An Editor for the ProFormA Format for Exchanging Programming Exercises

Uta Priss[1], Karin Borm[2]

Abstract: This paper describes an editor for an XML format for the creation and exchange of automatically evaluated programming exercises. It provides a brief description of the editor, its requirements and some of its challenges.

Keywords: automatically evaluated programming exercises, ProFormA format, teaching programming, XML exchange format

1 Introduction

The ProFormA format was developed as an XML format for the exchange of programming exercises that are automatically evaluated by computer software [St15]. In introductory programming classes it is becoming quite common to use software that automatically generates feedback for students. Traditionally such feedback was provided by (human) tutors. But in any large or even just moderately-sized class, it is virtually impossible for tutors to provide individualised and comprehensive feedback without a significant temporal delay. Although standard software engineering tools already provide a fair amount of feedback (for example compiler output) such feedback tends to be insufficient and potentially incomprehensible for novice programmers. Furthermore in order to evaluate not just syntax and run-time parameters but also whether students are learning programming concepts, lecturers may want to design specific tests that check whether a task has been implemented according to a specification. This can be achieved by writing unit tests which specify test cases and produce hints or error messages in case the students' code does not appear to achieve the desired goals. Thus automatic assessment of student submitted code mostly consists of compiling a set of standard software engineering tests and specifically designed unit tests. This topic is well researched in the literature and many tools for the purpose of automatic assessment of programming exercises already exist (cf. the review [Ih10] and the more recent [Bo17]).

Unfortunately, creating such exercises is time consuming. We estimate that it takes at least two hours to convert a textbook exercise into an automatically evaluated format because the

[1] Ostfalia University of Applied Sciences, www.upriss.org.uk
[2] Ostfalia University of Applied Sciences, k.borm@ostfalia.de

task description requires a higher precision, the unit tests need to be written and the correct functionality of the exercise needs to be well tested. Furthermore the initial learning curve for lecturers or tutors for how to use a software tool for creating programming exercises can be high. Thus, being able to share exercises with colleagues and being able to reuse exercises from a pool of exercises or repository can be an incentive to use such a system. Because different universities employ different learning management systems (LMS) for the delivery of exercises and LMS change over time, it is essential that exercises can be ex- and imported from LMS and repositories. The aim of the ProFormA format is therefore to facilitate sharing of automatically evaluated programming exercises across different tools and settings. The ProFormA format is an XML standard provided via an XSD schema file[3]. It was developed by a team of researchers and developers from several northern German universities. Further details about the ProFormA format can be found in Strickroth et al. ([St15], [St17]) and shall not be repeated in this paper.

The ProFormA format is already in use by at least five German universities and is expected to draw a larger user community over time [St17]. Each of the five universities is currently employing a different LMS which each has its own built-in editing facilities. At Ostfalia University the Praktomat software[4] which is a grader for student submitted code has been connected to the LMS LON-CAPA[5] which manages course and student data and provides a repository of exercises. Because LON-CAPA is a networked system that is used by many universities worldwide, its code cannot be modified without obtaining approval from the community. This can be a fairly slow process. While the connection between the Praktomat software and LON-CAPA was feasible without LON-CAPA code modifications via a middleware, it is currently not possible to modify the built-in editor of LON-CAPA. Thus the need arose to build a stand-alone editor[6] that requires only HTML and JavaScript and can be incorporated into LON-CAPA but also into any other similar web-based LMS or repository without major code modification. This editor is discussed in this paper. Section 2 provides a more detailed description of the editor and its requirements. Section 3 discusses some challenges and observations.

2 Requirements and Description of the Editor

This section briefly describes the editor, its requirements and some of its design decisions. Figs. 1-3 provide screenshots of the editor for a very simple “HelloWorld” Java program. The task description in Fig. 1 is formated as HTML because LMS tend to be web-based and require HTML task descriptions. HTML support is provided in the editor by code-highlighting, tag-completion and HTML preview. It would be possible to add a WYSIWYG editor to the HTML textfield if desired but the target user group (computer science lecturers)

[3] https://github.com/ProFormA/taskxml

[4] https://github.com/danielkleinert/Praktomat

[5] http://www.lon-capa.org

[6] A demo-version of the editor is available at http://media.elan-ev.de/proforma/editor/editor.html and the code is available via http://github.com/ProFormA/formatEditor.

might prefer a source editor. Other parameters shown in Fig. 1 pertain to the title, the language of the task description, the choice of programming language, parameters of the LMS and of the file that is to be submitted. In this example, the editor is configured for use at Ostfalia university where the LMS is LON-CAPA. The bottom two parts (about the submission and LON-CAPA) might need to be modified if a different LMS was to be used. LON-CAPA support can be included or omitted by editing the configuration file of the editor.

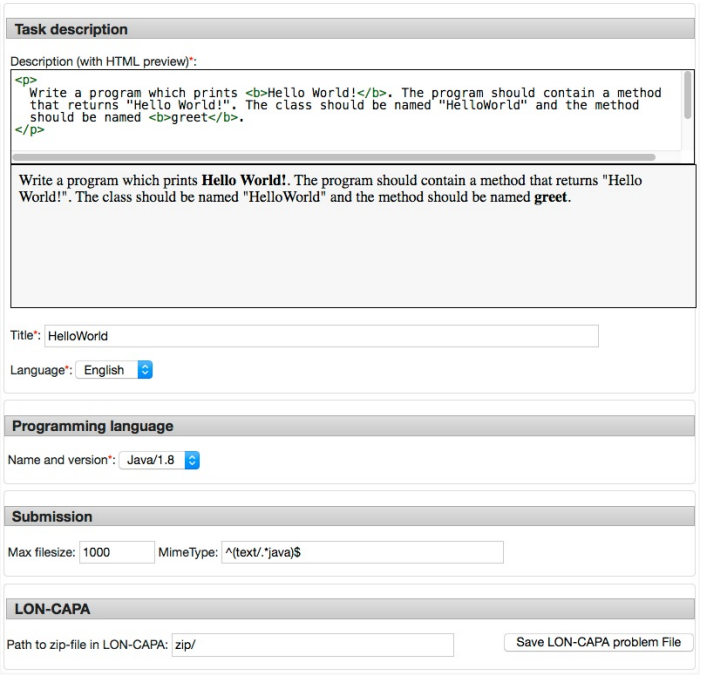

Fig. 1: Task description and general parameters

Fig. 2 shows an example of a model solution and a JUnit test. Files can be added via drag and drop, upload or copy/paste. The textareas that display the texts of the files provide some editing support using the CodeMirror JavaScript library. It is more likely that files will be created and tested in an IDE and just uploaded to the editor than that they are created with the editor. But if an already existing task is edited it can be convenient to modify the files within the editor instead of within the IDE. The attribute “class” corresponds to the choices that are available in the ProFormA format. In this case, both files are internal and therefore not shown to the students. Other choices are “template” for partially complete code, “library” and “instruction” which would all be shown to the students as part of the task description in the LMS.

Fig. 3 displays two tests for the “HelloWorld” example. Most of the parameters of the

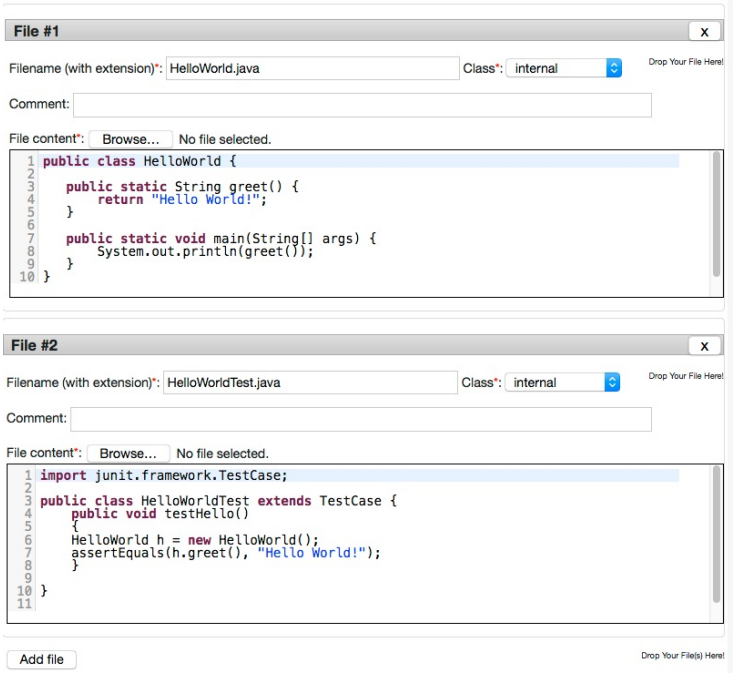

Fig. 2: Example of a model solution and unit test

compilation test shown in this example depend on the specific grading engine that is employed by the LMS. Because there tends to be a commonly used unit test framework for each programming language, parameters for unit tests tend to be fairly standard across different grading engines. The connection between the files (in Fig. 2) and the tests (in Fig. 3) is accomplished via IDs in the ProFormA format. The editor hides these IDs and establishes a connection via filenames instead.

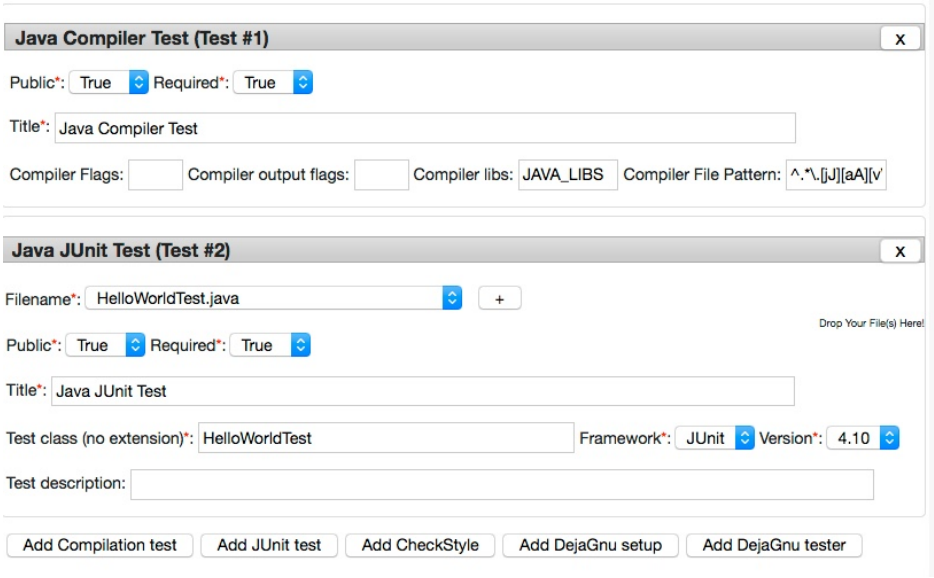

Fig. 3: Tests consist of parameters and references to files

The following requirements were identified for the editor:

1. Provide a web form for data entering
2. Generate the ProFormA task.xml (as XML or zip archive) from the web form data
3. Import a previously created task.xml into the web form (including import of older versions of the ProFormA format which are then converted to the current format)
4. Generate a file for each exercise as required by the LMS (e.g. LON-CAPA)
5. Advanced editing support (file upload, syntax highlighting of source code, auto-fill in options, drag and drop, menus, etc)
6. Form validation (check for completeness and consistency and schema validation of the generated task.xml)
7. Extensibility and modularity (new features, tests and programming languages should be easy to add)

8. Adaptability (should be easy to adapt with respect to other LMS and graders)

The requirements 1, 2, 5 and 6 are met by the editor. Requirement 3 is currently met for two versions of the format. Requirement 4 is met with respect to LON-CAPA. It is planned to support Moodle in the near future. In particular because of requirement 8, JavaScript is a suitable language in which to write the editor. Because LMS tend to be web-based, JavaScript code can be incorporated into any LMS that supports the ProFormA format. Although JavaScript is executed on the client, it requires no client-side installation and it is straightforward to deploy updates by changing the server-side copy of the code. Because JavaScript is a scripting language, users can fairly easily adjust the code to their own needs as long as the code is well structured and documented. The editor uses standard libraries (JQuery, JQuery-UI, CodeMirror, zip.js and xmllint.js). A few regression testing scripts are supplied with the editor for testing with Python, Selenium, Chrome and Firefox.

Requirements 7 and 8 were not well met by the initial versions of the editor. Recently the code has been modified to contain a separate configuration file. All information about versions, XML namespaces, test attributes that do not apply to all tests and graders or LMS specific meta-data has been gathered into this configuration file. Thus it is now much easier to change existing tests or to add new types of tests (for example in order to add a new programming language or framework) and to modify test parameters (for example for new versions of component software).

3 Challenges and Observations

At first sight, writing an editor that corresponds to an XML format should be completely straightforward. Our first attempt was actually to use a JQuery XML editor[7] which automatically generates a web form directly from an XML schema. Unfortunately, it appears that the nesting, structuring and use of XML elements and attributes in the ProFormA format does not necessarily correspond to the sequence in which lecturers create exercises. Thus the automatically generated web form was not user friendly. A fair amount of the data in the ProFormA format pertains to hidden background knowledge and dependencies, for example which tests are available for which programming languages. Such background knowledge is not encoded into the ProFormA format. Thus a file can be a valid ProFormA XML file but nevertheless contain a combination of programming language and tests which is not likely to be supported by any LMS or grader. For the editor we tried to assist users by adjusting some of its menus and auto-completion features based on such background knowledge in order to speed up the process of exercise creation and to avoid detecting errors only after the exercises have been uploaded into the LMS. The advantages of automatically evaluated programming exercises have to outweigh the amount of time and effort required for creating them. Thus unless the editor makes exercise creation seamless, it will not be used by many lecturers.

[7] https://github.com/UNC-Libraries/jquery.xmleditor

Some challenges still remain, in particular with respect to testing the editor. Theoretically a very simple test should be to import a file into the editor, save it, export it and compare it to its original version. But even plain XML file comparison requires use of some tools because the sequence of attributes (and potentially elements) in XML is not always determined, the ProFormA format contains some optional elements and attributes and browsers themselves may modify XML when they are reading and writing it. Even though some regression tests with Selenium[8] have been implemented, systematic testing of the editor still needs to be expanded. Some of the more advanced JavaScript functions are not equally supported by all types of browsers. For example, being able to create zip files from uploaded data is not necessarily cross browser compatible. Automated cross browser testing is still a challenge.

Our experience with users at Ostfalia so far has been positive. Users are aware that without the editor the exercises would need to be created manually by both configuring the grader and the LMS. We usually provide a couple of training sessions for new users who want to create programming exercises in order to explain all the steps that are required. There are still a number of features that should be improved. For example, the error messages that are displayed in the LMS in case there is a technical problem with an exercise are sometimes not sufficiently informative. Because of the complexity of the system, run-time errors can occur from within the LMS (for example if the file that executes an exercise contains an incorrect path for the task.xml file), within the grader (for example, German umlauts within source code sometimes cause problems) or potentially within the middleware (so far we have not encountered any run-time errors arising from the middleware, but it could happen). In any case, it can be difficult for inexperienced users to determine and fix the exact cause of an error. It is a question as to how much of this information belongs into the editor and how much belongs into the grader and LMS. It is best if problems within exercises are caught as early as possible during the development process of an exercise. It can be frustrating if errors occur only after the exercises have been implemented in the LMS because modifying an exercise from within the LMS still involves a number of steps.

A convenient feature of the editor is that it can read task.xml files in some older versions and convert them into the current version. Unfortunately, so far the editor has mostly been configured for use at Ostfalia. Thus, namespaces that are required by graders at other sites are not yet sufficiently supported. The editor will still read ProFormA files that contain other namespaces but it will ignore any data pertaining to namespaces for which it does not currently have a schema.

Acknowledgements

This work was partially supported by the German Federal Ministry of Education and Research (BMBF) under Grant 01PL16066H. The sole responsibility for the content of this paper lies with the authors.

[8] http://www.seleniumhq.org

References

[Bo17] Bott, Oliver; Fricke, Peter; Priss, Uta; Striewe, Michael (eds.). Automatisierte Bewertung in der Programmierausbildung. Digitale Medien in der Hochschullehre, ELAN e.V., Waxmann 2017.

[Ih10] Ihantola, Petri, et al. Review of recent systems for automatic assessment of programming assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research. ACM, 2010.

[St15] Strickroth, S.; Striewe, M.; Müller, O.; Priss, U.; Becker, S.; Rod, O.; Garmann, R.; Bott, O. J.; Pinkwart, N. (2015). ProFormA: An XML-based exchange format for programming tasks. eleed, 2015, 11.

[St17] Strickroth, S.; Müller, O.; Priss, U. (2017) Ein XML-Austauschformat für Programmieraufgaben. In: Bott; Fricke; Priss; Striewe (eds), Automatisierte Bewertung in der Programmierausbildung, Waxmann 2017.