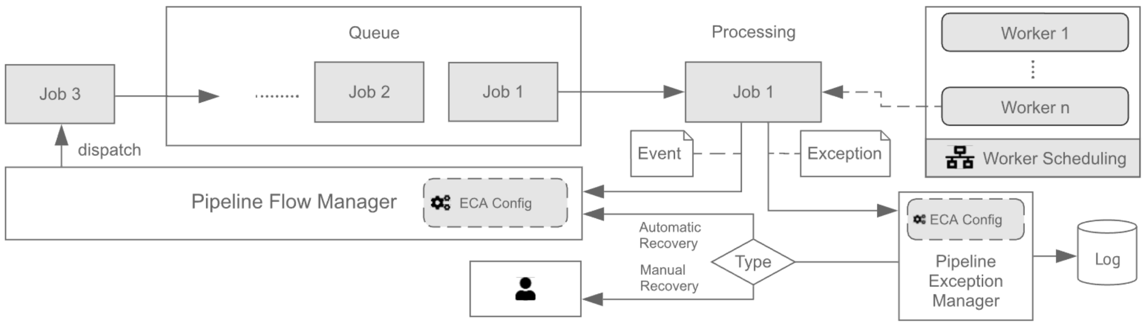

Figure 1: Overview of the system design.

ing. As jobs are loosely-coupled, worker processes can operate in parallel. The processing time of the entire pipeline can be regulated by the amount of workers in place. Depending on the workload of the web scraping endeavor, jobs can be distributed on different machines.

4.4 Error Handling

To keep the job implementations lightweight and to avoid the repetition of implementing similar error handling procedures in different jobs, the entire error handling logic is organized in the separate Pipeline Exception Manager component. An error inside a web scraping job should raise an exception based on a defined exception hierarchy. The handling of a specific exception happens then by an error handling strategy selected in the Pipeline Exception Manager.

Similar to the Pipeline Flow Manager, a configuration in the form of ECA-rules allows the system to select a corresponding error handling strategy. Depending on the type of error, either an automatic or a manual recovery can be carried out. An automatic recovery will create a new job corresponding to the exception and will dispatch this job via the Pipeline Flow Manager.

A generic form of a error handling strategy is a delayed retry of the same job, which raised the exception. This strategy could be employed to handle temporary issues. Also, more sophisticated strategies are conceivable, which e.g. adapt the inputs to a certain job or select an alternative job, which might be able to handle the task more appropriately.

For more severe issues, which cannot be resolved autonomously, the system will notify a corresponding user. Serious errors can often only be handled by system administrators or developers, whereas more harmless errors can also be solved by clerks working with the system. Ideally, there is a defined hierarchy in the organization for handling errors. For the case study in Section 5.1, such an organizational setting is described.

As additional metadata to a raised exception a `PipelineExceptionContext` is attached. This context is specific for each exception type and provides data for debugging and reproducing the error. E.g., for a specific scraping job which produced an error, the entire web page is stored as part of the error context. The stored website can be used to generate input for corresponding test cases to adapt or extend the web scraper job implementation.

5 EVALUATION

The proposed system was evaluated in a real-world case study (Section 5.1). Implementation details (Section 5.2) and results on the error handling (Section 5.3) are described in this section.

5.1 Case Study

The case study was developed with a company operating in the construction business within the EU. The company is regularly participating in bidding procedures for new projects. Within the EU, public institutions must select suppliers in the form of an open competitive bidding procedure if the procurement value exceeds a certain threshold.

Particularly in the construction sector, for smaller projects, municipalities or companies also use a bidding procedures to select appropriate suppliers. Therefore, tenders are published by companies and governments on different platforms and in different formats on the WWW to advertise for potential bidders.

A common platform for the publication of tenders is TED[1]. However, many states, cities or municipalities offer their own portals for publishing such tenders and within the various platforms tenders are published in highly heterogeneous formats.

[1] https://ted.europa.eu

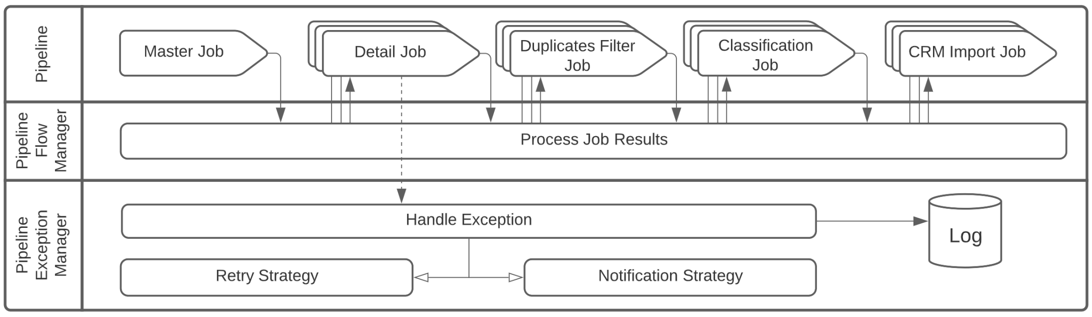

Figure 2: Overview of job processing pipeline of the case study.

Clerks working in the sales department of the company screen sources on the WWW as a daily routine. From the search interfaces of the portals, they collect relevant tenders for the different departments in the company. The corresponding data is extracted and entered into a company internal information system. This is a rather cumbersome and time-consuming endeavor for the clerks.

To automatize the whole process of collecting, extracting, classifying and storing tenders from the WWW, an implementation of the proposed system design was set in place. An overview of the respective processing pipeline is given in Fig. 2. In addition to the web scraping jobs, also two machine learning based jobs for doublet detection and tender classification are part of the pipeline, this paper only focuses on web scraping aspects. A terminal job, which exports positively classified tenders into the company's information system, is also part of the pipeline.

Within the case study, tenders from 16 different web portals are continuously scraped. This results in up to 2000 web pages which are scraped and processed on a daily basis.

5.2 Implementation Details

The pipeline for the case study was implemented in the programming language PHP using the Laravel[2] web framework. The framework offers a lightweight and flexible queue and job implementation, which facilitates the basis for the proposed pipeline-oriented system design.

Among the websites to be scraped for the case study, many implemented the master-detail design pattern, which resulted in consistent pipeline structures. For each master page, a corresponding master job was developed. The master job extracts primarily the links to the detail pages. Similarly, a separate job was developed for the detail pages, which then extracts detailed data about the tenders.

[2]https://laravel.com

The structure of the various web pages varied widely, ranging from unstructured and rather poorly organized HTML pages to well-structured XML or JSON files. Some websites are also implemented with JavaScript rendered content and reloaded parts of the web page dynamically. Therefore such web page were rendered inside the headless web browser engine Puppeteer[3] before starting the extraction procedure.

5.3 Error Handling Strategies

Within the case study automatic and manual procedures for error handling were set in place. In addition to the actual error handling procedures, an organizational structure in the form of hierarchical roles was established. The roles *clerk*, *administrator* and *developer* were defined.

As an automatic error handling procedure a delayed retry was used for jobs, which failed due to network issues. A job is retried twice, firstly with a delay of one hour and secondly with a delay of five hours. If the second retry failed, the failure is notified to an administrator, which tries to analyze and fix the problem or escalate the issue to a developer.

Furthermore, for other types of errors, such as any kind of representational or unclassified errors, an organizational procedure is defined. On a daily basis, the exceptions for each website are counted and if the exceptions are below a defined threshold, which is configured for each website, the exceptions are notified to clerks. The clerks can check the web pages, which produced the error, and either extract the data manually or ignore the web page. The web pages are also logged and can be analyzed by the developer at a later stage.

If the exception count is above a certain threshold, an administrator is notified. A high amount of exceptions for a certain website indicates a systematic error. The administrator can analyze the problem and either trigger a manual retry or escalate the problem to the

[3]https://pptr.dev/

developer. A developer could do a deeper analysis of the problem, based on the stored error context and could integrate necessary changes to adapt the implementation.

5.4 Web Scraping Results

For demonstrating the operation of the system a typical 5 week period from June 2021 to July 2021 is documented and respective results are displayed in Fig. 3, Fig. 4 and Fig. 5. An administrator would monitor the system with similar diagrams.

The daily amount of successfully scraped web pages is displayed in Fig. 3. The largest amount of successful scrapings typically happens on Fridays. On weekends only a few tenders are published, which results in a low amount of scraped web pages. Fig. 4 gives an overview of the amount of daily scraping errors. In the following list, notable incidents are summarized:

- At the beginning of Fig. 4 a higher error count was observable. This was caused by a redesign of a target website. The web scraper needed to be largely rewritten, which took several days of iterative implementation and testing work. This resulted in a higher amount of successfully scraped web pages after the rewrite of the web scraper, as unsuccessfully scraped tenders from the past have been caught up with the updated job implementation.
- On 07/13 a new type of tender was published on one of the websites. The new type had a different formatting for the tender title on the web page. More than 20 tenders could not be successfully scraped based on this formatting change. The implementation of the web scraping job could be adapted promptly by a developer and this type of error was solved.
- For most days several errors with unsystematic causes happen. Primarily, the errors originate from ill-formatted or missing data. Such errors often fall into the unclassified error category. If there is any pattern detectable in such errors a developer could adapt the respective implementation. Also a few networking errors did arise, such as network timeouts or page not found errors during the observed period.

The distribution of the different error types is displayed in Fig. 5. The majority of errors belong to the class of representational errors. These result from inputs to web scrapers, which don't comply with the expected representation. Within the 5 week period, network errors occurred only at a rather low frequency. All errors which let a web scraper fail in an unexpected way are declared as unclassified errors. These types of errors require an investigation by the developers.

6 DISCUSSION

The proposed system was implemented and used within a real-world use case. It was demonstrated that the system is capable of coping with challenges that typically arise in continuous and long-term web scraping endeavors. Specifically, the pipeline-oriented approach and the fine-grained encapsulation of sub-tasks into jobs has many merits for the domain of web scraping.

The system design is based on the assumption that scraping errors will happen continuously, as the web scraping targets are evolving. The proposed system includes proper logging and replication facilities to enable maintainers of the system to effectively adapt web scraper implementations to changes in the web scraping target.

In addition to the technical aspects it is important to have an organizational structure defined to handle errors in the web scraping process, which cannot be solved automatically by the system. For the use case, it turned out beneficiary to also involve non-technical clerks. For low-severity and seldom issues, such as an ill-formatted date, a clerk could manually extract the data and fix the issue quickly. The requirement for such an approach is, that the exact web page which produced the error can be pinpointed to the clerk, which the system's logging facility is capable of.

Some web pages require the execution of JavaScript to be rendered properly. For web scraping this requires rendering the web page using a headless browser. A headless browser can be configured to block the download of content, which is not required for the scraping task. This lowers the overhead for the server and speeds up the scraping process and should be considered a good practice for web scraping.

The code for web scraper implementations tend to get rather complex in terms of execution flows and conditional branching. Encapsulating each type of web page into it's own job implementation turned out to be helpful to keep the code units manageable and free of unintended side-effects.

For the web scrapers which were implemented in the case study, the code complexity particularly increased, when text anchors were required for identifying the relevant parts to extract. Depending on the structure of web pages CSS-selectors or XPATH-expressions are sometimes not capable of identifying

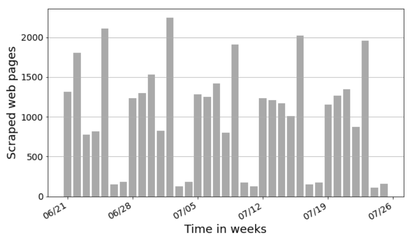

Figure 3: Scraped web pages.

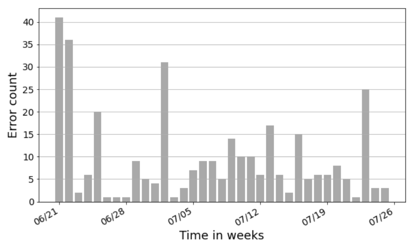

Figure 4: Error count.

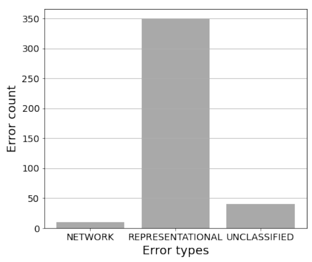

Figure 5: Error types.

the part of interest for extraction. Text anchors also turned out to be rather unreliable, especially for continuous and long-term usage. Therefore as another good practice the avoidance of text anchors should be a high priority, although for some subtle scraping tasks, text anchors are necessary.

7 RELATED WORK

Web scraping is employed in many use cases to collect and aggregate data from various sources from the WWW. In a case study for gathering biomedical data, the authors of (Glez-Peña et al., 2014) demonstrated the usefulness of web scraping to extract data from different sources. Also in the works of (Bonifacio et al., 2015) and (Kunang et al., 2018) it was demonstrated how the application of web scraping for gathering climate and weather data from different sources could increase the productivity of researchers in the field. Another example of the effectiveness of web scraping was demonstrated by the Vigi4Med (Audeh et al., 2017) tool. The tool scrapes forum discussions from more than 22 different medical websites. The resulting data is used as a complementary source to study adverse drug reactions. Although all presented approaches successfully employed web scraping in a specific context, the described implementations follow a rather ad-hoc approach. Issues which may arise in a long-term and continuous application of web scraping have not been addressed by the implementations.

Establishing a pipeline-oriented data processing system involves various challenges in the area of data cleansing, data collection and data analytics (Pervaiz et al., 2019), also challenges concerning infrastructure, organizational barriers and data quality are prevalent (Munappy et al., 2020). However, desirable properties such as fault-tolerance, auditability or error handling and mitigation strategies emerge from the application of the pipeline paradigm. Especially, in the context of long-term and reliable web scraping, such properties are inevitable.

The concept of event-driven pipeline processing has been successfully employed in the domain of computer vision workflows by the EPypes architecture (Semeniuta and Falkman, 2019). The architecture makes use of the event-driven paradigm in the context of distributed environments in order to support evolving computational graphs without sacrificing developer flexibility.

Triggerflow (López et al., 2020) is another system, which makes use of event-based control flow in order to orchestrate serverless workflows. Extensibility and flexibility are key properties named as advantages of the Triggerflow architecture.

Also in the domain of classical workflow management systems (WfMS) event-driven processing has been applied in order to adapt according to evolving needs. In (Goh et al., 2001) ECA-rules have been proposed to modify workflow executions at runtime to handle deviations in business processes.

8 CONCLUSION

In this paper, a pipeline-oriented system design to the rather error-prone and volatile endeavor of long-term and continuous web scraping was envisioned. A web scraper is typically built based on the assumption that the web page, which needs to be scraped, conforms to an expected structure. Due to the fact that websites are changing rather frequently (Ntoulas et al., 2004; Fetterly et al., 2004; Adar et al., 2009), web scraper implementations need to be adapted alongside these changes.

Our proposed system design is based on the assumption that failures in web scraping are expected. Fine-granular job implementations and proper logging for reproducing errors allow developers to effectively adapt implementations to changing scraping targets. In a real-world case study, we could demonstrate that our proposed system allows for effectively operating reliable and long-term web scraping endeavors.

REFERENCES

Adar, E., Teevan, J., Dumais, S. T., and Elsas, J. L. (2009). The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 282–291.

Audeh, B., Beigbeder, M., Zimmermann, A., Jaillon, P., and Bousquet, C. (2017). Vigi4med scraper: a framework for web forum structured data extraction and semantic representation. *PloS one*, 12(1):e0169658.

Bonifacio, C., Barchyn, T. E., Hugenholtz, C. H., and Kienzle, S. W. (2015). Ccdst: A free canadian climate data scraping tool. *Computers & Geosciences*, 75:13–16.

Dongo, I., Cadinale, Y., Aguilera, A., Martínez, F., Quintero, Y., and Barrios, S. (2020). Web scraping versus twitter api: A comparison for a credibility analysis. In *Proceedings of the 22nd International Conference on Information Integration and Web-Based Applications & Services*, iiWAS '20, page 263–273, New York, NY, USA. Association for Computing Machinery.

Fetterly, D., Manasse, M., Najork, M., and Wiener, J. L. (2004). A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237.

Glez-Peña, D., Lourenço, A., López-Fernández, H., Reboiro-Jato, M., and Fdez-Riverola, F. (2014). Web scraping technologies in an api world. *Briefings in bioinformatics*, 15(5):788–797.

Goh, A., Koh, Y.-K., and Domazet, D. S. (2001). Eca rule-based support for workflows. *Artificial intelligence in engineering*, 15(1):37–46.

Kappel, G., Rausch-Schott, S., and Retschitzegger, W. (2000). A framework for workflow management systems based on objects, rules and roles. *ACM Computing Surveys (CSUR)*, 32(1es):27–es.

Khder, M. A. (2021). Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, 13(3).

Kunang, Y. N., Purnamasari, S. D., et al. (2018). Web scraping techniques to collect weather data in south sumatera. In *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pages 385–390. IEEE.

Landers, R. N., Brusso, R. C., Cavanaugh, K. J., and Collmus, A. B. (2016). A primer on theory-driven web scraping: Automatic extraction of big data from the internet for use in psychological research. *Psychological methods*, 21(4):475.

Leotta, M., Clerissi, D., Ricca, F., and Spadaro, C. (2013). Improving test suites maintainability with the page object pattern: An industrial case study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 108–113. IEEE.

Lin, W., Qian, Z., Xu, J., Yang, S., Zhou, J., and Zhou, L. (2016). Streamscope: continuous reliable distributed processing of big data streams. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 439–453.

Liu, B. and Menczer, F. (2011). *Web Crawling*, pages 311–362. Springer Berlin Heidelberg, Berlin, Heidelberg.

López, P. G., Arjona, A., Sampé, J., Slominski, A., and Villard, L. (2020). Triggerflow: trigger-based orchestration of serverless workflows. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, pages 3–14.

Meschenmoser, P., Meuschke, N., Hotz, M., and Gipp, B. (2016). Scraping scientific web repositories: Challenges and solutions for automated content extraction. *D-Lib Magazine*, 22(9/10):15.

Mitchell, R. (2018). *Web scraping with Python: Collecting more data from the modern web.* " O'Reilly Media, Inc.".

Molina, P. J., Meliá, S., and Pastor, O. (2002). User interface conceptual patterns. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 159–172. Springer.

Munappy, A. R., Bosch, J., and Olsson, H. H. (2020). Data pipeline management in practice: Challenges and opportunities. In *International Conference on Product-Focused Software Process Improvement*, pages 168–184. Springer.

Noghabi, S. A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., and Campbell, R. H. (2017). Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645.

Ntoulas, A., Cho, J., and Olston, C. (2004). What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on World Wide Web*, pages 1–12.

Pervaiz, F., Vashistha, A., and Anderson, R. (2019). Examining the challenges in development data pipeline. In *Proceedings of the 2nd ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 13–21.

Saurkar, A. V., Pathare, K. G., and Gode, S. A. (2018). An overview on web scraping techniques and tools. *International Journal on Future Revolution in Computer Science & Communication Engineering*, 4(4):363–367.

Semeniuta, O. and Falkman, P. (2019). Epypes: a framework for building event-driven data processing pipelines. *PeerJ Computer Science*, 5:e176.

Simitsis, A., Wilkinson, K., Dayal, U., and Castellanos, M. (2010). Optimizing etl workflows for fault-tolerance. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 385–396. IEEE.