Yet Another Programming Exercises Interoperability Language

José Carlos Paiva
CRACS – INESC, LA, Porto, Portugal
DCC – FCUP, Porto, Portugal
jose.c.paiva@inesctec.pt

Ricardo Queirós
CRACS – INESC, LA, Porto, Portugal
uniMAD – ESMAD, Polytechnic of Porto, Portugal
http://www.ricardoqueiros.com
ricardoqueiros@esmad.ipp.pt

José Paulo Leal
CRACS – INESC, LA, Porto, Portugal
DCC – FCUP, Porto, Portugal
https://www.dcc.fc.up.pt/~zp
zp@dcc.fc.up.pt

Jakub Swacha
University of Szczecin, Poland
jakub.swacha@usz.edu.pl

Abstract

This paper introduces Yet Another Programming Exercises Interoperability Language (YAPExIL), a JSON format that aims to: (1) support several kinds of programming exercises behind traditional blank sheet activities; (2) capitalize on expressiveness and interoperability to constitute a strong candidate to standard open programming exercises format. To this end, it builds upon an existing open format named PExIL, by mitigating its weaknesses and extending its support for a handful of exercise types. YAPExIL is published as an open format, independent from any commercial vendor, and supported with dedicated open-source software.

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, programming learning

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.14

Category Short Paper

Funding This paper is based on the work done within the Framework for Gamified Programming Education project supported by the European Union's Erasmus Plus programme (agreement no. 2018-1-PL01-KA203-050803).

1 Introduction

Learning programming relies on practicing. Practicing in this domain boils down to solve exercises. Regardless of the context (curricular or competitive learning), several tools such as contest management systems, evaluation engines, online judges, repositories of learning objects, and authoring tools use a different panoply of formats in order to formalize exercises. Though this approach remedies individual needs, the lack of a common format hinders interoperability and weakens the development and sharing of exercises among different

educational institutions. Moreover, the existence of a common data format will increase innovation in programming education with a high practical impact, as it will help to save a lot of instructors' time that they would otherwise have to spend on defining new exercises or recasting existing ones themselves.

At the same time, all of these programming exercise formats focus on describing traditional programming exercises, such as blank sheet exercises, where the student is challenged to solve, from scratch, a presented problem statement. In fact, up to this date, there are no open formats that explore the fostering of new competencies such as understanding code developed by others and debugging. To enhance these skills, new types of exercises (e.g., solution improvement, bug fix, gap filling, block sorting, and spot the bug) can be defined and applied at different phases of a student's learning path. This diversity can promote involvement and dispel the tedium of routine associated with solving exercises of the same type. To the best of our knowledge, there are several formats for defining programming exercises but none of them supports all the different types of programming exercises mentioned.

This paper introduces a new format developed for describing programming exercises - the Yet Another Programming Exercises Interoperability Language (YAPExIL). This format is partially based in the XML dialect PExIL [4], but (1) it is a JSON format instead of XML, (2) transfers the complex logic of automatic test generation to a script provided by the author, and (3) supports different types of programming exercises.

The remainder of this paper is organized as follows. Section 2 surveys the existing formats, highlighting both their differences and similar features. In Section 3, YAPExIL is introduced as a new programming exercises format and four facets are presented. Then, Section 4 validates the format expressiveness and coverage. In the former, the Verhoeff model [6] is used to validate the expressiveness of YAPExIL. In the later, the YAPExIL coverage for new types of exercises is shown. Finally, Section 5 summarizes the main contributions of this research and discusses plans for future work.

2 Programming Exercises Format

The increasing popularity of programming encourages its practice in several contexts. In formal learning, teachers use learning environments and automatic evaluators to stimulate the practice of programming. In competitive learning, students participate in programming contests worldwide resulting in the creation of several contest management systems and online judges. The interoperability between these types of systems is becoming a topic of interest in the scientific community. To address these interoperability issues, several programming exercise formats were developed in the last decades.

In 2012, a survey [5] synthesized those formats according to the Verhoeff model [6]. This model organizes the programming exercise data into five facets: (1) Textual information - programming task human-readable texts; (2) Data files - source files and test data; (3) Configuration and recommendation parameters - resource limits; (4) Tools - generic and task-specific tools; (5) Metadata - data to foster exercise discovery among systems. For each facet of the model, a specific set of features was analyzed and verified the support of each format. In that time, the study confirmed the disparity of programming exercise formats and the lack or weak support for most of the Verhoeff model facets. Moreover, the study concludes that this heterogeneity hinders the interoperability among the typical systems found on the automatic evaluation of exercises. To remedy these issues, two attempts to harmonize the various specifications were developed: a new format [4] and a service for exercises formats conversion [5].

Since then, new formats were proposed to formalize exercises. In this section, we present a new survey that aims to compare existent programming exercise formats based on their expressiveness. Based on a comprehensive survey done of systems that store and manipulate programming exercises, we found about 10 different formats. Since some of them lack a published description we concentrated on 7 formats, namely: (1) Free Problem Set (FPS); (2) Kattis problem package format; (3) DOM Judge format; (4) Programming Exercise Markup Language (PEML); (5) the language for Specification of Interactive Programming Exercises (SIPE); (6) Mooshak Exchange Format (MEF); and (7) Programming Exercises Interoperability Language (PExIL).

The study follows the same logic as its predecessor, but with the following changes:

- Expressiveness model - the Verhoeff model is extended with a new facet that analyzes the support of new types of exercises. For this study eight types of programming exercises were tackled, namely: blank sheet, extension, improvement, bug fix, fill in gaps, spot bug, sort blocks, and multiple choice.
- Data formats - given that several years have passed since the last study, CATS and Peach Exchange formats are removed and new formats are added (Kattis, DOM Judge, PEML, and SIPE).

Each format is evaluated for its level of coverage of all features of each facet. The evaluation values range from 1 – low support to 5 – full support. Then, all the values are added and a final percentage is presented corresponding to the coverage global level of each format and facet, based on the extended Verhoeff model.

Table 1 gathers the current coverage level of the selected programming exercises formats.

Table 1 Coverage level comparison on programming exercises data formats.

Facets/Formats	FPS	KTS	DOMJ	PEML	SIPE	MEF	PExIL	**TOTAL**
1. Textual	2	3	3	3	3	4	5	66%
2. Data files	3	2	3	3	3	3	5	63%
3. Config	2	2	1	1	3	3	5	49%
4. Tools	1	3	3	2	2	3	5	54%
5. Metadata	2	2	3	3	3	3	5	60%
6. Ex. types	1	1	1	1	1	2	1	23%
TOTAL	37%	43%	47%	43%	50%	60%	87%	

Based on these values, we can see that, regarding format coverage, PExIL assumes a prominent role with 87% of facets' coverage rate based on the Verhoeff extended model. This is mainly because, despite being a format created eight years ago, it is still one of the most recent formats (excluding the PEML format). All the other formats cover more or less half of the facets.

Regarding facets coverage, one can conclude that, on the one hand, textual (66%), data files (63%), and metadata (60%) facets are the most covered. On the other hand, the support for different exercise types, beyond the blank sheet type (typical format), is scarce.

3 YAPExIL

Yet **A**nother **P**rogramming **Ex**ercises **I**nteroperability **L**anguage (YAPExIL) is a language for describing programming exercise packages, partially based in the XML dialect PExIL (**P**rogramming **Ex**ercises **I**nteroperability **L**anguage) [4]. In comparison to PExIL, YAPExIL (1) is formalized through a JSON Schema instead of a XML Schema, (2) removes complex

logic for automatic test generation while still supporting it through scripts, (3) supports different types of programming exercises and (4) adds support for a number of assets (e.g., instructions for authors, feedback generators, and platform information).

YAPExIL aims to consolidate all the data required in the programming exercise life-cycle, including support for seven types of programming exercises:

- `BLANK_SHEET` provides a blank sheet for the student to write her solution source code from scratch;
- `EXTENSION` presents a partially finished solution source code (the provided parts are not subject to change by the student) for the student to complete;
- `IMPROVEMENT` provides correct initial source code that does not yet achieve all the goals specified in the exercise specification (e.g., optimize a solution by removing loops), so the student has to modify it to solve the exercise;
- `BUG_FIX` gives a solution with some bugs (and, possibly, failed tests) to foster the student to find the right code;
- `FILL_IN_GAPS` provides code with missing parts and asks students to fill them with the right code;
- `SPOT_BUG` provides code with bugs and asks students to merely indicate the location of the bugs;
- `SORT_BLOCKS` breaks a solution into several blocks of code, mixes them, and asks students to sort them.

To this end, the YAPExIL JSON Schema can be divided into four separate facets: **metadata**, which contains simple properties providing information about the exercise; **presentation**, which relates to what is presented to the student; **assessment**, which encompass what is used in the evaluation phase; and **tools**, which includes any additional tools that the author may use in the exercise.

Figure 1 presents the data model of YAPExIL format, with the area of each facet highlighted in a distinct color. The next subsections describe each each of these facets.

3.1 Metadata Facet

The Metadata facet, highlighted in blue in Figure 1, encodes basic information about the exercise that can uniquely identify it and to which subject(s) it refers to. Elements in this facet are mostly used to facilitate searching and consultation in large collections of exercises and the interoperability among systems. For instance, an exercise can be uniquely identified by its `id`, which is a Universally Unique Identifier (UUID) of the exercise.

Furthermore, the metadata includes many other identifying and non-identifying attributes such as the `title` of the programming exercise, the `module` in which the exercise is in (i.e., a description of its main topic), the name of the `author` of the exercise, a set of `keywords` relating to the exercise, its `type` – which can be `BLANK_SHEET`, `EXTENSION`, `IMPROVEMENT`, `BUG_FIX`, `FILL_IN_GAPS`, `SORT_BLOCKS`, or `SPOT_BUG` –, the `event` at which the exercise was created (if any), the `platform` requirements (if any), the level of `difficulty` (one of `BEGINNER`, `EASY`, `AVERAGE`, `HARD`, or `MASTER`), the current `status` (i.e., whether it is still a `DRAFT`, a `PUBLISHED` or `UNPUBLISHED` exercise, or it has been moved to `TRASH`), and the timestamps of creation and last modification (`created_at` and `updated_at`, respectively).

3.2 Presentation Facet

The Presentation facet, highlighted in green in Figure 1, includes all elements that relate to the exercise visualization, both by the students and the instructors. More precisely, these will be the elements placed on the screen while the student solves the problem, and when the teacher firstly opens the exercise.

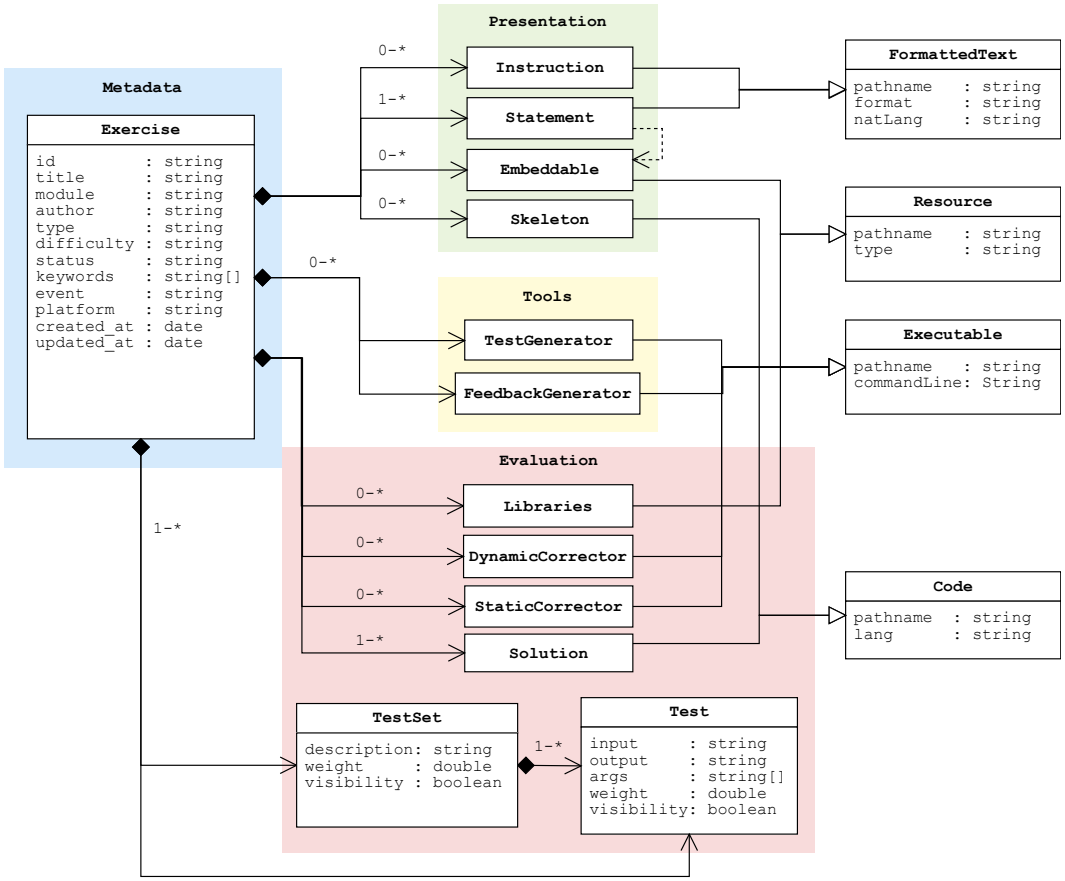

Figure 1 Data model defined by the YAPExIL format.

The supported elements include `instruction` – a formatted text file with instructions to teachers about how to deliver or some remarks on the exercise –, `statement` – a formatted text file with a complete description of the problem to solve –, `embeddable` – 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, from which they can start developing theirs.

All of these elements are allowed multiple instances, being required only a single statement in this facet to have a complete exercise. Hence, formatted text files may be translated to other natural languages or formats whereas code files can be written in several programming languages.

3.3 Assessment Facet

The automated assessment is the end goal of a programming exercise definition language. In order to evaluate a programming exercise, the learner must submit the source code to an evaluation engine. The evaluation engine will then use the necessary and available elements to judge it.

All the elements used in the evaluation belong to the Assessment facet, highlighted in red in Figure 1, and include `template` – code file containing part of a solution that wraps students' code without their awareness –, `library` – code library that can be used by solutions, either

in compilation or execution phase –, `static_corrector` – external program (and associated command line) that is invoked before dynamic correction to classify/process the program's source code –, `dynamic_corrector` – external program (and associated command line) that is invoked after the main correction to classify each run –, `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.

Each element in this facet also supports multiple instances, being required only a single solution and either a test or a testset with one test. Hence, multiple correctors, libraries, and test/testsets, and solutions in different programming languages may be provided.

3.4 Tools Facet

The Tools facet, highlighted in yellow in Figure 1, encompasses any additional scripts that may be used during the programming exercise life-cycle. These include external programs (and their associated execution command line) that generate (1) the feedback to give to the student about her attempt to achieve a solution (i.e., `feedback_generator`) and the test cases to validate a solution (i.e., `test_generator`).

4 Validation

YAPExIL was designed to fulfil two objectives: (1) support the definition of different types of programming exercises, in addition to the traditional ones; (2) cover all aspects of the model proposed by Verhoeff [6], while being simple and easily convertible. Hence, the validation of YAPExIL must prove the two objectives of this new data model for programming exercises.

The validation of the first objective consists of iterating through each proposed type of exercise and describing how YAPExIL can fulfil its requirements. Table 2 presents the results of this validation.

Table 2 Fulfilment of proposed exercise types.

Type	Fulfilment
`BLANK_SHEET`	Traditional programming exercise that only requires a statement, a test generator (or tests), and a solution. All these elements are part of YAPExIL.
`EXTENSION`	In addition to a `BLANK_SHEET` exercise, this one requires a skeleton, which is part of YAPExIL.
`IMPROVEMENT`	In addition to an `EXTENSION` exercise, this one needs to test other program metrics besides its acceptance. This is achieved through static and/or dynamic correctors.
`BUG_FIX`	This type requires the same elements as an `EXTENSION` exercise (skeleton can be the code with bugs).
`FILL_IN_GAPS`	This type requires the same elements as an `EXTENSION` exercise (skeleton can be the code with gaps marked with gap placeholder).
`SORT_BLOCKS`	This type requires the same elements as an `EXTENSION`, but multiple skeletons. This is supported by YAPExIL.
`SPOT_BUG`	This exercise requires a statement, a skeleton with bugged code, and one test where the output is the bug(s) location. All these elements are part of YAPExIL.

The evaluation of the expressiveness of programming assignments' formats has already been conducted in several works [2, 3, 6]. Table 3 validates the expressiveness of YAPExIL according to the model proposed by Verhoeff [6]. In particular, it iterates through each facet and correspondent features of the model and explains its fulfilment with YAPExIL.

Table 3 Fulfilment of Verhoeff expressiveness model for programming exercise packages.

Facet	Feature	Fulfilment
Textual	Multilingual	Multiple statements and instructions are supported and each of them can have a natural language associated (see Subsection 3.2).
	HTML format	Statements and instructions support HTML files (see Subsection 3.2).
	Other open standard text formats	Statements and instructions support Markdown files (see Subsection 3.2).
	Presentation formats	Statements and instructions support PDF files (see Subsection 3.2).
	Image	Embeddables can be images (see Subsection 3.2).
	Attach files	Embeddables can be attachments (see Subsection 3.2).
	Description	The description is provided in a statement. (see Subsection 3.2).
Data files	Solution	Solution is an element of YAPExIL (see Subsection 3.3).
	Skeleton	Skeleton is an element of YAPExIL (see Subsection 3.3).
	Multi-language	All kinds of elements with source code support multiple files and an associated programming language (see Subsection 3.3).
	Tests	Test is an element of YAPExIL (see Subsection 3.3).
	Test groups	Test elements can be part of a set (see Subsection 3.3).
	Sample tests	Tests can be public (see Subsection 3.3).
	Grading	Tests can have a weight (see Subsection 3.3).
	Feedback	Rich feedback can be provided by a generator.
Configuration & recommendation	Memory limit	May be provided as an argument of a test.
	Size limit	Checked through a static corrector command line.
	Time limit	May be provided as an argument of a test.
	Code lines	Checked through a static corrector command line.
Tools	Test gen.	Test generator is an element of YAPExIL (see Subsection 3.4).
	Feedback gen.	Feedback generator is an element of YAPExIL (see Subsection 3.4).
	Corrector	Static and dynamic correctors are elements of YAPExIL (see Subsection 3.3).
	Library	Library is an element of YAPExIL (see Subsection 3.3).
Metadata	Title	Title is an element of YAPExIL (see Subsection 3.1).
	Topic	Module is an element of YAPExIL (see Subsection 3.1).
	Difficulty	Difficulty is an element of YAPExIL (see Subsection 3.1).
	Author	Author is an element of YAPExIL (see Subsection 3.1).
	Event	Event is an element of YAPExIL (see Subsection 3.1).
	Keywords	Keywords are elements of YAPExIL (see Subsection 3.1).
	Platform	Platform is an element of YAPExIL (see Subsection 3.1).
	Management	Status, creation date, and update date are elements of YAPExIL (see Subsection 3.1).

5 Conclusion

There are plenty of programming exercise formats nowadays. The need for automated assessment of programming exercises led to a massive proliferation of formats following the emergence of automated assessment tools [6, 2, 3, 4]. Notwithstanding, despite some attempts to converge into a common format [5], the interoperability and reusability were left behind, resulting in the absence of a standard open format for representing programming exercises and working tools to convert among them.

In addition to the current inability to share and reuse exercises, the focus of existing formats, largely due to programming contests, has been on traditional programming exercises where the solver must develop a complete solution, starting from a blank file. However, different kinds of programming exercises can foster other programming skills and/or are more adequate at the different stages of learning programming (e.g., spot the bug, fill-in the gaps).

This paper introduces YAPExIL, a JSON format that aims to fulfill both of the identified gaps by building upon an existing (and failed) candidate to standard open format, PExIL. To this end, YAPExIL addresses the pinpointed flaws of PExIL: (1) the exaggerated complexity of the test generation mechanism; (2) using the slower and heavier language XML comparing to JSON format; and (3) the lack of support for non-traditional programming exercises.

YAPExIL is independent of the evaluation engine, heading all efforts to expressiveness and easy conversion from/to other programming exercise formats. Moreover, it is already being used as the main format of a collaborative web programming exercises editor, developed as open-source software [1]. This editor has an internal conversion mechanism for the most known formats, currently supporting two.

Our most immediate future work is the development of support for new formats within this tool, to build a large open collection of programming exercises (featuring gamification in a separate layer).

References

1 FGPE AuthorKit. `http://fgpe.dcc.fc.up.pt`, 2020.

2 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, November 2008. `doi:10.1145/1473195.1473240`.

3 A. Klenin. Common problem description format: Requirements, 2011. accessed on April 2019. URL: `https://ciiwiki.ecs.baylor.edu/images/1/1a/CPDF_Requirements.pdf`.

4 Ricardo Queirós and José Paulo Leal. PExIL: Programming exercises interoperability language. In *Conferência Nacional XATA: XML, aplicações e tecnologias associadas, 9.ª*, pages 37–48. ESEIG, 2011.

5 Ricardo Queirós and José Paulo Leal. Babelo - an extensible converter of programming exercises formats. *IEEE Trans. Learn. Technol.*, 6(1):38–45, 2013. `doi:10.1109/TLT.2012.21`.

6 Tom Verhoeff. Programming task packages: Peach exchange format. *International Journal Olympiads In Informatics*, 2:192–207, 2008.