Programming Exercises Interoperability: The Case of a Non-Picky Consumer

Ricardo Queirós
CRACS – INESC-Porto LA, Portugal
uniMAD – ESMAD, Polytechnic Institute of Porto, Portugal

José Carlos Paiva
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

José Paulo Leal
CRACS – INESC-Porto LA, Portugal
DCC – FCUP, Porto, Portugal

Abstract

Problem-solving is considered one of the most important skills to retain in the coming decades for building a modern and proactive society. In this realm, computer programming learning is vital to enrich those skills. Practicing in this area boils down to solve programming exercises. In order to foster this practice, it is necessary to provide students with the best of the breed automated tools and a good set of exercises in a fair quantity covering the curricula of a typical programming course. Despite the increasing appearance of automated tools such as program evaluators, gamification engines and sophisticated web environments, access to exercises remains problematic. In fact, although the existence of several code repositories (most for feed computer programming contests), the majority of them store the exercises in proprietary formats and without any access facilities hindering their use. This leaves no other option to teachers but to manually create programming exercises which is time-consuming and error prone, or simply, reuse the same exercises, from previous years, which is considered as a detrimental and limiting approach to enhance multi-faceted and creative programmers.

The article surveys the current interoperability efforts on programming exercises, more precisely, in terms of serialization formats and communication protocols. This study will sustain the selection of an API to feed a code playground called LearnJS with random programming exercises.

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

Keywords and phrases programming exercises format, interoperability, automated assessment, learning programming

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.5

Funding This paper is based on the work done within the Framework for Gamified Programming Education (FGPE) Plus: Learning tools interoperability for gamified programming education project supported by the European Union's Erasmus Plus programme (agreement no. 2020-1-PL01-KA226-HE-095786).

1 Introduction

The need for educational resources repositories has been growing in the last years since more instructors are eager to create and use digital content and more of it is available. This growth led many to neglect interoperability issues which are crucial to share resources and to reuse them on different domains [11].

One of these domains is computer programming. In this realm, the increasing popularity of programming contests worldwide resulted in the creation of several contest management systems fed by code exercise repositories. At the same time, Computer Science courses use programming exercises to encourage the practice of programming. Thus, the interoperability between these types of systems is becoming, in the last decade, a topic of interest in the scientific community [12]. In order to address these interoperability issues, several programming exercise formats and API were developed to expose these resources in a standard fashion.

This article explores the current state of programming exercise interoperability by surveying the most popular programming exercise serialization formats and API for exercise consumption. This study will sustain the selection of an API based on its simplicity and flexibility, to be consumed by a code playground called LearnJS.

The remainder of this paper is organized as follows. Section 2 surveys syntactic interoperability on programming exercises based on how data is serialized and how it is communicated between systems. In Section 3, one of the APIs previously studied is used to fed a code playground called LearnJS. Finally, Section 4 summarizes the main contributions of this research and presents some plans for future work.

2 Programming Exercises

Computer programming is a complex field [2]. One of the most efficient ways to learn to program is through solving exercises. For the practice to be effective, there must be some mechanism that provides feedback on the quality of the learner's resolution. Nowadays, most programming courses typically have large classes and, thus, it is difficult for the teacher to provide timely and consistent feedback [1].

In this context, a set of automated tools has emerged that promote the edition, test, and execution of programs and delivers fast and expressive feedback based on program evaluators. Moreover, several systems combine gamification elements to promote extrinsic and intrinsic motivation to foster students' engagement and loyalty [3]. Despite these advances, the scarcity of programming exercises is still a problem. Although there are several code repositories [11], 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.

In the next subsections two surveys on programming exercises interoperability are presented based on data and communication facets.

2.1 Formats

Nowadays, there is a plethora of exercise formats [9]. Most of them were created to formally represent exercises for computer programming contests and are often stored in contest management systems or code repositories.

CATS [1] is a format for programming assignments. The format encoded in XML describes the exercise metadata and a set of additional resources such as statement, tests, test programs, etc. All these files are wrapped up in a ZIP file to facilitate deployment.

[1] `http://imcs.dvgu.ru/cats/docs/format.html`

Freeproblemset (FPS) [2] is a transport file format for the storage of all information about a programming exercise. It aims to provide free exercises sets for managers of ACM/ICPC Online Judges by transporting data from one judge to another. The format uses XML to formalize the description of a programming exercise. It includes information on the exercise itself, test data, special judge data (optional) and solutions (optional).

Mooshak Exchange Format (MEF) is the internal format of Mooshak [3] defined as a system for managing programming contests on the Web [5]. Mooshak is being used in several Universities worldwide to support learning activity. In the competitive context, it was used as the official evaluation system for the IEEE programming contests for some years. MEF includes a XML manifest file referring several types of resources such as problem statements (e.g. PDF, HTML), image files, input/output test files, correctors (static and dynamic) and solution programs. The manifest also allows the inclusion of feedback and points associated to each test.

Peach Exchange Format (PEF) is a specific format for programming task packages used in Peach [4], a system for the presentation, collection, storage, management and evaluation (automated and/or manual) of assignments [14].

YAPExIL [7] is a language for describing programming exercise packages, which builds on top of the XML dialect PExIL [10]. Comparatively, YAPExIL (1) is formalized through a JSON Schema rather than an XML Schema, (2) replaces the logic for automatic test generation with a script provided by the author, and (3) adds several assets that underpin different types of programming exercises. Its JSON Schema can be divided into four separate facets: metadata, containing simple values providing information about the exercise; presentation, including components that are presented to either the student or the teacher (e.g., problem statement and instructions); assessment, involving what enters in the evaluation phase (e.g., tests, solutions, and correctors); and tools, containing any additional tools that complement the exercise (e.g., test generators).

In the last years, several approaches appeared to evaluate the expressiveness of programming exercises formats [4]. One of the most notable approaches is the model proposed by Verhoeff where he describes conceptually the notion of a task package as an unit for collecting, storing, archiving, and exchanging all information concerning with a programming task. The choice of the Verhoeff model over the alternatives is due to its more comprehensive coverage of the required features. This model organizes the programming exercise data in five facets: (1) Textual information – programming task human-readable, (2) Data files – source files and test data, (3) Configuration and recommendation parameters – resource limits, (4) Tools – generic and task-specific tools, and (5) Metadata – data to foster the exercises discovery among systems. Table 1 presents a comparative study of all the referred formats based on the Verhoeff model.

This study confirms the diversity of programming exercises formats highlighting both their differences and their similar features. This heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. Rather than attempting to harmonize the various specifications, or working on specific data ingestion approaches, a pragmatic solution could be, for instance, to provide a service for exercises formats conversion [9] based on a pivotal format in which the conversion is based.

[2] http://code.google.com/p/freeproblemset/
[3] https://mooshak2.dcc.fc.up.pt/
[4] http://peach3.nl

Table 1 Comparison of programming exercise formats.

Category	Feature	CATS	FPS	MEF	PEF	PExIL	YAPExIL
Textual	Multilingual			X	X	X	X
	HTML format	X	X	X	X	X	X
	LaTeX format			X			
	Image	X	X	X	X	X	X
	Attach files	X			X		X
	Description	X	X	X	X	X	X
	Grading						X
	Samples					X	X
Data files	Solution	X	X	X	X	X	X
	Skeleton						X
	Multi-language	X	X		X	X	X
	Tests	X	X	X	X	X	X
	Test-groups	X			X	X	X
	Sample tests		X			X	X
	Grading	X		X	X		X
	Feedback			X		X	X
Configuration Recommendation	Compiler						X
	Executor						X
	Memory limit	X	X		X		X
	Size limit						X
	Time limit	X	X				X
	Code lines						X
Tools	Compiler				X	X	X
	Test generator	X				X	X
	Feedback generator			X		X	X
	Skeleton generator					X	X
	Checker	X			X	X	X
	Corrector			X		X	X
	Library	X		X	X		X
Metadata	Exercise	X	X	X	X	X	X
	Author	X			X	X	X
	Event		X		X	X	X
	Keywords			X	X	X	X
	License					X	X
	Platform				X		X
	Management				X		X

2.2 API

There are several code repositories [11]. Despite its existence, few offer interoperability features such as standard formats for their exercises and APIs to foster its reuse in an automated fashion. The most notable APIs for computer programming exercises consumption are CodeHarbor[5], CrimsonHex, FGPE AuthorKit[6], ICPC[7], and Sphere Engine-[8]

[5] `https://codeharbor.openhpi.de/`
[6] `https://python.usz.edu.pl/authorkit/ui`
[7] `https://clics.ecs.baylor.edu/index.php/Contest/API`
[8] `https://docs.sphere-engine.com/problems/api/quickstart`

FGPE Authorkit [6] is a Web programming exercises management tool that was created as part of the Erasmus+ Project entitled Framework for Gamified Programming Education (FGPE). The Authorkit aims to foster the creation of gamified programming exercises. The Web application allow users to prepare gamified programming exercises divided into two facets: to create the content of exercises and to assign a gamification layer. The former allows to specify all the components of a programming exercise (e.g. problem statement or input/output test files). The latter allows the definition of game-based data such as rewards, rules, leaderboards and challenges. Both data types are stored in two dedicated formats – Yet Another Programming Exercises Interoperability Language (YAPExIL) and Gamified Education Interoperability Language (GEdIL) [13]. The Authorkit expose its functions, with a dedicated API, allowing users to import content from popular non-gamified exercise formats, and export it or share it with other peers, internally or via a GitHub repository where all exercise data is synchronized.

Sphere Engine is a set of API that enable creating coding assessment solutions and online code execution environments. It is composed by two API: compilers [9] and problems [10]. The former allows users to execute computer programs in a secure run-time environment and receive feedback on the resolution. The latter allows users to create, edit, and manage programming problems. In order to use the API one should have an API token. After authenticating into the Sphere Engine client panel, it is possible to get a token in the API Tokens section. The main features of the API are create, edit, and manage programming exercises, define test cases, and import/export exercises. Using the export endpoint, clients have access to all information about an exercise including a manifest file called config.xml with references for all the assets wrapped in a ZIP file.

ICPC API is a well-known API for accessing information provided by a Contest Control System or Contest Data Server. This API is meant to be useful, not only at the ICPC World Finals, but more generally in any ICPC-style contest setup such as an external scoreboard or a contest analysis software. The API makes available several objects as JSON elements such as exercises, teams and members, submissions, runs, awards, contest state and scoreboards.

CodeHarbor is a repository system which allows to share, rate, and discuss auto-gradeable programming exercises. The system enables instructors to exchange exercises through the proFormA XML format across diverse code assessment systems. Despite all these features, most of them are made through an user interface, and the API is only available for grading scenarios.

crimsonHex [11] repository aims to store programming exercises as learning objects. It provides a set of operations based in the IMS DRI specification and exposed through a REST API. The main functions are (1) the Register/Reserve function to book a new resource identifier from the repository, (2) the Submit/Store function push an exercise to the repository, (3) the Search/Expose function enables external systems to query the repository using the XQuery language, (4) the Report/Store function associates a usage report with an existing exercise, and (5) the Alert/Expose function notifies users of changes in the state of the repository using an RSS feed.

The comparison of API can be made through several approaches. Despite the plethora of options the most popular are: architectural styles, domain coverage, and analytics. The former explores architectural key aspects of the API such as design patterns, communication protocols and encoding types. The second focuses on the coverage rate of the API in relation

[9] `https://docs.sphere-engine.com/compilers/overview`
[10] `https://docs.sphere-engine.com/problems/overview`

to the features of the system. The latter focuses on engineering metrics such as performance and uptime, but also customer and product metrics such as engagement, retention, and developer conversion. In this study, the first two approaches will be used.

The three key aspects to be aware in the architectural styles facet are the (1) design philosophy/pattern (e.g., RESTful vs GraphQL), (2) communication protocol (e.g., HTTP vs WebSockets), and (3) encoding (e.g., human-readable text such as JSON vs Binary formats like ProtoBuf). Often, these three different aspects can be mixed together. For instance, one can use RESTful API over WebSockets but using a Binary Protocol (e.g. MessagePack).

Regarding the domain coverage of the API in the programming exercises domain, the most important features are the management of exercises such as create, read, update, delete, import/export exercises; the submission/grading features; the gamification features such as challenges and scoreboards and, finally, the user management features such as assigning user to teams.

Table 2 presents the comparison of the previous API based on two criteria: architectural styles and domain functions.

Table 2 Comparison of API from code repositories.

Category	Facet	CodeHarbor	crimsonHex	FGPE	ICPC	Sphere
Architectural Style	Design pattern	REST	SOAP REST	REST GRAPHQL	REST	REST
	Communication protocol	HTTP	HTTP	HTTP	HTTP	HTTP
	Encoding	XML	XML	JSON	JSON	JSON
Functions	CRUD	NO	YES	YES	YES	YES
	Import/Export	NO	YES	YES	YES	YES
	Submit/Grade	NO	NO	NO	NO	YES
	Game elements	NO	NO	YES	YES	NO
	Groups & Users	NO	NO	YES	YES	YES

Based on this study, one can conclude that, currently, the most popular approach to create web API is through RESTful API using the JSON format on the HTTP protocol. The coverage of the API is done essentially in the exercises management CRUD and import/export features.

3 Use case: LearnJS

This section presents the basic steps for the use of the FGPE Authorkit API by a code playground called LearnJS [8] defined as Web playground which enables anyone to practice the JavaScript language. More specifically, the playground allow users to solve programming exercises of several types (e.g. blank sheet, skeleton, find the bug, and quizzes) and to receive prompt feedback on their resolutions. In LearnJS, students can see videos or PDF of specific topics and solve exercises related with those topics with automatic feedback on their resolutions. Currently, the playground has two main components: (1) an Editor which allows students to code their solutions in an interactive environment and (2) an evaluator which evaluates the student's code based on static and dynamic analyzers. At this moment, a simple running prototype [11] (version 0.7.7) is available.

[11] `https://rqueiros.github.io/learnjs`

The use of the AuthorKit API by LearnJS will allow the support of a new type of exercise called random that, after selection, will make the playground use the API to fetch a random exercise from the Authorkit repository. The first step towards the integration is to register in the AuthorKit with the name, email, and password data in the request body. If the registration is successful, the next step is to log in using the registration data. This action will return a token that should be used in any subsequent requests.

In the AuthorKit, exercises are assigned to projects that act as containers. For that reason, the next step is to get a list of projects. Each project has a set of metadata such as an identifier, a title, a status, the number of contributors, and exercises. With a project identifier, it is possible to get all its exercises. An exercise is composed of several properties such as an identifier, a title, a type, keywords, a difficulty level, and a status. To obtain assigned assets such as the statement or tests it is necessary to explicitly append in the correspondent endpoint the type of assets to obtain.

Table 3 presents the endpoints used for this integration (base URL: `http://fgpe.dcc.fc.up.pt/api`.

Table 3 Endpoints of FGPE AuthorKit API used in the integration.

Functions	Method	REST API
Register	POST	/auth/register
Login	POST	/auth/login
GetProjects	GET	/projects
GetExercises	GET	/exercises?page=1&limit=6
GetExercise	GET	/exercises/:id
GetExerciseAssets	GET	/exercises/:id?join=statements&join=tests
ExportExercise	GET	/exercises/:id/export

Note that in order to effectively get the assets you should always use the suffix /contents in the endpoints. For instance, for each test, the contents of input and output are in /tests/:id/input/contents and /tests/:id/output/contents. The same logic can be used to fetch the statement of the exercise as shown in Listing 1.

Listing 1 Inject an exercise statement in the LearnJS GUI.

```
...
// Get statement ID
const resp = await fetch(`${baseUrl}/exercises/${id}?join=statements`)
const result = await resp.json();
const sId = result.statements[0].id

// Get statement (base64 string)
const statement = await fetch(`${baseUrl}/statements/${sId}/contents`)
const statementBase64 = await statement.text();

// Document fragment creation
let fragment = document.createDocumentFragment();
fragment.appendChild(window.atob(statementBase64))

// Injection of the statement in the playground UI through DOM
document.querySelector('#sDiv').innerHTML = fragment.body.innerHTML
...
```

The final result is shown in Figure 1 where the statement of the exercise appears in the top panel of the screen of the LearnJS GUI.

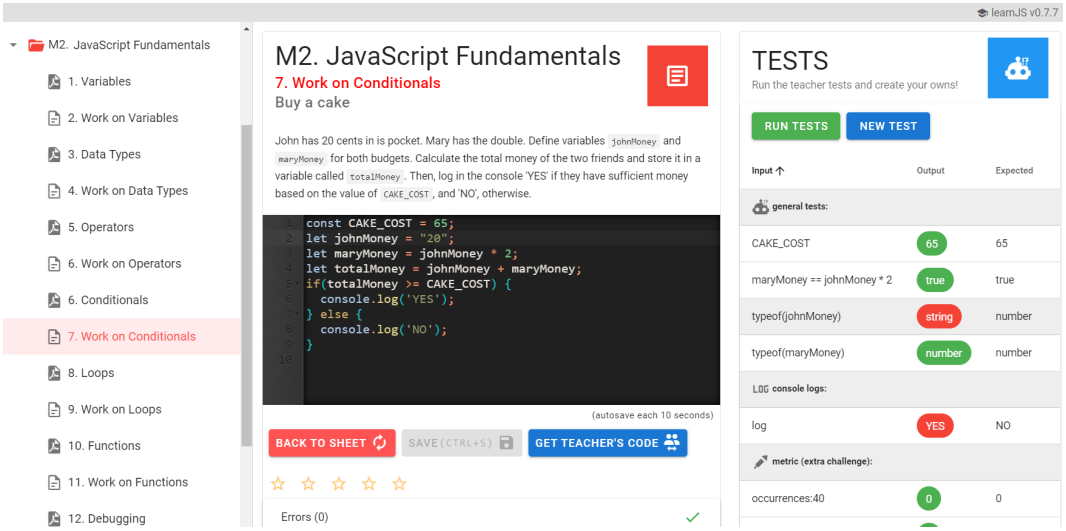

Figure 1 LearnJS playground GUI.

4 Conclusion

This article surveys the current state of programming exercises interoperability at two levels: data and communication. In the former, several exercise formats were identified and compared using the Verhoeff model. In the latter, several API have been studied based on their architectural styles and covered functions.

The ultimate goal of the survey is to support the best decisions to do in the process of integrating a new exercise type in the LearnJS playground. This integration consists of fetching exercises from the FGPE AuthorKit API and transforming them from their original format – YAPExIL – to the LearnJS internal format. As future work, the goal is to benefit from the gamified layer of the FGPE AuthorKit to gamify the LearnJS playground using the same API.

References

1 Kirsti M. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005. doi:10.1080/08993400500150747.

2 Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn? patterns of difficulties related to programming learning mid-stage. *SIGSOFT Softw. Eng. Notes*, 41(6):1–6, 2017. doi:10.1145/3011286.3011301.

3 Patrick Buckley and Elaine Doyle. Gamification and student motivation. *Interactive Learning Environments*, 24(6):1162–1175, 2016. doi:10.1080/10494820.2014.964263.

4 Stephen H. Edwards, Jürgen Börstler, Lillian N. Cassel, Mark S. Hall, and Joseph Hollingsworth. Developing a common format for sharing programming assignments. *SIGCSE Bull.*, 40(4):167–182, 2008. doi:10.1145/1473195.1473240.

5 José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003. doi:10.1002/spe.522.

6 José Carlos Paiva, Ricardo Queirós, José Paulo Leal, and Jakub Swacha. Fgpe authorkit – a tool for authoring gamified programming educational content. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 564, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3341525.3393978.

7 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.

8 Ricardo Queirós. LearnJS - A JavaScript Learning Playground (Short Paper). In Pedro Rangel Henriques, José Paulo Leal, António Menezes Leitão, and Xavier Gómez Guinovart, editors, *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*, volume 62 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.SLATE.2018.2.

9 Ricardo Queiros and Jose Paulo Leal. Babelo—an extensible converter of programming exercises formats. *IEEE Trans. Learn. Technol.*, 6(1):38–45, January 2013. doi:10.1109/TLT.2012.21.

10 Ricardo Queiros and José Leal. Pexil - programming exercises interoperability language. In *title = "XATA 2011: XML: associated technologies and applications"*, January 2011.

11 Ricardo Queirós and José Paulo Leal. crimsonhex: a learning objects repository for programming exercises. *Software: Practice and Experience*, 43(8):911–935, 2013. doi:10.1002/spe.2132.

12 Alberto Simões and Ricardo Queirós. On the Nature of Programming Exercises. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 24:1–24:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.24.

13 Jakub Swacha, José Carlos Paiva, José Paulo Leal, Ricardo Queirós, Raffaele Montella, and Sokol Kosta. Gedil—gamified education interoperability language. *Information*, 11(6), 2020. doi:10.3390/info11060287.

14 Tom Verhoeff. Programming task packages: Peach exchange format. *Olympiads in Informatics*, 2:192–207, January 2008.