IMPLEMENTATION of DSL for WEB SCRAPING

Shail K Shah¹, Shashank Shyam Shankar², Rachana N³, Preetha S⁴
¹,²,³,⁴Information Science and Engineering, BMS College of Engineering,

Abstract— The main goal of this project is to implement a DSL for Web Scraping. A Domain Specific Language or DSL in short is a language that is created for solving a single purpose. It is a language that is used in only one domain. In our project, that domain is web scraping. Our main aim is to create a simple scripting language with easy to use syntax with many features that help the user scrape the web easily. Currently, web scraping is a tedious process. At the moment, the majority of web scraping is done by the means of modules in high level languages. This would require the user and in-depth knowledge of the high-level language as well, and thus precludes many laymen from easy web scraping. This project will provide a DSL with highly simplified syntax which does not assume any skill from the user. Thus, anyone would be able to use this language to scrape the web with no previous knowledge of the domain. This DSL has been implemented using Python and its scraping libraries. With this, many features and functionalities can be implemented in the DSL thus providing an effective tool for web scraping without compromising on simplicity.

Keywords— Domain Specific Language, Web Scraping, Python, Beautiful Soup 4

I. INTRODUCTION

A domain-specific language (DSL) is a computer language which is focused on a specific application domain. This is not like a general-purpose language, which is widely applicable across domains. Our aim is to create a DSL for web scraping. Basically, web scraping is the procedure of extracting data from websites. It refers to the different methods used to fetch information from across the Internet. Usually, this is done with certain software that clone human Web surfing to collect specified bits of information from various websites. The purpose of making a DSL for web scraping when many modules already exist for the same purpose on various languages is simple. We aim to simplify web scraping for the end user so that no high-level technical knowledge or skill is required at all. One can scrape the data required with simple syntax without any technical know-how.

Our DSL will be an interpreted language. We aim to design a scripting language which can be run one line at a time, much like Python’s IDLE or the console in Javascript and one that can be executed in the form of scripts. We will be making use of one of the many modules available to Python for web scripting, such as, beautiful soup 4. Thus, a working Domain Specific Language has been created, with a wide range of functionality for scraping. The syntax of this DSL is highly simplified, thus making it very easy for even a non-technical layman to scrape data from the web. The DSL provides various functionality to scrape various forms of data such as images, videos, audio, files, ids, classes, text and so on (using get command) , and also provides functionality to view the details of the page, and the data contained in it(using view command). Furthermore, a help command has been added for the ease of the user. With the addition of variables and lists to the DSL, the user with a bit more familiarity can empower his scraping ability by performing extensive scraping and write neat code. Question is how do entities such as variables and lists come into use whilst performing scraping? The answer is quite straight forward actually. Variables are assigned to a particular value.

DOI:10.23883/IJRTER.2020.6028.LBIFZ
These values are URLs. Likewise, lists consist of a group of URLs. When a list of URLs is passed in as a parameter, the scraping is performed by looping through the list of URLs and scraping for data. This leads to a more concise form of code.

II. REVIEW OF LITERATURE

Randy et.al [1] formulated an understanding in the form of this research paper like ours is aimed at making web scraping easier for people with different backgrounds (Programmers and non-programmers). This language helps build scraping-DSL that can meet those needs, the researcher analyzed the tools used in web scraping to determine the right tool for the scraping-DSL engine, analyzed the website structure to determine the need for the scraping-DSL and the translator application, and analyzed the website content retrieval flow to determine the required language scrapingDSL elements. Non-programmer mentioned in this research is someone who is able to run the computer through the shell and less experienced in programming.

Thanhofer-Pilisch et.al [2] formulated a research paper aimed at diving into research established in DSL evolution to brief, formulate a structure and to analyze this area of research and explore trends and issues. The study indulges into the evolution of domain specific languages help researchers and practitioners to obtain a concrete understanding of existing and patterns of the various DSL.

Rodrigues et.al [3] proposed why DSLs are used, this paper is aimed at describing why certain DSLs are less efficient, the failure of deployment of DSLs. They help analyze any DSL based on efficiency, efficacy, easiness and user satisfaction.

This paper was presented with a systematic literature review (SLR) to verify the language and approach users take to build their DSL. A special protocol is used in order to structure the execution process of the DSL obtained from the SLR. On implementing the SLR protocol, only few papers were selected to be analyzed. Based on the protocol, the efficiency came down to 3 questions and they were: Understanding usability was important in designing of DLS, to verify what approaches/techniques were used to evaluate their ease along with efficient functionality and to identify problems that can arise with any DSL. These results helped understand the don’ts and abstractions of a project as a whole. Hence this bettered the framework and design of the DSL.

Abu Kausar et.al [4] proposed together a review paper is to throw some light on the web crawling previous work. This paper also discusses research undertaken that correspond to a web crawler. The paper has a clear depiction of the various techniques involved in web scraping such as general-purpose crawling, focused crawling and distributed crawling. These techniques help build a better understanding of powerful crawling methods.

Van Deursen et.al [5] explains on key terminology, risks, benefits, example domain-specific languages, implementation and design methodologies. Through this paper we are able take a look at how languages are categorized into general and specific languages. It opens up a broad view of categorizing the various available languages into the two broad classifications and enables deeper understanding of them. While it not only classifies the languages, it brings out the evolution and development of languages over time so as to bring a comparative and transformational analysis that deepens the understanding of dealing with developing programming languages and short-comings that one might encounter during development.

Pohjonen et.al [6] explains that traditional domain specific language has a go-to mapping idea of starting with a domain idea, then moving onto design models, finally onto source code. While this process remains tedious, the paper gives us a clear understanding of how to adapt and overcome the
errors made while traditionally mapping such as solving the given problem efficiently, designing the problem with structure and coding effectively. With the negation of resource-intensive and error-prone mappings with focused aim and same level of abstraction. For any given project modelling techniques are extremely crucial and this paper caters to all the requirements.

Hoisl et.al [7] explained that unified modeling language is popular and on the demand for coders to translate models into basic abstract of code is high. This domain specific language (DSML) is defined by multiple specification artifacts and is centralized on inter-related models, describing different views on DSML. There are a lot of fallacies in the traditional approach to building a DSML. The research paper based on 84 based UML DSML designs formulate an efficient way in creation of the optimized DSML. This gives a concrete understanding of how to formulate a structured method of modelling and creating a DSL based on the specifications.

Reeve et.al [8] proposed a research similar to ours which helped us better our understanding in developing a DSL for targeted audience.

In the field of biomedical engineering, often due to the enormous amount of information, there is no optimized way of going about extracting the critical information manually. If so, the there is an enormous amount of workload and is a tedious process. The DSL targets the field of biomedical industry. It extracts the critical information that is needed for implementing important operations. A “ROGUE” system is used compare system-generated strings. A series of Bio-Chain and Frequency distance is used to improve the base approaches. After evaluation of a set of 3 different papers, the model demonstrates a critical information which states that the abstract of the paper does not necessarily pertain to the entire contents of the paper. This data is critical for our DSL in order to get the required data.

Mulo, Emmanue et.al [9] proposed work for organizations to handle process driven SOA diagrams and handle processes through orchestration of devices, the model proposed is to monitor the infrastructure and implement a DSL for specification of compliance directives and to generate compliance for monitoring code. The model is aimed at evaluate the effort and productivity of a developer who is in-charge of specifying compliance directive.

Prakshta et.al [10] explained the task of retrieving information from the web using python and Beautiful Soup which is similar to our domain specific language aimed for information retrieval from the web. However, unlike our language which is fundamentally structured on python and Beautiful soup, their work in the paper fetches only URL’s in the given web page. Data retrieval is in an unstructured manner.

Data Analysis by Web Scraping using Python [11] explains the data extraction of that which is not visible (hidden) on the web is very hard and time consuming, traditional systems were of autonomous and heterogeneous nature of hidden web content, to extract this hidden content. But since it was ineffective, to search for this kind of data, the aim was to comply with user’s needs and help build an efficient interface that has indexing, query-processing, efficient data extraction process. EloisaVargiu et.al [12] have adopted scraping techniques in the Web advertising field. websites. Its main goal is to provide suggestions / recommendations of products and services to the increasing population of Internet users. This type of scraping is aimed at multimedia scraping. They achieve the desired result by scraping ads and pop-ups.

The aim of the paper [13] is to obtain information through indexing on the web with the help of a bot or web crawler. Web crawler is mostly used to get the required content from a given URL. This
content could be ranging with the variety of documents, text, URLs, multimedia files and with the recent events of big data and data analysis, we come to observe the extraction of datasets. There are different ways of storing the obtained data after scraping such as storing it to a .csv file or to JSON readable file. This paper covers topics on how to perform powerful, clean and efficient way of dealing with web-scraping.

**III. WORKFLOW OF DSL**

The above figure 1 shows work flow of the DSL. Step 1: Start by running the main.py file in the project module. Default readable mode of the DSL code is from a command line interface. But it could be read from a script or a .txt file as well.

Step 2: The input code from the user is now read from the CLI/Script and we use the split function to bifurcate the code so as to obtain the commands.

Step 3: We now convert the list of words into tokens after pre-processing the script. If there arises an error while converting into tokens, we then raise an error and exit from the CLI. These errors could be either a syntax error or parameter type error or order of sequence of commands error.

Step 4: These tokens are classified based on the 4 basic commands along with its arguments respectively and the script is ready to be run. The 5 basic commands are “help, get, view, mem”, out of which only get and view involve the operation of scraping the internet for obtaining data. There are a couple of ways in which scraping can be performed using the “get” and “view” command. Firstly, we could use the get command by specifying default syntax and specifying the URL. Alternate approaches in performing neat and multiple data extractions are by usage of list and variables. The usage of list and variable names are brought into use using the “let” command to assign values to the lists and variables. Once values of variables and lists are assigned, there is a separate table for lists.
and variables which keep a track of the variables in use which have been allocated with a value. These tables (list and variable) are accessed by the “mem” command. The mem command would simply display the name of the variables and lists active in the lifecycle of the running of DSL. A more advanced version of the “mem” command is “mem verbose” which displays the list and variable name along with the allocation values. Once we run the script, the first operation is to fetch the URL. If the URL cannot be fetched an error would be generated. Else, the URL is fetched and is set up to perform the required scraping operation. Let’s take a look at the execution of the different commands:

- If user has opted for the option: “VIEW”, the scraper would display the list of contents of the opted data from the given URL. The data could include: text, URL, images, videos, documents. view can be performed with list of URLs or variable names.
- If the user opts for option: “GET”, the scraper would get the contents of the URL for a given type of data retrieval. This operation when performed, would allow the downloading of content from the URL. There is an additional parameter called store which lets the user to store the contents retrieved from the URL to reside in a working directory. “get” command can also be used with list and variable names, to perform multiple extraction of data.
- If the user opts for the “HELP” command, it would provide the documentation of our domain specific language.
- The interpreter now checks for variation type of the script used along with its parameters and executes the code. Once the operation is completed. If the “store” command is specified, the data is stored in the directory.

Step 5: If “store” is not specified the output is displayed if any and operation is terminated.

End of operation

III.1. HIGH-LEVEL DESIGN

The above figure 2 is the high-level design of DSL. Where the user runs the main.py python module to get the DSL running. Then the user is able to view the DSL interpreter which is a command line interface, ready for execution of the DSL code to perform web scraping. User enters the code into the CLI. User if unfamiliar with the language, can use the “help” command to view the documentation of the usage of syntax to perform the required operations. User can either perform scraping using the default syntax and parameters or usage of variables and lists empowers user to expand their horizon.
of scraping. The output/error is displayed after the information is obtained from view command. This process continues till the “exit” command is entered.

3.2 Activity Diagram

![Activity Diagram](image)

**Fig 3: Activity Diagram**

The above figure 3 is the activity diagram of DSL. The user enters the domain specific language code into the CLI. The inputted DSL code is converted into respective tokens which are matched to a respective file module. The file checks for errors. After the parameters are obtained, the variables can be created and values could be assigned to them to simplify the operation. Now based on the type of data the user has requested, the URL will be scraped to obtain that information.

**IV. ALGORITHM**

Simplified algorithm of the main.py:

Determined mode of input
If mode of input is Command Line Interface:
Read the command
Break it into tokens
Call appropriate module and send the token list and line number (1 in case of CLI)
If mode of input is Script:
Open the script file
Store all commands
For each command:
Read the command
Break it into tokens
Call appropriate module and the token list and line number

Simplified algorithm for help.py
Load the help string
Display the help string on the user’s terminal

Simplified algorithm for get.py
Obtain the token list and line number from main.py
Read the second token to determine which resource is to be scraped
Based on the token, execute the code relevant to that resource
Error checking:
Check for unidentified tokens
Check for incorrect token usage
Check if all required sub options and details are present
If an error is present:
Report error along with line number
Else:
Scrape the resource using BeautifulSoup4 or Scrapy.
Obtain the resources
Either display the output, or store the output in a file or folder based on the options

Simplified algorithm for view.py

Obtain the token list and line number from main.py
Read the second token to determine what information the user desires
Based on the token, execute the code relevant to that resource
Error checking:
Check for unidentified tokens
Check for incorrect token usage
Check if all required sub options and details are present
If an error is present:
Report error along with line number
Else:
Obtain the details the user desires using BeautifulSoup4 or Scrapy
Write the details to the Report String
Either display the Report String or write it into a Report File based on the Options

**Simplified algorithms for let command:**

User enters command
Command is broken down into subcommands and tokens
Command is sent to the main module of let
Let module will get tokens and line number from main
Let module will check for incorrect tokens
If incorrect tokens are present:
    Report error and line number indicating the incorrect token
Error checking for reserved keywords
If the variable name is invalid or a reserved keyword:
    Report error that variable name cannot be reserved keyword
Else:
Return name / value pair to main
Main obtains name / value pair and enters it into the variable table
Simplified algorithm for list command:
(list.py)

User enters command
Command is broken down into subcommands and tokens
Command is sent to the main module of list
List module will get tokens and line number from main
List module will check for incorrect token usage
If tokens are used incorrectly:
Report error and line number indicating incorrect token
Return status of -1
Error checking for reserved keywords:
If the list name is invalid or a reserved keyword:
Report error that list name cannot be a reserved keyword
Return status of -1
Else:
Create a list of url values
Create a name / list pair
Return name / list pair to main
Main obtains name / list pair and enters it into the list table

Simplified algorithm for mem command:
(mem.py)

User enters command
Command is broken down into subcommands and tokens
Main module of mem is called
Main module in mem receives token list, line number, variable table snapshot, list table snapshot
Check tokens for incorrect usage or unknown tokens
If error is found:
Report the incorrect token and the line number
Return status of -1
Tokens and scanned and checked for ‘verbose’ token
If ‘verbose’ present:
Set verbose flag to 1
Else:
Set verbose flag to 0
Tokens are checked for ‘list’ and ‘vars’ token
If ‘list’ and ‘vars’ token both present:
Report error to the user that both tags cannot be present and indicate line no
Return status of -1
If ‘list’ token is present
Set modeOfOps flag to 1
If ‘vars’ token is present
Set modeOfOps flag to 2
Else:
Set modeOfOps flag to 0
Based on modeOfOps flag, and verbose flag, output is generated and displayed to the user
Return control to main
Simplified algorithm for freevar
(present in main.py)

User enters command
Command is broken into tokens
Loop through the tokens to obtain variable names to be deleted
List of variable names to be deleted is generated
For all variable name in the list:
If variable name does not exist in the variables table:
Report error saying variable doesn’t exist.
Return status of -1
Else:
Continue
For all variable name in list
Delete entry from variable table

Simplified algorithm for freelist
(present in main.py)

User enters command
Command is broken into tokens
Loop through the tokens to obtain list names to be deleted
List of list names to be deleted is generated
For all list name in the list:
If list name does not exist in the list table:
Report error saying variable doesn’t exist.
Return status of -1
Else:
Continue
For all list name in list
Delete entry from list table

Simplified algorithms for preprocessing commands with variables and list (to implement looping construct)
(present in main.py)

User enters command
Command is broken into sub commands and tokens
If sub command is ‘get’ or ‘view’:
Command must be preprocessed
Command tokens and line number is sent to preprocessor function
Error checking to ensure presence of required ‘from’ token
If token not present:
Print invalid syntax
Return status of -1
Obtain identified token after ‘from’ token
If identifier token present in variables table:
Replace identifier with variable value
Make a relevant call to the sub command module with replaced value
If identifier token present in lists table:
Loop is detected in command
New commands are generated
For every value in the list:
Replace identifier with value
Make a relevant call to sub command module with replaced value

4.1 Implementation
Our DSL will be a scripted language entirely written in Python. Python is an easy to use language, and is interpreted itself, much like our DSL. Python has a rich set of modules for web scraping, making it an excellent choice. The various Python modules used for scrapping are:
- Requests
- Lxml
- Beautiful Soup 4
All of these modules provide excellent web scraping tools, which we will be using for our DSL.

V. RESULT

The following are few snapshots of the output of few of the DSL commands:

Fig 4: Output of the command - get text from https://en.wikipedia.org/wiki/Wikipedia
In figure 4, all the text from the web page pointed to by the given URL is obtained and displayed on the user’s terminal.

Fig 5: Output of the command - view images from https://en.wikipedia.org/wiki/Wikipedia
In figure 5, all the details of the images on a particular web page, pointed to by the given URL is displayed on the user’s terminal. This includes a list of the filename, followed by the URL of that image file.
In the above figure 6 image, all the images on the page pointed to the particular URL provided by the user are found and downloaded and stored to the folder ‘dl_images’ as requested by the user.

In the above figure 7, we can see the following:

1. A url named ‘indiaUrl’ was created and assigned a url value
2. A url named ‘sAsiaUrl’ was create and assigned a url value
3. Mem command with ‘vars’ option is used to view active variables
4. Mem command with ‘verbose’ option are used to view values as well

In figure 8 we can see the following:

1. A list named l1 is created and assigned 3 urls
2. We use mem command with ‘list’ option to see the currently active lists
3. We use mem command with verbose option to see the value of the lists as well
4. This shows list has been created and stored in the memory
In the above figure 9, we can see that we have created three variables and two lists. When we execute the ‘mem’ command with no options set, we can see that it displays all the active variables and list residing in the memory.

The above figure 10 shows the mem command with verbose option. It displays all the active variables and lists, and also displays the values stored in the memory of these variables, thus displaying all the details of the memory.

In the above figure 11 you can see that a list of URLs of Wikipedia pages of two countries has been created and stored in a list named ‘countries. Now, we use a regular view audios command on the list. On executing this command, the command gets executed for all the URLs in the list, and displays the result to the user.
Fig 12: Result of using get audios all from countries store audio_files/

Figure 12 shows that if we use get audios all over the list countries, we can see that the audio files from both the urls have been obtained and stored in the required folder. Thus, it is possible to execute multiple commands without having to write it down explicitly, with the help of looping over a list (using preprocessing).

Fig 13: Deleting variables using freevar command

The above figure 13 show how variables can be deleted using the freevar command. At first, three variables are created. Then one variable is deleted, and then the memory is examined. Then, the remaining two variables are deleted and the memory is examined again.

The above screenshot shows two ways one can use the freevar command.
1. You can individually delete one variable at a time
2. You can delete multiple variables using the freevar command

Fig 14: Deleting lists using freelist command
The above figure 14 show how lists can be deleted using the freelist command. At first, three lists are created. Then one list is deleted, and then the memory is examined. Then, the remaining two lists are deleted and the memory is examined again. The above screenshot shows two ways one can use the freelist command.

1. You can individually delete one list at a time
2. You can delete multiple lists using the freelist command

VI. SCOPE FOR FUTURE WORK
The project has achieved the major goals it set out to achieve. It successfully created a simplified language for web scraping that provides a myriad of functionalities and tools. Despite having achieved all the major goals, there are many possible future improvements in the project.

Currently, our DSL is a simple line by line scripting language. There is no implemented decision control system (apart from looping over lists) in the current version of the DSL. Decision control can further simplify the use of the DSL and provide a lot more versatility to the language. With the help of decision control, development time and redundancy can further be reduced significantly.

Furthermore, regular expressions can be implemented in the DSL. Regular expressions can further make the task of web scraping easier. By the means of regular expressions, users can decide exactly what kind of files they would like to scrape, which the current version of the DSL does not have. With the help of Regular Expressions, multiple lines of code can be reduced to a single line, thus reducing development time of the project. These are the areas we would like to improve upon. By implementing these features in the DSL, we can reduce development time and reduce redundancy in the code, which would make the process of coding with our language faster.

VII. CONCLUSION
Thus, a working Domain Specific Language is created, with a wide range of functionality for scraping. The syntax of this DSL is highly simplified, thus making it very easy for a technical layman to scrape data from the web. Our DSL provides various functionality to scrape various forms of data such as images, videos, audio, files, ids, classes, text and so on (using get command), and also provides functionality to view the details of the page, and the data contained in it (using view command). The DSL also provides functionality on memory management. The user can create identifiers like variables and lists. These variables and lists can be assessed and even deleted once they serve their purpose. These variables and lists can also be used in commands, and the DSL also allows for looping over lists to reduce redundancy of code. Furthermore, a help command for the ease of the user has been added.

REFERENCES

I. Arifanto, Randy, Yudistira DW Asnar, and MM InggrianiLiem. "Domain Specific Language for Web Scraper Development." 2018 5th International Conference on Data and Software Engineering (ICODSE). IEEE, 2018.

II. Thanhoffer-Pilisch, Lang, Vierhauser, Rabiser(2017). “Thanhoffer-Pilisch, Lang, Vierhauser, Rabiser”

III. Rodrigues, IldevanaPoltronieri, Márciade Borba Campos, and Avelino F. Zorzo. "Usability Evaluation of Domain-Specific Languages: a Systematic Literature Review." International Conference on Human-Computer Interaction. Springer, Cham, 2017.

IV. Abu Kausar, Mohammad & Dhaka, Vijaypal& Singh, Sanjeev. (2013). “Web Crawler: A Review. International Journal of Computer Applications.” 63. 31-36. 10.5120/10440-5125.

V. Van Deursen, Arie, Paul Klint, and Joost Visser. "Domain-specific languages: An annotated bibliography." ACM Sigplan Notices 35.6 (2000): 26-36.

VI. Pohjonen, Risto& Kelly, Steven. (2002). Domain-specific modeling. Dr. Dobb's Journal. 27.
VII. Hoisl, Bernhard, and Stefan Sobernig. "Consistency Rules for UML-based Domain-specific Language Models: A Literature Review." ACES-MB&WUCOR@ MoDELS. 2015.

VIII. Reeve, Lawrence H., Hyoil Han, and Ari D. Brooks. "The use of domain-specific concepts in biomedical text summarization." Information Processing & Management 43.6 (2007): 1765-1776.

IX. Mulo, Emmanuel, Uwe Zdun, and SchahramDustdar. "Domain-specific language for event-based compliance monitoring in process-driven SOAs." Service Oriented Computing and Applications 7.1 (2013): 59-73.

X. Pratiksha Ashiwal, S.R. Tandan, Priyanka Tripathi, Rohit Miri. “Web Information Retrieval Using Python and BeautifulSoup”. IJRASET, IC Value: 13.98(2016): 2321-9653.

XI. Wu, Jiahao. (2019). Web Scraping Using Python: A Step By Step Guide.

XII. Armano, Giuliano & Cherchi, Giancarlo & Vargiu, Eloisa. (2003). “A Parametric Hierarchical Planner for Experimenting Abstraction Techniques” 936-941.

XIII. Mernik, Marjan, Jan Heering, and Anthony M. Sloane. "When and how to develop domain-specific languages." ACM computing surveys (CSUR) 37.4 (2005): 316-344.