Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

An architecture for modular grading and feedback generation for complex exercises

Michael Striewe

Paluno – The Ruhr Institute for Software Technology, University of Duisburg–Essen, Germany

A R T I C L E I N F O

Article history:
Received 30 April 2015
Received in revised form 3 February 2016
Accepted 15 February 2016
Available online xxxx

Keywords:
Software architecture
Component architecture
E-assessment systems

A B S T R A C T

Grading and feedback generation for complex open exercises is a major challenge in e-learning and e-assessment. One particular instance of e-assessment systems designed especially for grading programming exercises is JACK. This paper aims to discuss and evaluate key architectural concepts of JACK in terms of components, interfaces, and communication. It is shown how the architectural concept stands the test in an actual large scale deployment.

1. Introduction

Many topics in higher education make use of so-called open design exercises. In these kind of exercises, students provide their answers not by ticking boxes in a multiple choice form or by typing single numbers or words into gaps, but by handing in complex artefacts like short essays, program code, or diagrams. Grading and feedback generation for these kinds of artefacts is a major challenge in e-learning. In particular, solutions need possibly to be assessed in different ways to get a complete impression of the quality of the solution, where each individual analysis may be a long-running task. One typical instance of these kinds of exercises are programming exercises, where static and dynamic analysis of program code submitted by students can be performed. While this is primarily an aspect of intelligent analysis techniques, it also has various implications for the architecture of e-assessment systems. Hence designing an architecture for grading and feedback generation for open design exercises is also a general software engineering problem where several domain specific requirements have to be considered during the design process.

First, there might be multiple analysis techniques suitable for one specific exercise type but also analysis techniques that are general enough to be applied to more than one exercise type. For example, a programming exercise requires both static and dynamic analyses of the submission, which are two entirely different techniques. Static analysis is used to inspect source code without executing it in order to provide feedback to syntactic, structural, and stylistic errors. Dynamic analysis is performed by executing the code with several test cases in order to detect functional errors. At the same time, one single component for plagiarism detection can possibly be used on different types of exercises, such as programming exercises and short essay exercises. Hence, a general m-to-n relationship between exercise types and grading components can be assumed. In this relationship, the exercise type characterizes the artefacts that need to be submitted by the students, while a grading component can be any piece of software that is capable of generating feedback to an artefact. In order to avoid larger changes in the architecture when one exercise type or one grading technique is added, a modular architecture can be used in which both parts are loosely coupled and can be changed or extended independently of each other. This is a signif-

E-mail address: michael.striewe@paluno.uni-due.de.

http://dx.doi.org/10.1016/j.scico.2016.02.009
0167-6423/© 2016 Elsevier B.V. All rights reserved.

icant difference to systems offering no open design exercises but closed exercises like multiple choice or “click-on-image” questions, where one or a few simple comparisons or computations are sufficient to determine the correctness of an answer.

Second, sophisticated analysis techniques may require long-running checks. One option to increase system responsiveness in these cases is to handle grading as asynchronous tasks that can be run in parallel, which advocates for architectures that allow for load balancing and job queueing. It also supports the idea of a modular architecture, in which several instances of components for particularly long-running tasks can be deployed in parallel, while a lesser number of instances for components with short-running tasks is sufficient. Depending on the kinds of analysis techniques used, the queueing implementation must also obey dependencies between job. For example, in programming exercises a static check may be required to be performed before a dynamic check or a performance check should only be performed for solutions passing all test cases.

Third, complex exercises like programming tasks may involve security issues raising from complex grading operations, such as executing arbitrary code submitted by students. This advocates for architectural solutions where grading tasks happen in strictly isolated sandbox environments with limited access to critical system resources. One possible solution for this is to run critical grading components on separate servers, where no other data is stored than that necessary to grade this particular solution. This way, a malicious submission can neither access critical data, nor can it stop the whole system by blocking the grading component. Obviously, the latter point is already implied in the idea of asynchronous tasks as discussed above.

One particular instance of e-assessment systems designed especially with respect to these considerations is JACK. JACK is a framework for modular grading and feedback generation in complex exercise domains. It is a distributed web-based system written in Java, using EJB and OSGi as technical solutions for modular system design. This paper aims to discuss key architectural concepts of JACK and the benefits and drawbacks that could be observed by using the framework for eight years now. While previous papers on JACK and similar systems mainly discuss organizational issues, particular system features, or evaluations of grading quality, this paper focuses on technical aspects that are more general and thus not only relevant in the context of programming exercises. The overall conclusion of this paper is that the JACK architecture is suitable both from the perspective of software engineering and from the perspective of e-assessment systems. Special emphasis is laid on the aspects of extensibility and scalability, which makes the JACK architecture superior to other approaches.

The paper is organized as follows: Section 2 provides background information on the goals and genesis of JACK, both from a technical and didactical perspectives. Section 3 digs into the details of JACK’s architecture and discusses key interfaces, component collaboration, and communication in particular. Section 4 provides an informal evaluation of the quality of the architecture based on experiences with use and evolution of JACK through its past eight years of service. Section 5 discusses related work and section 6 concludes the paper.

2. Project context

JACK has been in use for more than eight years now and has been used at several universities in several different courses. At the time of writing there are seven deployments of JACK in actual use for programming exercises including one demonstration server[1] and one testbed.

2.1. Project history

The first version of JACK was designed in 2006/2007 in order to support a programming lecture for first-year students by delivering and grading programming exercises in Java [26,24]. The size of these exercises typically ranges from implementing a few methods in one class to implementing a few classes. However, JACK is not specifically limited to the analysis of small-sized projects, but is also able to handle larger projects.

Special emphasis was put right from the beginning on not only calculating grades, but also providing meaningful textual feedback that helps students to improve their solutions. The original version thus employed both static and dynamic analysis of program code as two separate components, where the former made use of another external component for graph transformations. The original version also supported content delivery to students both via a web-browser interface and via a plug-in for the IDE Eclipse, accessing JACK via a web-service interface. It was used at the University of Duisburg–Essen both for formative and summative assessments with students submitting homework exercises as well as mini-exams on a regular basis throughout the academic term. Usage figures showed up to 600 students per term and up to 1’000 submissions per exercise. The number of submission include both initial submissions and subsequent submission where students tried to improve previous errors. All submissions are checked completely and independent of each other. An initial exercise pool containing about 20 exercises (some with variants for different cohorts in exams) was created at this time.

Based on the first experiences with this framework, a technical redesign was performed in 2008/2009, replacing parts of the EJB components by equivalent OSGI implementations [23,27]. At the same time, existing grading components were refined and an additional component for static analysis of Java program code as well as a component for visualizing object oriented data structures were added. More components for analyzing Java exercises have been added over time, as well as

[1] The JACK demonstration server is available at https://jack-demo.s3.uni-due.de.