Design and Implementation of OpenDSA Interoperable Infrastructure

Hossameldin Latif Shahin

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Clifford A. Shaffer, Chair
Dennis Kafura
Edward A. Fox

June 30, 2017
Blacksburg, Virginia

Keywords: computer science education, web application, learning tools interoperability

Design and Implementation of OpenDSA Interoperable Infrastructure

Hossameldin Latif Shahin

(ABSTRACT)

OpenDSA is a system for creating rich eTextbooks that combine quality text with visualizations and interactive, auto-graded exercises. As OpenDSA gains recognition, its use increases each year. This mandates a scalable, reliable, and sustainable infrastructure to accommodate the fast-growing demand for OpenDSA access.

We implemented OpenDSA-LTI, an interoperable infrastructure which transforms OpenDSA from a standalone, self-contained eTextbook to an integrated learning tool communicating with a Learning Management System (LMS) through the Learning Tool Interoperability (LTI) protocol. OpenDSA-LTI delivers OpenDSA content and interactive materials to students through a reliable and secure LMS interface. LTI integration encourages OpenDSA adoption by providing easy, intuitive tools that help instructors to build and generate OpenDSA eTextbooks in their LMS courses. OpenDSA-LTI allows OpenDSA content developers to take advantage of various tools already provided by the LMS instead of reproducing these through their own proprietary services.

The OpenDSA-LTI extendable design allows for adding new LTI-compliant exercises to OpenDSA books. This changes OpenDSA developers' efforts to searching for learning tools instead of reimplementing them. As an example, instead of maintaining the original OpenDSA programming evaluation engine, we could easily replace it with the Code Workout online drill-and-practice system.

Since its launch in August 2016 until June 2017, OpenDSA-LTI has hosted 36 active courses offered by 25 different universities in 6 countries, 41 instructors have used OpenDSA-LTI to host their courses on the Canvas LMS, and the system has 2,729 registered students.

This work was supported in part by NSF grant DUE-1432008.

Design and Implementation of OpenDSA Interoperable Infrastructure

Hossameldin Latif Shahin

(GENERAL AUDIENCE ABSTRACT)

OpenDSA is a system for creating online textbooks that combine quality text with visualizations and interactive, auto-graded exercises. As OpenDSA gains recognition, its use increases each year. This mandates a scalable, reliable, and sustainable infrastructure to accommodate the fast-growing demand for OpenDSA access.

We built OpenDSA-LTI, an online web application which transforms OpenDSA from a standalone, self-contained textbook to a learning tool which any university can integrate in their learning systems.

OpenDSA-LTI delivers OpenDSA content and interactive materials to students through a reliable and secure interfaces. The new infrastructure encourages OpenDSA adoption by providing tools that help instructors to build and generate OpenDSA online textbooks in their institution's learning systems.

The OpenDSA-LTI extendable design allows for adding new exercises to OpenDSA online textbooks. This changes OpenDSA developers' efforts to searching for other learning tools instead of creating them from scratch. As an example, instead of maintaining the original OpenDSA programming exercises, we could replace it with the Code Workout online programming evaluation system.

Since its launch in August 2016 until June 2017, OpenDSA-LTI has hosted 36 active courses offered by 25 different universities in 6 countries, 41 instructors have used OpenDSA-LTI to host their courses on the Canvas LMS, and the system has 2,729 registered students.

This work was supported in part by NSF grant DUE-1432008.

Dedication

In memory of my sister, Hanan Shahin and my labmate, Taylor Rydahl.

Acknowledgments

First of all, all thanks due to ALLAH, may His peace and blessings be upon his prophet, Mohammad. I thank ALLAH for helping me during my journey until I successfully complete my master's degree.

I would like to thank my advisor, Dr. Clifford A. Shaffer, both for his guidance and his trust in me, and Drs. Dennis Kafura and Edward A. Fox for serving on my thesis committee.

I would also like to show my appreciation to the National Science Foundation which supported my work through grant DUE-1432008.

Thanks go to my friends and family for their support, and special thanks to my beloved wife, Doaa Altarawy, for her encouragement and continuous support.

Also, I would like to thank my little twins, Zeyad and Somaia, who kept wishing me the best of luck during my thesis writing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

OpenDSA is an open source eTextbook infrastructure combined with contents for a wide variety of Computer Science topics. At this time, content available in OpenDSA covers topics for a CS2-level course, a post-CS2-level Data Structures and Algorithms (DSA) course, Programming Languages, Finite Languages and Automata (FLA), and a senior-level algorithms course.

OpenDSA combines textbook-quality prose with visualizations and interactive, auto-graded exercises, allowing students to practice as much as they want. OpenDSA exercises are designed to provide students with immediate feedback as they progress through each exercise's steps or questions. Once a student has performed all of the exercise's steps or has answered all of the questions correctly, the exercise will display a green check mark indicator to show that the student has become proficient with the concept presented.

The OpenDSA framework makes it easy to add new visualizations, exercises, and topics. OpenDSA automatically handles exercise grading and gives immediate feedback, which makes it suitable for large classes.

1.1 Motivation

Since the project began in 2011, OpenDSA contributors have worked hard to enhance project content by adding new topics and creating more visualizations and comprehensive exercises. However, OpenDSA requires more than content to be suitable for class environments. The system has to support capabilities like account management, grade management, and activity logging. It is also necessary to support instructors' ability to follow their students' progress through the OpenDSA content and exercises. OpenDSA developers spent time and effort early in the project to design and implement these instructor's tools. Designing, implementing, and maintaining these tools made OpenDSA a stand-alone, one-size-fits-all

application. Thus, developers' focus was taken away from their primary goals of adding new topics, developing more exercises and visualizations, and conducting pedagogical studies [13] [19][34][17][9].

As the OpenDSA project was noticed beyond the developers' home institutions, instructors became eager to adopt it, and in some cases wanted to run an OpenDSA server by themselves. However, the original server infrastructure was not portable and was poorly documented, which made it hard for third parties to install. As a consequence, courses offered for instructors in other institutions were usually hosted on Virginia Tech servers, which raised critical scalability concerns and took further project resources away from core activities.

> "Teachers value a secure, central single point of entry into an on-line learning environment which enables students to move seamlessly between learning activities regardless of where the applications may be physically located." [5]

For the past decade, course material has increasingly been organized within learning management systems (LMS). An LMS creates a suitable learning environment for students: it typically contains discussion forums, accounts and login support, gradebooks, chat rooms, wiki pages, and multimedia content [5]. The LMS environment provides communication and collaboration facilities between individual learners, and between learners and teachers.

1.2 Research Objectives

The aim of this thesis is to document the design rationale and provide assessment for an integration of OpenDSA with LMSs to achieve three goals. The first goal is to take advantage of the robust instructor's tools and services already provided by an LMS, so that OpenDSA collaborators do not need to implement and maintain such standard functionalities. The second goal is to deliver OpenDSA contents through a trusted and convenient channel to instructors and students, which will lead to broader impact for the OpenDSA project. The last goal is to allow for relatively easy deployment of OpenDSA servers at third-party institutions.

Our research hypothesis is that online educational systems can be architected so as to keep the eTextbook content generating system, the LMS, and the various exercise types as separate software components, while allowing for a meaningful integration of the whole.

1.3 Solution Approach

Different integration protocols and approaches for improving the interoperability of learning tools were assessed against OpenDSA requirements and constraints. Based on our findings,

a set of development tools and communication protocols were chosen to be the basis for the new OpenDSA infrastructure.

The suitability of different protocols and Web frameworks as the proposed solution for integrating OpenDSA with an LMS were analyzed based on the following criteria:

- Changes and modifications to OpenDSA content code base that would be required.
- The ability to integrate OpenDSA with a range of LMSs.
- The infrastructure's overall performance overhead.
- The ability for other institutes to easily install the new infrastructure.
- The ability to augment OpenDSA with other learning tools (such as new exercise types) within an LMS.

We want to support integration with third-party learning tools and exercises. A good example, that we have already implemented, is CodeWorkout [29] integration. So, suitability for such integration is a criterion for the chosen protocol.

Based on the findings, we have designed and implemented a new infrastructure, called OpenDSA-LTI, which potentially integrates OpenDSA contents with any LMS that complies with the Learning Tool Interoperability (LTI) standard protocol [23]. To our knowledge, OpenDSA-LTI is the first infrastructure to integrate a standalone eTextbook system into a learning management system like Canvas[1] in a reusable way. The new infrastructure design considered the student, instructor, and developer points of views. The implemented infrastructure will hopefully:

- reduce students' unnecessary extra work in learning from multiple learning systems;
- ease OpenDSA book configuration and compilation for instructors;
- support OpenDSA book integration in any LTI-compliant LMS;
- allow instructors to embed individual visualizations and exercises in their LMS courses, in case they do not want to utilize the entire OpenDSA book; and
- provide instructors with the required tools to monitor students' learning progress.

In addition to being able to integrate OpenDSA with other learning tools within an LMS, a design goal for the integration protocol was to not require extensive changes in the existing OpenDSA content code base, or in the tools used to compile eTextbooks from those contents. It should also have widespread community support and evidence for continuity and progression. Finally, the selected protocol should not cause unnecessary performance overhead, which would harm the usability of OpenDSA and other learning tools.

[1] https://www.canvaslms.com

1.4 Major Contributions

The following are my major contributions to the OpenDSA project.

- Made OpenDSA more robust by mitigating known problems, including:
  - Missing interaction data and out-of-synch caching mechanism.
  - Gaming problems and an outdated Khan Academy exercise framework.
  - Manual configuration files preparation.
- Implemented the OpenDSA-LTI infrastructure, with the following features:
  - Through the LTI protocol, integrated OpenDSA content with the widest range of LMS.
  - Provided interfaces for instructors to configure and compile books within the LMS.
  - Designed the system to allow integrating any LTI-compliant learning tools (e.g. CodeWorkout exercises) to OpenDSA Books.

1.5 Structure of the Thesis

This thesis has the following structure. Chapter 2 describes the original OpenDSA infrastructure. Chapter 3 pinpoints some of the known limitations and issues in the original infrastructure, and explains how we mitigated and solved some of these problems. Chapter 4 assesses existing e-learning interoperability standards. Chapter 5 presents a detailed discussion about the new infrastructure's implementation. Finally, Chapter 6 presents conclusions and potential future enhancements to the OpenDSA system.

Chapter 2

The First Generation OpenDSA Infrastructure

The OpenDSA project architecture [32][33] comprises four components. The first component is a rich collection of content types, including the explanatory text, visualizations, and exercises. The second component is the compilation process; its main role is to generate a complete book in the form of HTML files from the different content types. The third component is the data collection server (DCS), which keeps track of student registrations, interaction data, and scores. The final component is the client-side framework, which has multiple responsibilities that can be briefly summarized as managing the communication between the compiled HTML pages and the DCS. All OpenDSA components are open source and available on GitHub. OpenDSA contents, the compilation process, and the client-side framework can be found at https://github.com/OpenDSA/OpenDSA. The original DCS is available at https://github.com/OpenDSA/OpenDSA-server.

In this chapter, I explain in detail about the major OpenDSA components and discuss how they work together to form the OpenDSA system.

2.1 OpenDSA Content

The basic OpenDSA unit of presentation is the module; each module explains one topic or concept, like QuickSort or Hashing. Modules were designed to be complete, and typically fit within a regular lecture period. Modules are written in a plain text markup language known as reStructuredText (ReST) [20], one ReST file for each module. Modules are compiled into HTML pages using Sphinx [7], a tool developed by the Python community for Python project documentation. The ReST/Sphinx combination is used to make a separation between content authoring and how the content is presented. Keeping the content in a plain text file makes collaborative authoring relatively easy. An OpenDSA book instance is a collection

of modules grouped together in ordered chapters. For each book instance, Sphinx creates the main landing page for the table of contents, from which the user can navigate to any module. A module typically contains prose, static diagrams, and interactive materials. There are multiple types of interactive materials: Slideshows (SS), Algorithm Visualizations (AV), Proficiency Exercises (PE), programming exercises, and Khan Academy-style exercises (KA).

All the interactive materials are implemented in HTML5, CSS, and JavaScript. Slideshows, AVs, and PEs are written using the JavaScript Algorithm Visualization library (JSAV) [27][28]. While slideshows are generated by Sphinx with inline HTML markup written in the module's HTML page, AVs, PEs, and KAs are embedded in the module as iframes. Iframe embedding is a design choice made to ensure that instructors can use OpenDSA interactive materials alone or as part of an OpenDSA book. KA exercises are developed using the open-source Khan Academy exercise framework[1]. The KA exercise framework supports multiple question types: True/False, Multiple Choice Questions (MCQs), and fill-in-the-blank.

In 2013, a snapshot of the KA framework was taken and made part of the project. The KA framework snapshot was extended to include a new type of question, referred to as proficiency exercise, where students are asked to manipulate a data structure to simulate a single iteration of an algorithm [31][1]. Generally, the data being manipulated are randomly generated. All OpenDSA interactive materials are capable of tracking student progress, and they provide feedback to show their current status. Slideshows show the number of slides visited and the number of slides remaining. PEs show the number of steps done correctly, steps done incorrectly, and the total number of remaining steps. KA exercises indicate whether the exercise was answered correctly in the form of smiling or frowning faces. KA exercises send and save students' state in the DCS so that they can solve the exercises over multiple sessions. Multi-part exercises show a status bar displaying the number of correct answers from the total number of questions to be answered to complete the exercise. Figures 2.1-2.4 show examples of these interactive materials.

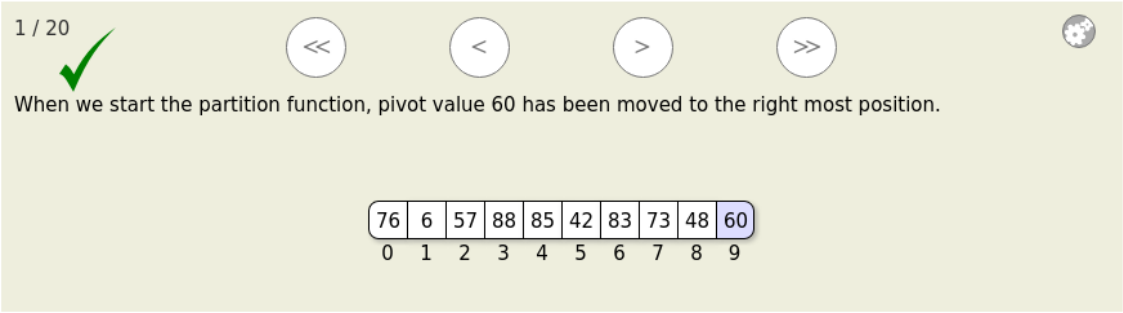

Figure 2.1: A slideshow that illustrates in multiple steps how a list of numbers is sorted by the QuickSort Algorithm

[1] https://github.com/Khan/khan-exercises

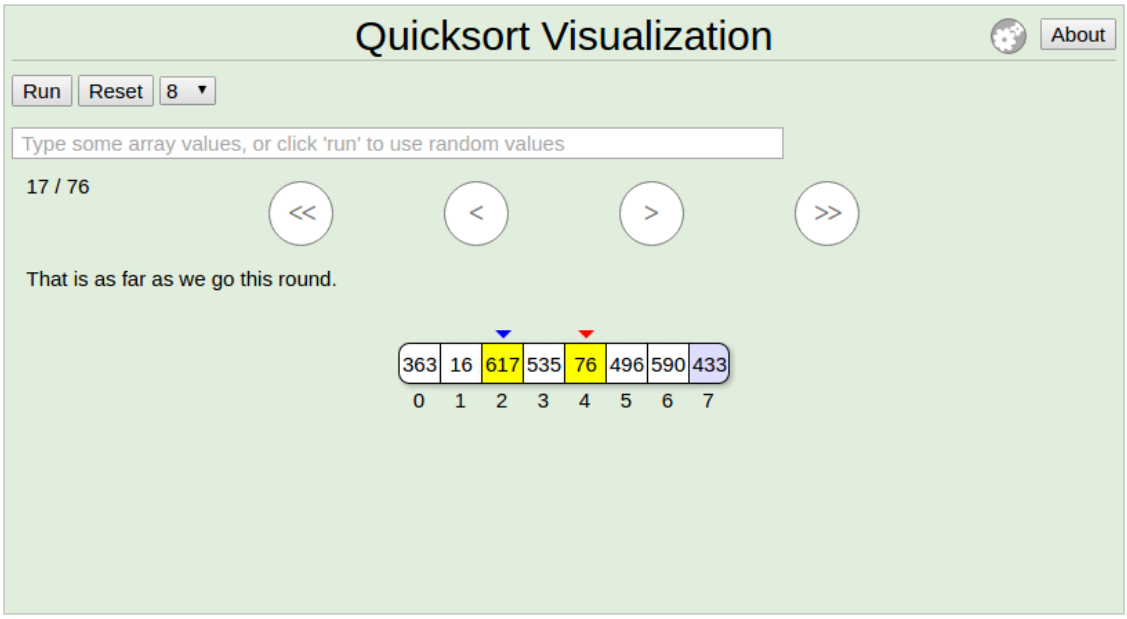

Figure 2.2: An Algorithm Visualization (AV): In addition to demonstrating QuickSort algorithm steps, AVs allow students to enter the initial data values to be used.

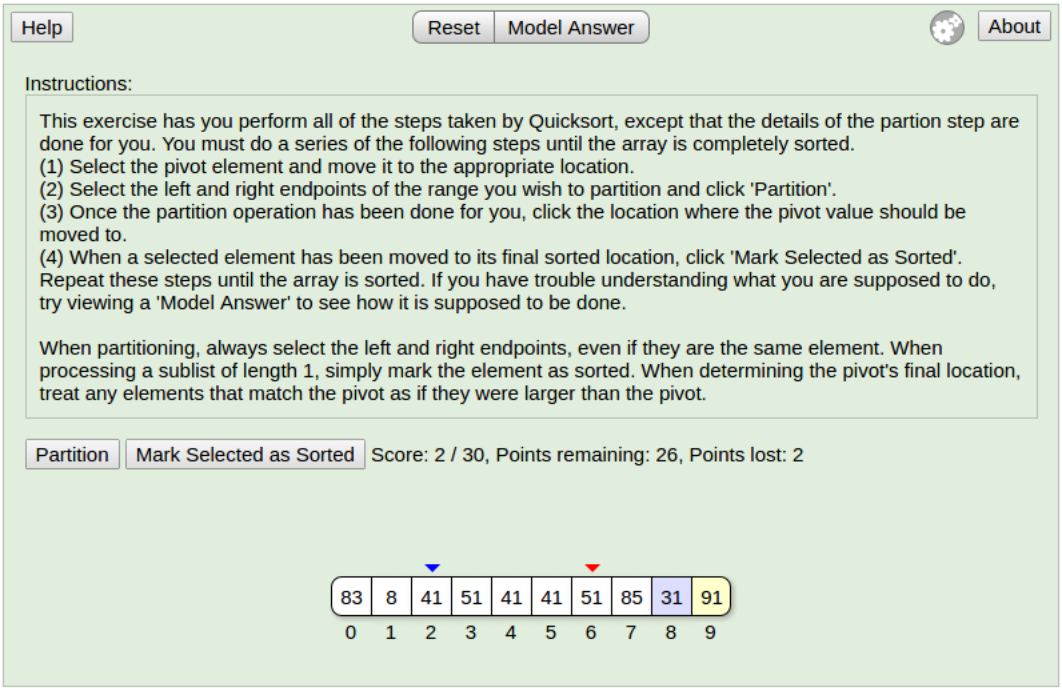

Figure 2.3: A Proficiency Exercise (PE): Students must simulate algorithm execution. They are asked to do a series of steps and change the data structure (an array in this case) until the algorithm completes. The score field shows the number of steps done correctly.

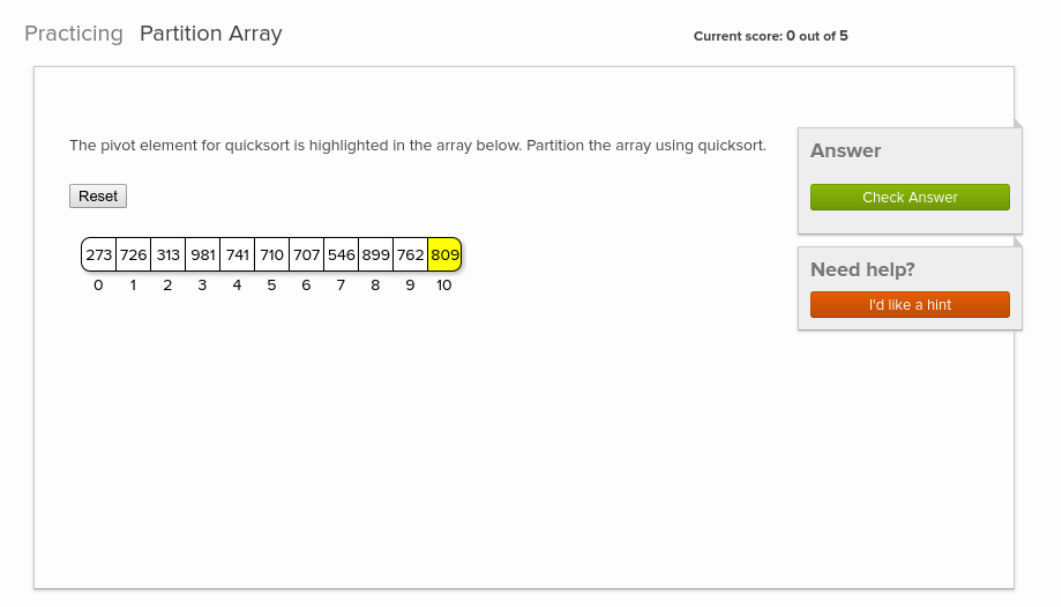

Figure 2.4: Khan Academy-style Proficiency Exercise (KA): The KA exercise has a pool of questions which is randomly presented to the students one by one. In this example, students are asked to simulate a single step or one iteration of an algorithm.

All OpenDSA interactive materials track the student progress, and they fire a "completed" event when the student finishes the material. The completion event is captured by the OpenDSA infrastructure to award the student the grade assigned to the exercise in a particular book instance. There are different requirements for each interactive material to be considered as "completed". A slideshow is completed by going through all the slides one by one until the student reaches the last slide. An AV is completed by advancing the algorithm visualization step by step until the algorithm is complete. A PE requires a student to perform a percentage of the total steps correctly to be marked as completed.

KA exercises are typically a set of questions; these questions make a pool that the KA framework presents one at a time at random. The student has to answer a predefined number of questions to accumulate the number of points required to complete the exercise. A student's attempt to answer a question will count only if it is correct, from the first trial, and without using hints. If a student attempt is incorrect, he will lose a point, and if he uses the hint, he will not get a point for the question. Table 2.1 compares each of the OpenDSA interactive materials' characteristics.

Table 2.1: OpenDSA interactive materials characteristics. All of the exercises are capable of giving immediate feedback, and they are self-grading. Only KA exercises save the student's state between sessions. Although slideshows and AVs can be configured to have a grade component, instructors usually assign grades to PEs and KAs only.

Type	Embedding	Feedback	Self-grading	State-saving	Grade-assigned
SS	inline	✓	✓	✗	Optional
AV	iframe	✓	✓	✗	Optional
PE	iframe	✓	✓	✗	✓
KA	iframe	✓	✓	✓	✓

2.2 The Authoring System and Compilation Process

Key elements in the compilation process are the compilation script and the configuration file. A compiled collection of OpenDSA modules is called a "book instance". The content of a book instance is defined using a configuration file. A configuration file is written in JSON format; it includes references to a set of OpenDSA modules grouped in ordered chapters. Configuration files are maintained and saved as part of the OpenDSA GitHub repository. There are standard configuration files for CS2, CS3 (data structures), programming languages, and senior algorithms book instances. Instructors who want to customize a configuration file for their specific needs usually copy an existing one and modify it as appropriate. The main purpose of the configuration file is to keep the content independent of specific settings related to a book context. Configuration files provide a set of variations that instructors can customize for their needs. Instructors can choose the book language, with a default fallback option to English if there is no module found written in the selected language. Instructors can set the programming language used for displaying code examples. Instructors can add, reorder, and remove modules. They can configure different exercise attributes like the number of points, threshold, and whether the exercise is required for module proficiency or not. The module's definition in the configuration file should identically mirror the module ReST file regarding the number and the order of module sections, as well as the number and the order of exercises defined in each section.

The compilation script uses a configuration file as input; it performs a series of tasks to validate the file, enforce default values, and collect and copy the set of module ReST files into the target book folder. Finally, the script calls Sphinx to compile the book instance in HTML pages, and configures them to use the DCS URL.

There are two major use cases for OpenDSA eTextbooks. The first is as an integral part of a course, where students are asked to perform the exercises before due dates, and exercise grades are part of the course final grade. The second is as a supplementary reading and practicing reference without grade tracking. In the latter case, the configuration process

would remove the DCS URL from the configuration file so that the compilation process generates a "Plain" book in the sense that students can use the book without registration. They can go to the book to practice, but no interaction data is stored, and no grades are tracked by the DCS.

2.3 Data Collection Server

The DCS in the first generation infrastructure was developed using Django[2] (version 1.3). Django is an open source Python-based Web framework based on Model-View-Controller design pattern. DCS was integrated with the A+ course management system (CMS), which was developed and used by OpenDSA collaborators in Finland [26]. A+ manages OpenDSA course creation, instructor registration, and student enrollments. An OpenDSA book instance is linked to a course taught by an instructor. DCS adopts a client-server architecture, where a content server hosts OpenDSA book module pages and serves them to a student browser in HTML format. Once a module page is loaded, all student interaction data and exercise attempts are communicated to the DCS, which is deployed on a separate server. Communication to the DCS is stateless, which means each request contains all the information required by the DCS end point to function properly, without the need to save client context information on the DCS side. The DCS database schema was designed to manage and persist a collection of information categories (a.k.a. resources). Below is a description of each resource category:

- **User resource**: To track and authenticate OpenDSA users.
- **Book resources**: A set of tables to store OpenDSA book instance information, which includes the book name and URL. They also manage the list of learning modules in the book and the list of interactive entities in each module. For each interactive entity, a name, a description, and a classification of the material type (e.g., SS, AV, PE, or KA) is stored.
- **Students interactions resources**: Interactions captured by the DCS have two main types:
  - **Interface interaction resources**: stores student click streams and module-level events.
  - **Exercise interaction resources**: stores student exercise attempts.

  We will discuss interaction types in more detail in the next section.
- **Student progress resources**: a set of tables that tracks student proficiency status and dates for each exercise or module.

[2] https://djangoproject.com

DCS exposes its resources through a set of standard APIs, which are not open for public use. Rather, DCS only authorizes requests from domains that are explicitly defined in its whitelist of content server domain names. The separation between the content server and DCS allows OpenDSA collaborators to install only the content server, and ask the OpenDSA team at Virginia Tech to add their domain name to the Virginia Tech DCS. This design flexibility helped with content dissemination, although it raised some performance concerns with the DCS deployed at Virginia Tech. A full description of the original OpenDSA DCS design can be found in [16].

DCS includes infrastructure for automated assessment of programming exercises. It allows students to perform programming assignments through an in-browser coding interface. When a student submits his code snippet, an attempt object is transmitted with the student code for evaluation. An automated assessment engine within DCS stores and evaluates student code. A complete description of the programming exercises embedded engine can be found in [21].

2.4 Client-Side Framework

The client-side framework consists of multiple JavaScript files that are loaded with each module page. The main role of this framework is to manage communications between the module page and DCS. It also provides common functionalities for all OpenDSA interactive materials. It provides a level of abstraction for all common behaviors that OpenDSA interactive materials might need. Interactive materials automatically inherit these common behaviors, which makes a content developer's job easier and helps them focus on the design and implementation tasks. The framework is mainly two files, `odsaMOD.js` and `odsaAV.js`. The former file serves as an abstraction for the module page, while the latter contains common code for all embedded AVs and PEs.

The client-side framework, among other responsibilities, mainly performs:

- User registration and authentication
- Transferring module information to the DCS to save new exercises in the database
- Buffering and transmitting user score data and interaction data to the DCS
- Managing user proficiency displays and keeping local proficiency in sync with the DCS

The client-side framework was designed with the goal of allowing users to use the book instance and see their progress even without the DCS being accessible. This goal was achieved by making use of the browser's local storage [35] to track users' scores and proficiency status when the book instance is not configured to use the DCS.

2.4.1 User Registration and Authentication

All pages in any book instance have links to allow users to register or login. When a new user opens a page, he can use the "Register" link to open a form that allows him to input his basic information. When the form is submitted, an AJAX request is sent to the DCS and a new user is created. The DCS sends a user session key back to the client-side framework, which in turn shows the new user as logged in, and stores the session key in the browser's local storage. Any subsequent communication between the client-side framework and the DCS would be identified by this session key. Each time a student logs in, the DCS creates a new session key. As a consequence, if a student logs in the second time, then the session key saved by the client-side framework becomes invalid. When the client-side framework sends any request using the invalid session key, the DCS rejects that request and returns an HTTP 401 error code which causes the framework to delete the session key, and to inform the student that he needs to log in again because his session is no longer valid.

2.4.2 Loading New Exercises into the DCS

The client-side framework, as well as OpenDSA contents, are hosted on the "Content Server". On the other hand, the DCS is designed to be deployed on a separate server. At Virginia Tech these two systems are hosted on different physical servers with two different URLs. OpenDSA's original architecture enforces this separation to make it easier for instructors to adopt the content server without the necessity of having a DCS if the instructor is not interested in DCS functionalities.

It was believed that once an instructor sets up his content server, he should be able to establish communication with a DCS instance deployed in another institution. Adding this flexibility to OpenDSA's original infrastructure introduces challenges, one of which is the scenario when an instructor adds more contents or interactive materials to any of his book instance modules. This situation mandates that the client-side framework communicates the entire book configuration to the DCS each and every time a module is loaded.

2.4.3 Score and Interaction Data Management

OpenDSA's interactive materials are self-grading, which means that the exercise decides by itself, without consulting the DCS, whether a student attempt is correct or not. Based on the completion criteria for each interactive material (discussed in Section 2.1), a score event is fired.

The client-side framework provides two listeners in a module page for score events. The first listener is for JSAV-based slideshow score events. The second listener listens to a `postMessage` score event generated by the embedded AVs and PEs. The framework vali-

dates the score events, stores them in the local storage, and then later sends them in batches to the DCS. The communication mechanism implemented by the framework for managing and transmitting score data is the same for interaction data (we'll talk about interaction data later in this chapter).

Since reading and writing data to the browser's local storage is not an atomic operation, there is a high chance that new score events might be stored in the local storage before the DCS sends the response back. So if the framework simply waits until it receives a response from the DCS, and then deletes the scores from the local storage, there is a high chance of losing score events in this process. To mitigate this situation, the framework would copy the score events and delete them immediately from the local storage, then send them to the DCS. Upon successful response from the DCS, the framework just deletes the copied score events. Otherwise, it rewrites them back to the local storage.

2.4.4 Managing User Proficiency

A student becomes proficient with an exercise when he can demonstrate "sufficient" knowledge about the concepts tested by the exercise. Based on the exercise type, the term "sufficient" translates to different criteria. Once a student achieves proficiency, the client-side framework sends the score event to the DCS and caches the proficiency status in the browser's local storage. The framework keeps the local proficiency cache synchronized with the DCS. So for each book, for each student, the framework stores the status for each exercise. Local exercise status can be "`SUBMITTED`" when the score is sent to the DCS, "`STORED`" when the DCS sends back a successful response, and "`ERROR`" when the DCS sends back a failure response. In the last case, the framework will change the proficiency indicator to show an error, and provides the students with a "Resubmit" link to submit the score manually. Figure 2.5 shows a state diagram of the local proficiency cache's different states.

For the content server to function properly without the DCS, the client-side framework has to save and maintain the student's proficiency in the browser's local storage. This caching mechanism allows the student to continue his progress even when there is no DCS or when the connection with the DCS server is temporarily unavailable, either due to a poor or no Internet connection. In any case, the student can still maintain his progress locally until he logs in, then the framework will synchronize his progress to the DCS.

Proficiency indication has two forms, a green check mark on the upper left corner of a slideshow, or a button placed near the top left corner of the embedded exercise. The button changes color based on the local cache status. The initial color is red, which means the student is not yet proficient. It becomes yellow when the status is "`SUBMITTED`" or "`ERROR`", and finally it turns green when the student score is stored by the DCS, and the local cache status becomes "`STORED`".

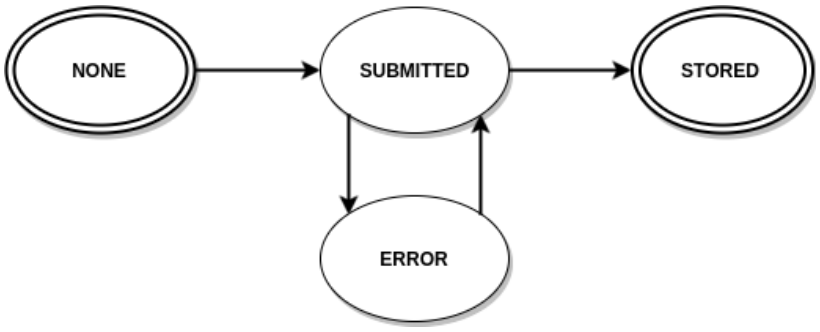

Figure 2.5: A state diagram for the exercise proficiency cache. Until a student starts the exercise, there is no state saved in the cache. When the client-side framework determines that a student achieves proficiency, a score object will be sent to the DCS, and the state changes to "`SUBMITTED`". If an error is responded to by the server, the state will change to "`ERROR`". Students can manually resubmit the score object again for exercises in the "`ERROR`" state. Finally, once DCS has saved the information successfully, the state changes to "`STORED`"

.

When "Module Complete" appears at the top of a module, it means that the student has already answered all the required exercises in this module. "Module Complete" is colored yellow or green. Yellow indicates that the student has achieved proficiency with all the exercises; however, one or more is not yet "`STORED`" in the DCS. Green indicates that all exercises are successfully marked as completed in the DCS.

When a book is configured to communicate with a DCS and a student is logged in, the client-side framework synchronizes the proficiency status for all exercises once the module page is loaded. The DCS becomes the ground truth when checking for student proficiency. Otherwise, the client-side framework has full authority to store and manage the student's proficiency.

2.4.5 Managing Student Login in Multiple Browser Tabs

It is important to keep all browser tabs consistent, therefore, the client-side framework developed a mechanism to check whether a student is logged in, and to ensure only one student is logged in during one session. If a student logs in through a tab and after that another student logs in through another tab on the same computer, the former student will be logged out, and both tabs will show the latter student username. Likewise, if a student logs in through a tab, opens another tab, then logs out, he will appear as logged out when he switches back to the first tab. The logic to maintain consistency between tabs was developed in the `loadModule()` function.

2.4.6 Transmitting the Data

The DCS collects two categories of data: student interaction events and exercise attempt events. Both event types are buffered in the browser's local storage as a list of objects, for each student, for each book instance. The buffered objects are sent to the DCS in batches to increase client-server communication efficiency by reducing the network traffic. Exercise attempt events fire whenever the client-side framework determines that the student has completed an "attempt" with an exercise. The precise information that is transmitted to the database server depends on the exercise type. Below is an example of information pieces included in the PE attempt object:

- `exercise` - the name of the exercise with which the attempt event is associated
- `module` - the module with which the attempt event is associated
- `score` - a decimal number between 0.0 and 1.0 indicating a student's attempt score
- `steps_fixed` - the number of steps fixed by the auto-grader during the attempt
- `submit_time` - the time in which a student finished the attempt
- `total_time` - the total time a student spent working on the attempt
- `uiid` - the unique instance identifier, which allows an attempt event to be tied to a specific instance of an exercise in a specific load of a module page
- `username` - a student username

The following is the attempt object attributes for a KA exercise:

- `exercise` - the name of the exercise with which the attempt event is associated
- `module` - the module with which the attempt event is associated
- `count_attempts` - attempts counter for how many times a particular question has been attempted
- `attempt_content` - student answer
- `correct` - "1" if the answer is correct, "0" otherwise
- `count_hints` - hints counter for how many times a particular question hint has been used
- `time_taken` - the total time a student spent working on the attempt
- `remote_adrr` - student IP address

When a batch of student attempts objects are sent to the DCS, a session key and book instance identifier are added to the request message.

Student interaction events, in turn, are two main types. The first is student click events, either on control buttons or on interface objects related to manipulating an exercise (such as clicking on the value in an array to show behavior in a sorting algorithm). The second type of interaction event relates to loading or unloading a module page, or changing focus into or out of the page. Interaction events will typically transmit data fields such as:

- `av` - the name of the JSAV-based material with which the event is associated (`null` string if it is a module-level event)
- `module` - the module with which the attempt event is associated
- `book` - the identifier of the book with which the event is associated
- `desc` - a stringified JSON object containing additional event-specific information
- `submit_time` - a timestamp in which the event has occurred
- `type` - the type of the event: module load/unload event or JSAV event
- `uiid` - the unique instance identifier, which allows an attempt event to be tied to a specific instance of an exercise in a specific load of a module page
- `username` - a student username

2.5 Putting It All Together

Any instructor who wants to use the OpenDSA system in his class will probably follow the following steps to set up his course.

First, the instructor has to select a set of OpenDSA learning modules. That usually happens by looking at OpenDSA sample book instances for different CS courses. If any of these standard books satisfies the instructor's needs, he can use them as is. Otherwise, he may choose to mix and match a collection of modules, reordering them in the way that satisfies his curriculum. Or he may find that a topic he wants to teach is not part of OpenDSA materials. In this case, he might decide to write the missing modules or interactive materials and contribute them to the project repository.

Second, the instructor asks the OpenDSA team at Virginia Tech to prepare a configuration file for his book instance. Whether he chooses a standard book or a customized one, he will end up with a configuration file that he then can further choose to tweak for a specialized version of his course. The instructor can choose the book language, select one or more of the programming languages for code examples presented in the book, adjust the number of points that each exercise is worth, and modify some other exercise-specific configurations like the grading method for a particular exercise.

Finally, the instructor has to decide whether he wants OpenDSA to be used only as reference material for his students, or if he wants OpenDSA to track his students' progress and save

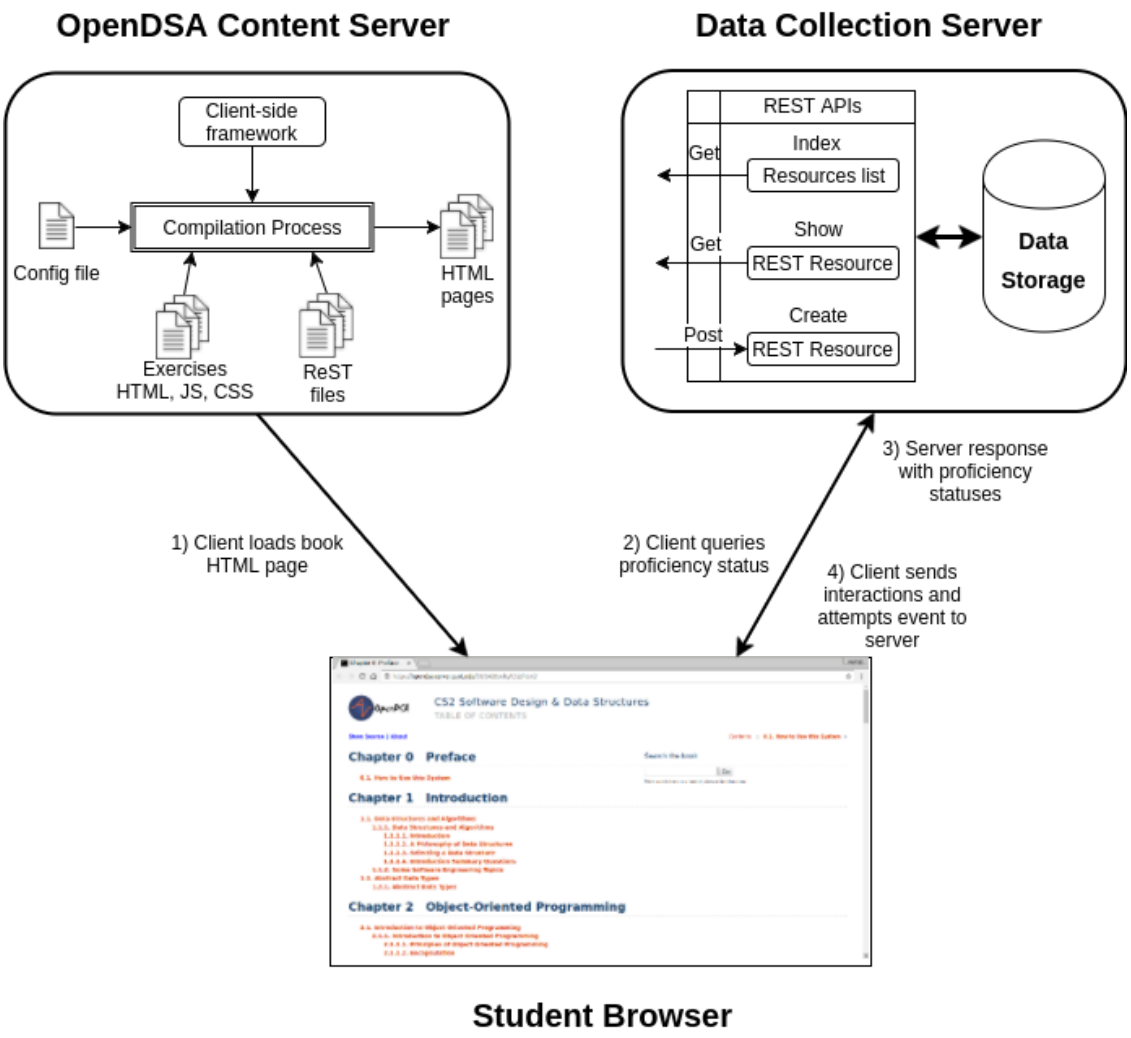

Figure 2.6: First Generation OpenDSA infrastructure

each student's exercise credit. In the latter case, the instructor has to define the DCS URL in the configuration file as well.

Once the instructor has a configuration file in hand, he can start the compilation process using the file on his content server. If the instructor did not have a content server available, then he would ask the OpenDSA team at Virginia Tech to host his course on the VT content server. Once the compilation process is completed, the newly generated book is comprised of multiple HTML pages, one for each learning module. All book pages are Web accessible. The instructor then shares the generated book URL with his students. Figure 2.6 depicts a high-level view of the OpenDSA client-server architecture.

When a student opens a book module, he loads one HTML page that contains prose describing the topic. In the module page, there may be interactive materials. Slideshows are defined

as inline HTML markups in the module page, while other visualizations and exercises are embedded in the module page through iframes. As illustrated in Figure 2.7, the module page loads the client-side framework with the `odsaMOD.js` file.

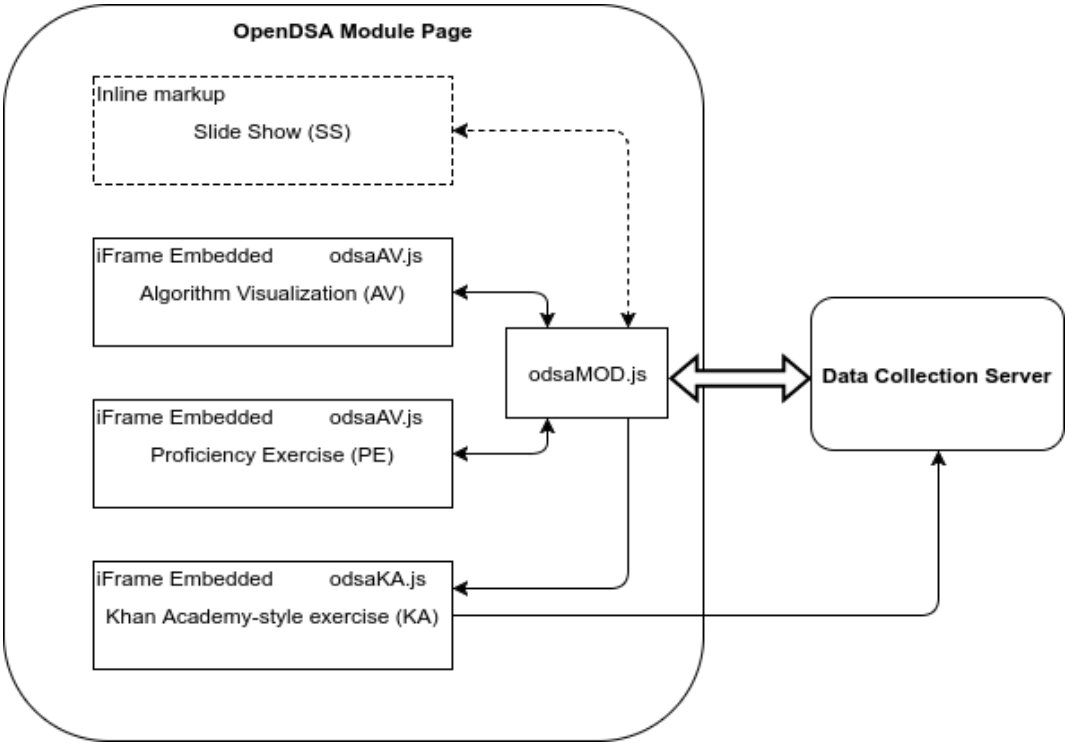

Figure 2.7: Client-side framework's role in a module page

The communication between `odsaMOD.js` and embedded exercises happens in two directions. The first direction is from `odsaMOD.js` to the AVs, PEs, and KA exercises. In this scenario, once the page is loaded, `odsaMOD.js` queries the DCS for all exercise proficiency statuses in the module; then it communicates each status to its embedded exercises. If a student was proficient, the exercise would show the proficiency indicator (the green check mark). The other direction is from AVs and PEs to `odsaMOD.js`. In this scenario, `odsaMOD.js` receives student interaction events and attempt events, which in turn are communicated to the DCS. As an exception, KA exercises send the attempts directly to the DCS without communicating with the `odsaMOD.js` library. The communication between any embedded exercise and the module page is handled by sending the data in a JSON format using HTML `postMessage`. `postMessage` is a Web messaging API introduced in HTML5 specifications allowing documents to communicate with one another across different origins or domains. The use of the `postMessage` API provided OpenDSA with more flexibility, where a module page can embed any external exercise hosted on a different domain. Given that they have already agreed on the message format, a module page and the external exercise can communicate score and proficiency information.

Chapter 3

Mitigating Original Infrastructure Problems

When I joined the OpenDSA project, I found that the majority of our collaborators were working on content development. Reading through the OpenDSA literature [8][16], I found known problems in the first infrastructure that had been reported by former collaborators.

In this chapter, Sections 3.1, 3.2, and 3.3 discuss initial efforts to mitigate various problems with the original infrastructure. Section 3.4 presents the reasons that led us to adopt a completely new infrastructure.

3.1 Khan Academy Exercise Framework

The Khan Academy (KA) exercise framework is a critical component in the OpenDSA project since it is used to deliver the majority of exercises. In this section, I will discuss work done to advance the KA exercise framework technically by enhancing students' experience, and pedagogically by preventing gaming behavior.

3.1.1 KA Framework Upgrade

The KA framework is used primarily for summary exercises, where one exercise contains a pool of questions. The framework picks one question at random and presents it to the student. In 2013, a snapshot of the KA framework repository was copied and made a part of the OpenDSA repository. Then the local copy of the KA framework was modified many times to satisfy OpenDSA requirements. On the other hand, the original framework was actively modified during the same time period, which led to a huge deviation between the two versions. We studied different options to rectify this situation and move forward. The

first option was to use a new exercise system, but since this option requires reimplementing all the existing exercises, we found it infeasible. The second option was to keep up with the latest version of the KA framework. However, we did not want to take a new snapshot and be in the same situation again after a few years. Thus, we decided to clone the KA framework under the OpenDSA account on GitHub, and reimplement OpenDSA requirements as a layer that overrides some of the KA framework functionalities, so that we can maintain compatibility and move forward by continuously pulling the original framework's latest modifications.

The latest KA framework uses RequireJS [10], which is an asynchronous JavaScript module loader. RequireJS aims at enhancing the loading time and improving the quality of the JavaScript code. Using RequireJS helped us to solve two main problems. First, OpenDSA KA summary exercises contain multiple individual questions, each in a separate file. The original framework loads these files synchronously, which made the overall page loading time relatively long. Using RequireJS, we could load the exercise files asynchronously which reduces the page load time dramatically. Second, the original framework requires the JavaScript code to be written in the same individual question HTML file, which made them harder to understand and maintain. Using RequireJS's dynamic loading feature, we could move the questions' JavaScript code to separate JS files. Also, we did not have to worry about naming conflicts, tracking of dependencies, and JS module loading order since RequireJS effectively handles these complexities. Upgrading to the latest version of KA framework was the ideal solution since the OpenDSA project has many collaborators working together at the same time as a team.

While upgrading to the latest KA framework enhanced students' experience and helped OpenDSA developers as well, it was not without limitations. The latest version of the KA framework does not provide a native method to recursively load summary exercises. OpenDSA KA summary exercises, especially the ones at the end of each chapter, are not implemented as a pool of individual questions, rather, They include a set of other summary exercises. We have implemented the asynchronous recursive loading of questions in the `odsaKA.js` layer that we wrote on top of the KA framework.

We have also added a new feature to `odsaKA.js` to let the framework differentiate between static questions and dynamically generated ones. This feature was required because we wanted the framework to remove static questions from the question pool once the student answers them correctly. On the other hand, randomly generated questions should be kept in the pool because the student will get a new instance of the question every time he loads the exercise.

3.1.2 Preventing the "Gaming" Problem

It is widely known among developers of eLearning systems that students will attempt to "game the system" [2][18]. This term refers to a range of behaviors that allow the student to gain credit without achieving the intended goal of learning the material. Through log

analysis [18], we discovered a number of ways in which students gamed the OpenDSA system, and we were able to take steps to deter some of this behavior.

One type of exercise commonly used in OpenDSA is a Summary, a collection of individual questions. There might be ten questions, and a student has to get five points to successfully gain credit for the exercise. Each time the student correctly answers a question, he gets a point, and wrongly answering a question loses a point (this is done to deter the gaming behavior of guessing). Students can also ask for a hint, which negates the question with respect to credit (this is done to prevent the misuse of the hints system). Each time the student correctly answers a question, a new question is selected at random. A given question might, therefore, be repeated later during the course of completing the Summary exercise. In the original system, reloading the browser window would cause a new random question to come up, without charging the student for a wrong answer. So some students would learn the answer to a few questions (possibly by abusing the hints system, or just by finding some relatively easy questions in the group). Then they would repeatedly reload the browser page until they got enough repeats of those questions to score the necessary points. We revised the system to remember the "current" question that a student is on. They are not allowed to progress to another question until the current one has been resolved. In particular, reloading the page will just repeat the question.

Then we discovered that students use the hints system to get closer to the answer, but before they submit the answer they would refresh the page to fool the system by answering the question after page reload to get the question credit. To discourage this behavior we also tracked the question on which a student uses the hints, then allow the student to answer the question without penalty or reward even if he refreshes the page. We also remember if a static question (one with no randomized component to make it different each time) has been completed, in which case we will not present it again until the student has been shown all of the other questions at least once.

While implementing the tracking mechanism, there were some cases that needed careful attention. The first is after a student had already gained credit for the exercise. In this case we no longer want to remove any question from the questions pool, on the contrary we want to avail all questions for the student to keep practicing. The second is when a question pool runs out of questions. This situation can happen because the tracking system keeps removing questions from the pool while the student was alternating between correct and wrong answers, but he still has not answered the five questions required to gain exercise credit. Once the pool runs out of questions, we make all the questions available again. The third is when the student has only one question left in the pool and he decides to use the hints on that question. This situation will result in a student being stuck with the last question forever without getting credit for it. In this situation, we also return all questions to the pool.

Preventing gaming behaviors not only discouraged students from adopting harmful behaviors; it has a direct impact on OpenDSA research. Researchers usually depend on the time

a student spends to solve a question as a measure of question difficulty. Preventing gaming is considered as an approach that produces more accurate data that researchers can depend on during analysis.

In the original implementation for Summary exercises, we were not properly tracking which sub-question was being answered at any given point in time. This made it difficult to determine which specific questions were too easy or too hard, which were being gamed, and which generated greater use of the hints system. We redesigned the tracking mechanism to log each individual question.

3.2 Client-Side Framework

As discussed in Section 2.4, the client-side framework acts as the brain of the module page. It is responsible mainly for communicating student interaction data and exercise attempts to the DCS. Below is a discussion of how we mitigated client-side framework known issues.

3.2.1 Missing Interaction Data

The mass of data communicated to any cyberlearning system is only useful if it can be collected accurately and then analyzed successfully [6]. OpenDSA researchers depend on interaction data to analyze students' learning behaviors and to get insights on how OpenDSA book instances are used. Researchers convert the interaction row data from syntactic actions (mouse clicks and page loads) to semantic behavior (time spent doing an exercise, exercise success rates, whether students use a given visualization). Most detailed behavior analysis involves tracking the amount of time spent by a user on an interactive element. Time measurement is a widely used method to analyze student behavior [3][2]. Daniel A. Breakiron, a former master's student at Virginia Tech who developed the client-side framework, has reported evidence that some of the data communicated to the DCS were lost [8]. We revised the client-side framework design and found out that there were specific scenarios during which interaction data was indeed lost.

Two students use the same browser: In this scenario, the client-side framework selects the interaction data of both students and sends them to the DCS with the session key of the last logged-in student. The DCS uses the session key to retrieve the student, then validates each interaction data record received and checks whether it belongs to that student or not. Since data sent has mixed objects for different students, the DCS rejects the entire request and sends back an error message to the client-side framework, which in turn removes all the interaction objects from the local storage. To rectify this situation, we changed the client-side framework to filter the interaction data and send only objects that belong to the current book and the currently logged in student.

When students close their browser abruptly: During data transmission, event data are removed from local storage and kept in memory. If the student closed the browser before the data were successfully transmitted to the DCS, the data would be lost. Likewise, if a student clears his local storage, then the data to be transmitted will be lost. We solved this situation by sending batches to the DCS more frequently to decrease the possibility of event loss. In addition to sending batches on a timely basis, we also send them whenever a student gets proficiency with any exercise.

Wrong mechanism for communicating interaction data: We found that copying the interaction data from and to the local storage, as discussed in Section 2.4.3, would make the data more likely to be lost. So we changed the mechanism to only copy the event IDs in the memory and leave the event objects in the local storage. When the DCS sends a successful response we would remove the events from local storage, otherwise, we would leave them to be sent as part of the next batch.

3.2.2 An Out-Of-Synch Local Proficiency Cache

In Section 2.4.4 we discussed the purpose of storing student proficiency information locally in the browser's storage. It is simply to allow students to use an OpenDSA book instance even without login or when there is no Internet connection. The idea seems reasonable; however, the implementation was problematic.

We received complaints from students that they have already completed an exercise but did not see the proficiency indicator. Even worse, sometimes students see that they are proficient with an exercise, but they did not get a grade for it in the gradebook. We investigated these cases and found that the local cache was not synchronized properly in some situations. To correct this issue we decided to define the DCS as the ground truth for student grades, and stop caching the proficiency information. This change requires that every time a module is loaded, it would send a query to the DCS for the module proficiency status as well as all its exercise statuses. When the client-side framework receives these statuses it will send them directly to each exercise to update its indicator. The gradebook page was no different; it also queries the DCS for all students' grades in a specific book instance. As a side effect, this solution requires students to always have an Internet connection while using OpenDSA.

3.3 Data Collection Server

The DCS is the central place to store and evaluate students' exercise attempts. It also provides services to store interaction data, manage student accounts, and manage book instances.

In this section, we discuss some of the issues with the original DCS that led in the end to a decision to entirely replace it with a new infrastructure.

3.3.1 Proprietary Mechanism for Handling Session Keys

Django provides full support for session handling. The Django session framework lets the developer store and retrieve arbitrary data on a per-site-visitor basis. It can store data on the client side or the server side and abstracts the sending and receiving of cookies. In the latter case, cookies contain a session ID, not the data itself. For each Django installation, developers should set up a `SECRET_KEY`, a unique, unpredictable value. `SECRET_KEY` is used to provide cryptographic signing for session data, which makes it protected against changes by the client so that the session data will be invalidated if tampered with.

Using cookies to handle session data is considered to be the most secure approach. Thus it is implemented and provided by default in almost all modern Web frameworks. Unfortunately, the DCS did not use the standard, secure way to handle sessions. Instead, it implemented a proprietary mechanism which not only made it insecure and an easy target for attackers, but also the mechanism put a restriction that only one session key per user is allowed. For example, if a student uses his laptop to log in, and later he logs in using a lab machine or his desktop at home, the old session key on his laptop is no longer valid, and he has to re-login again. We also believe that manual session handling collectively with the wrong mechanism used to handle interaction data has contributed to data loss as discussed earlier. The manual way the DCS handled session keys has also complicated the client-side framework and made it harder to maintain.

3.3.2 Automated Assessment Engine for Programming Exercises

The DCS contains an embedded engine to automate programming exercise evaluation. The engine was written in Python and was made part of the DCS. The DCS system provided endpoints used by the client-side framework to send students programming attempts. These attempts were different from other exercise attempt objects since they contain students' source code, and this code is evaluated on the server side rather than the client side. The DCS receives the source code, stores it, then triggers the evaluation engine. The evaluation process has multiple phases, and it was designed to run the code in a separate thread. The thread times out if it is taking too much time to run (typically around 2 seconds).

We received complaints from students that the program evaluation process was taking a long time. Right before exercises are due, students would submit their code and wait for a while. If they did not get a response, they would refresh the page to re-submit again which put more load on the evaluation engine and increased the failure rate. Due to the high load, the engine quickly became unstable and incorrectly evaluated students' code. Students were

confused because they could not tell whether the engine has a problem or their code is not correct.

We realized that the evaluation process exhausted the server resources, and it was a wrong decision to embed the engine in the DCS. We had two options, either we deploy the engine on a separate server or replace it altogether with a sophisticated, stable program evaluation system.

We chose the second option and decided to use CodeWorkout. It is a new, online, scalable drill-and-practice system. CodeWorkout provides many online repositories of programming questions that can be used in OpenDSA standard courses. It also facilitates creating new programming questions, which we found helpful to fill the gaps by implementing our specific exercises like recursion and binary tree exercises.

What also encouraged us to use CodeWorkout is that the application owners had plans to integrate it with the Canvas learning management system through LTI. It was our plan as well to integrate OpenDSA with Canvas, as we will discuss in detail in the next chapter.

The challenge now becomes: how to remove the old engine from the OpenDSA system in both the client-side framework and the DCS without breaking the system? Also: how to effectively integrate CodeWorkout in the two different offerings of OpenDSA (the "plain" books and book instances instructors used for credit) without affecting the students' experience?

In Chapter 5, we will present how we designed the new OpenDSA infrastructure to integrate not only CodeWorkout, but any LTI-compliant learning tool in the OpenDSA book instances.

3.4 Towards a Complete System

As can be seen from the description of the first generation infrastructure presented in Chapter 2 and the discussion in Sections 3.2 and 3.3, the communications between the client-side framework and the DCS are quite complicated because of improper handling of session keys. We mentioned in Section 2.4.1 that the DCS was developed using the Django Web framework version 1.3 released in March 2012[1]. This version was rather out-dated and no longer supported[2]. In addition, the DCS had to take on several critical tasks normally handled by an LMS. In particular, account login was particularly fragile. As a consequence, there were many problems and bugs with the original DCS. Also, the DCS was built on top of A+, which we wanted to replace for real LMS support. Finally, we wanted to be able to integrate 3rd party exercises like CodeWorkout, which DCS could not support.

Due to the DCS limitations, security concerns, and the fact that Django version 1.3 is out of date, we decided to decommission the DCS and design a new infrastructure that

[1] https://docs.djangoproject.com/en/1.11/releases/1.11/

[2] As of this thesis writing the latest Django version is 1.11, which was released in April 2017.

transforms OpenDSA from a standalone, self-contained eTextbook to an integrated learning tool communicating with an LMS.

In the next chapter, we evaluate the considerations for and against the available integration approaches for use by the new OpenDSA infrastructure.

Chapter 4

Interoperability Alternatives

As software technologies evolve, modern software systems become more complex. Thus, there is a necessity to divide such systems into independent components that collectively communicate to satisfy the overall system requirements [14].

Separating Web-based systems into components has been studied carefully over the last decade [14]. There are multiple approaches and methods proposed for separating these systems. These methods identify how best to partition a system, how independently each component of a system can be further developed and maintained, and how all the components can be integrated together using standard integration protocols.

An LMS is a complex software system; it has requirements that continuously change because Web technologies change or new studies suggest adding more features and capabilities. Therefore, many LMS vendors have designed their systems to be modular and extendable. Some provide a flexibility level that allows customers to extend the system by developing extensions or connecting to extensions created by other vendors [12].

Dagger et al. [12] present some of the approaches used by the majority of the LMS vendors to make their systems flexible and extendable. A number of standards have emerged recently that support interoperability of an LMS and separate learning content.

In this chapter, we will discuss the different ways by which LMSs provided extendability to their systems.

4.1 A Component-Oriented Approach

Component-oriented is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This approach emphasizes separation of concerns, where each component has a certain feature or concern it is meant to solve.

Although it improves the scalability and extendability of the system, this approach still requires the components to be implemented with compatible technologies. Moreover, adding new components might impact and require changing the existing parts of the system.

OpenEdX[1] is one example of an LMS with a component architecture, called XBlocks[2]. The OpenEdX LMS consists of multiple XBlocks that each focus on different features. XBlocks should be developed using Python, which is the programming technology that the OpenEdX platform uses. We considered OpenEdX and the XBlocks architecture as a potential basis for the new OpenDSA infrastructure when we started looking for interoperability options. We tested this option by developing XBlocks to integrate OpenDSA with OpenEdX. Unfortunately, the development workflow for XBlocks was not straightforward. However, that was not a major issue since we had to develop only a small number of XBlocks. A "module" XBlock could render OpenDSA book modules, and an XBlock would be needed for each OpenDSA exercise type. We ended up developing three different XBlocks, which can be found at https://github.com/OpenDSA/OpenDSAX. Although we had a successful proof of concept, we had many concerns. The main concern is the lack of support from the EdX community for implementing new XBlocks. It was a strong sign that the XBlocks architecture might not continue in the near future. Another concern was that OpenDSA would only work within the OpenEdX framework, since XBlocks do not integrate with other LMSs.

Appendix A provides more details about our experience with developing XBlocks for OpenDSA.

4.2 Plug-In Architecture

A plug-in architectural approach consists of a host system and multiple replaceable modules, each of which provides a specific feature of the system. Each plug-in adds a new feature without impacting the functionality provided by other plug-ins. Although plug-ins can be configured to suit users' needs, they are always used without modifications to their default configurations. Plug-ins require access to the host application database and file system, which make them susceptible to be broken if the host database was changed. The host applications provide APIs, which the plug-in must adhere to. The APIs specify the way for plug-ins to register themselves with the host application and a protocol for the exchange of data with plug-ins. The plug-in architecture has many advantages, like enabling third-party developers to create abilities that extend an application, reducing the size of the host system by separating its source code from the plug-in itself. However, the plug-in should be implemented using the same technologies as the host system, and they do not work across multiple systems.

Although Plug-in architecture is more loosely coupled than the component approach, it still

[1] http://docs.edx.org
[2] https://open.edx.org/xblocks

shares the same data models of the host system, like users, assignments, and courses.

4.3 Widgets

Widgets are small applications with limited functionality that can be embedded in a Web application. A widget usually plays an auxiliary role. It occupies a portion of a Web page and displays, in place, something useful from the information collected from the Web. Web widgets have multiple synonyms like portlet, Web part, gadget, and badge. Unlike plug-ins, widgets can be deployed in a separate server other than the system utilizing them.

A number of initiatives to produce standard specifications for widgets have emerged. Google Gadgets[3] and OpenSocial[4] aim to create widget platforms. Google's implementation of the platform is complex, with many internal dependencies. Even though it has many good features, the Google solution is a proprietary solution, and unfortunately, Google did not continue to contribute to open standards in this area. On the other hand, the W3C's widget specification initiative[5] is an open standard approach adopted by a larger group of technology leaders including Apple, Microsoft, Yahoo, Nokia, and Opera.

Although widgets mitigated some of the limitations on plug-ins and solved a few of the issues related to compatibility and ease of installation, widgets suffer from problems in security and privacy.

Wilson el at. [38] created a widget container for Moodle based on the W3C specifications. The container is used for placing widgets for collaboration features like chatting, voting, and a discussion forum. The authors had to make some extensions to W3C specifications to develop their container. They hoped to see a convergence between Google and W3C, which might lead to a more mature standard.

4.4 SCORM

The Sharable Content Object Reference Model (SCORM) standard of the Advanced Distributed Learning[6] (ADL) initiative is a set of specifications for creating and sharing e-learning material compatible with several LMSs. The specifications leverage standard Web technologies as well as existing learning technology specifications. SCORM specifications describe how the learning objects are packaged to create interoperable, plug-n-play, browser-based e-learning content. It also specifies how the learning content is described with metadata and how they operate at runtime [4].

[3] https://developers.google.com/gadgets
[4] https://developers.google.com/opensocial
[5] https://www.w3.org/TR/widgets/
[6] http://www.adlnet.org

Based on our review of existing literature, SCORM appears to be a widely adopted e-learning standard. However, compared to other emerging interoperability standards, SCORM was old. The latest version SCORM 2004 (4th edition) was released in 2009[7].

4.5 IMS Learning Tools Interoperability

The IMS Global Learning Consortium[8] has released a standard specification called Learning Tools Interoperability (LTI) [22]. The LTI specification consists of two parts, Full LTI and Basic LTI. Their goal is to improve integration of external learning content in LMSs. Both versions include the same main actors, called tool providers (TP) on the learning tool side, and tool consumers (TC) on the LMS side. Both TP and TC pass messages over the network and use HTTP for the service interfaces. In Full LTI, the TP and TC need an initial negotiation process to agree about the services and security policies. Full LTI allows supporting a wide diversity of learning tools and features, and the communications can be initiated from TC to TP and vice versa.

The Basic LTI is a subset of the Full LTI and it only exposes a single address and a single HTTP POST request from TC to TP. Therefore, the interactions between the LMS and the learning tool are limited. The messages between TC and TP in both Full and Basic LTI are signed with the OAuth protocol, and therefore the participants need to set up and share a key and a secret before initiating communication.

LTI has received significant support from the IMS community, and has a positive reaction from LMS vendors. In April 2015, IMS Global announced record levels of member growth and adoption of LTI Standards[9].

4.6 Summary

The integration approaches discussed in the previous sections have been used successfully at some point in existing LMSs. However, the biggest challenge to choosing the right approach is to understand its benefits and implications.

To choose the right approach we compared each one against the objectives we set in Chapter 1. We excluded component-based and plug-in approaches as they seem to be restrictive. Both approaches require the learning tool to be developed in the same technology used by the LMS. We could overcome this restriction by implementing a wrapper for each LMS technology. We decided against this workaround as it would complicate the overall solution

[7] http://www.adlnet.gov/adl-research/scorm/scorm-2004-4th-edition/
[8] https://www.imsglobal.org/
[9] https://www.imsglobal.org/pressreleases/pr150408.html

architecture.

One of the main objectives is to increase the ability to integrate OpenDSA with the widest possible range of LMSs. Therefore, we did not consider the component approach in particular, because it increases the coupling of the learning tools and the LMS.

Technically, widgets are more flexible than plug-ins. OpenDSA could be modified to provide its contents through a standard widget interface, and the W3C standard was a strong candidate. Moreover, authentication and authorization can be implemented using OAuth in the widget. However, the W3C widget specification does not include a standard way for exchanging learning-tools-specific data like student interaction events. Moreover, widgets do not provide a standard way to communicate student submissions and grades to the LMS. For these reasons, and because compatibility across LMSs would not be guaranteed, we excluded widgets.

We found the IMS LTI specifications most suitable for OpenDSA integration needs. LTI is increasingly supported by major LMS vendors. The IMS consortium community is growing, and specifications development is under continuous review and progression from LTI v1.0 (17 May 2010) until LTI v1.2 (5 January 2015)[10]. Compatible specifications from IMS under development as of the writing of this thesis include Common Cartridge [24] and Caliper Analytics [25]. LTI would standardize the process of building embedded links between OpenDSA and the LMS. LTI would enable us to make OpenDSA and LMS loosely coupled by building one wrapper over OpenDSA to integrate it with any LTI-compliant LMS. By adhering to an LTI interface, OpenDSA integration cost would decrease, OpenDSA adoption by instructors would increase, and we could transform the OpenDSA platform to a Software as a Service (SaaS) [37].

[10] https://www.imsglobal.org/activity/learning-tools-interoperability

Chapter 5

OpenDSA-LTI Implementation

This chapter is dedicated to present the implementation details of the new OpenDSA infrastructure, called OpenDSA-LTI, which can be accessed through http://opendsa.org. OpenDSA-LTI is a learning tool that integrates OpenDSA book contents with any LTI-enabled LMS. The new infrastructure followed the requirements and constraints in Chapter 1 and aimed at supporting multiple simultaneous courses and other services connected to it.

5.1 Technologies Stack

Since the new infrastructure would be developed as a Web application, the most important choice of technologies was choosing the Web framework on top of which the infrastructure will be built. A primary factor for choosing a Web framework is the programming language used for the LTI library implementation. We examined the IMS Global official account on GitHub[1] for various LTI implementations.

There were two main implementations for LTI; they use the Python and Ruby languages. The Python package (PyLTI 0.4.1)[2] was provided by the MIT Office of Digital Learning[3]. PyLTI supports LTI version 1.1.1 and LTI 2.0. It was originally written with OpenEdX as its LTI consumer; later an LTI provider was added to make it a complete implementation of the LTI specification. We examined the PyLTI repository[4] and found it was obsolete since 2015.

In contrast, an up-to-date and stable implementation for the LTI protocol was developed as

[1] https://github.com/IMSGlobal
[2] https://pypi.python.org/pypi/PyLTI/0.4.1
[3] http://odl.mit.edu
[4] https://github.com/mitodl/pylti

a Ruby library[5] and provided by Instructure[6] (the company that created Canvas).

Due to the LTI implementations and for other reasons described next, the Ruby on Rails framework[7] was chosen to implement OpenDSA-LTI. The Ruby on Rails framework has extensive documentation, active development, and a wide user base. In addition, many LTI learning tools (including CodeWorkout) and the Canvas[8] LMS are developed using Rails. This is an important consideration for us, since using Rails means that developers on the project would need to be familiar with only one Web framework.

Rails has a clear separation of different concerns; it enforces the Model-View-Controller (MVC) design pattern. The Model represents an object carrying data, View is the representation of the data, and Controller acts on both Model and View to control the data flow. Rails also have a powerful database layer that enforces the Active Record design pattern, which stores in-memory objects in relational databases.

The Rails community has developed a wide range of additional open-source modules called gems, which can be included in a new project to add new functionalities. They are publicly available and searchable through https://rubygems.org/. Third-party gems called Devise[9], CanCanCan[10], and ActiveAdmin[11] were utilized when implementing authentication, authorization, and an administration user interface, respectively.

Due to its database abstraction layer, Rails can utilize several databases with no modifications to the application's source code. No SQL queries were written in the OpenDSA-LTI source code, as all database access is handled through the Rails database abstraction.

During development, the OpenDSA-DevStack[12] virtual development environment was used. We developed OpenDSA-devStack to allow developers and collaborators to automatically create and configure a lightweight and portable development environment. In OpenDSA-devStack and production, a MySQL database is used due to its expected improvements in scalability, flexibility, and performance.

5.1.1 Software Choice

The relations between software components and services can be presented as a stack of layers and relations to external components and services. In this layered presentation, each layer shares a unified interface with the upper layers and with components and services on the

[5] https://rubygems.org/gems/ims-lti
[6] https://www.instructure.com
[7] http://rubyonrails.org/
[8] https://www.canvaslms.com
[9] https://github.com/plataformatec/devise
[10] https://github.com/CanCanCommunity/cancancan
[11] https://activeadmin.info
[12] https://github.com/OpenDSA/OpenDSA-DevStack

same layer. The layers of OpenDSA-LTI are shown in Figure 5.1. The upper layer in this architecture is OpenDSA-LTI itself.

OpenDSA-LTI's extendable design allows communicating with any LTI-enabled LMS. As of the writing of this thesis, we developed an adapter that compiles OpenDSA books in the Canvas LMS using APIs. The adapter can be extended to include other LMSs like Moodle[13], Blackboard[14] and Desire to Learn (D2L[15]). Once the book is compiled, OpenDSA-LTI is responsible for displaying the contents for these courses' users through LTI.

OpenDSA-LTI is built on the Rails framework, which maps URL addresses to Ruby functions and controls the requests and responses. Rails is also taking care of database communications and mappings between Ruby objects and database rows. Third-party Rails gems, such as Devise, CanCanCan, and ActiveAdmin, are represented in Figure 5.1 on the same layer as Rails. The software components on the upper layers are written in the Ruby programming language, and therefore are executed by a Ruby interpreter and uses Ruby standard libraries and gems.

OpenDSA-LTI runs on top of a Web server called NGINX[16], which is known for its high performance, stability, simple configuration, and low resource consumption. NGINX is an extendable Web server, A module called Passenger[17] is used to host the OpenDSA-LTI Ruby application. While OpenDSA-LTI can be run on top of other combinations of Web and application servers, to our knowledge, NGINX and Passenger are the best mix for Rails applications.

[13] https://moodle.org/
[14] http://www.blackboard.com
[15] https://www.d2l.com/
[16] https://www.nginx.com/
[17] https://www.phusionpassenger.com/

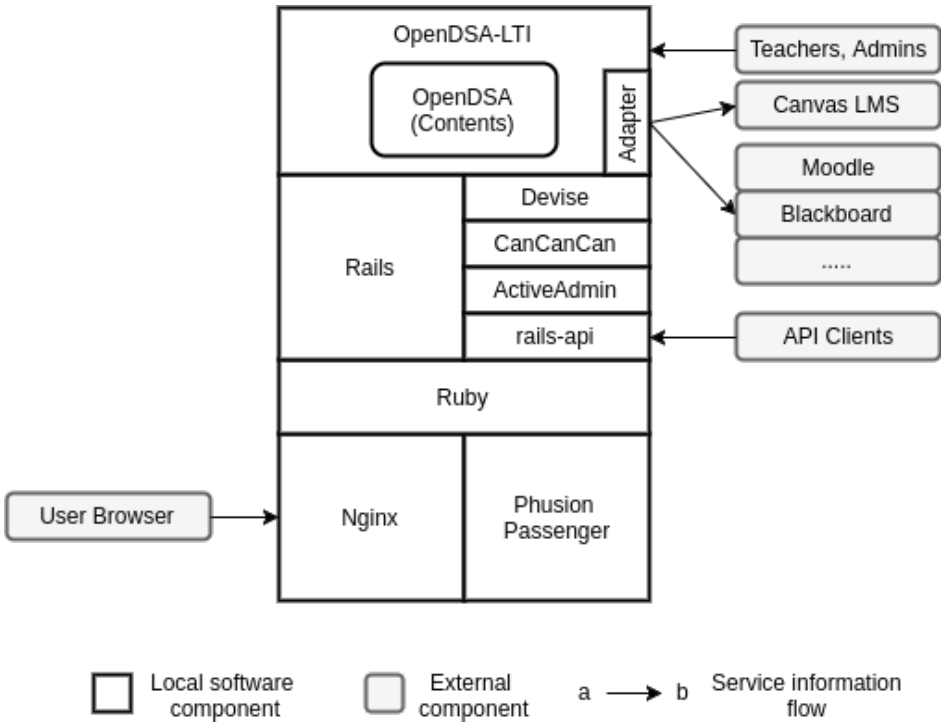

Figure 5.1: Software Stack for OpenDSA-LTI.

5.1.2 OpenDSA Redesign

In Section 3.3.1, we presented how the old DCS used a proprietary mechanism to manage authentication and session keys, which go against the standards for modern Web frameworks. In OpenDSA-LTI we used Devise, a popular authentication solution for Rails applications. Devise provides many services like encrypting passwords, email confirmations, and HTTP Authentication. In addition, many utility methods help secure the application and ease development and maintenance. Using Device not only reduced the OpenDSA-LTI system complexity but also simplified the client-side framework as well. For instance, unnecessary uses and session management methods were removed. Mechanisms to track user data using the local storage were modified to eliminate the use of additional unwanted data fields.

5.1.3 Licenses

All third-party applications, libraries, and components used in OpenDSA-LTI are published under open source licenses. OpenDSA-LTI itself is open source and licensed under the MIT License[18]. Table 5.1 summarizes the licenses for third-party components in OpenDSA-LTI.

[18] https://opensource.org/licenses/MIT

Table 5.1: OpenDSA-LTI software licenses.

Library Name	License Name
Bootstrap	Apache License v2.0
jQuery	MIT
Rails	MIT
NGINX	BSD-like
Devise	MIT
CanCanCan	MIT
ActiveAmin	MIT

5.2 Identity Management

Identity management in OpenDSA-LTI consists of two parts: authentication and authorization. Authentication refers to identifying users and controlling their login to the system. It is the process of verifying that "the user is who he claims to be". Authorization refers to a process of verifying that "the user is permitted to do what he is trying to do".

In the following subsections, we discuss how these aspects are implemented in OpenDSA-LTI.

5.2.1 Authentication

The authentication mechanism is implemented on top of the authentication framework provided by Devise. Devise supports, among other things, registration, database authentication, and password recovery.

The students' registration process is different from the instructor's process. As shown in Figure 5.3 step (3), when the student's browser submits the launch message to OpenDSA-LTI, the message contains the student's email address and his name. Upon receiving the message, OpenDSA-LTI will automatically register the student if he was not registered before, or authenticate him if he already exists. Instructors, on the other hand, need to manually register a new account in OpenDSA-LTI through `http://opendsa.org`. After instructors create a new account, they have to email the OpenDSA team to get instructor privileges. Since our system is open for public use, we added the instructor privilege request step to make sure that the system will not be misused.

OpenDSA-LTI instructors can update their personal information, reset their password, and

update their LMS access token stored in the system. An instructor's access token gives OpenDSA-LTI the necessary permission to manage his courses on his behalf on the target LMS.

Additional authentication mechanisms, such as Google, Facebook, and Twitter authentication can be later added to OpenDSA-LTI so that instructors can use their existing credentials from these platforms and do not have to register or remember new usernames and passwords to use the system.

Because we aim for OpenDSA-LTI to be widely adopted, we are planning to remove the instructor privilege request step and add the third-party authentication mechanism soon.

5.2.2 Authorization

While designing OpenDSA-LTI, we decided to utilize an existing flexible yet secure authorization model, rather than re-inventing the wheel and building it from scratch. We found the CodeWorkout permission model [29] is the most suitable for OpenDSA-LTI needs. The permission model allows any user to have two different roles, global role and course role. A global role defines a user as a system administrator, an instructor, or a regular user. A course role defines the user role in each course in which he is enrolled. For example, someone might have instructor access to the server, and be an instructor in one course, but a student in another course.

Instructors have the privilege to define their organization and courses, as well as to start new course offerings in different semesters. They also can upload and manage OpenDSA book configuration files. Finally, they can generate OpenDSA books in any LMS course after granting OpenDSA-LTI access to it.

Administrators are the super users; they can manage all aspects of the system, and they can perform actions on behalf of the instructors. For example, a typical scenario is when an instructor generates an OpenDSA book in his course and later the book content changes. In this case, the administrator can use the instructor access token to re-generate the book on behalf of the instructor.

Students by default will not have direct access to `http://opendsa.org`. Rather, they will be enrolled in the LMS course generated by their instructors and automatically registered in the system once they start to load any of the OpenDSA modules. Students are not even aware of opendsa.org or the fact that OpenDSA contents are loaded from outside their LMS course. That achieves one of the main goals of the infrastructure redesign, which enables students to move seamlessly between different learning activities regardless of where the learning tools may be physically located.

5.3 How OpenDSA-LTI Works

In March 2012, IMS launched LTI v1.1[19] (final version) in which both Basic LTI and Full LTI specifications were combined. This version includes updates and clarifications as well as support for the outcomes service, and it enabled communication to be initiated in both directions. The outcomes service provides OpenDSA-LTI with interfaces that allow setting, reading, and deleting student scores in the LMS. This version also provides support for IMS Learning Information Services (LIS) through outcomes service interfaces.

The LIS specification defines how systems manage courses, students, and outcomes (scores). As per the specifications, the LMS is not required to provide an implementation for the LIS services. Rather, the LMS can be configured to use the institute's existing student information systems. However, the LMS is required to provide OpenDSA-LTI with LIS pointers in the launch data. Figure 5.2 shows high-level interactions according to LTI v1.1.

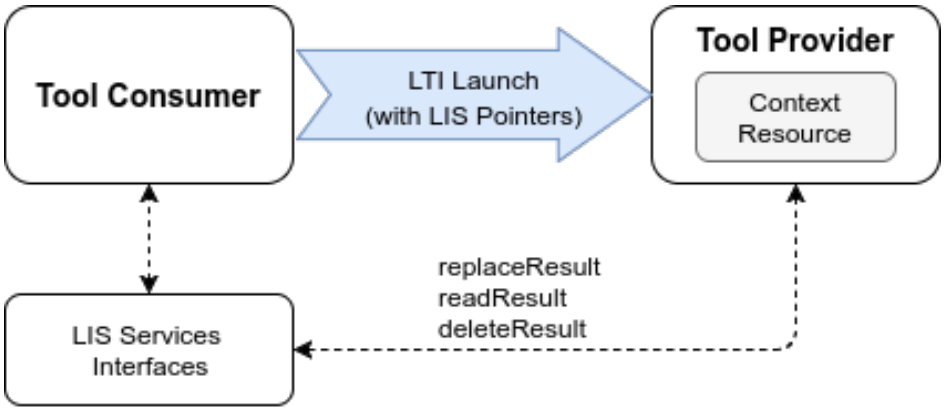

Figure 5.2: When the LMS launches a learning tool, it sends a launch message that contains an LIS URL. Later, the learning tool uses this callback URL to send grades back to the LMS.

In a context where student's grades are reported back to the LMS, OpenDSA-LTI calls the outcomes service interfaces, which support the following functionalities.

- `replaceResultRequest` - accepts a numeric grade (0.0 – 1.0) as a result for a particular activity.
- `readResultRequest` - returns the current score for a particular activity.
- `deleteResultRequest` - removes the score for a particular activity.

[19] https://www.imsglobal.org/specs/ltiv1p1

The LTI specification relies on the REST design model for Web applications [15]. LTI functions use RESTful Web services to exchange data between LMSs and external tools. Table 5.2 lists the LTI functions.

Table 5.2: LTI functions: *launch* is a POST message containing LTI parameters sent to the TP URL. *ReplaceResult* is a POST message sent to the LIS outcomes URL, and it contains two parameters: 1) LIS address to be replaced, and 2) a student grade on a scale from 0.0 to 1.0. *ReadResult* and *DeleteResults* retrieves and nullifies the student grade for a specific activity.

Function	REST method/URL	Parameters	Output
launch	POST/APP_URL	LTI_PARAMETERS	None
ReplaceResult	POST/LIS_OUTCOMES_URL	LIS_SOURCE_ID+GRADE	None
ReadResult	POST/LIS_OUTCOMES_URL	LIS_SOURCE_ID	GRADE
DeleteResult	POST/LIS_OUTCOMES_URL	LIS_SOURCE_ID	None

The launch function loads and executes a particular learning tool resource within the LMS. Listing 5.1 shows some of the launch parameters that the LMS sends to the learning tool. Two steps have to be done for students to be able to launch the LTI application. First, the LMS administrator or the course teacher should define the learning tool as an LTI application in the LMS by setting the name, launch URL, consumer key, and shared secret. Second, the teacher should create an activity in the course that refers to the LTI application.

Listing 5.1: Subset of LTI launch parameters

```
resource_link_id = 6a1735a118eafc8f996f161fec3923013e43f95a
resource_link_title = 01.01.01 - Data Structures and Algorithms
lis_person_name_full = Hossameldin Shahin
lis_outcome_service_url = https://.../tools/144033/grade_passback
lis_result_sourcedid = 144033-...-4b7e0cd604a9e0fcb2692964c25ae972
custom_ex_name = IntroSumm
```

When a student, later on, selects the learning tool, the LMS launches the tool by sending a HTTP POST request to the tool URL. Figure 5.3 shows a detailed description of the launch process.

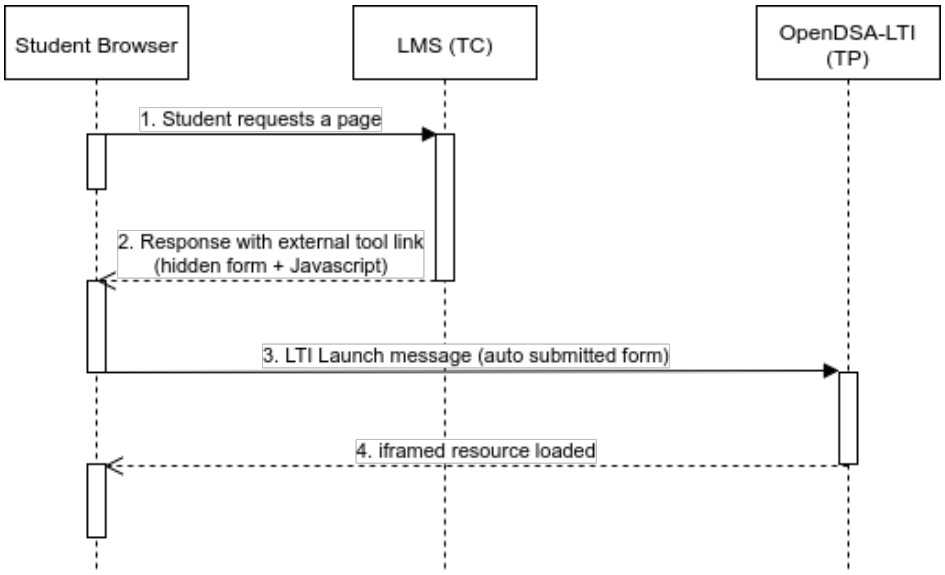

Figure 5.3: *launch* function sequence diagram

1. When the student clicks the OpenDSA-LTI link, the LMS performs these actions:
   (a) Create a launch message containing:
      i. User and activity context data, filtered by privacy policy defined in the LTI app.
      ii. All the custom fields that were configured during the LTI app definition.
      iii. A pre-configure key, which was shared with the OpenDSA-LTI.
   (b) Sign the message, using OAuth framework, based on a pre-shared secret.
   (c) Add the generated signature to the message.
   (d) Add a JavaScript code, which will automatically fire a POST request to the OpenDSA-LTI when the student loads the page.
2. Then, the LMS sends the reply message to the student.
3. The student browser automatically POSTs the launch request to OpenDSA-LTI, due to 1(d).
4. OpenDSA-LTI processes the request:
   (a) Extract the key from the message and search for it in the database to get the associated secret.
   (b) If the key is not found, raise an exception. Otherwise, sign the message using the secret in hand and check if the generated signature is equal to the received one. If so, the received request is valid, if not, the launch cannot be performed.
   (c) If the request is valid, send back the section HTML file. If the section is gradable, send the LTI callback URL along with the section HTML file to be used later in grade posting when the student completes the section correctly.

5.4 User Interfaces

There are different use cases and different actors for each use case when it comes to user interfaces. When students use OpenDSA books as an integral part of their course and perform exercises to get credit, their interface is the LMS course. Instructors, on the other hand, need to first configure the book, then generate the book in their target LMS course. Book configuration and course generation processes are performed using OpenDSA-LTI interfaces, written as part of the Rails Web application.

System administrators have a wider permission. They are responsible for setting up and managing the system. Since they are a small group of users and they do not usually use the system actively, their interfaces are often discarded. It is commonly assumed that admins are technically capable and can perform their jobs using direct database access or command line tools.

We believe that the more manual work the admins have to perform, the more problems will occur. Thus we decided to carefully design the interfaces that provide admins with all the functionalities they may need to perform their day-to-day tasks.

In OpenDSA-LTI we used a general-purpose user interface library called Bootstrap. Bootstrap is modern, simple, and responsive. Responsiveness is the feature that makes the interface layout fluid and supports multiple devices, since it can scale to different screen sizes and resolutions.

In the following sections, we present interfaces for students, instructors, and admins.

5.4.1 Student Interfaces

When students log in to their Canvas course, they are provided with standard menu items that allow them to navigate to different parts of the course. The Canvas LMS as shown in Figure 5.4 provides multiple views that serve different purposes.

When a student navigates to the "Modules"[20] menu item, a list of course modules will be shown. We designed OpenDSA book contents in Canvas to present a book chapter in each Canvas module. Under each chapter, a list of DSA topics is arranged as labels with the proper sequence number for easy tracking.

OpenDSA researchers are usually trying to know where in a module page that a user is viewing at a particular time. It is possible to track scrolling and window focus events, but generally, this requires using a special library. This makes it particularly difficult to tell what prose content in a given page is viewed and what is ignored. To better track where within the page the user is actually looking, we decided to divide the topic pages into multiple

[20]Canvas uses the term "Module" to represent a unit of a course, so to avoid confusion, we will use "topic" to refer to an OpenDSA module throughout this chapter.

sections. Multiple sections also satisfy Canvas requirements to have each gradable activity on a separate page.

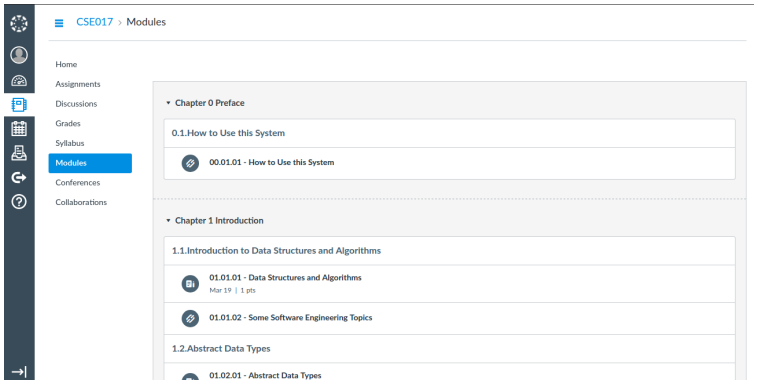

Figure 5.4: An OpenDSA Book compiled into a Canvas LMS course.

When a section is gradable, it is shown as an "Assignment", and a link to that assignment is placed in the correct position in the list of DSA topic sections. Gradable sections can also be accessed from the Canvas Assignments page, and they post a score back to the "Gradebook" when students complete them.

Figure 5.3 presented the LTI launch sequence of events to load a resource from OpenDSA-LTI. Likewise, launching a gradable section ("Assignment" in Canvas terminology) is no different except that the launch message contains the callback LIS_OUTCOMES_URL which is a Canvas endpoint that accepts student scores.

Figure 5.5 shows the LTI sequence of events taking place when a student completes a gradable section correctly.

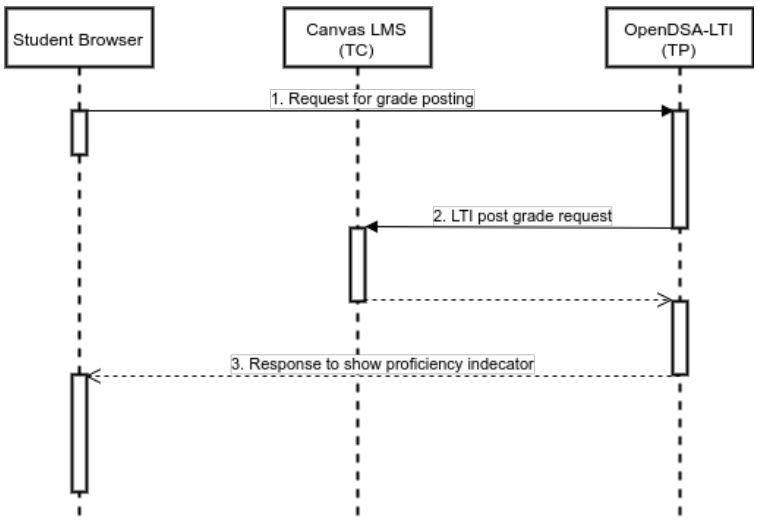

Figure 5.5: Sequence diagram to post a grade to Canvas LMS (*ReplaceResult* function)

1. When the student completes an exercise, the client-side framework performs these actions:
   (a) Fire a score event and prepare a score object. Add an LTI launch message (previously received from Step 4(c) in Figure 5.3) to the score object.
   (b) Send the score object to an OpenDSA-LTI assessment endpoint to give the student the exercise points.
2. OpenDSA-LTI authenticates the request as described in Figure 5.5 Steps 4(a) and (b). If the request is valid, OpenDSA-LTI sends a POST request to Canvas callback LIS_OUTCOMES_URL with the following parameters:
   (a) LIS_SOURCE_ID: the Gradebook field Id that holds the student's score
   (b) Student score as 1.0: which gives the student the full credit for the assignment
3. OpenDSA-LTI receives the Canvas Gradebook response and sends it back to the client-side framework, which shows the proficiency indicator to the student.

5.4.2 Instructor Interfaces

There are two main scenarios for how instructors would use OpenDSA in their LMS course. First, compile a complete book within an LMS course. Second, if the instructor does not want to include an entire book, instead, they want only to select and add interactive materials. In the following subsections, we discuss these scenarios in detail.

Integrating an OpenDSA Book to the Canvas LMS We created a simple 3-step process that allows the instructor to create, configure, and generate the book in his Canvas course.

First, the instructor opens "New Course Offering" page as shown in Figure 5.6. The page contains a form that collects the basic information to create a new course offering. Instructors who are using the system for the first time need to define their organization and course names. They can do that by following the link to the right of the organization and course dropdown element respectively. Then the instructor chooses one of the OpenDSA standard books to include in his course, or he can upload his custom configuration file by following the link beside the "Book Instance" drop down element. Finally, the instructor provides his Canvas access token for OpenDSA-LTI to be able to access the course and generate the book on his behalf. The access token is verified before the instructor submits the form. If it is valid, a green check mark is shown, otherwise, a red X mark is shown and the form will not submit until a valid one is entered.

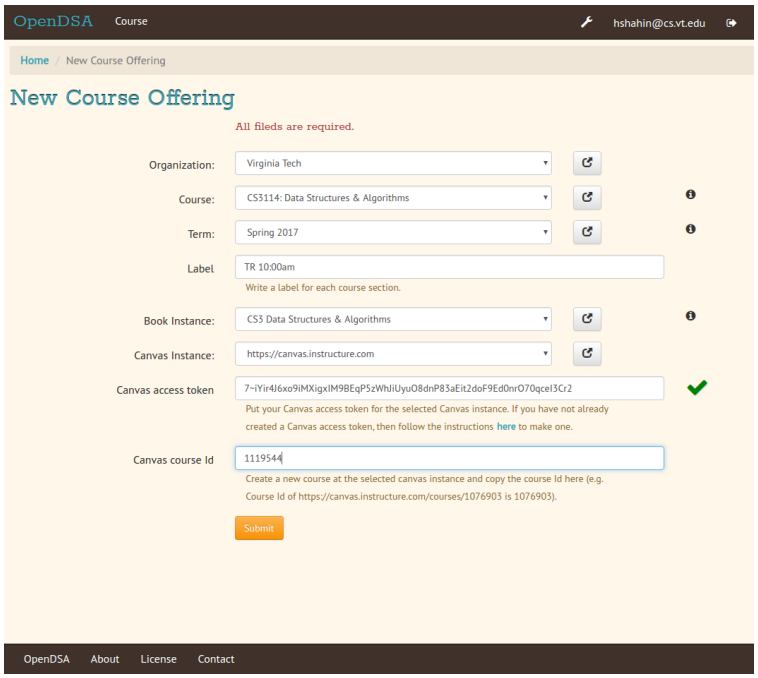

Figure 5.6: "New Course Offering" form is a page through which the instructor can select his organization and course, and create a new course offering in the selected semester.

Once submitted, the instructor will be directed to the second step. Figure 5.7 shows the book configuration tool. This tool allows the instructor to customize his book; he can manage topics, change the topics order, and change the exercises' number of points.

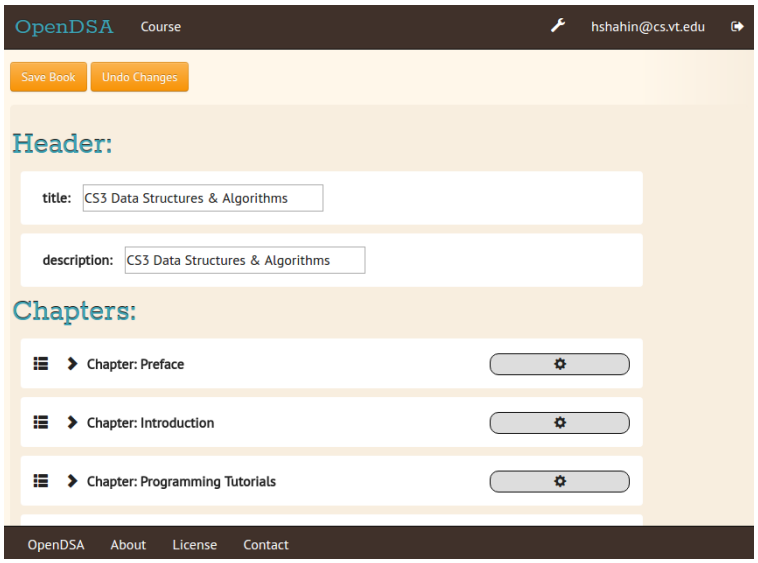

Figure 5.7: The book customization tool allows the instructor to design his book before generating it into a Canvas course.

The last step is to generate the book; the instructor clicks the "Generate Course" button and the system uses the customized book configuration and the access token to call Canvas APIs to generate the book. The final step takes a few minutes to complete because of the many API calls issued to Canvas. Since the last step is a long-running task, we implemented it to run in the background to avoid any performance degradation. We also added a progress bar element to visualize the progression of this operation.

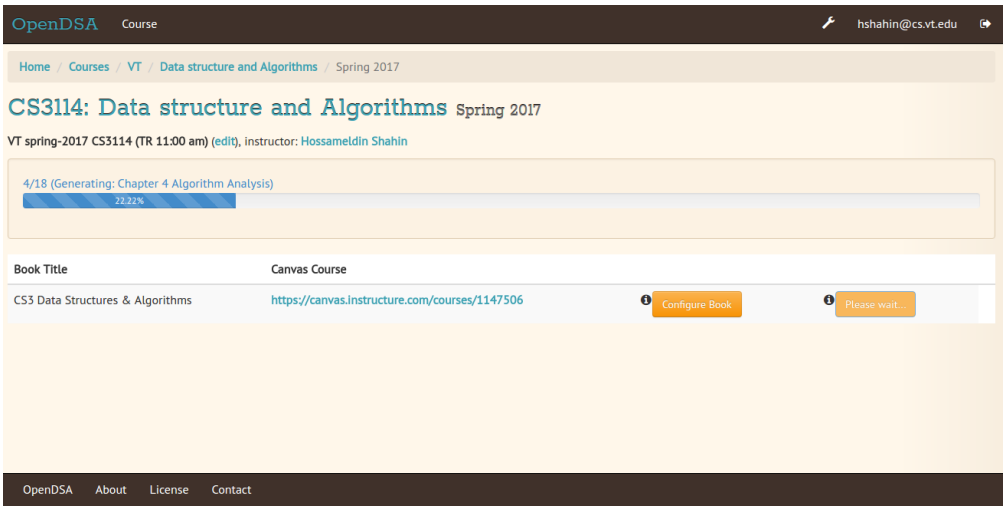

Figure 5.8: Canvas course generation is a background process that uses the Canvas APIs to create an OpenDSA book in a course.

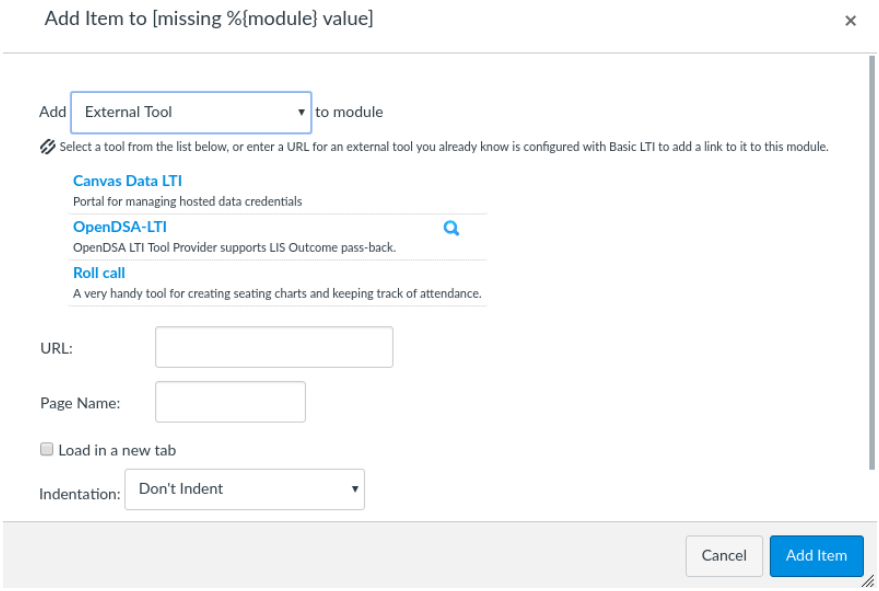

Figure 5.9: Canvas resource selection extension to standard LTI protocol.

OpenDSA Resource Selection A common scenario is when the instructor wants to add an individual visualization or exercise to his course modules. Canvas has implemented an extension[21] to the standard LTI protocol to allow selecting and customizing the learning tool resources. To satisfy this scenario, we developed a process that enables the instructor to manually define an OpenDSA-LTI app in his course. Then, when inserting content into a module, if the instructor picks "External Tools" he will see the configured app "OpenDSA-LTI" with a "find" icon as shown in Figure 5.9.

Clicking the app will bring up a new dialog where the instructor can pick one of OpenDSA's interactive materials to be inserted as a page or resource within the current module. Figure 5.10 shows a list of all interactive materials in the Quicksort topic. When an instructor clicks on any element in the list, the tool will then redirect the instructor to the LTI success URL with some additional parameters. Canvas will use these parameters to create a custom URL that will be used as the launch link.

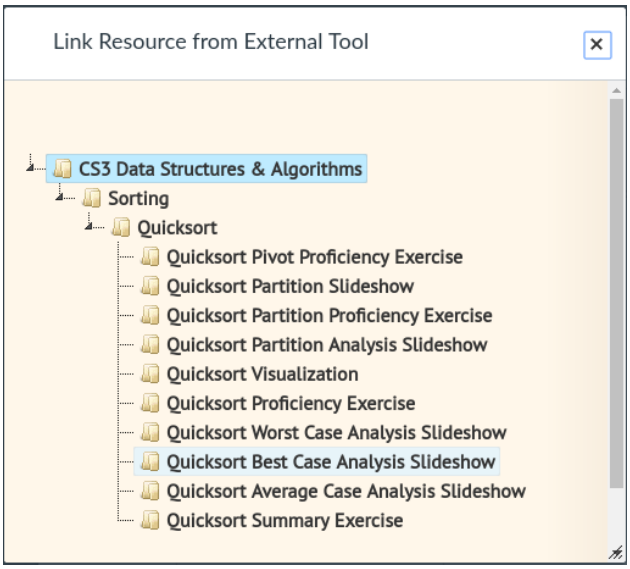

Figure 5.10: OpenDSA list of interactive materials from which instructor can choose to add to a module.

5.4.3 Admin Interfaces

In OpenDSA-LTI the system administrators grant regular users an instructor role, upload book configuration files to the system, and configure other learning tools used by OpenDSA-LTI like CodeWorkout. Admin activities are usually related to adding or modifying data to the database lookup tables like organizations and courses. In addition to modifying data, Admins archive the completed course offerings so they will not appear unnecessarily in

[21] https://canvas.instructure.com/doc/api/file.link_selection_tools.html

instructors' interfaces. Although most of the admin activities are related to database changes, direct access and modifying the database could introduce data integrity problems. The OpenDSA-LTI Database is created and managed by Rails Active Records, which contains a validation layer on top of Ruby objects. Rails Active Records also implement "callbacks", which are hooks into the life cycle of an Active Record object. These hooks automatically update values in other database tables and execute actions after an alteration to the object state. For example, archiving a course offering would archive the book associated with it. If the course offering archive status changed manually, the associated book would remain active.

We realized how important it is to have powerful administrator's interface in OpenDSA-LTI. However, every Web developer knows that creating an administration interface for their projects is an incredibly tedious task. Thus we chose ActiveAdmin, which is a framework for building administration style interfaces. With little effort, we created a highly customizable interface which saved development time and allowed us to focus on other mission-critical functionality. The initial version of the admin interface was inspired by CodeWorkout [29]. Then we added more functionalities that admins would mostly need to keep OpenDSA-LTI running flawlessly. Figure 5.11 shows the OpenDSA-LTI ActiveAdmin interface.

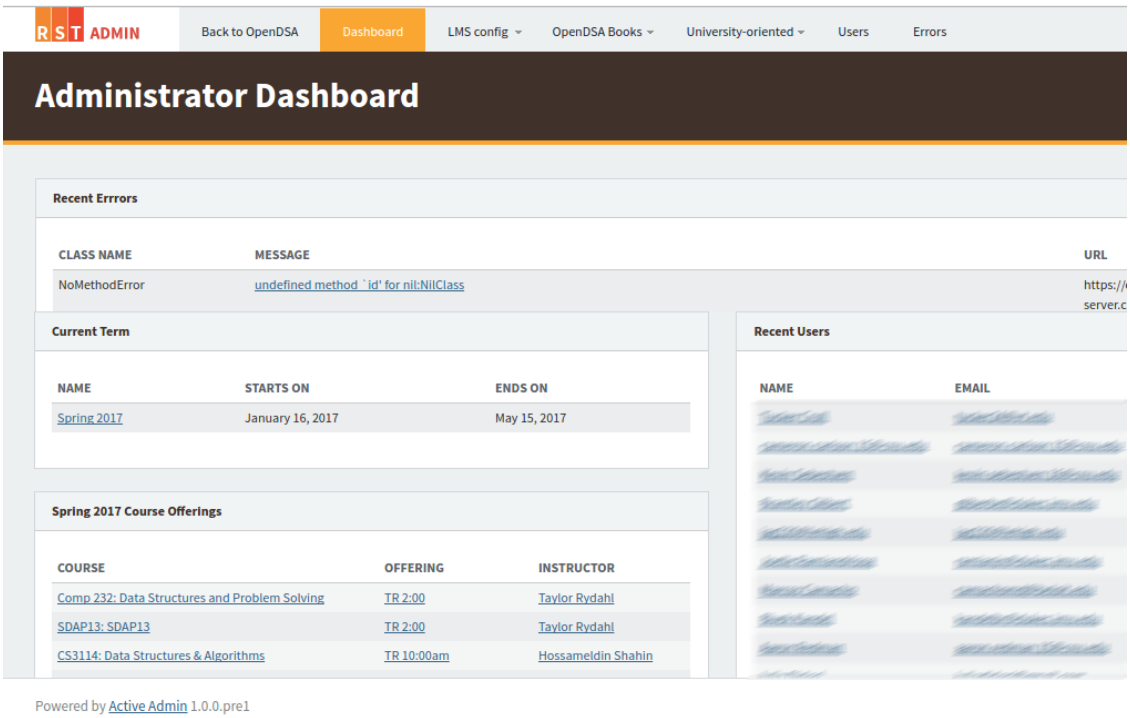

Figure 5.11: The ActiveAdmin dashboard provides a summary of the system. It shows the most recent errors, the recent logged-in users, and the courses offered during the current semester. The admin interface menus contain links to multiple pages that manage database lookup tables.

5.5 Security

eLearning systems security is important, especially if developed as a Web application accessible from the Internet. One of OpenDSA-LTI's design goal is to be open and easy to use. OpenDSA-LTI contains students' personal information like name, email address, and class performance information. These are considered as sensitive information that must be kept protected under the FERPA (Family Educational Rights and Privacy Act) which makes application security especially important. Security solutions such as SSL encryption and firewalls are insufficient in protecting the applications from malicious users and many threats.

The security metric timeline of a Web application is divided into three phases: design, deployment, and runtime [30]. In OpenDSA-LTI, we decided to identify the security flaws in the early design phase to minimize or completely remove any potential threats.

The security metric framework was based on top ten lists described by the Open Web Application Security Project (OWASP)[22]. The security threats were ordered based on how critical they are regardless of how common they are. The list includes:

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Cross-Site Request Forgery (CSRF)
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

5.5.1 Injection

> "Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization." [36]

[22] http://www.owasp.org

The most famous attacks on Web applications are SQL injection. These attacks aim at influencing database queries by manipulating Web application parameters. A popular goal of SQL injection attacks is to bypass authorization or read unauthorized data. OpenDSA-LTI was prepared for injections by sanitizing query parameters before making database queries. Sanitizing is also done even on data read from the database. Also, no "hand written" database queries are done in OpenDSA-LTI. Instead, queries are handled by the Rails Active Records.

5.5.2 Broken Authentication and Session Management

> "Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities." [36]

OpenDSA-LTI uses Devise for identity management. Devise is a flexible authentication solution for Rails. We are using cookies for the session management implementation. Sessions are generated on the server during the first request, and then given by the client on every subsequent request. A session could be stolen if the attacker can steal the cookie. The cookies could be stolen by someone listening to the HTTP traffic, or by reading them from the user's browser. We prevent tampering with session IDs by using SSL encrypted connections between OpenDSA-LTI and the users.

5.5.3 Cross-Site Scripting (XSS)

> "XSS flaws occur whenever an application takes untrusted data and sends it to a Web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface Web sites, or redirect the user to malicious sites." [36]

The Rails framework provides the `html_safe` method, which guarantees to escape all the data that is rendered on the page while sending the response back to the client. Escaping means that data inserted in the rendered page is modified so that user's browser does not interpret it as HTML tags, rather as textual content.

5.5.4 Insecure Direct Object References

> "A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data." [36]

As discussed in Section 5.2.2, OpenDSA-LTI implements a flexible user's role model that allows every user to have a different role at the level of a course. The permission system is based on the user role in a course whether it is an Instructor or a student. Students are not authorized to do any action using OpenDSA-LTI interfaces. Instead, students are only allowed to interact with OpenDSA materials through the LMS. Instructors, on the other hand, have a richer set of actions to perform when they are authenticated. While they can view all the courses offered by OpenDSA- LTI in any term, they can only manage their courses, configure OpenDSA book instances, and generate courses in a Canvas installation.

5.5.5 Security Misconfiguration

> "Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, Web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date." [36]

When we started the OpenDSA-LTI implementation, we decided to use Rails' latest version (4.2). Since release there were no security patches announced. Version 4.2 is still currently supported. On June 2016, Rails version 5.0 was released. It did not contain any security patches. Rather, it had major changes like creating APIs apps. Since OpenDSA-LTI does not avail any service through API yet, we decided not to immediately upgrade to that version. However, it is likely that similar security-related issues in third-party gems used by OpenDSA- LTI will be revealed later, and thus different parts of the system should be regularly updated with the latest security patches. In the deployment phase, we secured OpenDSA-LTI keys. Also, our database server is not accessible from the public Internet. Finally, our Passenger application server is not running in debug mode 3, which would expose a stack trace if an unhandled exception would be raised.

5.5.6 Cross-Site Request Forgery (CSRF)

> "A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable Web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim." [36]

The best practices and the guidelines for protecting a Web application from CSRF is to a) only allow POST requests to change data, b) create a unique token called the `authenticity_token` for each session, c) include the token in all HTML forms and d) verify that POST requests

contain the correct token for the session. Rails provides `authenticity_token` handling by default through the form tools. When the user views a form to create, update, or destroy a resource the Rails app creates a random `authenticity_token`, stores this token in the session, and places it in a hidden field in the form. When the user submits the form, Rails looks for the `authenticity_token`, compares it to the one stored in the session, and if they match, the request is allowed to continue.

5.5.7 Insecure Cryptographic Storage

> "Many Web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes." [36]

We use Devise for user identity management. Devise encrypts the user's password and saves it to the database. OpenDSA-LTI uses the user's email address as an identifier. There are two ways a user can register in the system. The first is by directly going to `opendsa.org` and signing up. In this case, a user is asked to enter his email address and a password. This step is typically done by instructors; then he asks the system administrator to grant instructor privileges. Students are not required to register in the system by themselves, rather they will be automatically registered when they open any of the OpenDSA modules in Canvas for the first time. The automatic registration process does not save a password for the student, in fact, the password field is left blank. If a student wants to login directly to `opendsa.org`, he will not be allowed until he provides a new password, which in turn will be encrypted and saved in the database.

5.5.8 Failure to Restrict URL Access

> "Many Web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway." [36]

Securing a Web application should not depend on hiding content, rather it should be implemented through a rigorous permission model. OpenDSA-LTI supports two categories of user roles: the Global Role and the Course Role. In their Global Role, a user can be an administrator, an instructor, or a regular user. At the course level, users can enroll as an instructor or a student. Each user has one Global Role. While this permission model provides flexibility, it was implemented to avoid any potential security issues. We use CanCanCan, a powerful authorization library for Ruby on Rails. CanCanCan works through a centralized

ability file in which we define for each user role what actions "can" or "cannot" be performed. All controllers in OpenDSA-LTI are configured to check the ability file before executing any of its actions. The checking step is applied by calling the `load_and_authorize` method on the controller. The ability file is a set of "Can" and "Cannot" statements which are further conditionally executed. For example, Administrators are given the ability to manage all courses for all instructors. While an instructor can manage courses, however, he is restricted to manage only his courses. Instructors can only view other instructors' courses without being able to make any changes. We are using ActiveAdmin, a Ruby on Rails plug-in for generating administration style interfaces. ActiveAdmin URLs are rather easy to guess, which would make them vulnerable if security checks were insufficient. Access to the admin area of the system is also managed by the CanCanCan ability file.

5.5.9 Insufficient Transport Layer Protection

> "Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly." [36]

To secure all the communication between users and the system, we have issued an SSL certificate for OpenDSA-LTI. Furthermore, all non-secured HTTP connections are redirected to the corresponding HTTPS URLs. When users open any OpenDSA-LTI views through Canvas, a signed LTI launch request is generated and sent to OpenDSA-LTI. The purpose of each LTI request is to load an OpenDSA section page in a Canvas iframe. Before the first LTI request is sent, the system generates a `consumer_key` and `shared_secret` for each user. As discussed in Section 5.3, the `key` and `secret` are communicated and used by both parties to identify who they are talking to. The `secret` is used to digitally sign every request going in either direction using OAuth. OAuth protocol itself is well designed, and requests are signed with timestamps and identifiers for preventing the repetition of requests. Generating `key` and `secret` for each user allows us to revoke them in case a user misused them or they got stolen.

5.5.10 Unvalidated Redirects and Forwards

> "Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages." [36]

We usually redirect successful POST requests to a "show" view which displays the modified resource to the user. Redirecting POST requests is important to avoid repeating requests in

case the user refreshes the page or moves back and forth in the browser history. All redirect URLs are generated by the OpenDSA-LTI server without using any user input in building those URLs.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The general theme of this thesis is to improve OpenDSA by achieving a number of goals. The first goal is to make OpenDSA a more robust system by mitigating the known issues and limitations. The second is to increase project adoption by integrating OpenDSA contents with the widest range of Learning Management Systems (LMS). We wanted to make it easier for students and instructors to use OpenDSA contents. We base our analysis on the assumption that keeping students in one central learning environment and reducing the use of parallel systems would improve learning and increase project adoption. Finally, integration with a delivery platform would transform OpenDSA from a monolithic system to Software as a Service (SaaS). That would offload from project developers and collaborators the need to implement instructor support tools that are already provided by every LMS.

We started by presenting the components of the original infrastructure: content types, compilation process, client-side framework, and data collection server. We discussed their technical specifications and requirements, and how they work together in one system. We discussed the problems reported by users and collaborators, and how we addressed these issues and solved them in each component. Our modifications range from fixing a problem to a complete overhaul in some situations.

Our review of interoperability and learning tool shows that there are different existing standards and solutions, all of which have their advantages and disadvantages. For our particular requirements and constraints, the chosen standard should support the majority if not all the LMSs. It does not require many changes to OpenDSA, and it can be used to add more learning tools to OpenDSA. We excluded the restrictive and tightly coupled standards, like the ones that required the use of a particular technology or which are used with a limited number of learning management systems.

For our needs and requirements, the Learning Tool Interoperability (LTI) standard was the most prominent. It did not depend on a specific technology and allowed us to integrate OpenDSA without re-writing it. We designed and implemented a system called OpenDSA-LTI, which is a wrapper that communicates with any LTI-enabled LMS. We were also able to simplify the OpenDSA client-side framework, so it does not have to deal with user authentication, session handling, or students' proficiency caching. Instead, it was simplified to focus on interaction and to score data communication.

We created interfaces for OpenDSA-LTI administrators and instructors. Instructors are the primary users of the system; they can define their organization, create courses, and create course offerings in a particular semester. Instructors use a three-step intuitive interface to create the course, customize the OpenDSA book, and generate the book contents into the LMS course. Instructors are not restricted to create an entire book. Instead, they can select OpenDSA individual interactive materials and make them part of their existing course.

We designed the new infrastructure with extendability in mind. The design allows for extending the existing adapter, which generates the book in an LMS, to accommodate new LTI-compatible LMSs. OpenDSA-LTI allows system admins to add new LTI-compliant learning tools to OpenDSA books. For example, we have replaced the old programming evaluation engine with CodeWorkout, a drill-and-practice programming system. Since CodeWorkout communicates with an LMS through LTI, our system could easily augment OpenDSA books with CodeWorkout exercises.

Since its launch in Fall 2016, we received an increasing number of requests from instructors to use OpenDSA-LTI. As of June 2017, OpenDSA-LTI has hosted 36 active courses offered by 25 different universities in 6 countries. 41 instructors have used OpenDSA-LTI to host their courses on the Canvas LMS, and the system has 2,729 registered students.

6.2 Future Work

OpenDSA is a fast-growing project, which presents many opportunities for future enhancements. Here we propose a potential direction to improve OpenDSA based on our experiences with LTI and the IMS Caliper [25] initiative.

LTI has changed the landscape of learning tools. Many vendors are now producing LTI-compatible tools. The space of learning tools has grown much larger. The variety of LTI tools allows instructors to design all their courses' activities to happen inside the LMS course. However, that mandates the activities to be scattered across multiple tools, each of which has its interaction capture mechanism and data format. Collecting a large amount of data is essential for understanding and improving student performance. It is especially important that the data is accurately collected and probably stored for predicting students' future outcomes.

Technologies have been developed to analyze larger data than ever. Machine learning methods and tools are designed to use data from different sources to learn and predict future student behavior.

To achieve such ultimate goals, there is a need to collect all of the students' interaction data in a standardized way across different learning tools, store the data in a centralized place, and then make it available for data analysis and visualization tools.

We can take OpenDSA-LTI as an example; the system is collecting a large amount of data. The system also allows for integrating CoreWorkout exercises in book instances. However, the data collected by CoreWorkout is not stored in the OpenDSA system, rather it is stored in its proprietary data store. To analyze students' performance, we should combine data from the two systems and the LMS, then store it in yet another ad-hoc system where we can roll up all the data for various kinds of analysis.

Caliper is designed to overcome this disconnectedness; it is a standard that enables the students' interaction data with learning tools and LMSs to be collected, stored, and transported. It is flexible enough to accommodate a wide variety of current and future interaction types. Since Caliper is an open standard, it will lower the cost of learning analytics. So, we are enthusiastic about OpenDSA adoption of Caliper, along with the LTI open IMS standard that we support. As OpenDSA-LTI has proven to be successful since launched in Fall 2016, we believe the prominent future path would be focusing on and encouraging other teams in the field to make more complete, and better, use of the students' interaction data through Caliper.

Appendices

Appendix A

OpenDSA XBlocks Architecture

A.1 Introduction

XBlock is an integration architecture for building courseware. An XBlock is similar in structure to a Web application plug-in and fills a similar niche. The OpenDSAX XBlocks attempt to host OpenDSA eBooks into the OpenEdX XBlock architecture.

OpenDSAX provides the following functionality:

1. Display of JSAV-based proficiency exercises, Algorithm Visualizations, and slideshows.
2. A user interface to allow course authors to select from a list of available exercises.
3. Course authoring tool to define exercise weight as well as other exercise configurations.
4. Student score tracking by the JSAV and Module XBlocks.
5. Student interaction data tracking and event logging.

A.2 Main Components

The OpenDSAX XBlock architecture consists of three XBlocks: Module, JSAV, and Content. The Module XBlock is the parent XBlock, and it serves as a container to the other two XBlocks. It provides JavaScript and CSS resources to its children. It is also considered a centralized place in which JSAV score and logging events are checked and validated. A Module XBlock receives score and logging events from children XBlocks through messaging.

The messaging facility is built on top of HTML5's cross-document messaging via Window.postMessage capabilities, which let the main page communicate with a page loaded

within an iframe and vice versa. This technology is working on all current versions of all major browsers [11].

Figure A.1 shows the main components for each OpenDSAX XBlock.

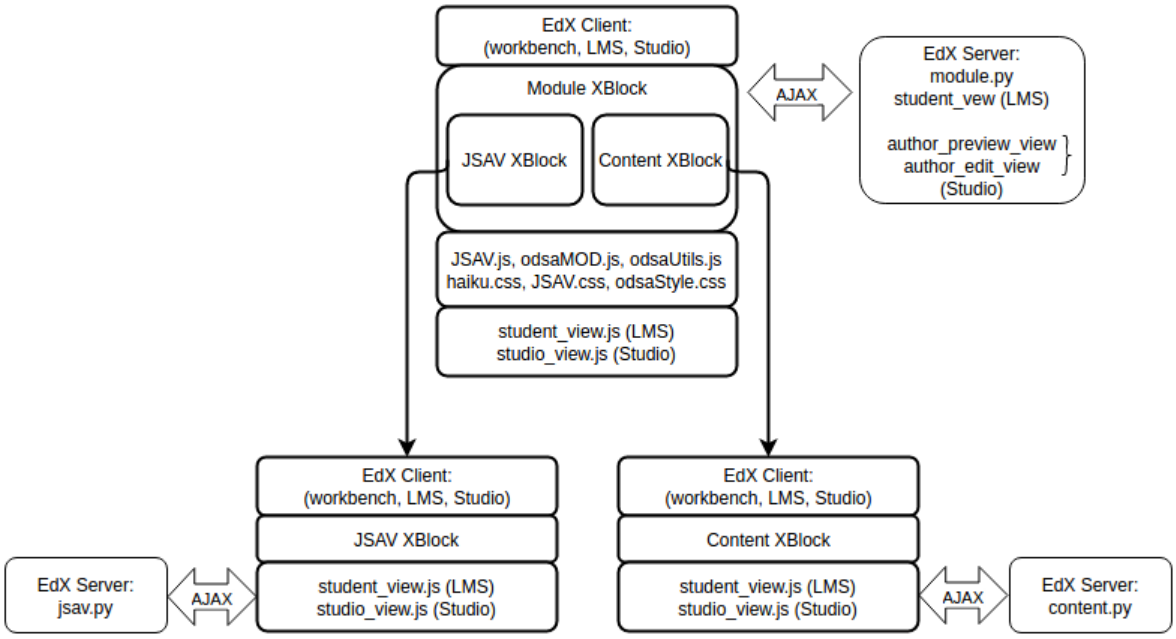

Figure A.1: OpenDSAX XBlocks main components.

A.3 How It Works

When a Module XBlock is loaded into EdX, it will contain one or more JSAV XBlocks. Each client-side JavaScript of the JSAV XBlock will trigger events as the student starts to interact with it. Depending on the JSAV XBlock instance type, events might be "message" events (which are triggered by iframed proficiency exercises and AVs) or "jsav-log-event" events (which are triggered by slideshows). Module XBlock client-side JavaScript will listen for those events and handle them. Events that have scoring data will be filtered and validated before the Module XBlock calls the `reportProgress` function of the corresponding JSAV XBlock to report the score back to JSAV server-side XBlock via an AJAX request. The server-side evaluation could have more rules to apply to scoring data than the client side. Only the server-side evaluation decides whether the student will be awarded proficiency and given the problem points. Server-side decisions will be sent back via AJAX response to the JSAV XBlock on the client-side. Accordingly JSAV and Module XBlocks on the client-side will update proficiency and score indicators based on server response.

Figure A.2 shows a sequence diagram that presents interactions happening between Module and JSAV XBlocks, and between client and server sides as well.

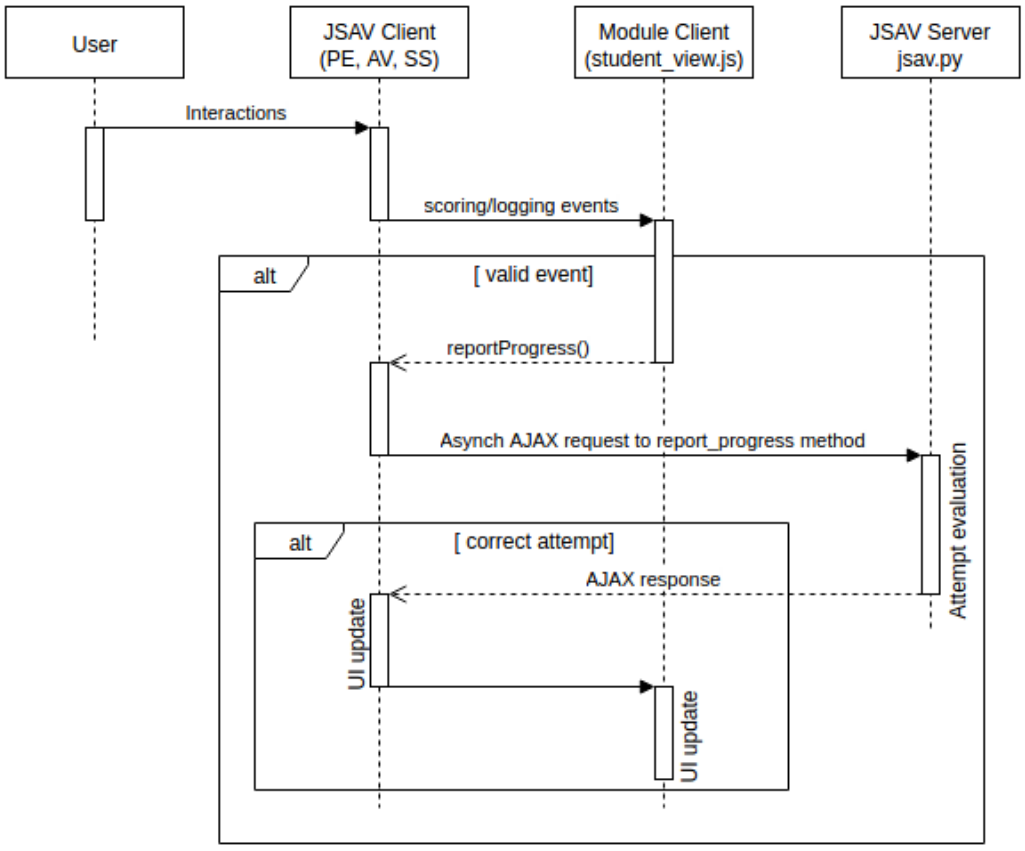

Figure A.2: OpenDSAX XBlocks sequence diagram.

Bibliography

[1] A. Korhonen, L. Malmi, P. Silvasti, J. Nikander, P. Tenhunen, P. Mard, H. Salonen, and V. Karavirta. TRAKLA2, http://www.cs.hut.fi/Research/TRAKLA2, last accessed 05-01-2017.

[2] R. S. Baker, A. T. Corbett, and K. R. Koedinger. Detecting student misuse of intelligent tutoring systems. In *International Conference on Intelligent Tutoring Systems*, pages 531–540. Springer, 2004.

[3] R. S. Baker, A. T. Corbett, K. R. Koedinger, and A. Z. Wagner. Off-task behavior in the cognitive tutor classroom: when students game the system. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 383–390. ACM, 2004.

[4] O. Bohl, J. Scheuhase, R. Sengler, and U. Winand. The sharable content object reference model (SCORM) - a critical review, http://ieeexplore.ieee.org/abstract/document/1186122. In *Computers in education, 2002. proceedings. international conference on*, pages 950–951. IEEE, 2002.

[5] S. Booth, S. Peacock, and S. P. Vickers. Plug and play learning application integration using IMS learning tools interoperability. In *Ascilite*, pages 143–147, 2011.

[6] S. Boyer, B. U. Gelman, B. Schreck, and K. Veeramachaneni. Data science foundry for MOOCs, http://ieeexplore.ieee.org/abstract/document/7344825. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

[7] G. Brandl. Sphinx: Python documentation generator. *http://sphinx.pocoo.org/index.html*, 2009.

[8] D. A. Breakiron. Evaluating the Integration of Online, Interactive Tutorials into a Data Structures and Algorithms Course, http://hdl.handle.net/10919/23107. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2013.

[9] D. A. Breakiron, E. Fouh, S. Hamouda, and C. A. Shaffer. Analysis of interaction logs for online tutorials. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 709–709. ACM, 2014.

[10] J. Burke. RequireJS, http://requirejs.org, last accessed 03-04-2017.

[11] Cross-document messaging. http://caniuse.com/#feat=x-doc-messaging, last accessed 03-24-2017.

[12] D. Dagger, A. O'Connor, S. Lawless, E. Walsh, and V. P. Wade. Service-oriented e-learning platforms: From monolithic systems to flexible services. *IEEE Internet Computing*, 11(3), 2007.

[13] M. F. Farghally, K. H. Koh, H. Shahin, and C. A. Shaffer. Evaluating the effectiveness of algorithm analysis visualizations. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 201–206. ACM, 2017.

[14] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[15] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[16] E. Fouh. *Building and Evaluating a Learning Environment for Algorithm and Data Structures Courses, http://hdl.handle.net/10919/51951*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2015.

[17] E. Fouh, D. Breakiron, M. Elshehaly, T. S. Hall, V. Karavirta, and C. A. Shaffer. OpenDSA: using an active eTextbook to teach data structures and algorithms. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 734–734. ACM, 2013.

[18] E. Fouh, D. A. Breakiron, S. Hamouda, M. F. Farghally, and C. A. Shaffer. Exploring students learning behavior with an interactive etextbook in computer science courses. *Computers in Human Behavior*, 41:478–485, 2014.

[19] E. Fouh, M. F. Farghally, S. Hamouda, K. H. Koh, and C. A. Shaffer. Investigating difficult topics in a data structures course using item response theory and logged data analysis. In *Proceedings of the 9th international conference on educational data mining*, pages 370–375, 2016.

[20] D. Goodger. reStructuredText Markup Specification. *http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html*, last accessed: 06-15-2017.

[21] S. Hamouda. *Enhancing Learning of Recursion, http://hdl.handle.net/10919/64249*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2015.

[22] IMS Global Learning Consortium. Learning Tools Interoperability Guidelines Version 1.0, https://www.imsglobal.org/ti/tiv1p0/imsti_guidev1p0.html, 2006.

[23] IMS Global Learning Consortium. IMS Global Learning Tools Interoperability Implementation Guide, https://www.imsglobal.org/specs/ltiv1p1/implementation-guide, 2010.

[24] IMS Global Learning Consortium. Profile, IMS Common Cartridge Version 1.0 Final Specification. *http://www.imsglobal.org/cc/ccv1p0/imscc_profilev1p0.html*, 2010.

[25] IMS Global Learning Consortium. Caliper analytics. *http://www.imsglobal.org/activity/caliper*, 2016.

[26] V. Karavirta, P. Ihantola, and T. Koskinen. Service-oriented approach to improve interoperability of e-learning systems, http://ieeexplore.ieee.org/abstract/document/6601947. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 341–345. IEEE, 2013.

[27] V. Karavirta and C. A. Shaffer. JSAV: the JavaScript algorithm visualization library. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 159–164. ACM, 2013.

[28] V. Karavirta and C. A. Shaffer. Creating engaging online learning material with the JSAV JavaScript algorithm visualization library. *IEEE Transactions on Learning Technologies*, 9(2):171–183, 2016.

[29] K. P. Murali. CodeWorkout: Designing an Online Drill-and-Practice System for Introductory Programming. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2016.

[30] E. A. Nichols and G. Peterson. A metrics framework to drive application security improvement. *IEEE Security & Privacy*, 5(2), 2007.

[31] J. Nikander, A. Korhonen, O. Seppälä, V. Karavirta, P. Silvasti, L. Malmi, et al. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education-An International Journal*, (Vol 3_2):267–288, 2004.

[32] C. A. Shaffer, V. Karavirta, A. Korhonen, and T. L. Naps. OpenDSA: beginning a community active-ebook project. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, pages 112–117. ACM, 2011.

[33] C. A. Shaffer, T. L. Naps, and E. Fouh. Truly interactive textbooks for computer science education. In *Proceedings of the Sixth Program Visualization Workshop*, pages 97–103, 2011.

[34] C. A. Shaffer, T. L. Naps, and S. H. Rodger. Using OpenDSA eTextbooks in Your Class. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 713–713. ACM, 2016.

[35] WebStorage. https://www.w3schools.com/html/html5_webstorage.asp, last accessed: 02-15-2017.

[36] D. Wichers. A list of the 10 Most Critical Web Application Security Risks, https://www.owasp.org/index.php/Top_10_2013-Top_10. *OWASP Foundation*, last accessed 03-04-2017.

[37] Wikipedia. Software as a service, https://en.wikipedia.org/w/index.php?title=Software_as_a_service&oldid=781150424, last accessed: 06-15-2017.

[38] S. Wilson, P. Sharples, and D. Griffiths. Distributing education services to personal and institutional systems using Widgets. In *Proc. Mash-Up Personal Learning Environments-1st Workshop MUPPLE*, volume 8, pages 25–33, 2008.