

ScraPE – An Automated Tool for Programming Exercises Scraping

Ricardo Queirós  

CRACS – INESC-Porto LA, Portugal
uniMAD, ESMAD/P.PORTO, Portugal

Abstract

Learning programming boils down to the practice of solving exercises. However, although there are good and diversified exercises, these are held in proprietary systems hindering their interoperability. This article presents a simple scraping tool, called ScraPE, which through a navigation, interaction and data extraction script, materialized in a domain-specific language, allows extracting the data necessary from Web pages – typically online judges – to compose programming exercises in a standard language. The tool is validated by extracting exercises from a specific online judge. This tool is part of a larger project where the main objective is to provide programming exercises through a simple GraphQL API.

2012 ACM Subject Classification Applied computing → Computer-managed instruction; Applied computing → Interactive learning environments; Applied computing → E-learning

Keywords and phrases Web scrapping, crawling, programming exercises, online judges, DOM

Digital Object Identifier 10.4230/OASlcs.SLATE.2022.18

1 Introduction

Programming courses are part of the curriculum of many engineering and science programs. These courses rely on programming exercises to foster practice, consolidate knowledge and evaluate students. The enrolment in these courses is usually very high, resulting in a great workload for the faculty and teaching assistants. In this context the availability of many and diversified programming exercises from different sources is of great importance [4]. Unfortunately, there are only a few sources to get, in an automatic way, programming exercises. Some notable examples are the online judges, which can be defined as repositories of programming exercises with automatic evaluation capabilities. These systems are often used by students around the world to train for programming contests such as the International Olympiad in Informatics (IOI)¹, for secondary school students; the ACM International Collegiate Programming Contests (ICPC)², for university students; and the IEEEExtreme³, for IEEE student members. Despite their usefulness, these systems do not have a simple mechanism to obtain programming exercises (e.g. an API). In fact, only a few offer interoperability features such as standard formats for their exercises and APIs to foster their reuse in an automated fashion. In this field, the most notable APIs for computer programming exercises consumption are CodeHarbor⁴, FGPE AuthorKit⁵, and Sphere Engine⁶. Still, they are not simple to use and expose a small number of exercises.

¹ <https://ioinformatics.org/>

² <https://icpc.global/>

³ <https://ieeextreme.org/>

⁴ <https://github.com/openHPI/codeharbor>

⁵ <https://github.com/FGPE-Erasmus/authorkit-api>

⁶ <https://sphere-engine.com/>



© Ricardo Queirós;
licensed under Creative Commons License CC-BY 4.0

11th Symposium on Languages, Applications and Technologies (SLATE 2022).

Editors: João Cordeiro, Maria João Pereira, Nuno F. Rodrigues, and Sebastião Pais; Article No. 18; pp. 18:1–18:7

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This poses a big problem for teachers who, due to lack of time, often resort to exercises from previous years. This recurrence hinders diversification and innovation in the practical part of programming courses, which is crucial for their evolution in this area.

This article presents a tool called ScraPE that allows, through a script formalized by a very simple domain-specific language (DSL), to extract data from Web pages (mostly online judges). The script defines a set of steps to navigate, interact and extract data to compose a programming exercise and its direct serialization to a standard language (YAPeXIL [3]). The tool will be used to mitigate the cold-start problem [5] in a larger project where the objective is to provide a simple and flexible GraphQL API for accessing programming exercises that can be consumed by several learning systems.

The remainder of this paper is organized as follows. Section 2 analyzes several of existing online judges to select the most suitable to feed a repository of programming exercises. Section 3 presents an automatic scraping tool to extract programming exercises. Then, in order to evaluate the effectiveness and efficiency of this approach, in Section 4, a report on the use of ScraPE in the TIMUS online judge is presented. The final section summarizes the main contributions of this research and plans future developments of this tool.

2 Online Judges

An Online Judge (OJ) is a system with a set of programming exercises that can be used by anyone to practice for programming contests. These systems can compile and execute your code, and test your code with predefined data. The code being submitted may run with restrictions, including time and memory limit, and other security restrictions. The output of the executed code will be compared with the standard output. The system will then return the result. When the comparison fails, the submission is considered unsuccessful and you need to correct any errors in the code, and resubmit for re-judgement.

Although there are several online judges, they do not provide any kind of API hindering its automatic consumption. In addition, those who provide these API return exercises in disparate formats, which leads to the need to use converters to harmonize formats. With this scarcity of exercises and given the difficulty of creating promptly good exercises, teachers often reuse exercises from previous years, which limits creativity [1].

In this section we survey online judges that present programming exercises. Since there are a large number of online judges, a set of criteria was applied to filter the set and thus obtain those that will be the most suitable to be used as a data source for the system to be implemented.

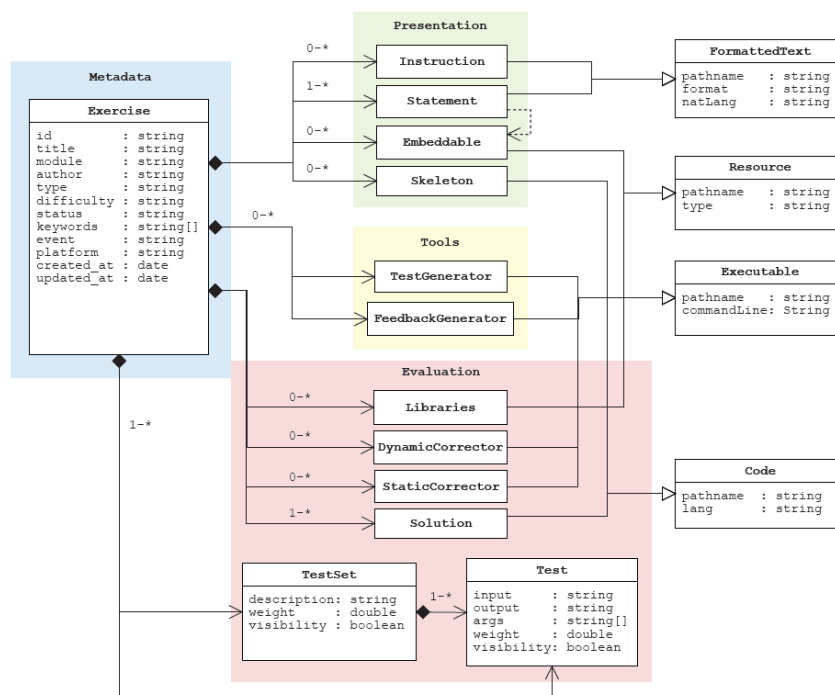
In a first phase we select 72 online judges. Then, in order to narrow the dataset we applied sequentially a set of criteria:

1. Statements in English language;
2. Statements in HTML format;
3. Public problem set (without the need to register/login in the OJ)
4. Minimum number of exercises ($n_{Ex} \geq 1000$)

Based on these filter criteria, only 17 OJs were selected. Then, all OJs were analyzed and validated according to their coverage in the YAPeXIL format [3]. The YAPeXil format is currently the most expressive format to represent a programming exercise [2]. It is formalized by a YAPeXIL JSON Schema (Figure 1) which can be divided into four separate facets:

- **Metadata** – which contains simple properties providing information about the exercise (i.e., a description, the name of the author of the exercise, a set of keywords relating to the exercise, the level of difficulty, the current status, and the timestamps of creation and last modification;

- **Presentation** – which relates to what is presented to the student (i.e. the statement – a formatted text file with a complete description of the problem to solve – embeddables – an image, video, or another resource file that can be referenced in the statement –, and skeleton – a code file containing part of a solution that is provided to the students);
- **Assessment** – which encompass what is used in the evaluation phase (i.e. solution – a code file with the solution of the exercise provided by the author(s), test – a single public/private test with input/output text files, a weight in the overall evaluation, and a number of arguments –, and test_set – a public/private set of tests);
- **Tools** – which includes any additional tools that the author may use in the exercise (i.e. generate the feedback to give to the student about her attempt to achieve a solution and the test cases to validate a solution).



■ **Figure 1** YAPExIL data model.

Each Online Judge was analyzed and its coverage in the 4 facets was verified. Table 1 presents the results of this study.

Based on these results, we can state that LeetCode, CodeChef, TIMUS, URI and Kattis are the OJs with higher YAPExIL coverage values, thus offering a higher guarantee that the exercises provided by the future API are more complete in terms of information for the end user.

3 ScraPE

ScraPE is a basic tool for scraping online judges on data related with programming exercises. The ultimate goal of this tool is to be used as a cold-start facilitator in a bigger system currently being developed which aims to provide a GraphQL API to anyone that want to get free programming exercises. This system will be based on a GraphQL server (Apollo)

18:4 ScraPE – An Automated Tool for Programming Exercises Scraping

■ **Table 1** Online judges comparison based on YAPExIL covereness.

Online Judges	#exercises	YAPExIL facets				TOTAL
		Metadata	Presentation	Assessment	Tools	
UVA	4300	20%	0%	0%	0%	5,00%
TIMUS	1157	95%	50%	0%	0%	36,25%
URI	2296	95%	50%	0%	0%	36,25%
Peking	3054	90%	45%	0%	0%	33,75%
Zhejiang	3179	75%	35%	0%	0%	27,50%
Kattis	3380	95%	50%	0%	0%	36,25%
LeetCode	2262	95%	50%	25%	0%	42,50%
CodeForces	78013	80%	25%	0%	0%	26,25%
DMOJ	4233	75%	25%	0%	0%	25,00%
Dunjudge	1707	80%	25%	0%	0%	26,25%
TopCoder	2122	65%	25%	0%	0%	22,50%
CodeChef	5001	95%	50%	25%	0%	42,50%
E-olymp	8325	85%	25%	0%	0%	27,50%
Toph	1548	90%	25%	0%	0%	28,75%
Hackerearth	1612	75%	25%	0%	0%	25,00%
LightOJ	1025	80%	25%	0%	0%	26,25%
Aizu	3023	85%	25%	0%	0%	27,50%

composed by a GraphQL schema, a resolver, a noSQL database where the exercises will be stored in YAPExIL format and a HTTP client to expose the API. Learning systems and/or individuals will use this API to feed their courses.

3.1 The schema

ScraPE uses a DSL to represent a script which is responsible by all the actions made on web pages from navigating to extracting data. The DSL is formalized as a JSON Schema. Listing 1 presents the action sub-schema, as will be explained below.

■ **Listing 1** Action schema.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "A schema to formalize an Action",
  "type": "object",
  "properties": {
    "page": { "type": "string" },
    "query": { "type": "string" },
    "type": { "type": "string" },
    "output": { "type": "string" },
    "actions": { "type": "array", "items": { "$ref": "#/defs/Action" } }
  },
  "required": ["type", "query", "output"]
}
```

The Action sub-schema is composed by five properties. The **page** property is the web page where the scraper will start extracting data. The **query** property represents a CSS selector that will be used to find the desired DOM nodes. The **type** property is a enumeration of all the action types that can be made in the selected element:

- GET – get DOM element(s) or attribute(s) based on a query;
- FILL – inject text in a selected text box or select an item in a selected combo box;
- CLICK – click in a selected button, radio button or checkbox.

The `output` property is a string which defines the name of the property to be created in the output file.

The execution of a query can result in multiple nodes. In this case, it is necessary to iterate over all of them and perform certain actions. This is the case, for example, of primary-secondary pages where a Web page has multiple links and where we need to enter and perform a set of actions on each of them. For this particular scenario we could have an `actions` array inside an `action` property.

4 Use Case: Timus Online Judge

Based on the results of Section 2, the effectiveness and efficiency of ScraPE was validated using one of the top-3 online judges as data source. The chosen one was the TIMUS Online Judge.

The first step was the construction of the JSON instance representing the script to be used in the scraping process. Since this process is (still) manual we use the Inspector tab of the browser Developer Tools to get all the desired CSS selectors.

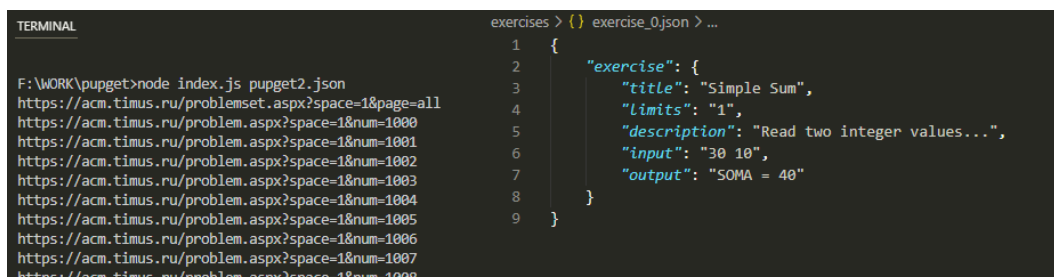
The next step was to refine the script with the action types and the output names for the output files being generated. It should be noted that the script generates an output file in an internal ScraPE format. Afterwards the file will be transformed into the target format (ideally YAPEXIL) using eXtensible Stylesheet Language (XSL) files. This way, it will be easier to scale ScraPE to other desired formats. The final script is presented in Listing 2

Listing 2 Final script.

```
"actions": [{
  "page": "problemset.aspx?space=1&page=all",
  "query": "td.name a",
  "type": "GET",
  "output": "exercise",
  "actions": [{
    "type": "GET",
    "query": "h2",
    "output": "title"
  }, {
    "type": "GET",
    "query": "div.problem_par:nth-child(1)",
    "output": "description"
  }, {
    "type": "GET",
    "query": ".problem_limits",
    "output": "limits"
  }, {
    "type": "GET",
    "query": ".sample td:nth-of-type(1) pre",
    "output": "input"
  }, {
    "type": "GET",
    "query": ".sample td:nth-of-type(2) pre",
    "output": "output"
  }
]}]
```

18:6 ScraPE – An Automated Tool for Programming Exercises Scraping

After the refinement phase, it is possible to run the ScraPE main script in the command line. Figure 2 shows, in the left, the output of the script running in command line and, in the right, one of the exercises extracted in a ScraPE internal format. Currently, exercises are generated in the file system, but the plan is to automatically store them in a specific NoSQL database.



```
TERMINAL                                     exercises > {} exercise_0json > ...
F:\WORK\pupget>node index.js pupget2.json
https://acm.timus.ru/problemset.aspx?space=1&page=all
https://acm.timus.ru/problem.aspx?space=1&num=1000
https://acm.timus.ru/problem.aspx?space=1&num=1001
https://acm.timus.ru/problem.aspx?space=1&num=1002
https://acm.timus.ru/problem.aspx?space=1&num=1003
https://acm.timus.ru/problem.aspx?space=1&num=1004
https://acm.timus.ru/problem.aspx?space=1&num=1005
https://acm.timus.ru/problem.aspx?space=1&num=1006
https://acm.timus.ru/problem.aspx?space=1&num=1007
https://acm.timus.ru/problem.aspx?space=1&num=1008

1 {
2   "exercise": {
3     "title": "Simple Sum",
4     "limits": "1",
5     "description": "Read two integer values...",
6     "input": "30 10",
7     "output": "SOMA = 40"
8   }
9 }
```

■ Figure 2 ScraPE exercises generation.

5 Conclusion

This article introduces an automated scraping tool called ScraPE. The purpose of the tool is not to compete with existing scraping tools, but to feed a database as a cold-start facilitator for an ongoing project. This database will be used in conjunction with an API to serve, in a flexible way, learning systems (or individuals) to get programming exercises in the YAPExIL format.

Currently, the tool is an ongoing work. In fact several parts of the process are not yet implemented such as: 1) the creation of a GUI editor to facilitate the script creation process; 2) the transformation of the ScraPE internal format to the YAPExIL format and 3) the storage of the exercises in a database.

After solving these three simple tasks the goal is to create the programming exercises API and integrate the ScraPE tool in order to provide a simple and universal way to everyone get programming exercises with specific filters (for instance, “give me an easy exercise in JAVA with arrays”).

References

- 1 Jackie O’Kelly and J. Paul Gibson. Robocode & problem-based learning: A non-prescriptive approach to teaching programming. *SIGCSE Bull.*, 38(3):217–221, June 2006. doi:10.1145/1140123.1140182.
- 2 José Carlos Paiva, Ricardo Queirós, and José Paulo Leal. Mooshak’s Diet Update: Introducing YAPExIL Format to Mooshak. In Ricardo Queirós, Mário Pinto, Alberto Simões, Filipe Portela, and Maria João Pereira, editors, *10th Symposium on Languages, Applications and Technologies (SLATE 2021)*, volume 94 of *Open Access Series in Informatics (OASICs)*, pages 9:1–9:7, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICs.SLATE.2021.9.
- 3 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Yet Another Programming Exercises Interoperability Language (Short Paper). In Alberto Simões, Pedro Rangel Henriques, and Ricardo Queirós, editors, *9th Symposium on Languages, Applications and Technologies (SLATE 2020)*, volume 83 of *OpenAccess Series in Informatics (OASICs)*, pages 14:1–14:8, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASICs.SLATE.2020.14.

- 4 Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003. doi:10.1076/csed.13.2.137.14200.
- 5 Vidhi Singrodia, Anirban Mitra, and Subrata Paul. A review on web scrapping and its applications. In *2019 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6, 2019. doi:10.1109/ICCCI.2019.8821809.