


Raccode: An Eclipse Plugin for Assessment of Programming Exercises

André Silva

Faculty of Sciences, University of Porto, Portugal


up201007410@fc.up.pt

 <https://orcid.org/0000-0002-7663-2456>

José Paulo Leal

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal


zp@dcc.fc.up.pt

 <https://orcid.org/0000-0002-8409-0300>

José Carlos Paiva

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal

up201200272@fc.up.pt

 <https://orcid.org/0000-0003-0394-0527>

Abstract

IDEs are environments specialized in support during the development of programs. They contain several utilities to code, run, debug, and deploy programs quickly. However, they do not provide the automatic assessment of programming exercises, which is required in both learning and competitive programming environment. Therefore, IDEs are often underestimated in these contexts and replaced by basic code editors. Yet, IDEs have unique features which are essential for programmers, such as the debugger or the package explorer. This paper presents Raccode, a plugin for assessment of programming exercises in Eclipse. This plugin integrates with Mooshak to combine the diverse capabilities of an IDE, like Eclipse, with the automatic evaluation of exercises, clarification requests, printouts, balloons, and rankings. It can be used both in competitive and learning environments. The paper describes Raccode, its concept, architecture and design.

2012 ACM Subject Classification Software and its engineering → Integrated and visual development environments

Keywords and phrases automatic evaluation, programming, IDE, learning, competition

Digital Object Identifier 10.4230/OASICS.SLATE.2018.4

Category Short Paper

Funding This work is partially funded by the ERDF through the COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, by National Funds through the FCT as part of project UID/EEA/50014/2013, and by FourEyes. FourEyes is a Research Line within project “TEC4Growth – Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact /NORTE-01-0145-FEDER-000020” financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF).



© André Silva, José Paulo Leal, and José Carlos Paiva;
licensed under Creative Commons License CC-BY

7th Symposium on Languages, Applications and Technologies (SLATE 2018).

Editors: Pedro Rangel Henriques, José Paulo Leal, António Leitão, and Xavier Gómez Guinovart

Article No. 4; pp. 4:1–4:8



Open Access Series in Informatics

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

1 Introduction

Integrated Development Environments (IDEs) are applications specialized in supporting programmers during the development of software, either simple or complex. They are designed to include all programming tasks in a single application. Therefore, IDEs provide a central interface containing every tool that a developer should need, including a code editor, a package explorer, a compiler, a debugger, and several build automation tools.

In competitive and learning programming environments, the automatic assessment of submitted programs is essential. Students and contest participants need timely feedback on their attempts, that can not be guaranteed by human judges. Several tools have been developed to provide this feature, which generally include embedded code editors trying to mimic IDEs. Hence, IDEs are often underestimated and set aside in these contexts to streamline the process.

However, this decision can have very negative outcomes. Besides not adapting future programmers to the tools that they will have to use later, it can slow down the development of the right solution to an exercise. For instance, the debugger can help to find bugs in programs, by running it step by step, stopping at some event or specified instruction (i.e., a breakpoint), and tracking the values of variables.

This paper presents Raccode, an Eclipse plugin that combines the several features of an IDE with the key characteristics of a tool that provides automatic assessment of programming exercises both in competitive and learning environments. Raccode integrates the REST API of Mooshak 2.0, providing Eclipse with automatic evaluation of programming exercises, support for clarification requests, tracking of printouts and balloons, rankings list view, among many others.

Eclipse is one of the most used IDEs among software developers. As an open source software, developers can contribute to the project or share their own products in the form of a plugin [10]. Eclipse has an environment called Eclipse Marketplace that provides an extensive listing of Eclipse-based solutions, such as plugins and bundles, which can be installed directly from the workspace using the Marketplace Client. These features together with the environments that Eclipse offers, such as Rich Client Platform – RCP [5, 3] – and Plug-in Development Environment – PDE [6] –, have led us to choose it as the first IDE to integrate Raccode. However, support for other IDEs is already planned.

The remainder of this paper is organized as follows. Section 2 reviews some systems that combine automatic evaluation with IDE-like features. Section 3 describes Mooshak and details its REST API, included in version 2.0. Section 4 presents Raccode, its concept, architecture, and design. Finally, Section 5 summarizes the main contributions of this work and next steps.

2 State of the Art

There are several online platforms and tools, such as CodeChef¹, CodinGame², or Enki [8], that combine automatic assessment of programming exercises with an IDE-like user interface and some of its features. However, they just include a very limited set of features when compared to IDEs, including a code editor with a weak support for code completion and a console log where the output is displayed.

¹ <https://codechef.com/>

² <https://codingame.com/>

CodeChef is an online competitive programming platform. It supports more than 35 programming languages, including Java, Javascript, C, C++, Python, and Pascal. The development environment contains a code editor, a selector for programming language, and a widget to provide custom inputs. The platform has two modes: practice and contest. The practice mode categorizes problems by difficulty, allowing users to solve at their own rhythm. The contest mode proposes a series of problems for participants to solve under specific time constraints. Users are ranked according to the number of problems solved, breaking ties with the total amount of time spent solving them. Every month more than 30 programming contests are realized. Users can award ranking points in both modes.

CodinGame is an online platform where programmers can learn and compete through game-based challenges. Most of these challenges require the user to develop a software agent to control the behavior of a character in a game environment, and provide a 2D game-like graphical feedback. The agent programmed by the player must pass all test cases (public and hidden) to solve the puzzle. Players can choose one of the more than 20 programming languages available. Once the exercise is solved, players can access, rate, and vote on the best solutions. The interface to develop agents presents the statement of the exercise on the left as well as the movie player, and the code editor and test cases on the right. The widget containing the test cases supports custom input and displays the output log, once the test runs.

Enki is a web-based learning environment with an IDE-like graphic user interface. It integrates with several kinds of tools. These tools include a gamification service to provide gamification features to students, an educational resources sequencing service to offer different learning paths, and an evaluator engine to give automatic feedback to students' solutions. The user interface includes windows for a code editor, a console log output, an error list, a test case editor, and a ranking list.

3 Mooshak 2.0 REST API

Mooshak [4] is a web-based system for automatic assessment in computer science. It was primarily designed for managing programming contests, such as ICPC contests, but the need for automatic assessment in pedagogical contexts has led to its adoption in computer science education. Since then, Mooshak assists educators in programming courses, providing instant feedback on practical classes and exams.

Recently, Mooshak 2.0 has been released. This version is a complete reimplementaion in Java and Google Web Toolkit (GWT) of the initial codebase. However, it also adds several new features, including a learning environment – Enki [8] –, a diagram assessment environment – Kora [1] –, and a REST (Representational State Transfer [2]) API.

The REST API of Mooshak 2.0 uses Jersey³, an open-source framework that is the reference implementation of the Java API for RESTful Web Services, extending it with several features to further simplify RESTful service. Jersey provides a Core Server to build annotation-based RESTful services, and to support JSON and the Java Architecture for XML Binding. It also includes a Core Client to facilitate the communication with REST services.

Mooshak 2.0 REST API contains endpoints for authentication and authorization (`auth`), `contests`, `problems`, `questions`, `printouts`, `balloons`, `languages`, and `submissions`. Most of the endpoints consume and produce JSON and XML, but some require other

³ <https://jersey.github.io/>

■ **Table 1** Main endpoints of the REST API of Mooshak.

Method	Endpoint	Consume	Produce	Description
POST	auth/login		JSON/XML	Authentication into a contest
GET	data/contests/contestId/rankings		JSON/XML	View rankings of a contest
GET	data/contests/contestId/problems		JSON/XML	List all problems of a contest/course
GET	data/contests/contestId/problems/problemId/view		JSON/XML	View a problem of a contest/course
POST	data/contests/contestId/problems/problemId/evaluate	form-data	JSON/XML	Evaluate a program of a contest/course
GET	data/contests/contestId/languages		JSON/XML	List all languages of a contest/course
GET	data/contests/contestId/languages/languageId		JSON/XML	Get a language of a contest/course
GET	data/contests/contestId/submissions/submissionId/evaluation-summary		JSON/XML	Get the summary of an evaluation
GET	data/contests/contestId/questions	JSON/XML	JSON/XML	List all questions of a contest/course
POST	data/contests/contestId/questions	JSON/XML	JSON/XML	Create a question in a contest/course
PUT	data/contests/contestId/questions/questionId	JSON/XML	JSON/XML	Update a question in a contest/course
POST	data/contests/contestId/printouts	form-data	JSON/XML	Create a printout in a contest
POST	data/contests/contestId/balloons	JSON/XML	JSON/XML	Create a balloon in a contest

formats, such as Form Data (e.g., the evaluation endpoint receives a file as input). The most important endpoints are summarized in Table 1.

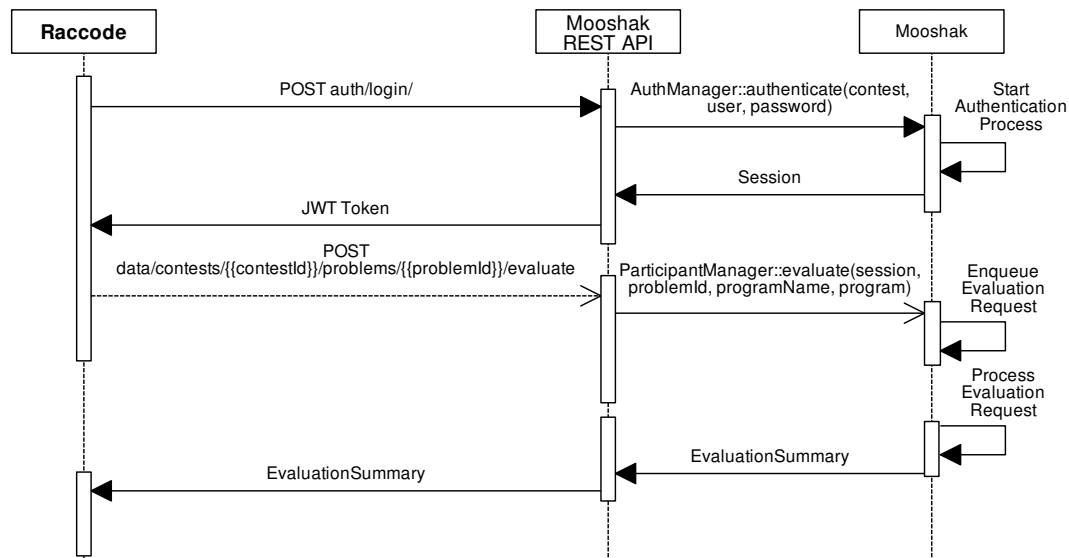
The authentication to the API uses JWT (JSON Web Token)⁴, a compact URL-safe form of representing claims that are transferred between two parties. Once the REST endpoint for login receives a request from a client, it extracts the user name, password and domain to which the user is attempting to connect, and leverages the work on the `AuthManager` which checks the credentials against the database. If it succeeds, a JWT Token is generated and sent back to the client, otherwise an error is returned. Thereafter, all requests must have the JWT Token in the `Authorization` header.

4 Raccode

The operation of the Raccode can be summarized by the sequence diagram presented in Figure 1. It starts by connecting to a server, using an host and port defined in Eclipse preferences. If no problem occurs and the connection is successful, this data is stored in the registries, so that it is not necessary to configure the server every time the plugin is started. From this moment, the user can proceed with the authentication in a contest.

There are two ways to login into a contest: via Eclipse wizards menu, following the path `File -> New -> Other... -> Raccode -> Login`; or just clicking on the button present on the toolbar with the label `New Problem`. If the user chooses a problem from a different contest, he needs re-authenticate. Nevertheless, the token given by server is also stored in a registry and, if it is the same among different contests, it is just needed to select the other

⁴ <https://tools.ietf.org/html/rfc7519>



■ **Figure 1** UML sequence diagram of an evaluation.

contest and click **Login**. If the problem belongs to the same contest, no action is required since Raccode keeps the JWT token given to the user for an hour, refreshing it when the time expires but the user remains active.

The second page of the login wizard presents a menu with the available problems and languages, allowing the user to select a problem. When pressing the **Finish** button, the project is created and added to Package Explorer. If the local machine does not have the selected language installed, an error message is presented explaining the situation and the project is not created.

The Problem view presents the problem statement, either in HTML or PDF. While solving the problem, the user can test his program with test cases received from the server (public tests) or its own test cases (user-defined tests). If the outputs do not match, he can always use the debugger to identify the problem.

Finally, to submit the code the user can use the menu **Raccode** -> **Submit** or click on the **Submit** button on the Eclipse toolbar. The program is then sent to the evaluator on server and a summary is returned, giving the user feedback and informing him if his program was accepted or not.

Raccode presents several tools on the perspective, including the ranking of the contest, the progress of the user (e.g., the number of attempts for the current problem, problems solved, among others), a listing of questions and answers of the selected problem as well as a means to submit questions, and listings for balloons and printouts if it is a contest.

In a general way, the Raccode plugin makes the **bridge** between Mooshak 2.0 and Eclipse. The following subsections present how Raccode integrates with the Mooshak 2.0 REST API and its design.

4.1 API integration

Mooshak has a REST API that enables a client to send HTTP requests to the server and get information easily. The documentation of the API describes how requests should be made, providing an example of a request and a response for each endpoint. Raccode consumes this API exactly as documented, displaying a message in a label or in a pop-up window if an exception occurs while making a request.

The project structure is split into two parts: one for requests and another for the User Interface (UI). By dividing the project into these two parts, it provides extensibility for other IDEs, since the API integration can be reused. The only code that needs to be modified is the UI part. The HTTP requests are performed using `URLConnection` library. The UI was developed using Standard Widget Toolkit (SWT) [9], a graphical widget toolkit to use with the Java platform. The aspect of the perspective is described in the next subsection.

The `request` package is divided in several classes corresponding to the resources being consumed from the REST API. Each of these classes leverages the request on the adequate method of the `RequestSender`, depending on the HTTP method being used. For instance, the `Auth` class contains methods to authenticate/authorize users. The `login` method invokes the `post` method of the `RequestSender`, which issues the request and processes the response to JSON, returning it. If everything goes fine the user gets logged in, and the response contains a token with a duration of an hour. During this time, every time the user submits a solution or wants to choose a new problem, he doesn't need to log in again. In the same way, if the user doesn't close the application or make logout past this hour, the token is only refreshed and the user can keep going to work. This is how Raccode works directly with Mooshak, making requests to obtain all resources needed for the good functionality of the plugin.

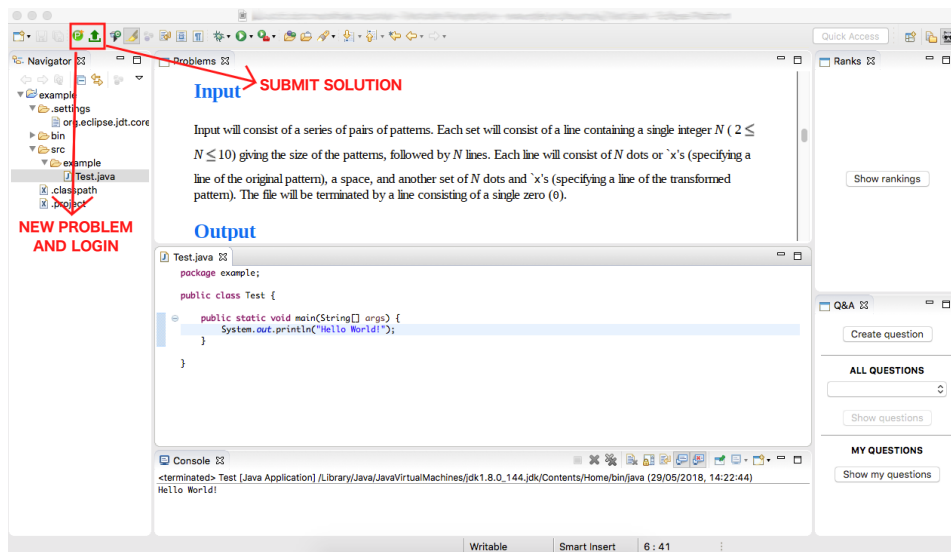
4.2 Design

In terms of design, Raccode uses a perspective to present the many tools needed to solve a problem. The default design aspect of Raccode includes a text editor, a list of problems, a package explorer, the console, the rankings list, and a Q&A (questions and answers) window. The toolbar of the perspective adds two buttons, one to get a new problem and another to submit a solution. As previously described, the new problem button opens a wizard that asks for user authentication, if it is not authenticated yet. Once logged in, and continuing in the same wizard, all problems in the contest become available and the user can choose which of them wants to solve, and in which language.

Furthermore, there is a page in Eclipse preferences to configure the connection to the server, in which Mooshak is added by default. Figure 2 presents the distribution of the several views, which were strategically positioned to fit the common Eclipse perspectives. Three of these windows are recognizable by any Eclipse user: package explorer, text editor, and terminal (console). The important here is to present the other three. The `Problem` view shows the problem statement, with the description, and some examples of input and output. The `Ranking` view allows the participant to have a notion of his progress (which problems solved with success, how many submissions were made, which problems remain to be resolved, etc). Furthermore, it is also useful to know the progress of the other participants, since it is possible to make competitions using Raccode. Each submission made by a participant is only accepted to run on the evaluation machine if the program has some difference from the previous program. After compiled and tested, a feedback message is returned, which can be `Accepted`, `Runtime Error` or `Compile Time Error`, for example.

When a participant of a contest has some doubts about a problem, he can submit a question through the `Q&A` view for teachers/judges to clarify their issues. If Mooshak is in the learning mode, other students can also answer the questions from their peers also.

Also, an a how-to manual about the work environment is provided, with information about submission of programs, the meaning of the feedback messages, troubleshooting, among others.



■ **Figure 2** Raccode perspective: distribution of the components (open to changes).

5 Conclusions

At the moment, Raccode is a work in progress. The connection between the REST API of Mooshak 2.0 and Eclipse is almost completed, only missing the submission and automatic creation of project parts. The design of the application is also complete with all views and wizards created, as well as the integration of Eclipse tools in the perspective.

Raccode has some issues with language verification. For instance, when the user wants to solve a problem in C++, it can not verify if the user has the required libraries installed to compile it.

Regarding improvements, the goal of Raccode is to integrate with other IDEs, since the market for IDEs is quite large. Another future improvement concerns the automatic installation of the required environment for compiling programs exactly as in Mooshak.

Raccode will be tested in an open environment with students from the Department of Computer Science of the Faculty of Sciences of the University of Porto (DCC-FCUP). This experiment aims to compare Enki with Raccode environment, in a small open course with a series of problems. The feedback will be taken through an online questionnaire based on the Nielsen's model [7], in Google Forms.

References

- 1 Helder Patrick de Pina Correia. Avaliação de diagramas no Mooshak 2.0. Master's thesis, Universidade do Porto, 2017.
- 2 Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.
- 3 Andreas Kornstadt and Eugen Reiswich. Composing systems with Eclipse rich client platform plug-ins. *IEEE Software*, 27(6):78–81, 2010.
- 4 José Paulo Leal and Fernando Silva. Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581, 2003.
- 5 Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse rich client platform*. Addison-Wesley, 2010.

- 6 Wassim Melhem and Dejan Glozic. PDE does plug-ins. Technical report, IBM Canada Ltd., 2003.
- 7 Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 206–213, 1993.
- 8 José Carlos Paiva, José Paulo Leal, and Ricardo Alexandre Queirós. Enki: A pedagogical services aggregator for learning programming languages. In *Conference on Innovation and Technology in Computer Science Education*, pages 332–337, 2016.
- 9 Matthew Scarpino, Stephen Holder, Stanford Ng, and Laurent Mihalkovic. *SWT/JFace in action*. Manning, 2005.
- 10 Lars Vogel. *Contributing to the Eclipse IDE Project: Principles, Plug-ins and Gerrit Code Review*. Lars Vogel, 2015.