Integrating OpenDSA and CodeWorkout with the Canvas LMS

Ayaan M. Kazerouni, Jackson Wonderly, Stephen H. Edwards, Clifford A. Shaffer
ayaan@vt.edu,jacdw94@vt.edu,s.edwards@vt.edu,shaffer@vt.edu
Virginia Tech
Blacksburg, VA

ABSTRACT

There exist numerous "smart learning tools", each serving different purposes. A primary challenge associated with these tools is getting them to work together smoothly and with minimal overhead for the end-user (e.g., instructors or students). We describe the infrastructure behind the integration of two learning tools with the Canvas LMS. This infrastructure is LTI compliant and is in use at several US universities.

OpenDSA is an e-textbook project that allows instructors to create custom books by putting together combinations of available modules, and CodeWorkout is an online system that serves short programming exercises and multiple-choice questions. We describe the implementation that allows OpenDSA textbooks to serve CodeWorkout exercises within Canvas. OpenDSA does this by being a "man in the middle", i.e., by functioning both as a tool consumer (facing CodeWorkout) and as a tool provider (facing Canvas).

KEYWORDS

interoperability, LTI, learning management system, smart content

ACM Reference Format:
Ayaan M. Kazerouni, Jackson Wonderly, Stephen H. Edwards, Clifford A. Shaffer. 2019. Integrating OpenDSA and CodeWorkout with the Canvas LMS. In *Proceedings of SPLICE '19.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

OpenDSA [3] is an open e-textbook project that allows users to choose from available modules to put together a custom textbook for a course. CodeWorkout [1] is an online system that serves short programming exercises and multiple-choice questions, either in a public setting (for voluntary practice) or in a course setting (with deadlines, time limits, and other policies in place). OpenDSA includes several CodeWorkout exercises in its modules, which eventually make into textbooks used in several US universities.

In this paper we describe the implementation of the interoperability between OpenDSA and CodeWorkout. Specifically, we describe how the two tools can be used together in a single course, embedded in the Canvas Learning Management System (LMS). While on the backend OpenDSA and CodeWorkout are separate entities that maintain their own databases and servers, on the frontend they are presented as a single learning tool, giving the instructor and students a more cohesive experience. This is possible using the LTI protocol for communication.

SPLICE '19, February 27, 2019, Minneapolis, MN

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

We describe some terms:

- **Learning Management System (LMS)**: A system to organize and deliver learning content and monitor students. Two examples are Canvas and Moodle.
- **Tool Provider**: Sometimes also called "smart content", these tools are learning tools providing specific pieces of functionality for an instructor or student. For example, OpenDSA provides e-textbook capabilities, and CodeWorkout provides programming exercises. Both are tool providers.
- **Tool Consumer**: A system that provides extension points to which tool providers may attach their functionality. For example, Canvas is a tool consumer that "consumes" the functionality provided by both OpenDSA and CodeWorkout.

Under the LTI protocol, tool providers embed tool consumer windows using *i-frames*, and communication between the two takes place by transferring user, assignment, and grade information. As a concrete example, to serve CodeWorkout programming exercises within Canvas, CodeWorkout would be the *tool provider*, and Canvas the *tool consumer* that displays CodeWorkout windows and receives assignment outcomes from CodeWorkout.

3 PROBLEMS FACED IN THE PAST

In the past, we used OpenDSA and CodeWorkout as independent tool providers to Canvas. Both systems communicated independently with Canvas, unaware of each others' existence. OpenDSA would initialize the course modules using the Canvas API, making Canvas point to OpenDSA modules where appropriate, and CodeWorkout exercises where appropriate. This led to certain "pain points", both technical and pedagogical.

Grade passback. OpenDSA had neither control over nor access to the information being sent from CodeWorkout to Canvas. As an interactive e-textbook, OpenDSA contains numerous proficiency exercises of its own, including multiple-choice questions and algorithm analysis exercises. Programming exercises relating to a topic would ideally be treated as part of the proficiency exercises for that topic. However, because of the independent LTI communication undertaken by CodeWorkout, OpenDSA had no idea of whether or not students had completed the programming exercises associated with a module. That is, OpenDSA could not tell if the student had *truly* completed the module, only that they had completed the parts provided by OpenDSA.

Content integrity. As a textbook, OpenDSA necessarily contains prose alongside its numerous interactive widgets (proficiency exercises, algorithm visualizations, etc.). The primary idea was to have students read the prose *as well as* practice using these widgets. CodeWorkout programming exercises can be thought of as just another widget included within OpenDSA, similar to Khan Academy multiple choice questions.

With CodeWorkout functioning as a separate LTI tool provider, however, its programming exercises would manifest as separate Canvas modules, i.e., each one on its own page, by itself [1]. This breaks the flow of the "textbook", and worse, separates the prose from the exercise, making it difficult and unattractive for the student to read the related prose before or while working on the exercise.

Unobtrusive embedding. CodeWorkout's main use case is not to be embedded in an LMS or e-textbook. CodeWorkout serves numerous courses and contexts that may use OpenDSA, Canvas, both, or none. This means that any infrastructure that aims to embed CodeWorkout exercises in itself must do so using an unobtrusive protocol, i.e., that can work with CodeWorkout's existing extension points. An example of such a protocol is LTI, and that is what we have used here (§4).

4 OUR APPROACH

Fundamental to the problems described above is that OpenDSA and similarly classed tools like MasteryGrids [2] function on a more *managerial* level than tools such as CodeWorkout. That is, OpenDSA curates, organizes, and presents content to the student, often from multiple sources, of which CodeWorkout is just one.

According to the definitions in §2, both OpenDSA and CodeWorkout are *tool providers* to the Canvas LMS, a *tool consumer*. However, to students and instructors, these relationships are not evident or intuitive. That is, a student is first and foremost interacting with a **Canvas module**, which happens to be an **OpenDSA section** or chapter. That section or chapter might or might not include a related CodeWorkout programming exercise. Our current design does not implement this mental model, and this has led to considerable technical debt trying to keep the three systems (Canvas, OpenDSA, and CodeWorkout) functioning in harmony.

Our solution was to change OpenDSA's LTI communications such that, where it used to function only as a tool provider to Canvas, it now also functions as a tool consumer to CodeWorkout. OpenDSA embeds CodeWorkout exercises in itself, and CodeWorkout sends grade and user information to OpenDSA, not to Canvas. This simple change comes with a number of virtues:

LTI compliance. It is still completely LTI compliant, with only small unobtrusive changes being made to CodeWorkout to facilitate it. This approach would allow OpenDSA to embed multiple tool providers into its e-textbooks, as long as they are LTI compliant.

Conceptual integrity. We now implement the mental model described above. OpenDSA is the textbook being embedded in Canvas, and CodeWorkout is simply a programming exercise widget being embedded in OpenDSA.

Content integrity. OpenDSA can embed CodeWorkout exercises alongside the prose and other widgets, thus maintaining the integrity of the textbook.

Grade passback. Grades for a module's deliverables can be organized and presented less disparately. For example, Canvas can display that the student has completed all assignments associated with the module on Binary Search Trees, including algorithm visualizations, proficiency exercises, and CodeWorkout programming exercises. This is made possible because CodeWorkout no longer maintains its own channel of communication with Canvas. Instead, it sends all data through OpenDSA, which can receive it and interpret it as it sees fit. This communication between the two learning tools is hidden from Canvas, which only knows that it is embedding an OpenDSA module.

[1] This is due to the fact that LTI tool providers are embedded inside modules or assignments, and in Canvas, we typically have one module or assignment per page.

REFERENCES

[1] Stephen H. Edwards and Krishnan Panamalai Murali. 2017. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 188–193. https://doi.org/10.1145/3059009.3059055

[2] Tomasz D. Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. 2014. Mastery Grids: An Open Source Social Educational Progress Visualization. In *Open Learning and Teaching in Educational Communities*, Christoph Rensing, Sara de Freitas, Tobias Ley, and Pedro J. Muñoz-Merino (Eds.). Springer International Publishing, Cham, 235–248.

[3] Clifford A. Shaffer, Ville Karavirta, Ari Korhonen, and Thomas L. Naps. 2011. OpenDSA: Beginning a Community active-eBook Project. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research (Koli Calling '11)*. ACM, New York, NY, USA, 112–117. https://doi.org/10.1145/2094131.2094154