


Moozz: Assessment of Quizzes in Mooshak 2.0


Helder Correia

CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal
up201108850@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

Quizzes are a widely used form of assessment, supported in many e-learning systems. Mooshak is a web system which supports automated assessment in computer science. This paper presents Moozz, a quiz assessment environment for Mooshak 2.0, with its own XML definition for describing quizzes. This definition is used for: interoperability with different e-learning systems, generating HTML-based forms, storing student answers, marking final submissions and generating feedback. Furthermore, Moozz also includes an authoring tool for creating quizzes. The paper describes Moozz, its quiz definition language and architecture, and details its implementation.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases quiz, automated assessment, authoring, XML, feedback, e-learning

Digital Object Identifier 10.4230/OASICS.SLATE.2018.3

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).

1 Introduction

Mooshak [5] is a web system that supports automated assessment in computer science. It evolved from a programming contest management system, supporting different contest models, to a pedagogical tool used in introductory computer science courses. Although Mooshak was initially targeted for text-based computer programming languages, it was later extended to support visual languages, such as EER (Extended Entity-Relationship) and UML (Unified Modeling Language).

Quizzes are a widely used form of assessment, not only in computer science, and they are widely supported among e-learning systems. Quizzes can also be used in computer science



© Helder Correia, 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. 3; pp. 3:1–3:8



OpenAccess Series in Informatics

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

contests, in particular in those targeted to younger students, to develop their computational thinking skill, as is the case of Bebras [2]. Hence, a system supporting automated assessment in computer science, both in competitive and pedagogical settings, should also have a quiz assessment environment.

In fact, the previous version of Mooshak already has an incipient form of quiz assessment. Nevertheless, it lacks support for several types of questions, standard quiz interoperability languages, and integration with other assessment modes. Meanwhile, version 2.0 has a pedagogical environment, named Enki [6], that integrates different kinds of assessment in a single course, including code and diagram assessment.

This paper presents Moozz, an assessment environment for quizzes integrated into Mooshak 2.0. Moozz supports all the standard question types, including multiple choice (select one or multiple), gap filling, matching, short answer, and essay, as well as questions with media files (e.g. images, sounds, and videos). Moozz uses its own XML definition for describing quizzes, named Moo, providing eXtensible Stylesheet Language Transformations (XSLT) to convert to and from other quiz formats, such as IMS Question and Test Interoperability specification (QTI) [8] and the GIFT format¹.

An authoring tool is also included in Moozz to facilitate the creation of quizzes. This tool allows to import and export quizzes in the supported formats, insert multimedia content, and add questions, answers and feedback messages.

The remainder of this paper is organized as follows. Section 2 presents Mooshak 2.0, and the formats and standards supported by Moozz. Section 3 describes the architecture of Moozz and details its implementation. Finally, Section 4 summarizes the contributions of this work and identifies opportunities for future developments.

2 Background

This paper presents Moozz, a quiz assessment environment that aims to integrate in the pedagogical tool of Mooshak 2.0, named Enki [6]. This environment has several features, such as the compatibility with IMS QTI specification and the GIFT format, and the support for Bebras quiz competitions. This section aims to provide some background on Mooshak, particularly on the tool in which Moozz integrates, and to describe the supported formats.

2.1 Mooshak 2.0

Mooshak [5] is a system for managing programming contests on the web, which provides automatic judging of submitted programs, submission of clarification requests, reevaluation of programs, tracking of printouts, among many other features. It supports a wide range of programming languages, such as Java, C, VB, and Prolog.

Even if Mooshak was initially designed to be a programming contest management system for ICPC contests, educators rapidly found its ability to assist them in programming courses [3] to give instant feedback on practical classes, evaluate and mark assignments, among other uses. This has motivated the development of several extensions specifically for learning, such as a plagiarism checker and an exam policy.

Recently, Mooshak was completely reimplemented in Java with Graphic User Interfaces (GUIs) in Google Web Toolkit (GWT). Besides the changes in the codebase, Mooshak 2.0 gives special attention to computer science learning, providing a specialized computer science languages learning environment – Enki [6] –, which not only supports exercises using typical programming languages, but also diagramming exercises.

¹ https://docs.moodle.org/25/en/GIFT_format

Enki blends assessment and learning, integrating with several external tools to engage students in learning activities. Furthermore, Enki's GUI mimics the aspect of an IDE, and attempts to achieve their powerful extensibility. As a typical IDE, such as Eclipse and NetBeans, the GUI of Enki is divided into regions, each one containing several overlapping windows, organized using tabs. These regions are resizable, and their windows can be moved among different regions. A window holds a unique component, capable of communicating with other components. Therefore, it is possible to add any number of components required by a specific assessment environment and link them to the evaluation engine with relative ease. This has already been done for the diagram assessment.

2.2 Question and Test Interoperability (QTI)

The IMS Question and Test Interoperability (QTI) specification describes a data model for representing question and test data, as well as their corresponding results. This specification enables authoring and delivering systems, question banks, and Learning Management Systems (LMSs) to exchange question and test data [8]. Among other things, this common format can facilitate populating question banks, transmitting results and information about the learner between the various components of an e-learning ecosystem, and incorporating questions designed by several IMS QTI users into a single assessment.

The IMS QTI uses XML to store information about assessments. Its data model can be seen as a hierarchy of elements whose contents and attributes are XML tags [7]. There are three key elements in this model: **assessment**, **section** and **item**. The **assessment** element contains a set of questions, which can be organized using **section** elements. The **section** element indicates a group of questions, enabling authors to separate each subtopic and calculate the score obtained for each section as well as the overall score. An **item** is a question with all the associated data, such as score, answers, layout and feedback.

The results of the IMS QTI are specific to a participant, but can contain data of more than one assessment. The core data structures for reporting results are the **summary**, which contains global statistics of the assessment, such as the number of attempts, and the results of the internal tree of the **assessment**, **section** and **item** elements.

2.3 Bebras

Bebras is a community building model for concept-based learning of informatics [2]. It is designed to promote informatics learning in school through short tasks about simple concepts [1]. These tasks are the main component of Bebras. They are generally accompanied by a story or media element, to attract the attention of the children, and try to teach one or more concepts of informatics. Besides covering a wide range of topics, these tasks can be designed to help in the development of core computational thinking skills, such as abstraction, decomposition, evaluation and generalization.

From the practical point of view, Bebras tasks are just quiz questions with multimedia elements. The Bebras model can be used both in competitive and learning environments. Furthermore, it has already been used in several individual and team competitions across the globe.

2.4 General Import Format Template (GIFT)

The General Import Format Template (GIFT)² is a format created by the Moodle community to import quiz questions from a text file using markup language [4]. It supports multiple-choice, true-false, short answer, matching, fill in the blank and numerical questions.

The markup language of GIFT uses blank lines to delimit questions. Questions are written with the following syntax `::title:: question { answers }`. The syntax of the answers depends on the type of question. For instance, in multiple choice questions the correct answer(s) are prefixed with an equal sign (=) and the wrong answers with a tilde. To add feedback, a hash (#) can be used after each answer followed by the feedback message. Comments are preceded by double slashes (//) and are not imported. A full description of the language can be found on the Moodle page dedicated to the format.

3 Moozz

Moozz is an assessment environment for quizzes. It supports multiple choice (select one or multiple), gap filling, matching, short answer, and essay questions, as well as questions with media files. Moozz has its own XML language, named Moo, for storing and interchanging quizzes. Moo can be converted to and from different formats, such as GIFT and IMS QTI. Hence, the questions present in assessments can be saved in a question bank and reused in other assessments. It also contains an authoring tool for creating quizzes complying with Moo.

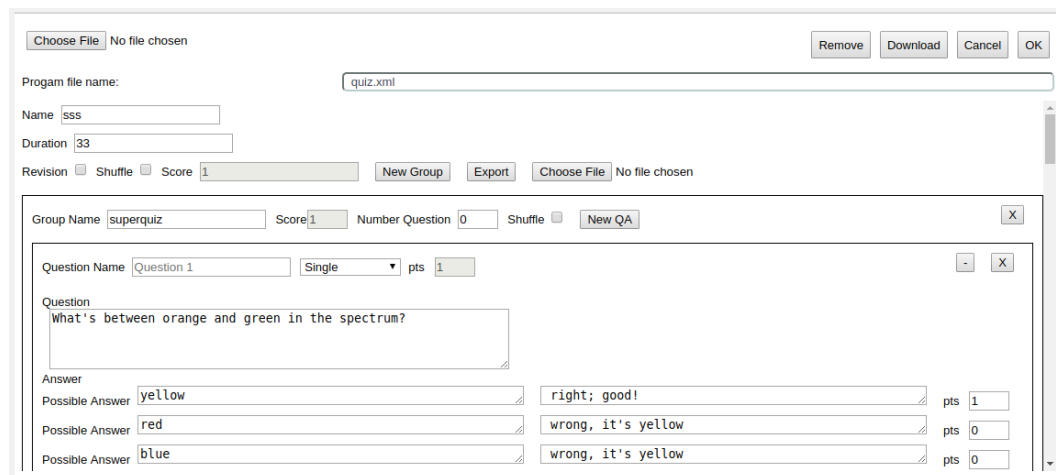
3.1 Authoring Tool

Moozz provides an authoring tool for quizzes. This tool can import and export quizzes in one of the following formats: Moo (XML), GIFT (plain text), IMS QTI (XML), and JSON. Besides that, it allows to create question groups, and add, edit, and remove questions. Questions can also have notes for each possible choice, question, or group. The quizzes are stored in Moo XML in Mooshak.

The GIFT format was extended to support the concept of groups in Moozz. Each group starts with `$name:numberQuestion`. `name` is the name of group whereas `numberQuestion` is the number of questions in the group to be selected for the exam. Two blank lines must be used to separate two groups of questions and one blank line must be left to separate each question from the next one.

Moozz supports several kinds of questions, such as: single-select answer, multiple-select answers, short answer, numerical, fill in the blank, matching, and essay questions. In `single-select answer` questions only one alternative can be marked as right and erroneous responses can be scored negatively. By default, a wrong answer has score zero and the correct answer scores one, but this can be modified in the Quiz Editor. A `multiple-select answer` question is used when two or more answers must be selected in order to obtain full credit. The multiple-select answer option is enabled by assigning partial answer weight to multiple answers, while allowing no single answer to receive full credit. The rating of each option can be defined and by default, a wrong answer has score zero. In `short answer` type, all the possible answers must be written, and it will be 100% credited if it matches exactly any of the correct responses, or zero, otherwise. A `numerical question` is similar to a `short answer` question, but the answer is a number. Numerical answers can include an

² https://docs.moodle.org/25/en/GIFT_format



■ **Figure 1** Screenshot of the Moozz authoring tool.

error margin, an interval and a precision for a correct answer. A **fill in blank** question is like a **short answer** question, but the answers are presented in an HTML element select. In the **boolean** question type, the answer indicates whether the statement is true or false. There can be one or two feedback strings. The first is shown if the student gives the wrong answer, and the second if the student gives the right answer. In **matching** questions, there are two arrays. One array with the keys and another with the values. Each key matches one and only one value. These questions do not support feedback messages. Finally, an **essay** question allows any text response, and is graded manually.

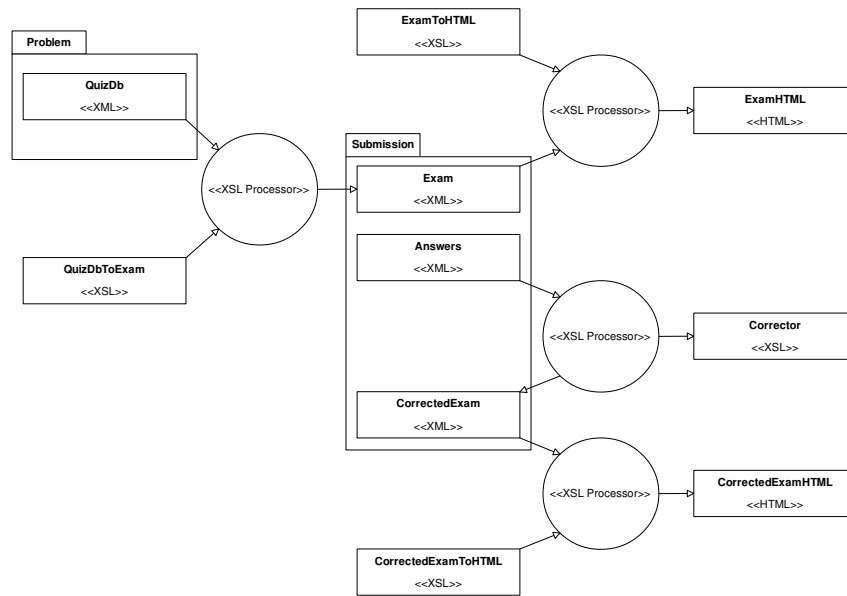
Figure 1 presents the first version of the authoring tool embedded in Mooshak administrator GUI.

3.2 Architecture

There are four types of XML files stored in Moozz which are Moo-compliant: **QuizDb**, **Exam**, **Answers**, and **CorrectedExam**. **QuizDb** is the question bank XML file containing all the questions used in assessments, which is created in the authoring tool of Moozz or imported in one of the supported formats. **Exam** is the XML file that contains the subset of questions of an actual exam. **Answers** has the answers of a student to an exam. **CorrectedExam** has the exam with feedback, classification and grade for each question.

Some of these files are generated from each other during the quiz assessment workflow (e.g., **Exam** is generated from an XSL Transformation applied to **QuizDb**). The **Exam** and **CorrectedExam** are also transformed into an user-friendly format to be displayed to the student using XSLT. Therefore, most of the work in Moozz consists of XML manipulations. Figure 2 presents the architecture of Moozz, particularly the transformations conducted in its core.

When the user request for a new exam, a transformation **QuizDbToExam** is applied on **QuizDb**, which is present on the problem directory. This transformation aims to select randomly N questions from the whole question bank. The outcome of this transformation is an **Exam** XML file, which is stored in the submission directory reserved to the current participant for subsequent requests. Before being sent to user, this XML is transformed into HTML through an **ExamToHTML** transformation. After solving the exam, answers are sent to Moozz in JSON and converted to XML in a Java class **JSONHandler**. The result is an **Answers**



■ **Figure 2** Diagram of the architecture of Moozz, highlighting the XSL Transformations carried internally.

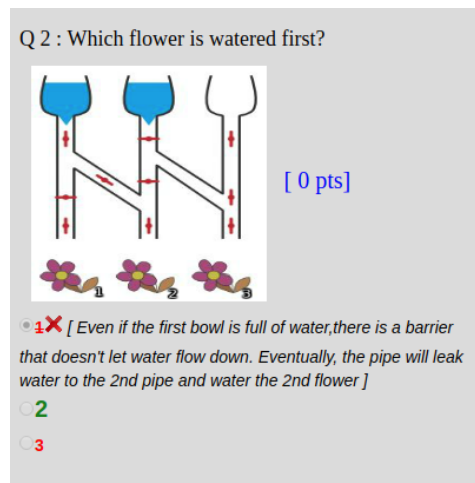
XML file, which is also stored in the submission directory. The quiz evaluation is then executed. The evaluation consists of applying a **Corrector** transformation to **Answers XML**. This transformation outputs a **CorrectedExam**, which is saved in the same directory. Finally, **CorrectedExam XML** is converted to **CorrectedExamHTML** through **CorrectedExamToHTML** to present the feedback to the student.

3.3 User Interface

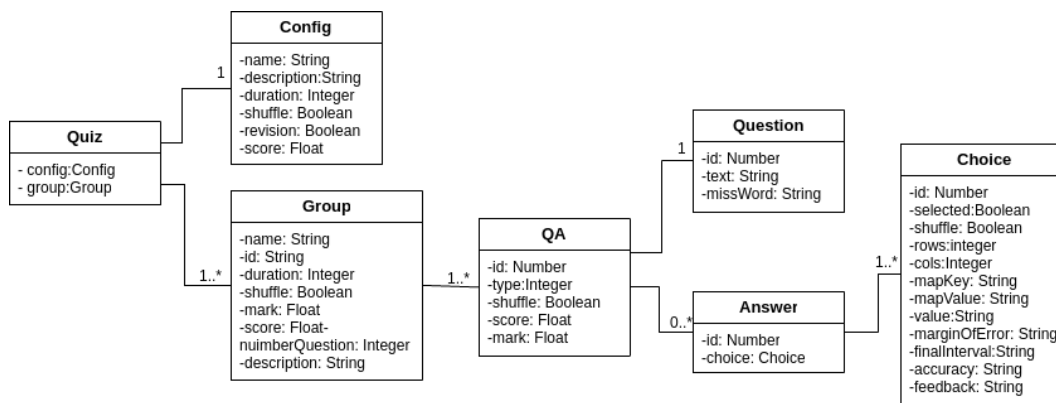
The client-side of Moozz follows the Model-View-Presenter (MVP) design pattern, integrating seamlessly in Enki within a single window. Since Enki uses GWT, the Viewer is also implemented with it. In this sense, the component defines a Java interface **MoozzView** with methods to update the view, such as `setQuiz(String html)`, and a class named **MoozzViewImpl** that implements the interface and displays the quiz. The presenter part is implemented in **MoozzPresenter**. This class receives the commands inputted by the user in the view, and invokes the necessary methods on the RPC interface of the Moozz service.

The data received from the server is either an **ExamHTML**, if the exam is not solved, or a **CorrectedExamHTML**, if the exam was already submitted. These HTML files are just an excerpt of an HTML, not a complete HTML page, containing the formatted elements to be displayed to the user. The excerpt is inserted into the container reserved for the quiz, after some Javascript pre-processing steps and CSS styling to make it more user-friendly. The answers submitted by the students are sent in an XML-formatted string complying with the Moo language.

On multiple choice questions, feedback is displayed only for the selected answer. For true-false questions, there can be one or two feedback strings. The first is shown if the student gives the wrong answer. The second if the student gives the right answer. Figure 3 presents an example of feedback in Moozz.



■ **Figure 3** Example of feedback presented in Moozz.



■ **Figure 4** Data model of the Moo language.

3.4 Moo Language

Moozz is able to import quizzes in different formats, therefore it is required a common quiz format, capable of storing the quiz and its configurations (e.g., time and number of questions per exam). A natural candidate for this role is the Question and Test Interoperability (QTI) standard. However, QTI revealed to be too complex and does not support configurations needed for quizzes in Mooshak, so a new language based in QTI is proposed, named Moo. As QTI, Moo is an XML language with its own XML Schema definition. As depicted in the simplified data model of Figure 4, Moo stores questions and settings such as the duration, and name of the quiz. Questions are organized in groups and each group stores information, such as the name, grade, and number of questions to appear on the exam.

Questions and answers of a group are stored in a type called QA. This type saves the question and its answers (if applicable) as well as configurations, such as the name of the question, the type, and the score, which is the sum of the positive scores of the answers. Each QA has one or more elements of type Choice. The Choice elements save different data, according to the type of the question. For example, in multiple, single, short-answer and boolean types, it includes the response text, feedback for each option, score and mark. The recorded data for numeric types depends on their subtype: exact answer (response value and

the margin of error), range answer (initial and final interval values), and precision answer (value and accuracy). In questions of type matching, it saves the key and the value. The essay type just saves the question, since the answer is a free-text introduced by the student. The essay type does not support feedback. The questions and answers texts accept inline HTML tags for including media or text formatting.

4 Conclusions and Future Work

Mooshak is a system that supports automated assessment in computer science. It has been used both in competitive and learning environments, supporting the assessment of visual and text-based programming languages. This paper presents an assessment environment for quizzes in Mooshak 2.0, named Moozz. Moozz supports all the standard question types, including multiple choice, true/false, short answer, numerical, fill in the blank and matching, and questions with media formats. It uses XSL Transformations to support the most common quiz formats, namely IMS QTI and GIFT.

Moozz includes a quiz authoring tool that is embedded into the administrator GUI of Mooshak. This editor is capable of importing and exporting quizzes in different formats, inserting multimedia elements, and add any of the supported question types with feedback information for each answer.

This environment is a work in progress. Currently, the development phase is almost completed, only missing the XSL Transformation to comply with IMS QTI. The next phase is the validation, which will be conducted in a real exam scenario with text, visual and quiz based exercises.

References

- 1 Valentina Dagiene and Sue Sentance. It's computational thinking! Bebras tasks in the curriculum. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 28–39, 2016.
- 2 Valentina Dagiene and Gabriele Stupuriene. Bebras-a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education*, 15(1):25, 2016.
- 3 Ginés García-Mateos and José Luis Fernández-Alemán. A course on algorithms and data structures using on-line judging. *ACM SIGCSE Bulletin*, 41(3):45–49, 2009. doi:10.1145/1562877.1562897.
- 4 Gaurav Kumar and Anu Suneja. Using Moodle – an open source virtual learning environment in the academia. *International Journal of Enterprise Computing and Business Systems*, 1(1):1–10, 2011.
- 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.
- 6 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.
- 7 Niall Sclater and Rowin Cross. What is IMS question and test interoperability. *Retrieved*, 7(22), 2003.
- 8 Colin Smythe, Eric Shepherd, Lane Brewer, and Steve Lay. IMS question & test interoperability: an overview, 2002. Final Specification, version 1.2.