Mooshak's Diet Update: Introducing YAPExIL Format to Mooshak

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

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

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

Abstract

Practice is pivotal in learning programming. As many other automated assessment tools for programming assignments, Mooshak has been adopted by numerous educational practitioners to support them in delivering timely and accurate feedback to students during exercise solving. These tools specialize in the delivery and assessment of blank-sheet coding questions. However, the different phases of a student's learning path may demand distinct types of exercises (e.g., bug fix and block sorting) to foster new competencies such as debugging programs and understanding unknown source code or, otherwise, to break the routine and keep engagement. Recently, a format for describing programming exercises – YAPExIL –, supporting different types of activities, has been introduced. Unfortunately, no automated assessment tool yet supports this novel format. This paper describes a JavaScript library to transform YAPExIL packages into Mooshak problem packages (i.e., MEF format), keeping support for all exercise types. Moreover, its integration in an exercise authoring tool is described.

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.2021.9

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/FGPE-Erasmus/yapexil-mef-converter`; archived at `swh:1:dir:017c9838b4dfb8d45092d31e55804671f3449de2`

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

Learning programming demands a tremendous amount of practice. Practice in this domain consists of developing a program that attempts to solve a problem, receive feedback on it, and use this feedback to further improve the solution until it meets all requirements. As the number of students enrolled in programming courses grows, educational practitioners rapidly started to search for tools such as contest management systems, evaluation engines, and online judges to support them in delivering timely and accurate feedback to students during exercise solving.

There is a panoply of different tools providing automated assessment of programming assignments, but the vast majority targets a single type of activity by design: blank-sheet programming exercises, where the student is challenged to code a solution from scratch to a presented problem statement. One of such tools is Mooshak, an open-source web-based system for managing programming contests, which includes automatic judging of submitted programs. Even though Mooshak handles custom static and dynamic analysis scripts and, thus, enables the assessment of non-traditional programming exercises, its format – the Mooshak Exchange Format (MEF) – does not forecast support for other types of programming activities.

Only recently, to the best of the authors' knowledge, an open format for describing programming exercises that explores the development of new competencies through different types of activities, such as understanding unknown source code and debugging, has been published – the Yet Another Programming Exercises Interoperability Language (YAPExIL). YAPExIL ships with direct support for seven types of activities, namely blank-sheet, solution improvement, bug fix, gap filling, block sorting, and spot the bug, to be applied at different phases of a student's learning path or solely to dissipate the monotony associated with solving exercises of the same type.

However, YAPExIL is a completely decoupled format, i.e., it is neither associated nor targets any specific tool. Hence, due to its recency, there is still no automated assessment tool that can work with problem packages adhering to the YAPExIL format. This paper introduces a JavaScript library to transform YAPExIL packages into problem packages adhering to the MEF format, ready to be consumed by Mooshak. This conversion has minimal loss of the encoded content in YAPExIL, maintaining support for all its types of programming exercises.

The remainder of this paper is organized as follows. Section 2 surveys the mentioned formats, highlighting both their differences and similar features. In Section 3, the YAPExIL to MEF conversion JavaScript library is introduced. Then, Section 4 demonstrates how the different exercise types can be described in MEF. Finally, Section 5 summarizes the main contributions of this research and presents some plans for future work.

2 Related Work

This section describes the programming exercise formats previously mentioned, namely MEF [1] and YAPExIL [3]. According to [3], which follows an extended Verhoeff model [5] to evaluate expressiveness, these are the two most expressive programming exercise formats (excluding PExIL [4], which is a subset of YAPExIL).

2.1 Mooshak Exchange Format (MEF)

Mooshak [1] is a web-based contest management system originally developed to manage programming contests over the Internet. Like many other systems of similar categories, it features its own internal format to describe programming exercises called **M**ooshak **E**xchange **F**ormat (MEF). A package adhering to MEF includes an XML manifest file in the root, containing metadata and references to other types of resources present in the package such as problem statements (e.g., PDF and HTML), images, input/output test files, static and dynamic correctors, and solutions. The manifest file can also hold points (i.e., the weight of the test within the overall problem grade) and feedback messages for each test case.

Due to its wide adoption in educational contexts to assist teachers in delivering timely and accurate feedback on programming assignments, version 2 of Mooshak heads most efforts towards its improvement for educational contexts. In particular, it extends MEF to support

code skeletons for students to start from as well as to give the possibility to have multiple solutions, public and private tests, and different editors for separate kinds of exercises (e.g., code and diagram).

2.2 Yet Another Programming Exercises Interoperability Language (YAPExIL)

Yet **A**nother **P**rogramming **Ex**ercises **I**nteroperability **L**anguage (YAPExIL) [3] is a language for describing programming exercise packages, which builds on top of the XML dialect PExIL (**P**rogramming **Ex**ercises **I**nteroperability **L**anguage) [4]. Comparatively, YAPExIL (1) is formalized through a JSON Schema rather than an XML Schema, (2) replaces the complex 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).

The two central goals of YAPExIL are (1) to fully cover Verhoeff's expressiveness model [5]; (2) to support the definition of different types of programming exercises beyond the traditional ones. Both of these goals have been achieved and proven [3]. In particular, it includes support for seven different types of programming exercises:

- `BLANK_SHEET`, which challenges the student to write her solution to a given problem statement starting from a blank sheet;
- `EXTENSION`, that asks the student to complete a partially finished solution source code, without modifying the provided parts;
- `IMPROVEMENT`, which provides a correct initial source code that does not yet fulfil all the goals set in the exercise specification (e.g., develop a solution with less than 2 loops), so the student has to modify it to solve the exercise;
- `BUG_FIX`, that challenges the student to fix a solution source code in which a few bugs have been planted;
- `FILL_IN_GAPS`, which provides code with missing parts and asks students to fill them with the right code;
- `SPOT_BUG`, which asks students to merely indicate the position of the bugs in a solution source code with a few bugs;
- `SORT_BLOCKS`, that asks students to sort the broken and shuffled blocks of code of a correct solution.

3 YAPExIL-to-MEF Converter

The conversion from YAPExIL to MEF has been developed as a JavaScript library[1], using version ES2017. The library exports two API methods for its clients, namely

`yapexil2mef(pathToZip, outputPath = 'output.zip', options = {})`, that reads a ZIP archive with a YAPExIL programming exercise located at `pathToZip` and writes a new ZIP archive at `outputPath` with a MEF programming exercise.

[1] `https://github.com/FGPE-Erasmus/yapexil-mef-converter`

`yapexil2mefStream(zipStream, outputStream, options = {})`, which receives an input stream `zipStream` from a ZIP archive containing a YAPExIL programming exercise and writes a MEF programming exercise to `outputStream`.

Both methods receive an `options` parameter containing a list of files to ignore and supported image and statement extensions.

Following YAPExIL facet structure, the actual conversion can be seen as a four-step process, including metadata, presentation, evaluation, and tools steps. In the metadata step, only the `title`, `module`, `type`, and `difficulty` map into MEF format, while `author`, `status`, `keywords`, `event`, and `platform` are left out (except if platform matches `-timeout N`, then `N` is set as the default problem timeout). The `title` is set as the `Name` of the MEF programming exercise. Moreover, it is also part of the `Title` field, which consists of a concatenation of the `module` with the `title` of the YAPExIL exercise. The `difficulty` maps almost directly to MEF's `Difficulty` as both are 5-level scales (i.e., from `beginner`, `easy`, `average`, `hard`, and `master` to `VERY_EASY`, `EASY`, `MEDIUM`, `DIFFICULT`, and `VERY_DIFFICULT`, respectively).

The `type` of exercise requires some extra work, as it is an important field in respect to the support for different types of exercises and has no related property in the target format. While the `BLANK_SHEET`, `EXTENSION`, `IMPROVEMENT`, and `BUG_FIX` exercises can be solved using a common code editor and only differ on the type of code skeletons provided by the author, the `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS` exercises may require appropriate user interface elements to be solved. Hence, from the evaluation engine perspective, the latter need to keep some sort of indicator about the kind of editor to display in the user interface, whereas the former uses the default editor. As MEF supports code, diagram, and quiz questions, there is already the property `Editor_kind` with a similar purpose, but a different set of possible values: `CODE`, `DIAGRAM`, and `QUIZ`. This set has been extended with three new values, namely `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS`, to maintain support for all exercise types. Table 1 summarizes the conversion rules applied in the metadata step.

Table 1 Conversion from YAPExIL metadata facet to MEF.

YAPExIL	MEF
title	Name; Title
module	Title
author	-
difficulty	Difficulty (map values)
status	-
type	Editor_kind (extend and map values)
keywords	-
event	-
platform	-

In the presentation step, the instruction files, Markdown and additional problem statements, and non-image embeddables are dropped out. Table 2 presents the conversion of YAPExIL presentation facet to MEF. A single problem statement, either HTML or text, goes to the `Description` field, plus a PDF statement which is saved in `PDF` field. Hence, at most two problem statements are kept. All embeddables with a supported image extension are stored as type `Image`. Finally, each `skeleton` is directly persisted in the equivalent type `Skeleton`, unless a `Template` for the same file extension exists in the evaluation facet.

Table 2 Conversion from YAPExIL presentation facet to MEF.

Element	YAPExIL	MEF
Instruction	M	-
Statement	M	1 (Description) + 1 (PDF)
Embeddable	M	M (only images)
Skeleton	M	M

In that case, `templates` are applied to solutions during the conversion, as they are not supported in Mooshak. The same happens with `solutions`, otherwise each `solution` is directly mapped into `Solution`. For both kinds of correctors, static and dynamic, the commands are merged into a large corrector and persisted into MEF, whereas the `library` files are ignored. In regards to tests, YAPExIL follows a tree structure, where intermediate nodes depict `testsets` and leaf nodes represent `tests`. In MEF, this tree is flattened into a list of `Test`. Table 3 outlines the key points of the conversion from the YAPExIL evaluation facet to MEF.

Table 3 Conversion from YAPExIL evaluation facet to MEF.

Element	YAPExIL	MEF
Library	M	-
DynamicCorrector	M	M (merged)
StaticCorrector	M	M (merged)
Solution	M	M
Template	M	applied to each Solution and Skeleton
TestSet	M	flattens to a list of Test
Test	M	M

The tools facet of YAPExIL is completely dropped out as its integration would require profound changes in MEF and, specifically, in Mooshak 2 with little gain.

4 Validation

The primary goal of the presented conversion is that the resulting MEF package can fully describe the same 7 types of programming exercises supported in YAPExIL. Traditional programming exercises only require a statement, tests, and a solution and, thus, support is kept. In addition to a `BLANK_SHEET` exercise, `BUG_FIX`, `EXTENSION`, `SPOT_BUG`, and `FILL_IN_GAPS` exercises require a skeleton, which is also part of MEF. An `IMPROVEMENT` exercise has the same requirements from the previous ones, but needs to evaluate other program metrics besides correctness, using either static or dynamic correctors. From Section 3, we see that corrector scripts are merged into a single script and, thus, there is no loss of value. Finally, the `SORT_BLOCKS` type has the particularity of requiring multiple skeletons (the blocks), which finds no barrier in MEF.

Besides the capability of representing each of the exercises' components, the user interface delivering the different types of exercises to the student must adapt according to the type of exercise. To this end, the format needs to keep either the type of exercise or an adequate editor to display. This is the role of `Editor_kind` of MEF, which has been extended with the values that require adaptations in the user interface (i.e., `FILL_IN_GAPS`, `SPOT_BUG`, and `SORT_BLOCKS`) beyond `CODE`.

Furthermore, an experiment to assess the library performance and functional correctness has been conducted with an example programming exercise package in YAPExIL format for each type of activity. All activities are variants, dependent on its type, of the same problem which challenges students to write a Python function that computes the derivative of $f(x) = x^b$ at point $x = m$, given b and m. The tests were developed using Jest, a JavaScript library for creating, running, and structuring tests for any project built using JavaScript or TypeScript. A total of 1000 runs has been performed on the test machine (Ubuntu 18.04.5 LTS 64 bits with i7-5820K CPU @ 3.30GHz x 12 and 32GB RAM) under the same conditions and their execution times registered separately for each type of activity. The conversion includes reading the archive from disk and write the new archive to disk. Figure 1 presents the a boxplot of the performance measurements. All tests were correct from the functional perspective.

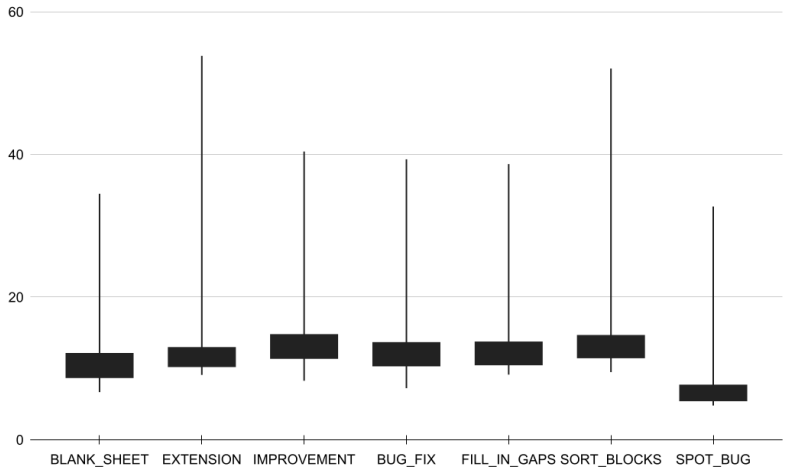

Figure 1 Boxplot of the performance measurements of the conversion for each type of activity (1000 runs). Times are in milliseconds.

On the one hand, the conversion of `SPOT_BUG` is the fastest as it contains only one test file with the line and column location of the bug, whereas all the others have six test cases. On the other hand, `IMPROVEMENT` and `SORT_BLOCKS` are the more time-consuming. The former for adding a static corrector to do the additional checks, the latter for having three skeletons rather than one.

5 Conclusions

Learning how to program relies on practice. Practice programming boils down to develop solutions to described problems, receive feedback on them, and improve them until they meet all requirements. Providing accurate and timely feedback to large classes is not a feasible task for teachers, who often resort to automated assessment tools to mitigate this issue. However, each tool typically adheres to its own programming exercise format and has no compatibility with other formats, precluding the share and reuse of exercises among institutions. Furthermore, the exercises proposed are always homogeneous, only varying the problem statement and requirements, i.e., the student is challenged to code a solution to a given problem statement, starting from an empty file.

This paper presents a JavaScript library to convert programming exercise packages adhering to YAPExIL, a JSON format supporting support for seven types of programming activities (blank-sheet, solution improvement, bug fix, gap filling, block sorting, and spot the bug), into packages complying with MEF, the internal format of Mooshak. The resulting package has no loss regarding the capability to describe each type of activity, requiring a minimal change to MEF (more options in an enumeration). Moreover, the library is already integrated into a collaborative web programming exercises editor, developed as open-source software [2], to allow exporting programming exercises directly as MEF packages, as presented in Figure 2.

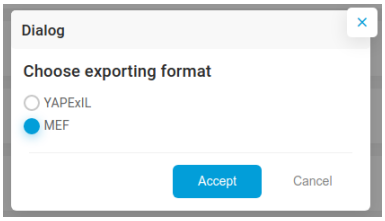

Figure 2 Export exercise dialog in FGPE AuthorKit.

The work described in this paper is part of the Framework for Gamified Programming Education (FGPE) project. This project includes the creation of about 500 programming exercises, most of which are already available online and are now supported by Mooshak 2. The consecutive next steps are to finish the gamified learning environment presenting such exercises to students.

References

1 José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.

2 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.

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 Ricardo Queirós and José Paulo Leal. Making programming exercises interoperable with PExIL. In José Carlos Ramalho, Alberto Simões, and Ricardo Queirós, editors, *Innovations in XML Applications and Metadata Management*, pages 38–56. IGI Global, 2013. doi:10.4018/978-1-4666-2669-0.ch003.

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