Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nearly all higher education institutions today use a *Learning Management System* (LMS) [20], with more than three-quarters of all students reporting that their institution's LMS is used for most or all of their courses [27]. In the past, learning tools usually operated independently of the LMS, or perhaps relied on integration approaches that were not standardized across LMS [19]. The lack of communication between learning tools and LMS caused difficulties for instructors, students, and the developers of learning tools. Instructors who wanted to use third-party content in their course had to manage a separate gradebook for each tool they used, requiring the manual aggregation of grades from different tools. Students had to maintain a separate account for each of the different tools used in their courses. Learning tool developers often had to develop and maintain course administration infrastructure that duplicated functionality that an LMS provides, or had to deal with the non-standardized integration approaches supported by LMS.

In 2010 the IMS Global Learning Consortium released the *Learning Tools Interoperability* (LTI) standard[1], which defines a standardized way for learning tools to be securely connected with platforms such as LMS. Since LTI was released, it has seen considerable adoption by both LMS and learning tools. Prominent LMS that support LTI include Moodle, Canvas, Blackboard Learn, Brightspace, and Sakai [13]. Major educational content and tool providers that have adopted LTI include McCgraw-Hill, Cengage, Pearson, and others [13].

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

1.1 Motivation

The OpenDSA eTextbook system [26, 47] is one example of a learning tool provider that supports LTI [48]. OpenDSA uses LTI to allow for its textbooks, auto-graded exercises, and interactive visualizations to be integrated with LMS. OpenDSA also relies on LTI to integrate content from other learning tool providers into its textbooks.

However, OpenDSA's integration with LMS and tool providers had/has some issues:

- OpenDSA modules, which were designed to be viewed on a single web page, were split up among multiple pages in the LMS
- OpenDSA had no communication with the third-party tools that it included content from, instead relying on the LTI functionality of the LMS.
- OpenDSA only supported integration with the Canvas LMS
- OpenDSA does not have access to interaction data captured by the third-party tools that OpenDSA uses content from, limiting the ability of OpenDSA researchers to understand how OpenDSA textbooks are being used.

OpenDSA also had some issues that were unrelated to interoperability:

- The OpenDSA book configuration format was verbose and required book configuration files to be frequently updated.
- Customization of OpenDSA books was not friendly to non-expert users since it required that users be familiar with OpenDSA's book configuration format.

1.2 Objectives

The primary aim of this thesis is to further explore the problem of connecting educational systems in an effort to improve interoperability. We attempt to achieve this aim by designing, implementing, and discussing interoperability solutions in the context of the OpenDSA system that address the issues listed Section 1.1.

A secondary aim of this thesis is to increase the impact of the OpenDSA project by making its content more widely accessible and reusable. We attempt to achieve this aim by developing improvements to the OpenDSA book configuration system to address the issues listed in Section 1.1, and by enabling OpenDSA content to be consumed in the form of stand-alone exercises and stand-alone modules. We also compare OpenDSA's book configuration system to the configuration systems of two other eTextbook platforms: Runestone Interactive [2] and zyBooks [3].

1.3 Contributions

The contributions of this thesis are as follows:

- A description and discussion of steps taken to improve OpenDSA interoperability
  - Implemented LTI tool consumer functionality, improving integration with external tools used in OpenDSA
  - Serving individual OpenDSA exercises and visualizations through LTI (improving upon a prototype implementation)

[2] https://runestone.academy
[3] https://www.zybooks.com/

  - Support for embedding stand-alone OpenDSA exercises and visualizations in arbitrary HTML pages
  - Serving stand-alone OpenDSA modules through LTI
  - Expanded the number of LMS OpenDSA has support for

- Identification of some limitations and issues related to the LTI standard and its implementation by LMS
- A discussion of the problem of sharing learning analytics data among educational systems, and a proposed architecture
- Improvements to OpenDSA book infrastructure
  - A more concise book configuration format
  - A book configuration GUI

1.4 Structure of Thesis

Chapter 2 covers some background information, primarily describing the OpenDSA system as it existed prior to this work. Chapter 3 covers some improvements made to the OpenDSA book configuration system. Chapter 4 describes and discusses steps taken to improve the interoperability of the OpenDSA system, along with some remaining issues and limitations. Chapter 5 describes and discusses the problem of sharing analytics data among educational systems, and outlines an architecture that could be used for this purpose. Finally, Chapter 6 presents conclusions and future work.

Chapter 2

Background

2.1 Learning Tools Interoperability

Learning Tools Interoperability (LTI) is a standard published by the IMS Global Learning Consortium that defines a way to securely connect learning applications and tools with platforms like learning management systems (LMS) [5]. LTI 1.0 was released in 2010 [14], and the LTI standard has since been adopted by most major LMS [13]. Under LTI, learning applications and tools that provide content or services are referred to as *tool providers*, while LMS and other platforms that consume content or services from tool providers are referred to as *tool consumers*.

A tool consumer requests a particular resource from a tool provider by sending an LTI launch request, which is simply an HTTP `POST` request containing parameters specified in the LTI standard. Information in an LTI launch request includes the context of the launch (e.g. course information), information on the user that triggered the request, information on the resource being requested, and more. If the resource being requested is graded, then the launch request will include information the tool provider can use to send a score back to the tool consumer. Some concrete benefits of LTI include eliminating the need for instructors to manage separate gradebooks for each tool they use in their course (provided those tools support LTI), and eliminating the need for students to register separate accounts for each tool used in their course. Through the use of iframes, LTI also enables students and instructors to access external tools without leaving the LMS.

In order to authenticate requests between a tool provider and tool consumer, LTI 1.1 prescribes the use of OAuth 1, though this has changed to OAuth 2 in LTI 1.3 [16]. This requires that the tool provider and tool consumer use a common key and secret to sign their requests.

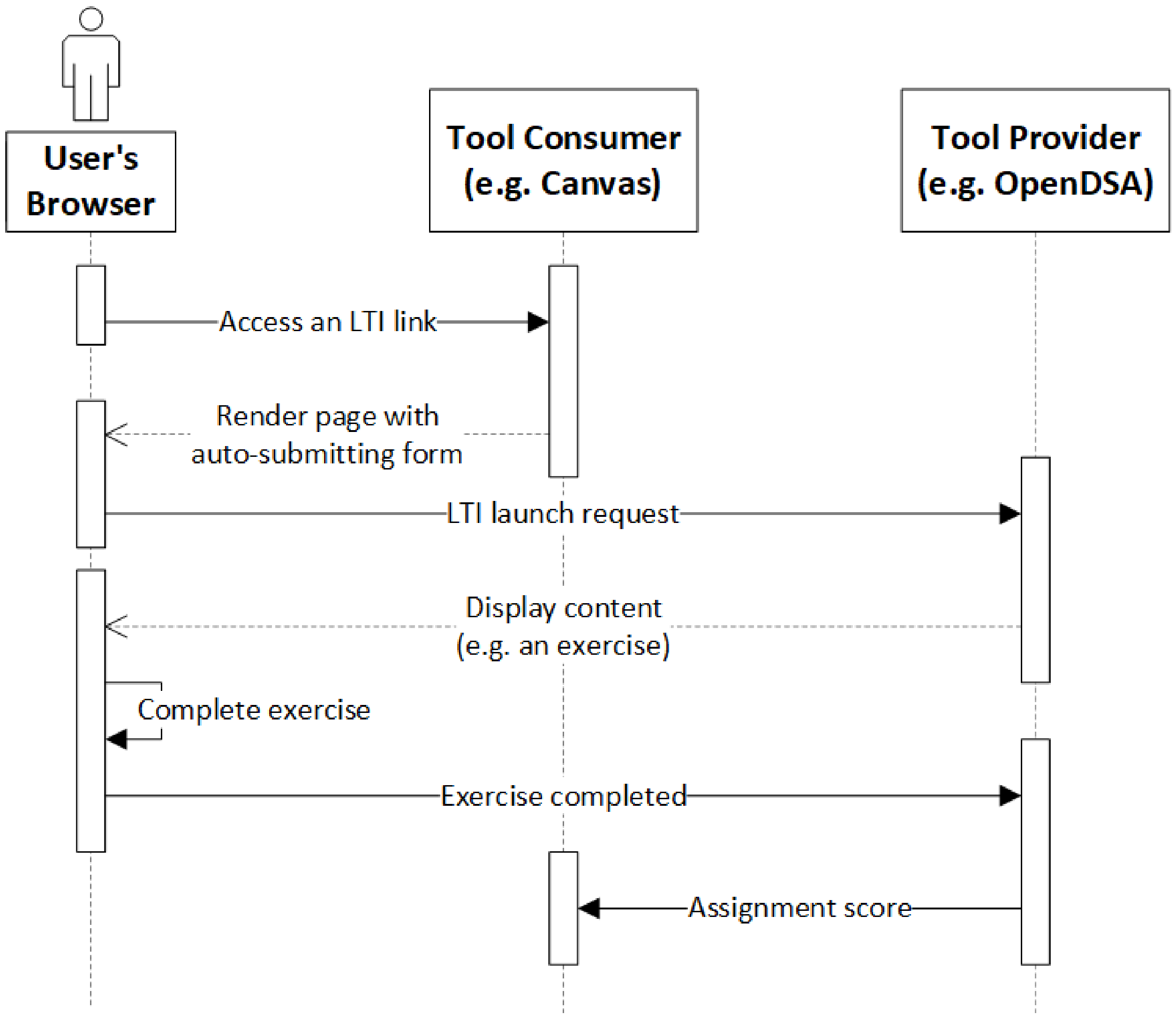

User's Browser
Tool Consumer (e.g. Canvas)
Tool Provider (e.g. OpenDSA)
Access an LTI link
Render page with auto-submitting form
LTI launch request
Display content (e.g. an exercise)
Complete exercise
Exercise completed
Assignment score

Figure 2.1: A rough outline of an LTI flow for a graded assignment

Figure 2.1 roughly illustrates a simple LTI workflow for a graded assignment. The basic steps are as follows:

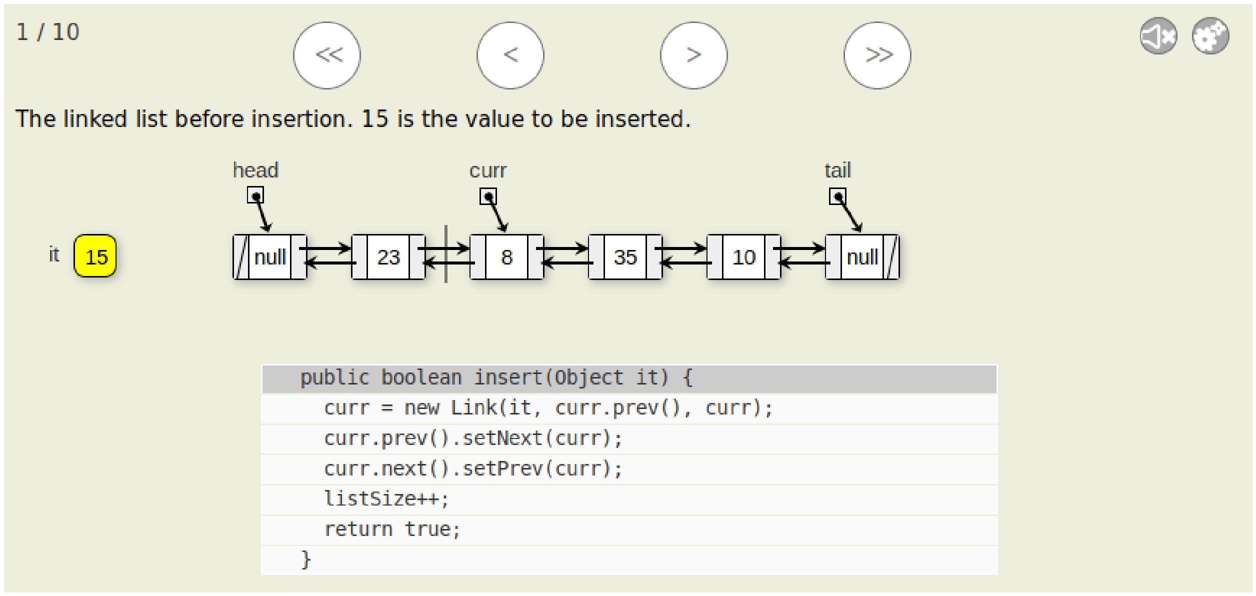
1. In an LMS, a user accesses an assignment linked to a tool provider
2. The LMS sends an LTI launch request to the tool provider
3. The tool provider receives the request and displays the assignment
4. The user completes the assignment
5. The tool provider sends the user's assignment score to the LMS by sending an HTTP `POST` request to the outcome URL from the LTI launch request

The user's assignment score would then appear in the gradebook in the LMS.

2.2 OpenDSA

OpenDSA[1] is an open source[2] eTextbook project providing infrastructure and materials to support Computer Science courses on a variety of topics, including Data Structures and Algorithms (DSA), Formal Languages, Finite Automata, and Programming Languages [26, 47]. OpenDSA content combines textbook-quality prose with interactive algorithm visualizations and auto-graded exercises, allowing students to practice as much as needed. OpenDSA's algorithm visualizations are created using the JavaScript Algorithm Visualization (JSAV) library [33]. OpenDSA textbooks are also modular, allowing them to easily be customized to fit the needs of various courses.

[1]https://opendsa-server.cs.vt.edu
[2]https://github.com/OpenDSA



The linked list before insertion. 15 is the value to be inserted.

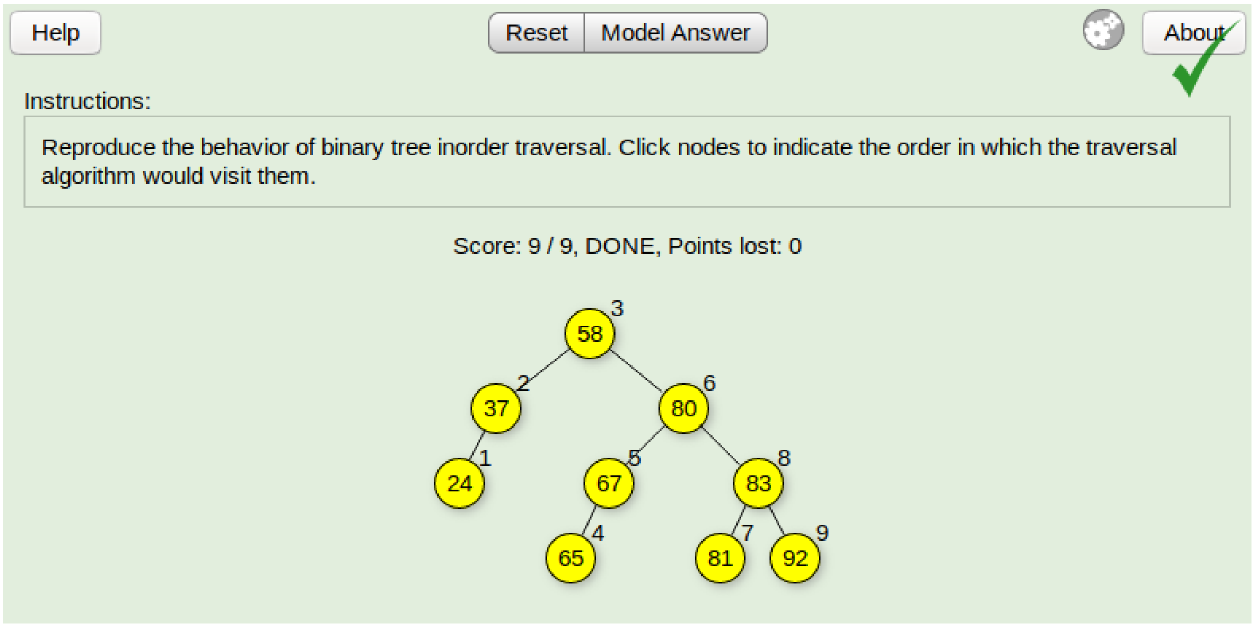
```
public boolean insert(Object it) {
  curr = new Link(it, curr.prev(), curr);
  curr.prev().setNext(curr);
  curr.next().setPrev(curr);
  listSize++;
  return true;
}
```

Figure 2.2: A JSAV-based OpenDSA slideshow on inserting into a doubly linked list

2.2.1 OpenDSA and LTI

OpenDSA was originally a stand-alone monolithic system requiring instructors to manage a separate gradebook, and requiring OpenDSA developers to maintain course administration infrastructure. In order to move OpenDSA away from this model, prior work turned OpenDSA into an LTI tool provider [48], enabling the integration of OpenDSA content with the Canvas LMS and potentially any LMS that supports LTI. This allows instructors and OpenDSA developers to take advantage of the support an LMS provides, and removes the need to maintain a separate OpenDSA gradebook since LTI allows scores to be sent to the gradebook in the LMS. This also benefits students, who no longer have to deal with managing a separate OpenDSA account.

To facilitate the integration of OpenDSA eTextbooks with the Canvas LMS, OpenDSA uses the Canvas API to populate a Canvas course with modules and assignments that correspond to sections or modules in an OpenDSA eTextbook. Prototype functionality was also implemented to enable instructors to add individual OpenDSA exercises and visualizations



Reproduce the behavior of binary tree inorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

Score: 9 / 9, DONE, Points lost: 0

Figure 2.3: A JSAV-based OpenDSA exercise requiring the user to emulate the behavior of a binary tree inorder traversal

to their Canvas course without needing to configure and compile a textbook.

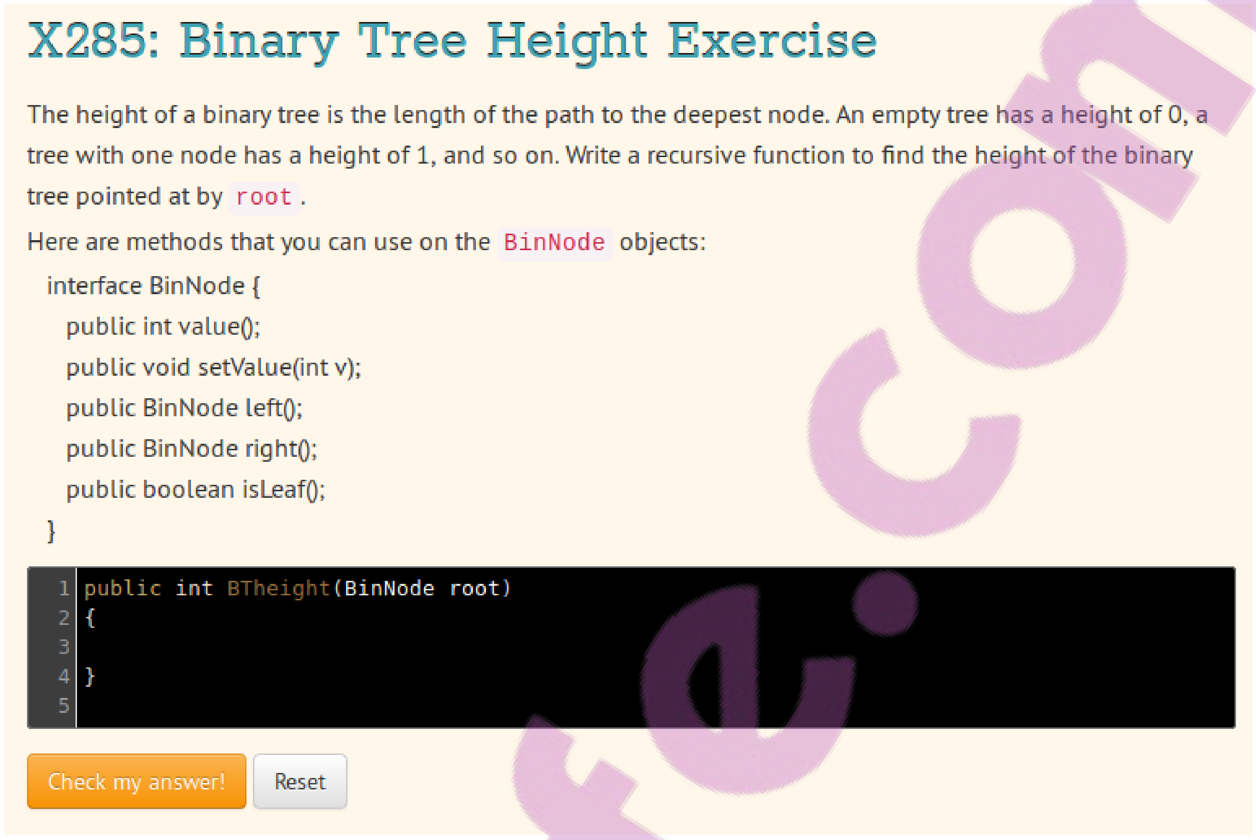
CodeWorkout

CodeWorkout[3] is a free, open-source solution for practicing small programming problems [22]. Like OpenDSA, CodeWorkout is an LTI tool provider which enables CodeWorkout exercises to be integrated with LTI tool consumers. OpenDSA uses CodeWorkout to provide short programming exercises for certain topics.

2.2.2 Books

The primary way that OpenDSA content is consumed is in the context of an eTextbook. Each OpenDSA book is divided into one or more *chapters*. Each chapter includes one or more

[3]https://codeworkout.cs.vt.edu/


X285: Binary Tree Height Exercise

The height of a binary tree is the length of the path to the deepest node. An empty tree has a height of 0, a tree with one node has a height of 1, and so on. Write a recursive function to find the height of the binary tree pointed at by `root`.

Here are methods that you can use on the `BinNode` objects:

```
interface BinNode {
  public int value();
  public void setValue(int v);
  public BinNode left();
  public BinNode right();
  public boolean isLeaf();
}
```

```
public int BTheight(BinNode root)
{

}
```

Figure 2.4: A CodeWorkout exercise

modules, with each module covering a single topic, such as Binary Search or Linked Lists. An OpenDSA module is further divided into one or more *module sections*. Each of these module sections can contain prose, images, diagrams, auto-graded exercises, and interactive visualizations. OpenDSA books are customizable, such that an instructor may choose which modules they want in their book, how to organize those modules into chapters with names of their choosing, and how the modules in each chapter are ordered. This customization allows an instructor to create a book to suit the needs of their specific course. The settings for exercises and visualizations in each module can also be customized.

There are two types of OpenDSA books:

1. LTI books

2. Stand-alone HTML books

LTI Books

In the first iteration of the OpenDSA system, books were only offered in a stand-alone HTML format [26]. Later, the OpenDSA server infrastructure was rewritten to be centered around LTI [48]. This current OpenDSA server is developed with Ruby on Rails [4]. The version of the LTI standard that OpenDSA currently supports is version 1.1 [5].

When an instructor wants to use an OpenDSA book in their course, they fill out a course creation form on the OpenDSA website where they specify the details of their course (e.g. the name of the course, semester, etc.), select the book instance they want to use, select their LMS instance, and provide the ID of their course in the LMS. The instructor also provides an API key for the LMS instance, which allows OpenDSA to make requests to the LMS API on behalf of the instructor. Next, OpenDSA uses the API of the LMS to automatically create links and assignments in the instructor's LMS course corresponding to the specific book instance the instructor has selected. After creating the links and assignments in the LMS, the book instance is compiled to generate the corresponding HTML files. In order to allow for support of course generation for multiple LMS, OpenDSA uses an adapter pattern, such that a separate adapter can be created for each LMS OpenDSA wants to support [48]. Currently, OpenDSA has a course generation adapter for only the Canvas LMS.

The original OpenDSA LTI system created an assignment and/or link for each section of each module in an OpenDSA book, meaning each module in a book would be split up over multiple pages in the LMS. This was mainly driven by limitations in the Canvas LMS requiring only a single grade for each assignment. Issues with this approach, and changes made to resolve those issues, are discussed in Section 4.1.

[4] https://rubyonrails.org/

Stand-alone HTML Books

OpenDSA still offers stand-alone HTML versions of books, without any LTI functionality. However, these versions of books are now intended to be used only as supplemental reading material. These books are not tied to a specific course offering, users are not required to be logged in to access these books, and exercise scores are not recorded.

Book Configuration

The content and layout of an OpenDSA book is controlled by a JSON configuration file. This configuration file controls the global book settings, the chapters in the book, the modules to be included in each chapter, and the settings for every section, exercise, and visualization included in every module. In Chapter 3 we discuss issues with the prior OpenDSA book configuration infrastructure, and present improvements made to resolve these issues.

Book Authoring and Compilation

OpenDSA uses *reStructuredText (RST)*[5] as its authoring format. RST is a plain-text markup syntax that is commonly used in the Python programming language community for technical documentation. In the OpenDSA system, each RST file contains a single OpenDSA module, which is designed to be displayed on a single web page [26]. *Sphinx*[6] compiles RST files into other formats, including HTML and LaTeX. OpenDSA uses Sphinx to compile RST files into HTML files.

Sphinx also allows for the creation of custom *directives*, which are defined by a Python script. When Sphinx comes across an instance of one these custom directives during the

[5]http://docutils.sourceforge.net/rst.html
[6]http://www.sphinx-doc.org/

compilation process, the script defining the directive is run. The script is passed any arguments that are provided by the directive instance, and the script generates the final output (e.g. an HTML snippet) based on those arguments.

For example, OpenDSA defines a Sphinx directive for each of the exercise and visualization types that it offers:

- **avembed**: This directive is used for exercises and visualizations that have their own separate HTML page, and are displayed within an iframe in the module page.
- **inlineav**: This directive is used for exercises and visualizations that are included directly in the module page, rather than being displayed in an iframe.
- **extrtoolembed**: This directive is used for exercises and visualizations that are provided by external tools (e.g. CodeWorkout). These exercises and visualizations are displayed in an iframe in the module page.

Listing 2.1: An example avembed directive instance found in an RST file

```
.. avembed:: Exercises/Binary/DefSumm.html ka
   :long_name: Tree Definition Summary Exercises
```

Besides compiling RST files using Sphinx, the OpenDSA book compilation process also includes pre and post-processing steps. The pre-processing step is run before Sphinx and performs several actions, including validating the format of the book configuration file, verifying that the RST files listed in the configuration file actually exist, adding a header to each RST file, and more. The post-processing step is run after Sphinx has generated the module HTML files. The post-processing script makes corrections to chapter and section numbers, and makes any other necessary corrections.

2.2.3 Stand-alone LTI Exercises and Visualizations

Previous work developed a prototype feature that allowed for individual OpenDSA exercises and visualizations to be served through LTI [48]. This prototype feature worked by creating specially formatted module RST files that contained only exercises and visualizations, and nothing else (i.e. no prose). Each exercise and visualization was placed in its own section in the RST file. A book configuration file was then created that included all of these specially formatted module files. The book configuration was then uploaded to the OpenDSA website, where it was compiled and marked as being an exercise-only book. Since at the time each OpenDSA module intended to be served over LTI was split up during compilation so that each section was on a separate HTML page, this resulted in a separate HTML page being created for each exercise and visualization. This was an acceptable approach for a prototype since it allowed for existing book infrastructure to be used with only minor modifications. However, the following limitations made it unsuitable as a permanent solution:

- The exercise-only book had to be recompiled whenever the collection of available exercises and visualizations needed to be changed.
- The data model did not make sense semantically since the exercises and visualizations were linked to a special book, but that "book" wasn't actually being used as a book.
- The prototype did not allow for accurate interaction data to be captured since a given stand-alone exercise or visualization could be used in multiple course offerings, but the the exercises and visualizations in the special exercise-only book could only be tied to a single course offering (i.e. whichever course offering the book was tied to).
- The settings of exercises and visualizations could not be customized by different instructors.

The prototype feature also included support for an extension to the LTI standard supported only by the Canvas LMS to allow instructors to add individual OpenDSA exercises and visualizations to their Canvas course through a graphical interface. We discuss our implementation that improves upon this prototype in Section 4.2.2.

2.2.4 Client-side Framework

The OpenDSA client-side JavaScript framework [4, 35] has several responsibilities. When an OpenDSA page first loads, the client-side framework initializes the exercises and visualizations that are included in that page. The client-side framework is also responsible for communicating with the OpenDSA server to retrieve a user's progress on exercises, and to record exercise attempts. The client-side framework also records rich click-stream data, capturing information when a user loads the page, presses a button, and other events.

Chapter 3

Improvements to the OpenDSA Book Configuration System

This chapter covers improvements made to the OpenDSA book configuration system. First, we describe some issues with the prior OpenDSA book configuration system. Next, we present steps taken to simplify the format used to store OpenDSA book configurations in files, so as to make manually creating and maintaining OpenDSA book configurations easier. We then present a graphical user interface we developed that makes it easier for non-expert users to create and customize their own book configurations without having to manually modify JSON configuration files. We also include a comparison of OpenDSA's book configuration system to the book configuration systems of two other eTextbook platforms.

3.1 Prior Issues

Prior to this thesis, OpenDSA book configuration files were always hand-crafted, and had to be manually kept in sync with the corresponding module RST files. So, for example, if the name of a section changed, any configuration files containing the module with that section had to be updated. If a new exercise was added to a module, it had to be added to every configuration file containing that module. In addition, settings for each and every exercise and visualization had be specified in the configuration file. Many exercises and visualizations have the same or similar settings, so this resulted in verbose configuration files with a large amount of duplication. We will refer to this book configuration format as the “full” configuration format. A sample of the full configuration format is show in Listing 3.1.

Listing 3.1: A sample of the full JSON book configuration format

```
    "chapters": {
        "Preface": {
          "Intro": {
            "long_name": "How to Use this System",
            "sections": {}
          }
        },
        "Introduction": {
          "Background/IntroDSA": {
            "long_name": "Data Structures and Algorithms",
            "sections": {
              "Data Structures and Algorithms": {
                "IntroSumm": {
                  "long_name": "Introduction Summary Questions",
                  "required": true,
                  "points": 1.0,
                  "threshold": 5
                }
              },
              "Some Software Engineering Topics": {
                "showsection": false
              }
            }
          }
        }
```

There was also another configuration format that had been developed that required only listing the modules to be included in the book. This format is referred to as the “simple” configuration format. A sample of this configuration format is shown in Listing 3.2. A Python script parsed through the RST files of the modules in the simple configuration file and generated a full configuration file by adding all of the sections and exercises to the configuration. One downside of this approach was that it did not allow for different settings to be specified for each exercise or visualization, but rather it used the same settings for every exercise and visualization of a given type. However, the settings for specific exercises and visualizations could be manually changed after generating the full configuration file.

Listing 3.2: A sample of the simple configuration format

```
  "chapters": {
      "Preface": [
        "Intro"
      ],
      "Introduction": [
        "Background/IntroDSA",
        "Design/ADT"
      ],
```

The process of configuring OpenDSA books was also not friendly to newcomers since it requires that the creator of the configuration to be familiar with the JSON configuration format that OpenDSA uses, and be aware of the different options and their meaning. This is a significant barrier to entry, and as a result it was common for the maintainers of OpenDSA to create configuration files on behalf of instructors who wanted to use OpenDSA.

3.2 A New Book Configuration Format

In an attempt to resolve some of the issues with the original book configuration formats, we developed a new configuration format. The main feature of this new configuration format is that it allows for default settings to be specified for each exercise and visualization type, but also allows for those default settings to be overridden for specific exercises and visualizations. A sample of the new configuration format is shown in Listing 3.3. This new format is significantly less verbose than the full configuration format since settings do not have to be specified unless they deviate from the defaults. In addition, it is not required to list each individual section, exercise, and visualization in the configuration unless the settings are being overridden for a given section, exercise, or visualization. To give an idea of the conciseness of the new format, one of the configuration files in the full format is 3,236 lines and 98,977 characters long, while the same configuration in the new format is only 623 lines and 14,220 characters long.

Listing 3.3: A sample of the new configuration format

```
    "glob_ka_options": {
      "threshold": 5,
      "points": 1.0,
      "required": true
    },
    "chapters": {
      "Preface": {
         "Intro": {}
      },
      "Introduction": {
         "Background/IntroDSA":{
            "Some Software Engineering Topics":{
               "showsection":false
            }
         },
         "Design/ADT":{
            "IntroADTSumm":{
               "threshold":4
            }
         }
      }
```

This new format also somewhat mitigates problems associated with changes in RST files. For example, when renaming a module section in an RST file, the full configuration format requires that any configuration file containing the module with that section be updated. The new configuration format only requires configuration files that override the default settings for that section be updated. The same applies for exercises and visualizations.

This new format was implemented by developing a Python script that we named `simple2full.py` that is based on a script created by a previous OpenDSA developer [48]. The script is run prior to the main OpenDSA compilation script and expands the configuration file into the full configuration format. The script works by parsing the RST files of modules listed in a configuration to determine what sections, exercises, and visualizations are in each module. The script uses the default settings for each section, exercise, or visualization when generating the full configuration unless the default settings have been overridden for a given section, exercise, or visualization. The script is also automatically run on configuration files before saving the configuration to the OpenDSA database since the database needs to store the complete settings for each individual section, exercise, and visualization.

3.3 Book Configuration GUI

In order to make it easier for non-expert users to create and customize their own book configurations, we developed a graphical web interface. This interface allows users to load one of the reference configurations that are created and maintained by OpenDSA developers. The user can then modify the reference configuration to suit their needs, and save it to the OpenDSA database so that it is available for them to choose when they create their course. Users can also load a previous configuration that they saved to the OpenDSA database, upload a configuration file that they have saved locally, or build their own configuration from scratch. The user interface for loading a configuration is shown in Figure 3.1.

The interface also allows users to modify book meta-data and global settings such as title, description, and the programming language(s) used for code examples. In addition, the user can set the default settings for each exercise and visualization type. A portion of the user interface for customizing meta-data and global settings is shown in Figure 3.2.

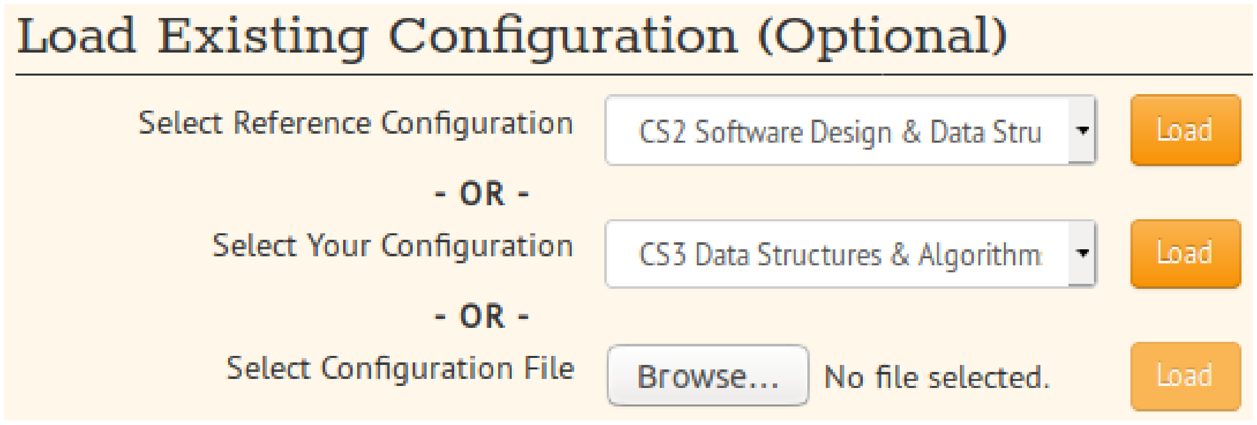

Load Existing Configuration (Optional)
Select Reference Configuration
CS2 Software Design & Data Stru
Load
- OR -
Select Your Configuration
CS3 Data Structures & Algorithm
Load
- OR -
Select Configuration File
Browse...
No file selected.
Load

Figure 3.1: The user interface for loading a book configuration

The user can customize the content of their book using a two-pane drag-and-drop interface shown in Figure 3.3. Each pane of the selection interface displays a tree view using the jsTree library[1]. The left pane of the interface displays the chapters and modules that are included in the book, while the pane on the right displays modules that are available to be included in the book. Users can add a chapter to the book by clicking a button located just above the left-hand module pane, which will display a modal window requiring the user to enter a name for the chapter.

Users can add a module to the book by using their mouse to click and drag a module from the right-hand pane to the left-hand pane. It is also possible to select multiple modules in the right-hand pane using the ctrl or shift key and then drag them all to the left-hand pane at once. Chapters and modules can be removed by right clicking on the tree node and selecting the context menu entry labeled “Remove”. Expanding a module node displays the sections contained in that module, while expanding a section node displays the exercises and visualizations in that section. The name of each exercise and visualization is preceded by an icon indicating what type of exercise or visualization it is. Right clicking on an exercise or visualization node shows a context menu with entries labeled “Edit Exercise Settings”

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

and “Reset Settings”. When a user clicks on “Edit Exercise Settings”, a modal window is displayed allowing the user to customize the settings for the exercise or visualization. If an exercise or visualization has settings that are different from the default settings, its node in the tree view is labeled with an “M” to indicate that it has been modified.

Once a user is satisfied with their book configuration, they can click “Save Configuration” button at the bottom of the page to save the configuration to the OpenDSA database. The user can then select the configuration when they go to create a course. Alternatively, the user can click the “Download Configuration” button to download a JSON file containing the configuration. If the user has loaded a pre-existing configuration, then there will also be a button labeled “Update Configuration” which will save the changes to the existing configuration.

3.4 Issues Addressed

The new book configuration format addresses the issue of the original “full” configuration format being verbose, since it only requires that the settings for sections and exercises be explicitly listed if those settings differ from the default settings. Since the new configuration format only requires that sections and exercises be listed in the configuration file if the settings are different from the default settings, this also somewhat mitigates the issue of configuration files having to be kept in sync with the module RST files, although it does not completely eliminate it. For example, if a section in a module RST file is changed, only configuration files that are overriding the default settings for that section have to be updated with the new configuration format, whereas the old configuration format requires updating every configuration file containing the module with that section. The new format also addresses the issue of the “simple” configuration format not allowing for exercise and section settings to be customized.

The book configuration GUI mitigates the issue of configuring OpenDSA books not being friendly to newcomers, while still allowing for book customization on par with crafting a book configuration file manually. The GUI makes it so that users do not have be familiar with the JSON book configuration format that OpenDSA uses in order to create or customize their own book configuration. Users also do not have to be familiar with the possible settings available for different book options since the options and possible settings are listed and enforced by the GUI.

3.5 A Comparison With Other eTextbook Platforms

Two other interactive eTextbook platforms that allow for some level of book configuration are Runestone Interactive [2] and zyBooks [3]. In this section we compare the book configuration systems of these two eTextbook platforms with OpenDSA's system.

3.5.1 Runestone Interactive

Runestone Interactive [42] is an open-source platform for interactive Computer Science eTextbooks that offers a library of ready-made books. Like OpenDSA, Runestone uses reStructured Text (RST) as its authoring format, with Sphinx being used to compile the RST files into HTML. Runestone textbooks are each stored in a separate Git repository that contains all of the RST files that will be included in the book, with all of the RST files stored in a directory named `_source`. These RST files are organized into folders within the `_source` directory, with each folder containing the RST files for a single chapter. In order to

[2] https://runestone.academy
[3] https://www.zybooks.com/

control what RST files are included in a book, Runestone uses the Sphinx `toctree` directive, which directs Sphinx to generate a table of contents based on the RST files listed under the directive. Each chapter's folder contains an index RST file with a `toctree` directive that lists the relative path to each of the RST files that are to be included in the chapter in the order that they are to be included. All of the chapter folders are contained in a parent folder, along with a main book index RST file with a `toctree` directive that lists the relative file path for each of the chapter index files in the order that the chapters are to be included in the book. Runestone does not currently offer a GUI for customizing books.

Thus, modifying a Runestone book would require modifying potentially multiple index RST files, depending on what customizations were desired. In order to re-order the chapters in a book, one needs to modify the order that the chapter index files are listed in the book's main index file. In order to rename a chapter, one has to modify the chapter's RST file. To re-order, remove, or add content to a given chapter, one must modify the index RST file of that chapter. However, having a separate index file for each chapter is not a strict requirement of Sphinx; it is also possible to have a single index file that directly lists the RST files for all of the chapters.

Comparison To OpenDSA

While OpenDSA allows for the content included in a book to be customized using a single configuration file, customizing a Runestone book would require modifying potentially multiple files. Like Runestone, an OpenDSA book also has an index RST file that lists all of the RST files that are included in the book using the Sphinx `toctree` directive, but this index RST file is automatically generated by the OpenDSA book compilation script based on the OpenDSA book configuration file, rather than being manually created. OpenDSA also does not require manually collecting the RST files that are to be included in a book. Instead, OpenDSA stores all of the RST files available under the RST folder of the main OpenDSA repository [4]. When compiling a book, the OpenDSA compilation script automatically copies the RST files that are listed in the book configuration file to a separate directory created for the book being compiled.

One downside of Runestone's approach compared to OpenDSA is that if an RST file was to be included in multiple Runestone books, it would require having a copy of the RST file in multiple folders/repositories. If a change was made to an RST file that was included in multiple books, one would have to ensure every copy was updated. An example of two copies of an RST file that are unsynchronized at the time of this writing can be seen here [5] [6]. This is not an issue for OpenDSA since OpenDSA only has a single source copy of each RST file.

3.5.2 zyBooks

zyBooks [51] is a proprietary eTextbook platform that offers a catalog of pre-configured eTextbooks for STEM courses. Since zyBooks is closed-source, it is unclear how a zyBook configuration is stored. However, zyBooks does provide a GUI to allow instructors to customize the content in one of the pre-configured textbooks to some extent [7]. The GUI allows instructors re-organize the chapters and sections in a book using a drag-and-drop interface, add new chapters, rename chapters, mark sections as being optional content, and mark sections as being unused content in order to hide them from students.

[4] https://github.com/OpenDSA/OpenDSA/tree/master/RST

[5] https://github.com/RunestoneInteractive/fopp/blob/b0cb5be74793bcaeab3c26ede250abfad2ddf78e/_sources/Debugging/BeginningtipsforDebugging.rst

[6] https://github.com/RunestoneInteractive/thinkcspy/blob/dd2a30fc8864ad222934d95f59c4317c276c0de6/_sources/Debugging/BeginningtipsforDebugging.rst

[7] https://zybooks.zendesk.com/hc/en-us/articles/360007903633-How-to-configure-your-zyBook

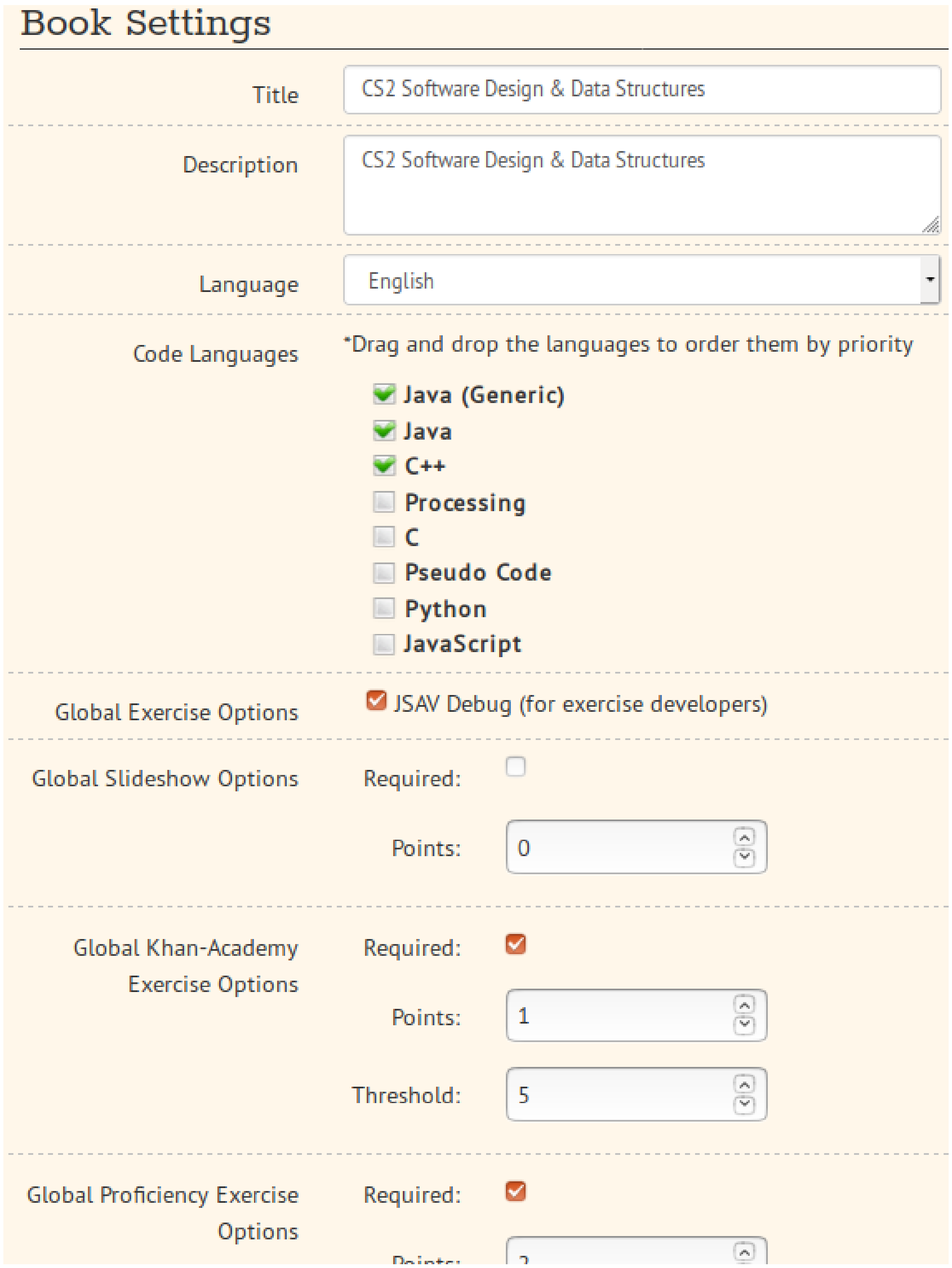
Comparison to OpenDSA

The zyBooks and OpenDSA configuration GUIs have a number of similar features. Both systems allow users to re-organize the content of their book using a drag-and-drop interface, and to remove content from their book. Both systems also allow users to add new chapters, rename chapters, and remove content from their book.

One difference between zyBooks and OpenDSA is that OpenDSA does not allow for content to be explicitly marked as optional. Another difference is that zyBooks does not appear to allow instructors to add additional sections to their book. That is, a given zyBook contains a pre-defined set of sections, and the GUI tool does not provide a way to add additional sections beyond this pre-defined set, though sections can be moved to different chapters. OpenDSA's book configuration GUI provides users more control over the content in their book by also allowing them to add additional content not already in the pre-defined configuration. OpenDSA's book configuration GUI also allows users to customize the settings for individual exercises in a book, while zyBook's GUI does not.

3.5.3 Summary

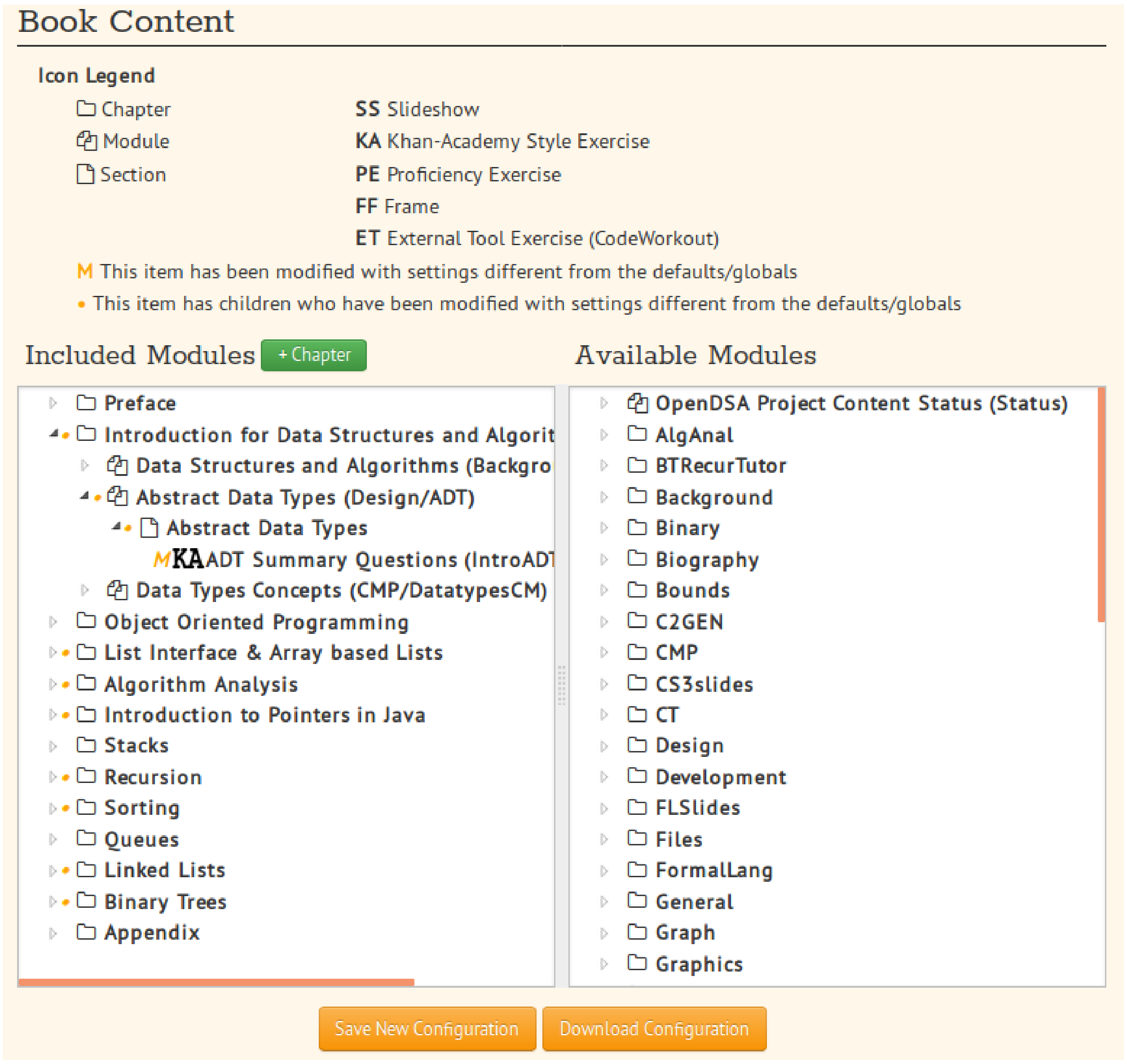
OpenDSA, Runestone Interactive, and zyBooks all provide pre-configured books, and also allow for some level of customization of these pre-configured books. Both OpenDSA and zyBooks provide a GUI with a drag-and-drop interface to allow users to re-organize the content of a book or remove content from a book, however OpenDSA's GUI also allows for a user to add additional content that is not already included in the pre-configured book. Runestone does not provide a GUI tool for configuring books, but its configuration system does allow for re-organizing the content of a book, and potentially adding content not already included in a pre-configured book. Compared to Runestone's configuration format, OpenDSA's format would appear to be more robust since it does not require maintaining multiple copies of an RST file in order to include that file in multiple books. OpenDSA's system also allows for all book customization to be done in a single configuration file, while customizing a Runestone book would require modifying potentially multiple files. Finally, Runestone's book configuration format and the zyBooks configuration GUI do not appear to allow for exercise settings to be customized (e.g. the threshold for a student to receive credit on an exercise), while OpenDSA's configuration GUI And configuration format both allow for this.



Book Settings

Title	CS2 Software Design & Data Structures
Description	CS2 Software Design & Data Structures
Language	English
Code Languages	*Drag and drop the languages to order them by priority ✓ Java (Generic) ✓ Java ✓ C++ ☐ Processing ☐ C ☐ Pseudo Code ☐ Python ☐ JavaScript
Global Exercise Options	✓ JSAV Debug (for exercise developers)
Global Slideshow Options	Required: ☐ Points: 0
Global Khan-Academy Exercise Options	Required: ✓ Points: 1 Threshold: 5
Global Proficiency Exercise Options	Required: ✓

Figure 3.2: The user interface for customizing book meta-data and global settings



Book Content

Icon Legend

🗀 Chapter	**SS** Slideshow
Module	**KA** Khan-Academy Style Exercise
🗋 Section	**PE** Proficiency Exercise
	FF Frame
	ET External Tool Exercise (CodeWorkout)

M This item has been modified with settings different from the defaults/globals

• This item has children who have been modified with settings different from the defaults/globals

Included Modules + Chapter

- 🗀 Preface
- • 🗀 Introduction for Data Structures and Algorit
  - Data Structures and Algorithms (Backgro
  - • Abstract Data Types (Design/ADT)
    - • 🗋 Abstract Data Types
      - M KA ADT Summary Questions (IntroADT
  - Data Types Concepts (CMP/DatatypesCM)
- 🗀 Object Oriented Programming
- • 🗀 List Interface & Array based Lists
- • 🗀 Algorithm Analysis
- • 🗀 Introduction to Pointers in Java
- 🗀 Stacks
- • 🗀 Recursion
- • 🗀 Sorting
- 🗀 Queues
- • 🗀 Linked Lists
- • 🗀 Binary Trees
- 🗀 Appendix

Available Modules

- OpenDSA Project Content Status (Status)
- 🗀 AlgAnal
- 🗀 BTRecurTutor
- 🗀 Background
- 🗀 Binary
- 🗀 Biography
- 🗀 Bounds
- 🗀 C2GEN
- 🗀 CMP
- 🗀 CS3slides
- 🗀 CT
- 🗀 Design
- 🗀 Development
- 🗀 FLSlides
- 🗀 Files
- 🗀 FormalLang
- 🗀 General
- 🗀 Graph
- 🗀 Graphics

Figure 3.3: The user interface for customizing the contents of a book

Chapter 4

Interoperability Improvements and Issues

This chapter presents improvements made to OpenDSA's interoperability with LMS and other LTI tool providers. In doing so we explore ways of integrating educational systems, and also provide examples and implementation details that others might find useful when developing their own LTI-enabled systems. We also discuss some interoperability issues that we ran into related to LTI, identifying potential areas for improvement.

4.1 Becoming an LTI Tool Consumer

Previous work implemented functionality to allow OpenDSA to utilize content from other LTI tool providers in OpenDSA textbooks [48]. In particular, OpenDSA uses the Code-Workout system [22] to provide short programming exercises. In the following sections we describe OpenDSA's integration with CodeWorkout. First we describe the original approach for integrating CodeWorkout exercises that resulted in CodeWorkout receiving LTI launch requests directly from the LMS. Next we describe how we turned OpenDSA into an LTI tool consumer, thus enabling a new approach where CodeWorkout receives launch requests directly from OpenDSA, and sends assignment scores to OpenDSA instead of the LMS. Although we discuss these approaches in the context of OpenDSA and CodeWorkout, other

LTI-enabled tools can be integrated with OpenDSA without any significant tool-specific changes.

4.1.1 Original Approach

In order to incorporate CodeWorkout exercises in OpenDSA books, the original OpenDSA LTI implementation used the Canvas LMS API to add CodeWorkout exercises as separate pages and assignments in Canvas during OpenDSA's course generation process. This meant that Canvas was the entity sending LTI launch requests to CodeWorkout and receiving CodeWorkout scores, leaving OpenDSA entirely out of the loop, as illustrated in Figure 4.1. As a result OpenDSA did not know when or even if OpenDSA users accessed CodeWorkout exercises, nor when or if users completed the exercises, nor what scores users achieved on the exercises.

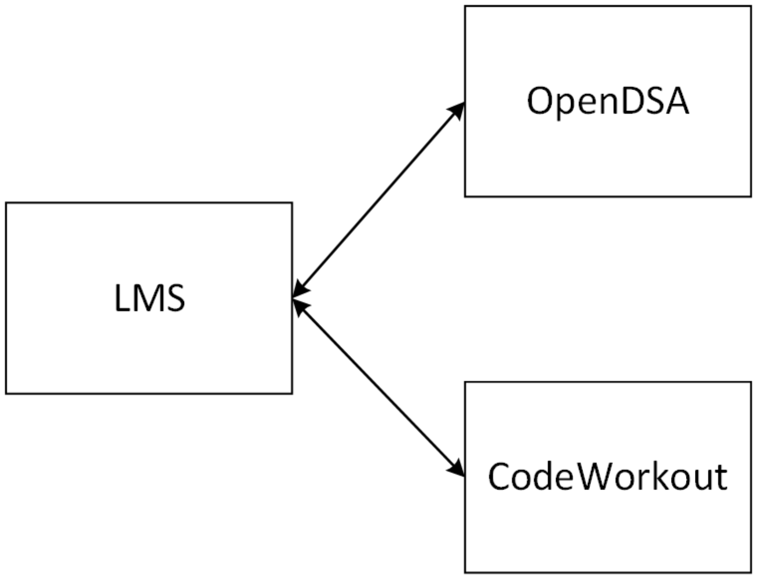


Figure 4.1: The original pattern of communication between OpenDSA, CodeWorkout, and the LMS showing the lack of contact between OpenDSA and CodeWorkout

In fact, not only was every CodeWorkout exercise placed on its own page in Canvas, but every section in an OpenDSA module had its own page in Canvas as well, with at most one gradable exercise per section. A main reason for this was that Canvas only allows one gradebook entry per assignment. A major advantage of every OpenDSA module section being on its own web page was that it allowed more precise tracking of user interaction data in terms of tracking what part of an OpenDSA book the user was looking at. The main issue with this approach was that OpenDSA modules were originally designed to be displayed on a single page, so placing each module section on a separate page could negatively impact the cohesiveness of the content. One example is that OpenDSA exercises sometimes refer to content that is located in a preceding section of the module, so in some cases the exercises were placed on separate pages from the content they referenced. Another problem was that OpenDSA module sections that contained only prose would not show up on the assignments page in a Canvas course since they were not graded. As a result, these ungraded sections were missed or ignored by some students who only looked at the assignments page, even though the sections contained relevant content.

4.1.2 New Approach

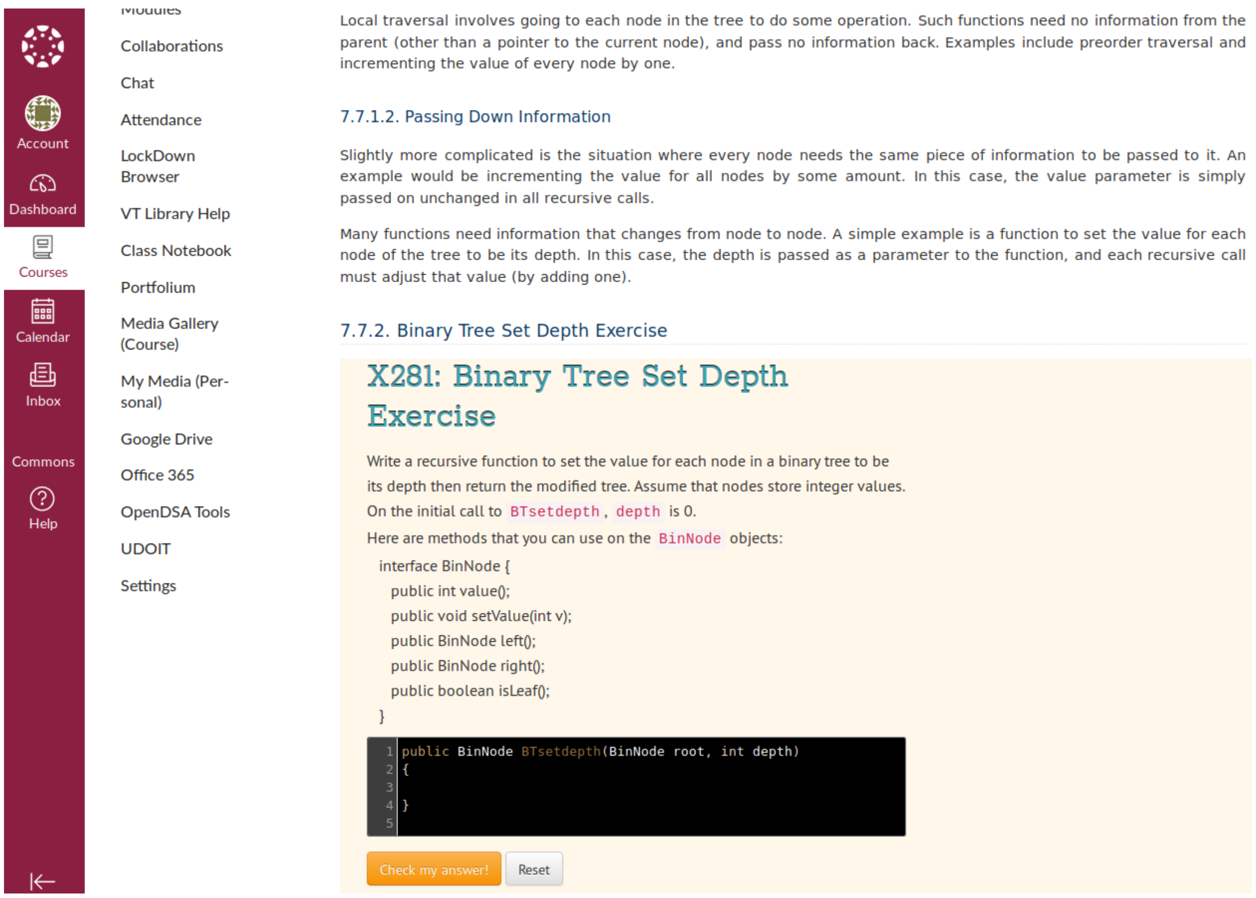
In order to make the experience of OpenDSA LTI books more similar to the original intended design of a single OpenDSA module being a single web page, and address other issues described in the previous section, it was decided that OpenDSA should take a new approach that doesn't involve splitting up modules into individual sections. To make this new approach possible required addressing two challenges:

1. Allowing multiple graded exercises per LMS assignment

2. Handling external (i.e. CodeWorkout) exercises

Addressing the first challenge is fairly straightforward. OpenDSA simply aggregates the scores of all exercises in a module and then sends the aggregate score to the LMS, rather than the individual exercise scores. When a module is launched by a user for the first time, OpenDSA creates a record in the database to track the user's progress on the module. The module progress record also stores the parameters from the LTI launch request that are necessary to send the score to the LMS so that the module score can be sent at a later time. Whenever a user completes an exercise, the score for the module containing the exercise is recalculated and sent to the LMS. Since the LTI standard requires scores to be between 0 and 1, the aggregate score is calculated by simply adding up the points a user has earned on each exercise in a module and then dividing by the total possible points. A disadvantage of this is that if an OpenDSA module has multiple graded exercises, then students and instructors will only see the aggregate module score in the LMS gradebook, and not scores for the individual exercises. However, since OpenDSA uses a mastery-based approach where a student either receives zero credit or full credit for an exercise, this is not a major issue.

The simplest way to address the second challenge would have been to continue making each CodeWorkout exercise a separate page and assignment in the LMS, but this would not solve the issues discussed in Section 4.1.1, such as the inability of OpenDSA to know when or if a user has completed a CodeWorkout exercise, and the negative impact on content cohesion that results from placing exercises on pages separate from the associated prose. Instead the second challenge was addressed by turning OpenDSA into an LTI tool consumer, allowing OpenDSA to send standard LTI launch requests to CodeWorkout. This approach allows for OpenDSA to display CodeWorkout exercises in an iframe on the same page as native OpenDSA content as shown in Figure 4.2. This approach also solves the problem of OpenDSA not knowing when or if students access CodeWorkout exercises, and also the problem of OpenDSA not knowing what scores students receive on CodeWorkout exercises since CodeWorkout sends student scores directly to OpenDSA. When OpenDSA receives a score from CodeWorkout, the score for the OpenDSA module containing the CodeWorkout exercise is recalculated and then sent to the LMS. The new communcation pattern between the LMS, OpenDSA, and CodeWorkout is illustrated in Figure 4.3. It should be noted that this new communication pattern was implemented without making any changes to the CodeWorkout codebase.



Local traversal involves going to each node in the tree to do some operation. Such functions need no information from the parent (other than a pointer to the current node), and pass no information back. Examples include preorder traversal and incrementing the value of every node by one.

7.7.1.2. Passing Down Information

Slightly more complicated is the situation where every node needs the same piece of information to be passed to it. An example would be incrementing the value for all nodes by some amount. In this case, the value parameter is simply passed on unchanged in all recursive calls.

Many functions need information that changes from node to node. A simple example is a function to set the value for each node of the tree to be its depth. In this case, the depth is passed as a parameter to the function, and each recursive call must adjust that value (by adding one).

7.7.2. Binary Tree Set Depth Exercise

X281: Binary Tree Set Depth Exercise

Write a recursive function to set the value for each node in a binary tree to be its depth then return the modified tree. Assume that nodes store integer values.
On the initial call to `BTsetdepth`, `depth` is 0.
Here are methods that you can use on the `BinNode` objects:

```
interface BinNode {
   public int value();
   public void setValue(int v);
   public BinNode left();
   public BinNode right();
   public boolean isLeaf();
}
```

```
public BinNode BTsetdepth(BinNode root, int depth)
{

}

```

Figure 4.2: An OpenDSA module with a CodeWorkout exercise displayed in Canvas via LTI

LTI Tool Consumer Implementation Details

This section discusses some of the more technical details of OpenDSA's LTI tool consumer implementation.

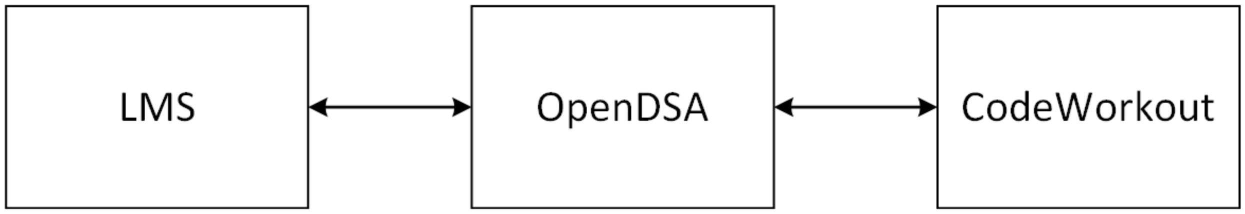


Figure 4.3: The new pattern of communication between OpenDSA, CodeWorkout, and the LMS

When a user accesses an OpenDSA module from within an LMS, the LMS sends an LTI launch request to the OpenDSA server. OpenDSA finds or creates the module progress record for the user, which also stores the `lis_outcome_serivce_url` and `lis_result_sourcedid` parameters from the LTI launch request which will later be use to send the user's module score to the LMS. OpenDSA then returns the HTML page for the requested module which is displayed within an iframe in the LMS. When an OpenDSA module page containing a CodeWorkout exercise is loaded in the user's browser, an iframe that will contain the CodeWorkout exercise is generated by the OpenDSA client-side framework. However, the target URL of this iframe is an endpoint on the OpenDSA server, not the CodeWorkout server. When the iframe is first loaded, a `GET` request is automatically sent to an endpoint on the OpenDSA server. The `GET` request contains a URL parameter specifying the ID of the exercise instance, which is used to retrieve the exercise information from the OpenDSA database, along with information on the course offering the exercise instance is associated with. The exercise information includes the name of the LTI tool provider that the exercise is from, allowing OpenDSA to look up the LTI launch url, consumer key, and shared secret for that tool provider in the OpenDSA database. Using this information, OpenDSA generates the parameters for an LTI launch request. Some of the key parameters OpenDSA sends in the LTI launch request are described in Table 4.1. All of the parameters listed are standard parameters defined in the LTI specification [5].

After generating the LTI launch parameters, OpenDSA uses an OAuth library to gener-

Parameter	**Description**
context_id	In LTI, a "context" usually refers to a course, but it can also refer to a project or other collection of resources. We use the ID of the OpenDSA course offering.
context_label	This is a short label for the context. We use the OpenDSA course offering's course number (e.g. "CS 3114").
context_title	This is a plain text title of the context. We use the full name of the OpenDSA course (e.g. "Data Structures and Algorithms").
lis_outcome_service_url	This is the URL that the tool provider (e.g. CodeWorkout) should post scores to.
lis_result_sourcedid	This is an opaque identifier that identifies a unique entry in the gradebook of the tool consumer (e.g. OpenDSA). We use a string in the format "{user_id}_{exercise_instance_id}". The tool provider includes this parameter when sending the score back to the tool consumer.
resource_link_id	This is an opaque identifier that uniquely identifies an LTI link in the tool consumer. We use the ID of the OpenDSA exercise instance.
resource_link_title	This is the title of the resource. We use the name of the exercise.
roles	This a listing of the role(s) of the user (e.g. Learner or Instructor).
lis_person_name_given	This is the first name of the user who triggered the LTI launch.
lis_person_name_family	This is the last name of the user who triggered the LTI launch.
lis_person_contact_email_primary	This is the email of the user who triggered the LTI launch.

Table 4.1: A description of some key LTI launch parameters

ate OAuth parameters with CodeWorkout's consumer key and shared secret. Next, OpenDSA generates an HTML page containing a hidden HTML form that includes all of the LTI and OAuth parameters. The HTML page also contains a single line of JavaScript that automatically submits the form when the page is rendered by the user's browser, which results in the LTI launch request being sent to the CodeWorkout server. The CodeWorkout server authenticates the request using the OAuth parameters, then displays the exercise specified by the LTI standard `resource_link_title` parameter, which matches the name of a CodeWorkout exercise. It should be noted that using the `resource_link_title` to identify the exercise is not required by the LTI specification. LTI also allows for custom parameters (which are denoted by the prefix "`custom_`"), so another approach would be to define a custom parameter that OpenDSA and CodeWorkout both agree would be used to specify the exercise being requested.

After the user completes the CodeWorkout exercise, CodeWorkout sends the user's score to the URL specified by the `lis_outcome_serivce_url` from the launch request sent by OpenDSA, along with the `lis_result_sourcedid` parameter from the launch request. When OpenDSA receives the score from CodeWorkout, the `lis_result_sourcedid` parameter is used to retrieve the user and the exercise instance that the score is for and update the user's score for that exercise. OpenDSA then looks up the module that the exercise is contained in, finds the user's progress record for that module, calculates the new aggregate module score for the user, and then sends the score to the LMS using the `lis_outcome_serivce_url` and `lis_result_sourcedid` parameters that were saved with the user's module progress record when OpenDSA first received the LTI launch request from the LMS.

4.1.3 Discussion

OpenDSA's LTI tool consumer implementation described above is not specific to CodeWorkout. Thus, it can potentially be used to incorporate content from other tool providers that conform to the LTI standard. For example, OpenDSA's tool consumer functionality has already been used to integrate dozens of exercises and visualizations from the Advanced Content Server (ACOS) project[1] into OpenDSA [41]. Enabling the integration of these ACOS exercises and visualizations into OpenDSA required only adding entries for the ACOS LTI launch endpoints to the OpenDSA database, and adding entries to a static look-up table used when compiling non-LTI OpenDSA books.

As an LTI tool consumer, OpenDSA can now integrate third-party content while maintaining many of the benefits of developing the content natively, without OpenDSA developers having to spend the time and resources to develop the content themselves. OpenDSA is also now able to display content from other tool providers inside iframes alongside native OpenDSA content, resulting in a more seamless integration than the previous approach of setting up third-party content as separate pages in the LMS. In addition, OpenDSA also now has knowledge of when users complete third-party exercises, as well as what scores they receive. To a user of OpenDSA, a third-party exercise is now functionally no different from a native OpenDSA exercise. The most notable disadvantage that currently remains in using third-party content is that OpenDSA does not receive interaction data from the third parties. Chapter 5 discusses the problem of sharing interaction data among educational systems in more detail.

[1]https://github.com/acos-server/acos-server

4.2 Improving OpenDSA Content Reusability

While OpenDSA textbooks can be customized to include only the content an instructor wants, this is primarily at the module level. An instructor may not want to use an entire OpenDSA textbook, or even an entire OpenDSA module. Alternatively, an instructor may want to use OpenDSA content in their course, but are restricted to an LMS for which OpenDSA does not currently support course generation. Another possibility is that some other learning tool or platform may want to incorporate OpenDSA content to be served to their own users for their own purposes. In this section we present modifications made to OpenDSA infrastructure to allow for stand-alone (i.e. outside of a book) OpenDSA exercises, visualizations, and modules to be served through LTI (and otherwise), in order to support the use cases described above.

4.2.1 Supporting LTI Content Selection

Before LTI-enabled content can be used in an LMS, an LTI launch link for that content must first somehow be configured in the LMS. OpenDSA uses the Canvas API in order to add LTI launch links to a Canvas course during the course generation process for an OpenDSA book. This is an acceptable approach for OpenDSA due to the need to configure many different LTI launch links at once, but this is not a standards-based approach since it requires an LMS-specific adapter to be created for each LMS that OpenDSA wants to support. One LMS-agnostic approach is for tool providers to provide an interface on their website that instructors can go through to find the launch URL for each specific piece of content they want. This is a less than ideal approach that requires the instructor to navigate to the tool provider's website, copy the LTI launch information from the tool provider's interface, then paste the launch information into the LMS interface.

An approach used by some tools is to provide a generic launch URL that displays a resource selection interface the first time a particular link is launched by an instructor. The tool can determine if it is the first time a link is being launched by checking if it has seen certain parameters in the launch request before, specifically the `context_id` and `resource_link_id`. Once a resource is selected, that resource can then be associated with the particular `context_id` and `resource_link_id` so that later launches of that link will immediately load the resource. A downside of this approach is that if a course is copied, then the IDs will change and the links will have to be reconfigured.

An early solution for this problem was developed and implemented by the Canvas LMS in the form of an extension to the LTI standard[2]. This extension enables a feature where the instructor can select from a list of pre-configured LTI tool providers when creating a new assignment. Canvas then sends an LTI launch request to the selected tool provider, including an extension parameter `ext_content_return_types=select_link` indicating that the tool should display an interface allowing the instructor to select a resource to add to their course. This interface is displayed in a modal window, so the instructor never has to leave the LMS. When the instructor selects a resource in the tool provider's interface, the tool provider then appends the LTI launch URL of the selected resource to the URL specified by the `launch_presentation_return_url` parameter from the launch request, and redirects the user to that URL. Canvas is then able to automatically fill in the launch URL for the resource in the assignment creation form. This is the method that OpenDSA initially used for LTI resource selection (Section 2.2.3).

In 2016, IMS released a better solution: a new LTI feature which they call *Deep Linking* [7] (formerly named *Content-Item Message*). A standard LTI launch request has an `lti_message_type` parameter with a value of `basic-lti-launch-request`. Deep Linking

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

introduces the `ContentItemSelectionRequest` and `ContentItemSelection` message types. A rough outline of the workflow of LTI Deep Linking is illustrated in Figure 4.4, and is fairly similar to the Canvas extension described above. The basic steps are as follows:

1. The user selects a tool provider they want to add content from
2. The LMS redirects the user to the tool provider via an LTI launch request with an `lti_message_type` of `ContentItemSelectionRequest`
3. The tool provider displays an interface allowing the user to select one or more resources
4. The tool provider redirects the user back to the LMS using a `POST` request with information on the resource or resources that were selected, and an `lti_message_type` of `ContentItemSelection`

Deep Linking is an improvement over the Canvas extension for several reasons:

- It can potentially allow a user to add multiple resources to their course at once
- The response from the tool provider to the LMS can include more extensive information on the resource(s) selected
- The response from the tool provider to the LMS is verified using OAuth

The more extensive resource information provided by the tool provider can enable the LMS to automatically populate fields such as an assignment title and description.

OpenDSA Deep Linking

OpenDSA uses Deep Linking to allow instructors to add stand-alone (i.e. outside of a book) OpenDSA exercises, visualizations, and modules to their LMS course. Due to the similarities

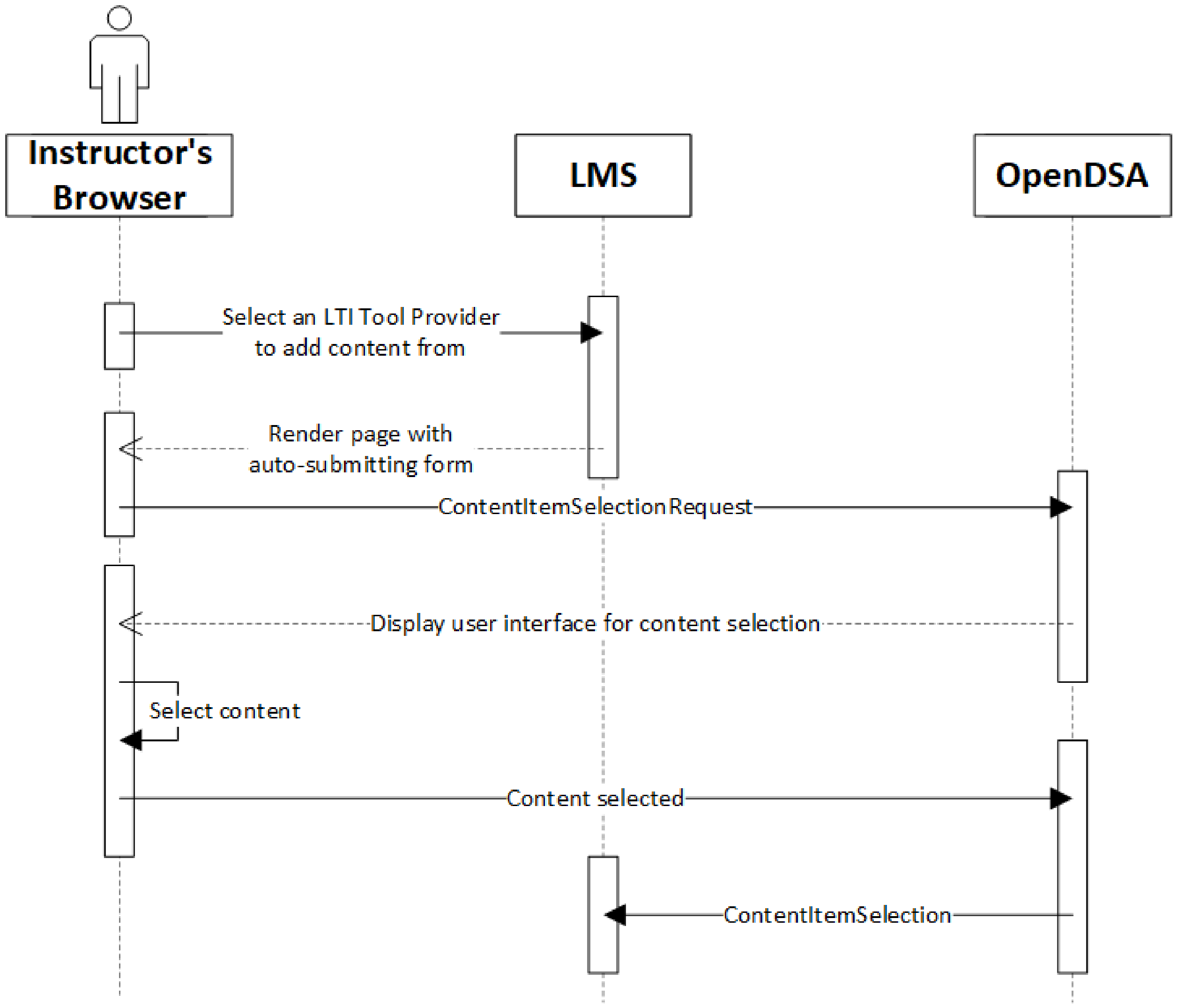

Instructor's Browser
LMS
OpenDSA
Select an LTI Tool Provider to add content from
Render page with auto-submitting form
ContentItemSelectionRequest
Display user interface for content selection
Select content
Content selected
ContentItemSelection

Figure 4.4: A rough outline of the LTI Deep Linking workflow

between Deep Linking and the Canvas LTI extension for resource selection, OpenDSA support for Deep Linking could be built on top of the existing resource selection infrastructure. The OpenDSA LTI Deep Linking work flow is as follows:

1. The OpenDSA server receives an LTI launch request with an `lti_message_type` of `ContentItemSelectionRequest`.

2. The OpenDSA server verifies the request using OAuth and displays an interface for

selecting content. The `content_item_return_url` and `oauth_consumer_key` values from the launch request are saved in JavaScript variables on the client.

3. The user selects an OpenDSA resource they want to add to their course.

4. The OpenDSA client sends an AJAX request to the OpenDSA server with information on the resource that was selected, along with the `oauth_consumer_key` and `content_item_return_url`.

5. The OpenDSA server receives the AJAX request and does the following:

   (a) Generates a JSON object storing the information on the selected resource according to the Deep Linking specification [8], converts it to a string, and stores it in the `content_items` parameter.

   (b) Sets the `lti_message_type` parameter to `ContentItemSelection`.

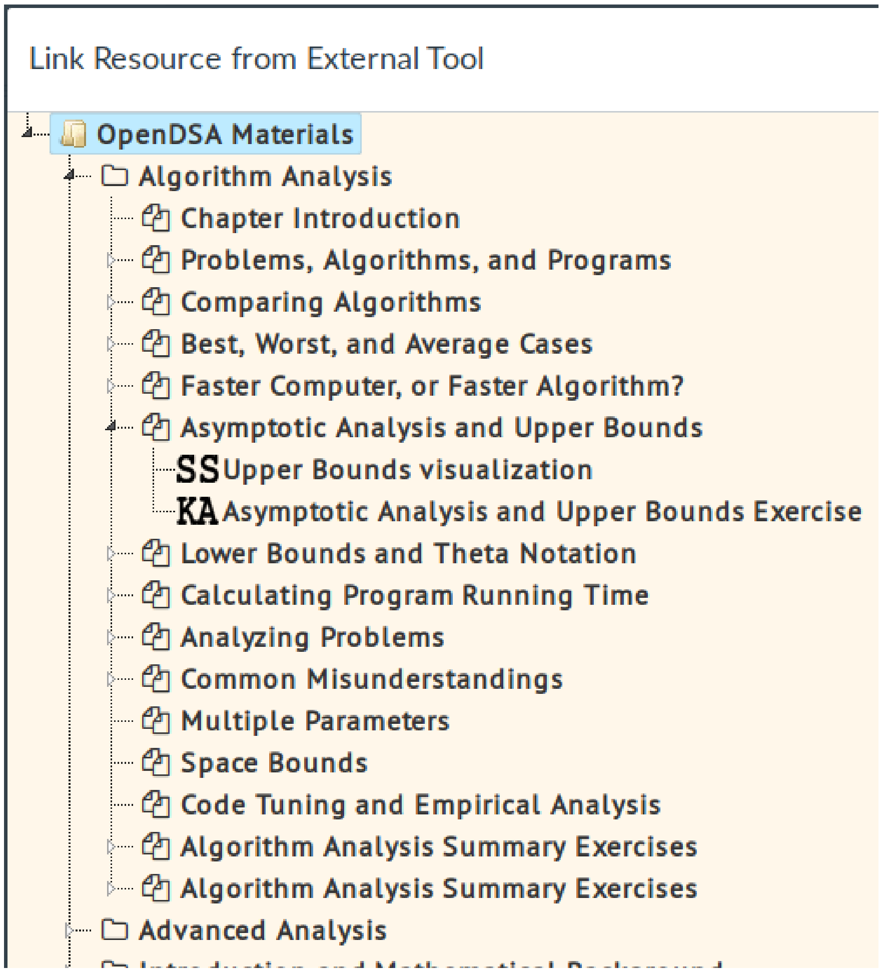
   (c) Looks up the shared secret associated with the `oauth_consumer_key` and generates OAuth parameters.

   (d) Returns all parameters to the OpenDSA client.

6. The OpenDSA client generates a hidden HTML form with input fields storing all of the parameters received from the OpenDSA server. The `action` attribute of the form is set to the value of the `content_item_return_url` parameter from the LTI launch request sent by the LMS.

7. The OpenDSA client automatically submits the form using JavaScript, resulting in a `POST` request being sent to the `content_item_return_url`.

The end result is that the LMS receives the information necessary to set up a link that launches directly to the specific resource the user selected.


Link Resource from External Tool

- OpenDSA Materials
  - Algorithm Analysis
    - Chapter Introduction
    - Problems, Algorithms, and Programs
    - Comparing Algorithms
    - Best, Worst, and Average Cases
    - Faster Computer, or Faster Algorithm?
    - Asymptotic Analysis and Upper Bounds
      - SS Upper Bounds visualization
      - KA Asymptotic Analysis and Upper Bounds Exercise
    - Lower Bounds and Theta Notation
    - Calculating Program Running Time
    - Analyzing Problems
    - Common Misunderstandings
    - Multiple Parameters
    - Space Bounds
    - Code Tuning and Empirical Analysis
    - Algorithm Analysis Summary Exercises
    - Algorithm Analysis Summary Exercises
  - Advanced Analysis

Figure 4.5: The current OpenDSA interface for LTI resource selection shown in the Canvas LMS

LMS Support and Differences

Prior to this work, OpenDSA only supported LTI resource selection for the Canvas LMS through the Canvas LTI extension. OpenDSA now implements LTI Deep Linking and explicitly supports resource selection for Canvas, Moodle, and Blackboard Learn. It is possible that OpenDSA's Deep Linking implementation also works with other LTI-compliant LMS, but it has not been tested.

While implementing and testing support for Deep Linking in OpenDSA, a number of differences between LMS were noted. Perhaps the most important difference noted is that Canvas does not seem to entirely conform to the LTI Deep Linking specification. For one, Canvas requires the the launch URL be explicitly specified for the item being added, however the specification [8] lists the URL as an optional field, with the tool consumer being directed to assume the default launch URL for the tool provider if the field is omitted. If the URL field is omitted, Canvas shows a rather unspecific error message: "There was a problem retrieving a valid link from the external tool". Another deviation from the specification is that Canvas does not honor custom parameters specified by the tool provider in the `ContentItemSelection` message. The Deep Linking specification dictates that the `ContentItemSelection` message may optionally include a set of custom parameters for items of type `LtiLinkItem` [8]. The tool consumer should then include these custom parameters in subsequent LTI launch requests to the link being created. When custom parameters are specified in a `ContentItemSelection` message to Canvas for an `LtiLinkItem`, the custom parameters are *not* included in subsequent LTI launch requests to the created link.

Another difference is that the response to a `ContentItemSelection` message from a tool provider can vary between LMS. Moodle will automatically populate the fields of an assignment creation form and allow the user to review the settings before actually creating the assignment. Canvas simply fills in the launch URL in the assignment creation form and nothing else (e.g. the assignment title field is left blank). On the other hand, Blackboard immediately creates the assignment based on the information in the tool provider's `ContentItemSelection` message. Since Blackboard immediately adds the resource to the course without providing an interface to finalize the resource settings, the tool provider must provide an interface for the instructor to specify whether the resource should be graded, and if graded, how many points it is worth. The tool provider must then include this information in the `content_items` parameter when sending the resource information to the LMS.

Details on this can be found in the Deep Linking specification [8]. The Deep Linking standard also specifies that a tool consumer should include the parameter `auto_create=true` in the `ContentItemSelectionRequest` if it immediately and automatically persists the resource information it receives in a `ContentItemSelection` message, and that a value of `false` should be assumed if the parameter is omitted, but Blackboard does not include this parameter even though it *does* perform automatic persistence.

Another difference between Blackboard and other LMS is that Blackboard does not provide a way for instructors to pre-configure LTI tool providers at the course level. The only way to pre-configure an LTI tool provider in Blackboard is for a Blackboard administrator to do so. Blackboard does allow instructors to set up links and assignments that link to LTI tool providers that have not been pre-configured, but the instructor must provide their OAuth consumer key and shared secret for every link they want to add, and they must also know the LTI launch URL (i.e. by copying it from the tool provider's website). On the other hand, Canvas and Moodle do allow instructors to pre-configure LTI tool providers at the course level.

One advantage that Blackboard does have over Canvas and Moodle is that Blackboard supports adding multiple resources at one time through Deep Linking, which can help save time for instructors. Whether or not a tool consumer allows adding multiple resources at one time can be determined by the `accept_multiple` parameter in the `ContentItemSelectionRequest` [8]. At the time of this writing, Moodle appears to be planning support for adding multiple resources at a time with Deep Linking, though the status of the plans has not been updated since August 2017[3]. Canvas does not appear to have any public plans to support adding multiple resources at once. OpenDSA could potentially take advantage of LMS like Blackboard that support adding multiple resources at a time

[3]https://docs.moodle.org/dev/LTI_ContentItem_Enhancements

through Deep Linking to allow adding an entire OpenDSA book to a course, which would eliminate the need for LMS-specific adapters for course generation. However, this requires further exploration.

4.2.2 Stand-alone Exercises and Visualizations

OpenDSA has hundreds of interactive visualizations and auto-graded exercises. For the remainder of this section, we will use "exercises" to refer to both exercises and visualizations or the sake of brevity. As described in Section 2.2.3, previous work [48] created a prototype implementation for allowing individual OpenDSA exercises to be served through LTI, outside of the context of a book. In this section we describe the implementation of a more robust system for serving stand-alone exercises, both through LTI and in any arbitrary HTML page.

LTI Support

The first step in improving the prototype implementation for stand-alone exercises was to allow for an exercise to be directly associated with a course offering, rather than a book. To that end, a new database table named `inst_course_offering_exercises` was created to store such associations. This table also stores the settings for each particular exercise instance.

The next step was to enable the creation of an HTML page for displaying a particular exercise without having to configure and compile a book. The approach we took takes advantage of the support for template pages in the Rails framework. We created template pages for the different exercise types offered by OpenDSA, with the template pages mimicking the HTML that is usually generated during the OpenDSA compilation process, but with placeholders for values that are specific to each exercise. The Rails templating engine then generates a complete HTML page by replacing the placeholder values with the actual values at runtime when a particular exercise is requested.

However, before we can generate an HTML page for an exercise, we need to know what values should be used. For exercises that are displayed in an iframe we need to know what file the HTML page for that exercise is stored in. For exercises that are displayed without an iframe, we need to know what JavaScript and CSS files are required by that exercise. Initially, the OpenDSA database did not store this information; the only place this information was stored was in the module RST files. While the information could be manually input into the database, this would not be a scalable approach. Instead, an automated process was developed for transcribing the information from the RST files to the database.

One problem that arose was determining which JavaScript and CSS files are needed by a particular exercise. This is only a problem for exercises using the `inlineav` RST directive, since those exercises are loaded directly in the module page, as opposed to exercises using the `avembed` directive which have their own HTML page and are displayed inside an iframe in the module page. Originally, if a CSS or JavaScript file was required by one or more `inlineav` in a certain module, then it was included near the top or bottom of the RST file as an RST `odsalink` or `odsascript` directive. These directives are replaced with standard HTML `link` and `script` tags when the RST file is compiled. We could simply include all of the JavaScript and CSS files from a particular module when we generate the HTML page for an exercise from that module, but this is wasteful and can sometimes lead to errors if one of the scripts is expecting a certain HTML element to be present and that element doesn't exist. Instead we implemented two pseudo options, `:links:` and `:scripts:`, for the `inlineav` directive to allow for JavaScript and CSS files to be associated with a particular exercise. These pseudo options are converted to `odsalink` and `odsascript` directives in a pre-processing step prior to the RST file being compiled.

We also expanded the functionality of the `simple2full.py` script described in Section 3.2 so that it can also record the values of the `:links:` and `:scripts:` pseudo options from the RST files, as well as the path of HTML file associated with each `avembed` directive, and store the values in the full configuration file it generates. We then developed a Rake[4] task that runs the `simple2full.py` script and transcribes the values from the generated JSON configuration file to the database. The Rake task is automatically run whenever an update is deployed to the OpenDSA server so that the database always contains up-to-date exercise information.

When an instructor goes to add an OpenDSA resource to their course, the list of exercises is loaded from the database and displayed for the instructor to pick from. When an exercise is selected, the instructor is shown an interface allowing them to customize the settings for the selected exercise. After the instructor finalizes their selection, the launch URL for the selected content is generated, with the URL including the unique ID of the exercise, and the encoded exercise settings. The instructor's selection is then sent to the LMS by the Deep Linking process described in Section 4.2.1. When that exercise is later launched, OpenDSA does the following:

1. Verifies the request using OAuth.
2. Automatically signs in the user based on the `lis_person_contact_email_primary` parameter
3. Determines which LMS instance the launch request came from based on the origin of the request.
4. Uses a combination of the the LMS instance and the `context_id` parameter in the launch request to find a matching course offering record in the OpenDSA database.

[4]Rake is a build utility for Ruby that is similar to Make - https://github.com/ruby/rake

The `context_id` uniquely identifies a course offering (or other similar context) within a particular instance of an LMS. If no course offering is found, then one is automatically created.

5. Uses a combination of the course offering and the `resource_link_id` parameter from the launch request to find the `inst_course_offering_exercise` record in the database. The `resource_link_id` uniquely identifies a particular link within a context. If no record is found, then one is automatically created.

6. Finds or creates a record in the database to track the user's progress on the exercise. If the launch request includes `lis_outcome_service_url` and `lis_result_sourcedid` parameters, then they are stored with the exercise progress record.

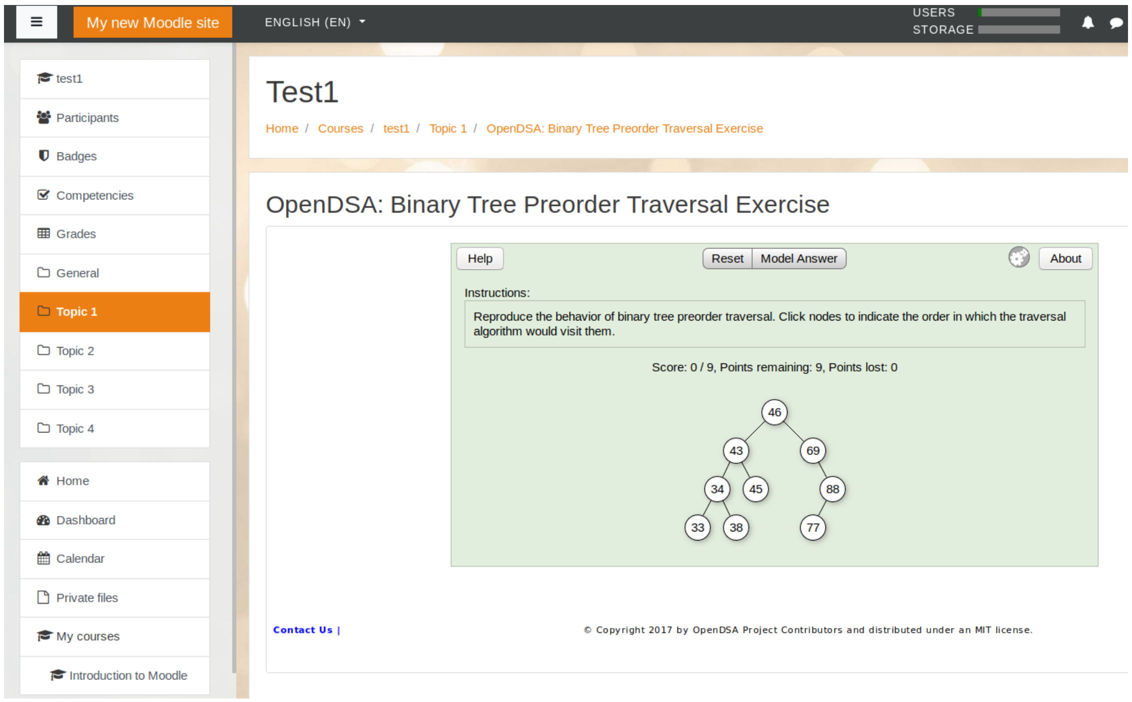
7. Renders the HTML page for the exercise which is shown to the user.

When a user completes a stand-alone exercise, OpenDSA uses the `lis_outcome_service_url` and `lis_result_sourcedid` parameters stored with the user's exercise progress record to send the score to the LMS.

This feature was made available to instructors in December of 2017. As of July 2019 the feature has been used in one course offering which was held during the Spring 2019 semester and used the Canvas LMS. This course offering used 37 exercises and visualizations which were used by a total of 64 students.

Plain HTML

An instructor who was using OpenDSA requested the ability to use some OpenDSA exercises in a context where LTI was not supported. We implemented basic support for displaying OpenDSA exercises in a standard non-LTI iframe so that they could be embedded in any



Instructions:

Reproduce the behavior of binary tree preorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

Score: 0 / 9, Points remaining: 9, Points lost: 0

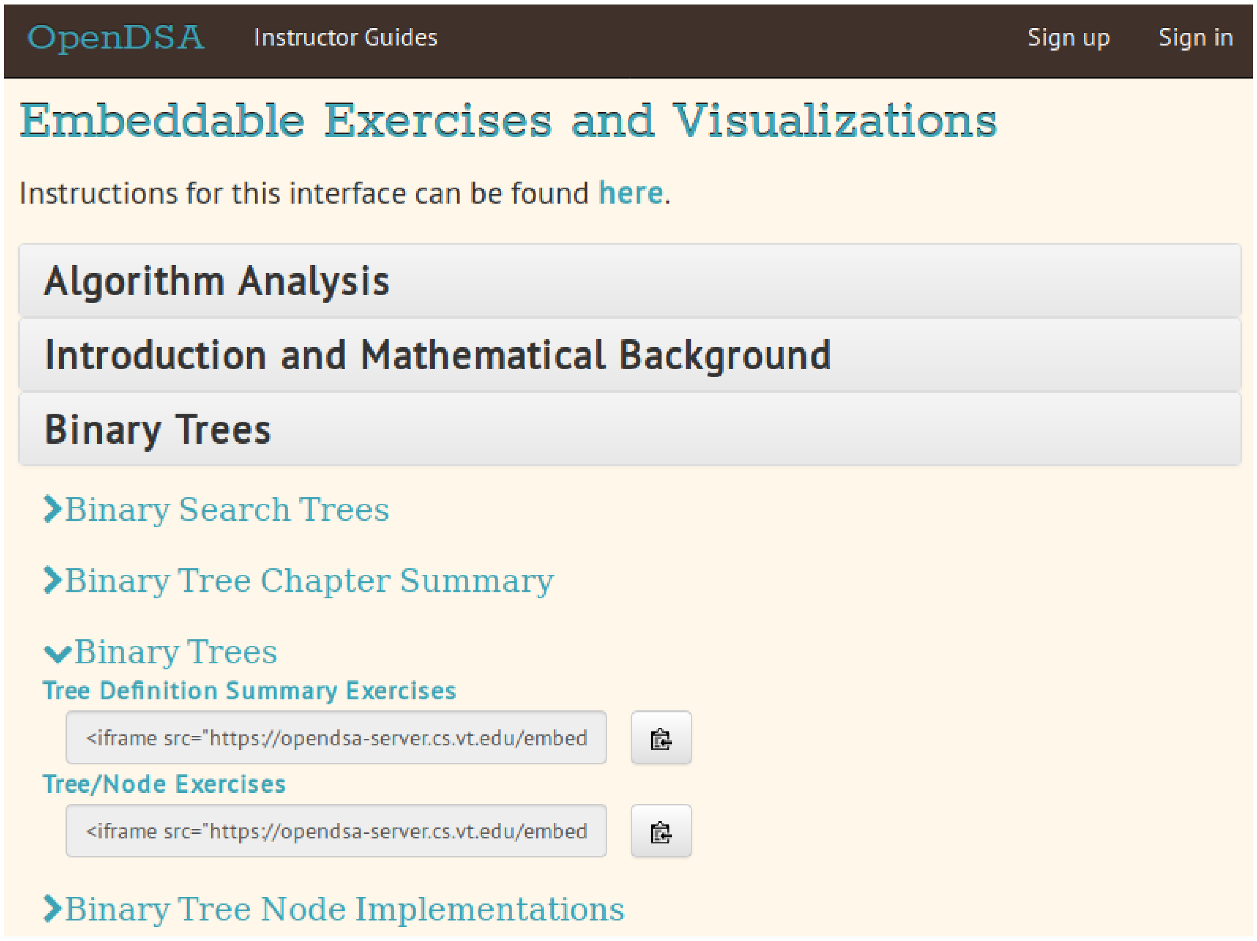
© Copyright 2017 by OpenDSA Project Contributors and distributed under an MIT license.

Figure 4.6: A stand-alone OpenDSA exercise displayed in Moodle via LTI

arbitrary HTML page. Of course the downside to this approach is that exercises served this way cannot report scores. This required little additional work since it is essentially a subset of the functionality needed to serve exercises through LTI. We also implemented a simple web interface to allow instructors and others to find the HTML embed code for any exercises they want to use (Figure 4.7).

4.2.3 Stand-alone Modules

Besides books and individual exercises, we also wanted instructors to be able to add an individual OpenDSA module to their course. Some instructors might want to use OpenDSA modules but are using an LMS that OpenDSA doesn't support course generation for, or only want to use a few modules and don't want to go through the whole process of setting up a book.



Embeddable Exercises and Visualizations

Instructions for this interface can be found here.

Algorithm Analysis

Introduction and Mathematical Background

Binary Trees

Binary Search Trees

Binary Tree Chapter Summary

Binary Trees

Tree Definition Summary Exercises

<iframe src="https://opendsa-server.cs.vt.edu/embed

Tree/Node Exercises

<iframe src="https://opendsa-server.cs.vt.edu/embed

Binary Tree Node Implementations

Figure 4.7: The interface for finding OpenDSA exercises to embed in an HTML page

Since OpenDSA modules are stored in RST files that need to be compiled to create HTML files, a decision had to be made as to how compilation should be handled. One approach would be to compile a new HTML file every time an instructor adds a module to their course. One of the problems with this is that the compilation of RST files requires a non-trivial amount of time and server resources, so this approach could have scalability issues. This could also have a negative impact on user experience since an instructor might have to wait while the module is compiled. The approach that we went with instead was to compile a single version of each module which is then served to everyone who requests it, thus allowing the modules to be compiled only once, and in advance of being requested.

A primary concern with compiling a single module version to be used by everyone is that the module can change over time. While this is not much of a problem if only the prose changes, it is a problem if an exercise is added or removed from the module, or if the structure of the module otherwise changes, because this could cause database records for the module to become out of sync. Our solution to this problem was to implement a simple module version system that takes advantage of the fact that OpenDSA uses Git[5].

We first created a database table named `inst_module_versions`. As the name might suggest, this table stores information on the versions of different modules. We then expanded the functionality of the Rake task described in Section 4.2.2. This Rake task is automatically run when an update is deployed to the OpenDSA server. The Rake task first goes through the RST files of OpenDSA modules and looks at the git commit hash of the commit that the version of the file was last modified in. The git commit hash is compared to the git commit hash stored by the record for the current version of that module in the `inst_module_versions` table. If the hashes don't match, or if no module version record exists, then a new version of the module needs to be compiled. The Rake task then generates a JSON configuration file containing all of the outdated modules and uses it to compile the module RST files into HTML files. We might have instead used the date the file was last modified according to the file system to determine if the file has changed, but using the git commit hash has the added benefit of being able to determine exactly which version of the RST file was used to compile a module's HTML file, which, for example, would be useful if we ever need to recompile an older module version for some reason.

Some changes were also required for the OpenDSA script used to compile RST files to HTML files. Normally when multiple modules are compiled at once, if one module references another module then that reference will be turned into a hyperlink to the referenced module.

[5]`https://git-scm.com/`

Since we are compiling modules that are intended to stand on their own, the script was modified to include an option to disable any hyperlinks to other modules. For LTI books the compilation process would also normally embed the IDs of the exercise instances in the module into the HTML files as data attributes, but since the same HTML for a stand-alone module can be used in multiple course offerings, the exercise instance IDs can vary, so they are left blank for now.

After the compilation is complete, the configuration file is used to create database records for the new module versions in the database. These module versions are also marked as template versions. Later, when an instructor adds a module to their course, the database record for the template module version is cloned, with the clone being tied to the instructor's course offering. The template approach allows for OpenDSA to supply default settings for a module, but still allows for the possibility of an instructor customizing the module settings.

The next step after creating the stand-alone modules was to allow instructors to add a stand-alone module to their course by making the modules available in the OpenDSA resource selection interface. We simply expanded the functionality of the existing interface used for selecting stand-alone exercises to allow for modules to also be selected. When a module is selected, OpenDSA supplies an LTI launch URL to the LMS that includes the ID of the module as a query string parameter. At this time we do not allow the customization of module settings, such as the settings for exercises in the module, but this functionality can be added in the future with purely client-side changes.

The launch process used for a stand-alone module is as follows:

1. Verify the request using OAuth
2. Automatically sign in the user based on the `lis_person_contact_email_primary` parameter

3. Determine which LMS instance the launch request came from based on the origin of the request

4. Use a combination of the the LMS instance and the `context_id` parameter in the launch request to find a matching course offering record in the OpenDSA database. The `context_id` uniquely identifies a course offering (or other similar context) within a particular instance of an LMS. If no course offering is found, then one is automatically created.

5. Use a combination of the course offering and the `resource_link_id` parameter from the launch request to find the `inst_module_version` record in the database. The `resource_link_id` uniquely identifies a particular link within a context.

   (a) If no record is found, then the module ID in the launch request is used to retrieve the latest module version record that is marked as being a template record. The template record is then cloned, with the course offering id and `resource_link_id` being set on the clone. This is also where any custom module settings that might be specified in the launch request would be applied to the clone.

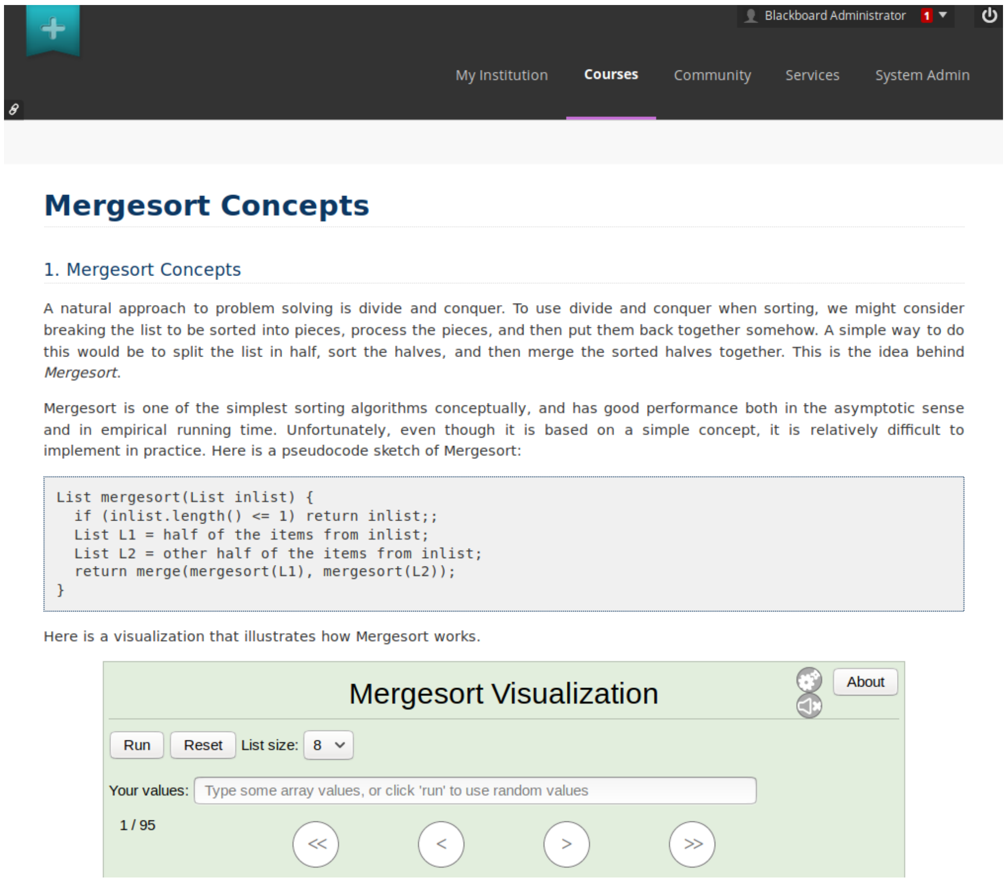
6. Find or create a record in the database to track the user's progress on the module. If the launch request includes `lis_outcome_service_url` and `lis_result_sourcedid` parameters, they are stored with the module progress record.

7. Render the HTML page for the module, which is then shown to the user. The path to the HTML file is stored in the `inst_module_version` record. In order to pass the IDs and settings for the exercise instances to the client-side framework, the module HTML page is wrapped in a template page that includes templated JavaScript code to save the exercise settings in a JavaScript variable.

8. The OpenDSA client-side framework applies the exercise IDs and settings from the JavaScript variable created in previous step, then initializes the exercises in the module.

From then on the process is essentially the same as the process for modules in a book, as described in Section 4.1.2.

This feature was made available to instructors in June 2019, and as of July 2019 it has not yet been used.



Mergesort Concepts

1. Mergesort Concepts

A natural approach to problem solving is divide and conquer. To use divide and conquer when sorting, we might consider breaking the list to be sorted into pieces, process the pieces, and then put them back together somehow. A simple way to do this would be to split the list in half, sort the halves, and then merge the sorted halves together. This is the idea behind *Mergesort*.

Mergesort is one of the simplest sorting algorithms conceptually, and has good performance both in the asymptotic sense and in empirical running time. Unfortunately, even though it is based on a simple concept, it is relatively difficult to implement in practice. Here is a pseudocode sketch of Mergesort:

```
List mergesort(List inlist) {
  if (inlist.length() <= 1) return inlist;;
  List L1 = half of the items from inlist;
  List L2 = other half of the items from inlist;
  return merge(mergesort(L1), mergesort(L2));
}
```

Here is a visualization that illustrates how Mergesort works.

Mergesort Visualization
About
Run
Reset
List size: 8
Your values: Type some array values, or click 'run' to use random values
1 / 95
<<
<
>
>>

Figure 4.8: A stand-alone OpenDSA module displayed in Blackboard via LTI

4.3 Issues and Limitations

This section discusses several issues and limitations related to LTI.

4.3.1 An Unsupported Use Case

We ran into a scenario after turning OpenDSA into an LTI tool consumer that we found was not covered by the LTI standard. When a student completes an OpenDSA exercise, a check mark is displayed on that exercise to indicate to the student that it is in fact completed. This was not possible to do for CodeWorkout exercises that are completed after a module page loads, or any other exercises that OpenDSA might include through LTI, since the OpenDSA client does not have an adequate way of knowing when the CodeWorkout exercise is completed. One might say that OpenDSA should just rely on CodeWorkout to indicate to the user that the exercise is completed, but OpenDSA might also want to display something like a progress bar to the user that shows how close they are to completing the entire OpenDSA module. The Mastery Grids system [39] also needs similar information for its status display. While the OpenDSA server knows almost immediately when a student completes a CodeWorkout exercise (since CodeWorkout sends a score to the OpenDA server), there is no message sent to the OpenDSA client. If the OpenDSA client wants to find out if a CodeWorkout exercise has been completed, then it would have to send a request to the OpenDSA server, but it would either need to regularly poll the server to check the user's progress (which would not be efficient), or else somehow know when it should send a request to check the user's progress. Otherwise the OpenDSA client would not know that the CodeWorkout exercise has been completed until the user reloads the page.

A solution to this problem is to use the JavaScript `window.postMessage`[6] function.

[6] https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

This function allows for cross-origin communication between two browser frames or windows. For example, this allows for a page displayed in an iframe to send a message to the page containing the iframe, or vice versa. In this way, a CodeWorkout exercise being displayed in an iframe within an OpenDSA module could send a message to the OpenDSA module page to let it know when the exercise has been completed, when the user's current score has changed, and possibly other events so that that OpenDSA client could react if necessary.

The use of the `window.postMessage` function in the context of LTI is not unprecedented. Some LMS, such as Canvas[7], support the use of `window.postMessage` to allow a tool provider to request that the iframe it is being displayed in be resized, among other things, though it does not appear to be a part of the LTI standard and it is not well documented. An example of code that can be used by a tool to request to be resized is shown in Listing 4.1. It would be beneficial if the LTI standard would define a set of message types for client-side communication that could be implemented by tool consumers and tool providers, since there seems to be some use for such capabilities.

Listing 4.1: A message being sent by a tool to its parent page to request an iframe resize

```
    parent.postMessage(JSON.stringify({
        subject: 'lti.frameResize',
        height: 1000
      }), '*');
```

4.3.2 Variation in LMS Implementations

The differences between LMS implementations has only seemed to be a problem for OpenDSA in the context of Deep Linking, which was covered in Section 4.2.1. These differences, such as Blackboard's lack of an interface for finalizing assignment settings for individual assign-

[7]Canvas' implementation for LTI postMessage support - https://git.io/fjo0B

ments created through Deep Linking, can increase the effort required by developers of tool providers. Deviations from the LTI standard, such as in the case of Canvas requiring Deep Linking parameters that are supposed to be optional, or silently ignoring certain parameters, can also necessitate more effort from tool developers, especially when error messages given by the LMS aren't specific enough to pinpoint the problem. For example, determining the meaning of one Canvas error message required looking through the Canvas source code to see that the message was being triggered by the omission of a certain field that is listed in the LTI Deep Linking specification as being optional.

4.3.3 Sharing Analytics Data

LTI is a major step in solving the problem of securely integrating educational systems. It allows systems like LMS or OpenDSA to include third-party content and have it appear and behave as though that content is a native part of the platform. However, one issue that still remains is the problem of sharing analytics or interaction data among systems that are connected through LTI. This problem is discussed in more detail in the next chapter.

Chapter 5

The Future of the E-Learning Ecosystem

5.1 Background

The use of LMS among educational institutions today is nearly universal [20]. The LTI standard allows instructors to more easily use third-party tools and content in their course by providing some level of integration between the learning tools and the LMS, enabling the sharing of scores and basic contextual information. LTI can also be used to integrate learning tools with platforms besides LMS, or even with other learning tools. OpenDSA serves as an example of this with its use of LTI to enable integration of OpenDSA content with various LMS, and potentially other LTI tool consumers, and to enable the integration of content from other learning tools, such as CodeWorkout, into OpenDSA.

5.2 Problems, Goals, and Issues

From the perspective of OpenDSA, it would be beneficial to be able to integrate external content and get the same benefits as if it were internal content. As it stands, OpenDSA is able to use LTI to integrate content from third-party tools and receive basic score information, but it is not able to collect the more detailed interaction data that those tools might capture. For example, OpenDSA knows what scores students get on CodeWorkout exercises used in an OpenDSA module, but more detailed information such as the amount of time students spend on each exercise attempt is kept siloed within the CodeWorkout database. This can make it difficult to get a complete picture of students' performance, and consequently make it difficult for instructors and researchers to perform analysis to explore what might be impacting students' performance. This problem will only get worse as the amount of external content that OpenDSA uses increases.

From a broader perspective, multiple third-party tools may be used in a course, with the LMS and various learning tools each potentially generating a large amount of rich interaction data. It would be beneficial if the analytics data generated by the various educational systems used in a course could be shared in a standard format so that instructors and researchers can get a more complete understanding of student performance, and potentially gain insights into the learning process.

5.2.1 Goals

From a very high level, the goal is to be able to share and aggregate analytics data that is generated by various educational systems. Some more concrete goals are as follows:

- For OpenDSA, the goal is to be able to aggregate the interaction data that OpenDSA generates with the interaction data generated by CodeWorkout content that OpenDSA uses, and by content from any other LTI tools that OpenDSA might use.

- A broader goal is to be able to aggregate all of the interaction data that is generated by the various tools used in a given course, and at a given university.

5.2.2 Issues

A summary of some of the main issues that must be addressed to achieve the above goals are listed below. These issues are discussed in more detail in the rest of the chapter.

- Various educational systems must agree to share data with each other.
- Various educational systems must agree on a standard data format that is capable of describing the variety of interactions supported by various educational systems.
- There needs to be a system that stores and aggregates the data from various educational systems.
- There needs to be a way to tie together entities such as users and courses across systems so that data from various systems can be aggregated.
- There needs to be an architecture that describes how the various systems are connected.

Ethical and Privacy Concerns

There have been several papers that have discussed the ethical and privacy issues surrounding learning analytics [44, 49]. Some examples of issues that should be considered are the level of transparency in how students' data is used, the amount of control that students have over how data on them is collected and used, whether data collected from one course should be made available to the teaching staff of another course, etc. These issues are perhaps even more important when student data is being shared among different educational systems.

5.2.3 Example Use Cases

The ability to share standardized analytics data among educational systems opens up a number of possibilities. With a standardized data format, tools can be created that act on that data. Dashboards and reporting tools can be created to provide instructors with an overview of students' performance across multiple systems, or a detailed view of an individual student's performance. Dashboards and reports could also be made available to students to allow them to monitor their own performance. Automated alert systems can be created that inform instructors when a student is falling behind, or even predict when a student is in danger of falling behind [43]. Recommender systems can be implemented to provide personalized suggestions of relevant resources to students to help them close gaps in their knowledge [18]. Leaderboard systems can be created that grant points to students based on their performance and interactions with the various systems used in the course [21]. Researchers could be allowed access to the data to perform custom analyses.

While versions of the methods and tools described above already exist, they tend to act on non-standard data formats. With a standard data format, such tools can have a much wider impact since they would be able to act on data from a larger number and variety of educational systems.

5.3 Learning Analytics Specifications

In order to enable the interoperability of analytics data between educational systems, the systems need to agree on a shared data format. There are currently two prominent specifications for detailed learning analytics data: xAPI and Caliper Analytics. This section describes these two specifications and discusses their similarities, differences, and levels of support from the community.

5.3.1 xAPI

The Experience API, or xAPI, is a “technical specification that aims to facilitate the documentation and communication of learning experiences. It specifies a structure to describe learning experiences and defines how these descriptions can be exchanged electronically.” [31] xAPI is supported by the Advanced Distributed Learning (ADL) Initiative, a division of the US Department of Defence. The specification is open source and licensed under the permissive Apache 2.0 license. Version 1.0 of the xAPI specification was released in October 2013, while the latest version (1.03) was released in September 2016. The next version of the xAPI specification has not been planned [31].

A core component of xAPI is the `statement` object. These statements capture events in the form of actor-verb-object, where the statement describes an actor performing an action on some object. Statements can also include information on the context of the event, the result of the event, and extension properties. These statements are generated by what xAPI calls a *Learning Record Provider (LRP)* when an learner/actor has a learning experience, and are represented using JSON. The LRP sends the xAPI statements to a *Learning Record Store (LRS)*, which stores the statements and allows for them to later be retrieved by a *Learning Record Consumer (LRC)*.

The structure of xAPI statements is strictly defined by the xAPI specification, but the actual content of statements is flexible. In fact, the xAPI specification does not itself define any verbs, actors, objects, contexts, or extension properties, but rather intends that communities will define their own vocabularies to fit their needs, with each verb, activity, object, etc. being assigned its own unique identifier. The term *profile* is used in the xAPI community to refer to a set of rules and vocabularies intended to address a particular use case. For example, the cmi5 project[1] is a community created xAPI profile that defines rules and a standard xAPI vocabulary for use with traditional LMS.

Listing 5.1: An example xAPI statement [38]

```
{
  "actor": {
    "name": "Sally Glider",
    "mbox": "mailto:sally@example.com"
  },
  "verb": {
    "id": "http://adlnet.gov/expapi/verbs/experienced",
    "display": {"en-US": "experienced"}
  },
  "object": {
    "id": "http://example.com/activities/solo-hang-gliding",
      "definition": {
        "type": "http://adlnet.gov/expapi/activities/course",
        "name": { "en-US": "Solo Hang Gliding" }
      }
  },
  "context": {
    "instructor": {
      "name": "Irene Instructor",
      "mbox": "mailto:irene@example.com"
    },
    "contextActivities":{
      "parent": { "id": "http://example.com/activities/hang-
         gliding-class-a" },
      "grouping": { "id": "http://example.com/activities/hang
         -gliding-school" }
    }
  },
  "timestamp": "2012-07-05T18:30:32.360Z"
}
```

The openness and flexibility of xAPI has been described as both one of its main advantages and disadvantages [1, 2]. While the flexibility can enable xAPI adopters to adapt

[1]https://github.com/AICC/CMI-5_Spec_Current

to new and changing use cases rather quickly, it also raises the risk of different organizations defining conflicting vocabularies with different identifiers being used to refer to the same concept. These conflicting vocabularies can of course cause issues when attempting to aggregate data from different systems. In order to mitigate this risk, xAPI encourages communities to publish their vocabularies and share best practices. Developers have also created the xAPI registry[2] to provide a place to find vocabularies that have already been defined. In addition, there is an IEEE working group working on standardizing the xAPI information model format and communication protocol [32].

Besides the structure of statements, the xAPI specification also thoroughly defines how statements are transferred between a Learning Record Provider, a Learning Record Store, and a Learning Record Consumer. For example, the specification defines a minimum set of HTTP endpoints that an LRS must implement, including how the LRS should respond in different situations, though the LRS may support additional endpoints not in the specification. The specification also dictates that an LRS should validate the syntax of xAPI statements it receives, but that it should not validate semantics. It is also specified that an LRS should support at least one of several authentication methods, including OAuth 1.0 and HTTP Basic Authentication. To help ensure compliance with the specification, the ADL Initiative provides an automated test suite that evaluates whether an LRS correctly implements mandatory requirements[3]. There are also xAPI client libraries available[4] for several languages, including JavaScript, Java, Python, PHP, and .NET.

[2]https://xapi.com/registry/
[3]https://lrstest.adlnet.gov/
[4]https://xapi.com/libraries/

Support for xAPI

As of July 2019, there are currently around 200 recognized products and vendors that have adopted or are in the process of adopting xAPI [37]. Many training and educational systems and platforms used by businesses have adopted xAPI. It is also a US Department of Defense (DoD) policy that learning resources used by organizational entities within the DoD should support xAPI [30].

The major LMS used in higher education provide some level of support, but it does not appear overly robust in some cases. Moodle does not natively support xAPI, but there are plugins that implement support for xAPI features[5,6]. Sakai can be configured to support logging xAPI statements to an LRS[7]. The xAPI list of adopters indicates that Blackboard is in the process of implementing support for xAPI, but we were unable to find any information in Blackboard's documentation. There is a third-party plugin for Blackboard that adds support for logging some xAPI statements to an LRS[8]. The Canvas documentation indicates that there is limited support for xAPI[9].

5.3.2 Caliper

Caliper Analytics, or simply Caliper, is a standard that "provides a structured approach to describing, collecting and exchanging learning activity data", and defines an API for "transforming and exchanging event data from different applications to specific target endpoints for storage, analysis, and use" [11]. Caliper is supported and published by the IMS Global Learning Consortium. Caliper is published under a license that allows for the specifica-

[5]https://moodle.org/plugins/logstore_xapi
[6]https://moodle.org/plugins/mod_tincanlaunch
[7]https://github.com/Apereo-Learning-Analytics-Initiative/SakaiXAPI-Provider
[8]https://github.com/jiscdev/blackboard-xapi-plugin/wiki
[9]https://canvas.instructure.com/doc/api/file.xapi.html

tion to be distributed freely, but forbids modifications not authorized by IMS. Version 1.0 of the Caliper specification was released in October 2015, while the current version of the specification, Version 1.1, was released in January 2018.

The Caliper specification includes an information model that "defines a set of concepts, rules, and relationships for describing learning activities" [11]. The different activity domains modeled by Caliper are described in *Metric Profiles* that each include one or more *Event* types. Each event type defines a vocabulary of actions that might be performed by different actors (e.g. learners and instructors). Each event type also defines a set of properties for that event type, many of them optional, though additional extension properties not defined by the specification can be added. One of the optional event properties is *LtiSession*, which can be used to capture contextual information for events that occur in the context of an LTI tool launch, including the parameters from the LTI launch request. Events are captured in an actor-action-object form, where an event describes an actor performing an action on an object.

As an example, the Caliper specification includes the *Assessment Profile* which can be used to capture information about a person interacting with assessments. The Assessment Profile supports four event types, with each of those events supporting different actions. One of the event types, the *AssessmentItemEvent*, models a learner's interaction with an individual assessment item, such as a question on a quiz. Actions supported by the AssessmentItemEvent are `Started`, `Skipped`, and `Completed`. The Caliper specification also defines the *Basic Profile* which provides a generic Event that can be used to describe actions that have not yet been modeled by Caliper. *Serious games* are an example of a use case that is not sufficiently modeled by the current set of Caliper Metric Profiles [46].

Caliper uses the JSON Linked Data (JSON-LD) format. The JSON-LD format is similar to normal JSON, and can be treated like normal JSON, but it explicitly allows for links to other objects (e.g. a URL) to be included in objects. Those links can then be dereferenced to retrieve the linked objects. JSON-LD also includes a type system, where a given object can have one or more types specified. A JSON-LD document includes a *context* property, which defines the vocabulary (i.e. the object types and properties) that is used in that document. Listing 5.2 shows a sample Caliper ForumEvent. Note that many properties, such as `actor`, are simply URLs. Those URLs could be dereferenced to retrieve a concrete representation of the objects. It would also be valid if the objects themselves were directly included in the event instead of just the URLs.

Listing 5.2: An example Caliper event capturing a user subscribing to a forum [11]

```
{
  "@context": "http://purl.imsglobal.org/ctx/caliper/v1p1",
  "id": "urn:uuid:a2f41f9c-d57d-4400-b3fe-716b9026334e",
  "type": "ForumEvent",
  "actor": "https://example.edu/users/554433",
  "action": "Subscribed",
  "object": "https://example.edu/terms/201801/courses/7/
     sections/1/forums/1",
  "eventTime": "2018-11-15T10:16:00.000Z",
  "edApp": "https://example.edu/forums",
  "group": "https://example.edu/terms/201801/courses/7/
     sections/1",
  "membership": "https://example.edu/terms/201801/courses/7/
     sections/1/rosters/1"
  "session": "https://example.edu/sessions/1
     f6442a482de72ea6ad134943812bff564a76259"
}
```

In addition to the information model, the Caliper specification also defines the *Sensor API*, which provides an interface for instrumented applications to communicate with data consumers. IMS provides implementations of the Sensor API for several popular languages[10], including Java, Python, JavaScript, PHP, Ruby, and .NET. The Caliper specification refers to the application collecting and transmitting the data as the *Sensor*, while the recipient of

[10] https://www.imsglobal.org/caliper-analytics-11-public-repos-sensor-apis

the data is the *Endpoint*. It is specified that OAuth 2.0 should be used for authentication. When the endpoint receives the data, the data is persisted to a data store (i.e. an LRS). Version 1.0 of the Caliper specification [6] states that a standards based LRS is outside of Caliper's initial scope, and the latest version (1.1) does not indicate that this has changed. The current specification also defines how an Endpoint should respond in a limited number of scenarios, but it does not describe any APIs for retrieving data from an Endpoint.

Support for Caliper

As of July 2019, the number of products that have adopted Caliper appears fairly limited. There are are total of 11 products and organizations that are certified by IMS as supporting Caliper [13]. Included among these is Blackboard and Canvas. Moodle is not listed among those certified, however there is a Moodle plugin that can be installed to add support for logging Caliper events[11]. It is possible that there are other products that have adopted Caliper but have not been certified by IMS since only affiliate and contributing members of IMS can access the Caliper certification service.

5.3.3 Discussion

Though there are some significant differences between xAPI and Caliper, there are also some significant similarities. For example, both specifications use an actor-verb-object form to capture events (though Caliper uses the term *action* instead of verb). In 2016, the organization behind xAPI, the ADL Initiative, joined IMS as a contributing member and announced that IMS and the ADL Initiative were exploring potential areas of alignment for the two specifications [29]. It appears the possibility of alignment is still being explored, as

[11]https://moodle.org/plugins/logstore_caliper

a presentation on the progress of the alignment efforts was given at the IMS Europe Summit in November 2018 [12]. Some suggest that it is likely that both specifications can co-exist, with each specification serving different niches [28]. For example, it has been suggested that Caliper could be a better fit for the more defined parts of the learning ecosystem, including LMS and large applications that want to interact with an LMS, while xAPI may be more suitable in the professional learning context, and for recording events that occur outside the LMS [28].

Another difference between the two specifications is the level of control that supporting organizations have over the specifications. xAPI is an open source specification and published under a license that allows for the specification to be freely modified and republished. In addition, anyone is free to attend meetings of the xAPI specification working group. In contrast, the Caliper specification is tightly controlled by IMS and modifications not authorized by IMS are forbidden. The Caliper specification working group is open only to organizations that are contributing members of IMS, and membership fees can range from $1,500 to $55,000 depending on the size of the organization[12]. Many major LMS organizations are contributing members of IMS, including Moodle, Instructure (the makers of Canvas), Blackboard, Sakai, and Desire2Learn (the makers of Brightspace).

Since Caliper is strictly controlled by IMS, it may be slower to adapt to new and changing uses cases than the more open xAPI community. If Caliper is too slow to adapt to a use case, this could lead to some developing their own extensions to Caliper, which could lead to conflicts if the Caliper specification eventually expands to cover that use case. However, this strict control over the specification can also help ensure consistency in the information model. On the other hand, xAPI's flexibility and openness can potentially allow communities to more quickly adapt, but also comes with an increased risk of different organizations using

[12]https://www.imsglobal.org/imsmembership.html

conflicting vocabularies.

In terms of the data format, both Caliper and xAPI use a JSON format. Caliper uses the JSON-LD data format which includes a type system and links to other objects, while xAPI uses plain JSON. Some have suggested that xAPI should adopt JSON-LD due to the benefits it provides over standard JSON [1].

The xAPI specification more thoroughly describes the communication between the entity that generates the data and the entity that stores the data than Caliper does. For example, xAPI describes APIs for retrieving data from an LRS once it has been stored, while Caliper does not. xAPI also provides an automated test suite to ensure that LRS comply with the specification. Caliper does not currently provide any sort of testing or certification for Caliper endpoints (e.g. LRS), though IMS anticipates that this will be offered in a future release of the Caliper specification [10].

On the other hand, Caliper does provide automated tests for Caliper Sensors [9], with Caliper Sensors being equivalent to xAPI Learning Record Providers (LRP). The Caliper Sensor tests evaluate whether the data a sensor sends conforms to the different Metric Profiles defined by the Caliper specification. xAPI does not provide tests for LRP, however this could be because the xAPI specification only defines the basic structure of xAPI statements and leaves the vocabulary up to the relevant communities to define.

Finally, xAPI currently seems to be supported by significantly more products than Caliper. This could be due to the fact that Caliper is newer than xAPI. It might also be that the gap isn't as wide as it seems since certification for Caliper is only available to IMS members, but non-IMS members might still have adopted Caliper, though this is only speculatory.

	xAPI	**Caliper**
Supporting Organization	Advanced Distributed Learning (ADL)	IMS Global Learning Consortium
Basic Event Form	actor-verb-object	actor-action-object
Freedom to Modify/Republish	Freely modifiable and republishable	Freely republishable. Modifications must be approved by IMS.
Working Group Meetings	Open to public	IMS members only
Event structure	Defined by specification	Defined by specification
Event vocabulary	Defined by communities	Defined by specification
LRS/Endpoint Behavior	Thoroughly defined	Sparsely defined in current version
Testing	LRS/Endpoint testing. No tests for data providers.	Testing for Sensors (data providers). No LRS/Endpoint tests.

Table 5.1: A summary of the comparison between xAPI and Caliper

5.4 Learning Record Stores

Regardless of whether an application supports xAPI, Caliper, or even both, the data that is generated needs to be stored somewhere, and be made available for later retrieval. The xAPI specification defines the concept of a *Learning Record Store (LRS)* for this purpose. The xAPI specification defines an LRS as “a server (i.e. system capable of receiving and processing web requests) that is responsible for receiving, storing, and providing access to Learning Records.” [31] An LRS also serves as a place to aggregate learning data from multiple applications.

5.4.1 Features and Issues

The xAPI website provides an overview aimed at those who are considering building their own LRS that lists some of the core features that should be satisfied by an LRS, and mentions some issues that should be considered [36]. The Advanced Distributed Learning Initiative also published a document to aid those looking to choose an LRS [3], and it includes many features and issues that should be considered. In this section, we discuss some of these LRS features and issues, as well as others.

Authentication and Authorization

As with most applications, an LRS must ensure that a user (or application) is who they say they are (authentication), and that a user can only access resources that they have permission to access (authorization). Authentication can be done in multiple ways, but the main methods specified in either the xAPI or Caliper specifications are:

1. Basic HTTP authentication (i.e. username and password)[13]

2. OAuth 1.0[14]

3. OAuth 2.0[15]

An LRS should grant or restrict access to features and data as appropriate, based on the user's permissions. Accordingly, an LRS must implement some sort of access control system, such as role-based or attribute-based access control. An LRS should also provide an interface to allow administrators to manage users and permissions.

Data Validation

An LRS should ensure that the data it receives conforms to the appropriate specification so as to avoid accepting malformed data. There are a number of open source xAPI validation

[13]HTTP 'Basic' authentication RFC - https://tools.ietf.org/html/rfc7617
[14]OAuth 1.0 RFC - https://tools.ietf.org/html/rfc5849
[15]OAuth 2.0 RFC - https://tools.ietf.org/html/rfc6749

tools available [45]. An LRS might also have safeguards in place to avoid accepting duplicate data.

Data Storage

Once an LRS has validated the data, the data must then be persisted. Given the semi-structured nature of the xAPI and Caliper information models, e.g. the large number of optional fields and the potential for custom/extension fields, a fully relational database may not be appropriate. One could simply store any non-standard data as raw JSON in a plain-text catch-all column, but this would make any queries on that data problematic. Using a NoSQL database, such as a document store[16], is one option since they allow for flexible data schemas, however NoSQL databases also tend to have diminished querying capabilities, which can be a concern when developing analytics and reporting tools. Using PostgreSQL can enable a hybrid approach since it is primarily a relational database, but it also has data types that support storing, indexing, and querying JSON data[17]. In one case a NoSQL database (MongoDB) was used for primary storage, with data being pulled to a secondary relational database (PostgreSQL) for analytics and reporting purposes due to the more robust querying capabilities [1].

For an LRS that may be used for years, and be used to store large amounts of data from multiple applications, the LRS might also want to have a mechanism for archiving older data.

[16]also known as a document-oriented database
[17]https://www.postgresql.org/docs/current/datatype-json.html

Querying

Once the data has been stored, an LRS should also provide mechanisms for that data to be retrieved, e.g. via REST[18] APIs. The Caliper specification does not currently define any APIs for data retrieval, meaning LRS developers have to define their own non-standard APIs, at least for now. For an LRS that supports xAPI, the LRS must, at a minimum, implement the REST interfaces that are defined in the xAPI specification [31]. Some have found that the minimum REST interfaces defined by the xAPI specification are not sufficient for their purposes [1]. Criticisms include the lack of aggregate queries, and the fact that all matching statements are returned in full.

An LRS could also be a good use case for GraphQL[19], which is a query language for APIs that allows for clients to more precisely control what data is returned by the server. For example, a client can request only the specific fields that they need, which can help keep data transfer costs down.

Privacy and Security

Since LRS store sensitive student/learner data, privacy and security issues are of particular concern. As such, steps should be taken to address common risks. The Open Web Application Security Project (OWASP) publishes a list of critical web application security risks that web application developers should be aware of[20]. For example, to avoid the exposure of sensitive data during transmission, an LRS should require that incoming and outgoing traffic is encrypted using the Transport Layer Security (TLS) protocol[21].

[18] REST stands for Representational State Transfer
[19] https://graphql.org/
[20] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[21] TLS protocol - https://tools.ietf.org/html/rfc8446

Location

An LRS can be its own stand-alone system, or it can be built into another system, such as an LMS [3, 36]. Some suggested benefits of a dedicated stand-alone LRS is that it can provide more flexibility in configuration and scalability, easier system upgrades, and better performance and reliability [3].

Data Forwarding

An LRS might support the ability to forward data to other data stores or applications. For example, it could be useful for an LRS to allow an application, such as an analytics tool, to set up something like a webhook[22] so that certain data that the LRS receives is automatically forwarded to the application. This can allow for tools that utilize the data in the LRS to maintain up-to-date copies of relevant data without having to poll the LRS to check for updates. This feature could also be used to forward data from one LRS to another LRS.

One concern that a feature like this raises is that the LRS should be able to gracefully handle transmission failures. For example, if the application that the LRS tries to trasmit data to is down or otherwise unresponsive, the LRS might attempt to resend the data after some time.

Other Features and Issues

Besides core features related to storing data, an LRS might also have built-in analytics tools, such as a customizable data analytics engine and analytics dashboards [3]. An LRS might also have special features to facilitate its use by multiple organizations (multi-tenancy), such as supporting multiple separate data stores. An LRS should also use a scalable architecture

[22] `https://sendgrid.com/blog/whats-webhook/`

to allow the system to expand as the number of users and the amount of traffic increases [3].

5.4.2 Examples of LRS

In this section we present a sample of currently available LRS, including open source and closed source LRS.

Open Source

Learning Locker[23] is an LRS for xAPI that is open source via the GNU General Public License version 3.0. Learning Locker is a Node.js application and uses MongoDB as its database. Learning Locker also includes customizable reports and dashboards, statement forwarding, multi-tenancy, and other features. Learning Locker provides multiple endpoints for querying data, including aggregate queries.

Apereo OpenLRW[24] is an LRS for both xAPI and Caliper that is open source via the Educational Community License version 2.0. OpenLRW is written in Java and uses MongoDB as its database. OpenLRW provides multiple endpoints for querying data, including aggregate queries.

Callisto[25] an LRS for Caliper that is open source via the Apache License 2.0. Callisto is a Ruby on Rails application and uses PostgreSQL to store Caliper events. Callisto is still a work in progress, though at the time of this writing there have been no commits to the GitHub repository in some months. Callisto provides endpoints for querying data, but this does not include aggregate queries.

[23] https://github.com/learninglocker
[24] https://github.com/apereo-Learning-Analytics-Initiative/OpenLRW
[25] https://github.com/openedinc/callisto/

Cloud LRS [26] is another LRS that supports both xAPI and Caliper, and is open source under a license that allows for it to be freely used and modified for educational, research, and not-for-profit purposes. Cloud LRS is a Node.js application and uses PostgreSQL as its database. Cloud LRS appears to have a limited REST API at this time, supporting saving or retrieving only a single Caliper or xAPI statement at a time.

Closed Source

There are a number of closed source LRS available. However, based on marketing materials, the companies that develop these LRS seem to be focused on the needs of businesses and professional organizations, rather than academic institutions. Below are a couple of examples.

Watershed LRS[27] is an LRS for xAPI. While the core LRS is offered for free, more advanced versions with built-in analytics start at $1,600 per month. The features of Watershed LRS include statement forwarding, analytics dashboards, a custom report builder, and an aggregation API.

Rustici LRS[28] is another LRS for xAPI. Pricing information is not listed on Rustici Software's website. The features of Rustici LRS include statement forwarding and multi-tenancy.

5.4.3 Concerns for Data Providers

The following are several issues that should be considered when developing an application that will send data to an LRS.

[26] https://github.com/ets-berkeley-edu/cloud-lrs
[27] https://www.watershedlrs.com
[28] https://rusticisoftware.com/products/rustici-lrs/

When an existing application adopts a learning analytics data specification, developers need to determine how to map the data that the application collects to whichever standard data format they adopt. When an organization or application first adopts an LRS, it must be decided whether pre-existing analytics data that has been collected should be sent to the LRS. This could be a challenge for some organizations and applications, depending on the amount of pre-existing data.

An application should also consider how often to send data to the LRS. The data could be sent immediately as it is generated, but performance might be improved by sending the data in batches.

Applications also need to be able to gracefully handle cases where there is a failure transmitting data to the LRS, such as if the LRS is down or unresponsive. For example, if there is a transmission failure, a simple solution might be to wait a short period of time and try again. A more robust solution might be to store the data locally after a certain number of failed attempts, then either try again after a longer period time, or wait for manual intervention by an administrator.

5.5 Architectures for Sharing Learning Analytics Data

Thus far we have discussed several different components, including the LMS, learning tools, analytics tools, and the LRS, but there also needs to be some sort of architectural framework that defines how all these different components are connected. In this section we present motivating examples, discuss several challenges involved with such an architecture, and discuss proposed architectures.

5.5.1 Motivating Examples

Since OpenDSA textbooks can contain CodeWorkout exercises, OpenDSA researchers would like to be able to access the interaction data that is collected by CodeWorkout, and potentially other LTI tool providers that OpenDSA might include content from. This would allow OpenDSA researchers and developers to gain a more complete understanding of how OpenDSA textbooks are being used by students, and perhaps how these interactions affect student performance. A suitable architecture should facilitate this.

To give a more complex example, CS 3114 at Virginia Tech is a junior-level Computer Science course on data structures and algorithms. A number of different systems are used in CS 3114:

1. An LMS (Canvas)
2. OpenDSA (and consequently, CodeWorkout, and potentially other LTI tool providers), linked with the LMS through LTI.
3. Web-CAT, an automated grading system for programming assignments [23]. Web-CAT is linked with the LMS through LTI.
4. A plugin for the Eclipse IDE that captures data on various events as students work on programming assignments, with the data being sent to the Web-CAT server [34, 40].
5. Piazza [29], a question and answer platform where students can ask questions and have discussions. Piazza is linked with the LMS through LTI.

A suitable architecture should enable the collection and aggregation of data from all of these systems.

[29] https://piazza.com/

5.5.2 Challenges

A major problem that must be addressed is that there needs to be a way to tie together or align certain entities (e.g. users, courses) across systems. For example, since different systems might use different identifiers to refer to user accounts that actually represent the same person, these systems either need to agree to use the same ID, or there needs to be a mechanism to tie together user accounts with different IDs that represent the same person. For example, the Learning Locker LRS provides a "Personas" feature, which allows for different user accounts that represent the same person to be grouped together, though it still must first somehow be determined that two different user accounts in fact represent the same person.

It has been suggested that that since many LMS use *Universally Unique Identifiers* (UUIDs) to identify users, and since these UUIDs are included in LTI launch requests from these LMS, that these UUIDs could be reused by LTI tool providers to associate data for users across systems [24]. However, not all prominent LMS currently use UUIDs to identify users. For example, Moodle does not. Though if all LMS did agree to use UUIDs, then this could be a valid approach for many cases. One caveat is that this approach does rely on an LMS acting as a central hub for all software tools used in a course, which may not always be the case.

An approach that could handle cases where not all tools are accessed through the LMS could be to use an identifier defined at a level higher than the LMS, such as the ID of the student in a university's Student Information System (SIS), or the student's email address. Both of these pieces of information can be passed in LTI launch requests using standard parameters already defined by the LTI specification [5], and are already passed by many LMS (including Canvas, Moodle, and Blackboard), but they could also be used by systems that aren't accessed from within an LMS. However, use of email addresses could raise privacy concerns.

Another concern is defining how a given application should know where to send the analytics data that it collects, and how to aggregate data collected from various, possibly disconnected, tools used in a course. For an LMS, and for tools which are used outside of the context of the LMS, this could be a configuration option within the LMS or tool itself. For tools used in the context of an LMS, this could be added to existing LMS workflows for configuring LTI tools, such that when an LTI tool is configured in the LMS, any configuration required to allow that tool to send data to the desired LRS (e.g. specifying a key and secret) could also be done. The LMS could then include the URL of the LRS endpoint in the LTI launch request to the tool provider.

5.5.3 Architectures

The *Total Learning Architecture* (TLA), sponsored by the Advanced Distributed Learning Initiative, is a collection of specifications that defines "the data models, interface specifications, centralized software services, communication protocols, and application methods necessary to support interoperability of learning and development data across the talent management enterprise" [50]. In the TLA, interaction data from various activity providers is collected in a central LRS. The data from the LRS is used for various purposes, including building learner profiles, and recommending learning activities.

Flanagan and Ogata propose another architecture that also includes measures to protect students' privacy [24, 25]. In this architecture, all the learning tools used in a course are linked to an LMS through LTI. Similar to the TLA, each learning tool sends the event data it collects directly to a central LRS. The data from the LRS is then used by an analytics tool that is linked to the LMS via LTI in order to provide insights to students and instructors. The privacy of students is protected by using the UUIDs from the LMS, which are taken from the LTI launch request and the LMS API, to identify users and courses in the LRS. This reduces the amount of personally identifiable information that the LRS stores, providing some measure of anonymity. An LMS plugin is used to translate the UUIDs back to the actual user and course information when displaying analysis results to instructors and students.

The architectures described above are not entirely adequate for OpenDSA's case, or at least they do not adequately describe how a case such as OpenDSA's should be handled. OpenDSA would like to aggregate its own interaction data with interaction data collected by CodeWorkout (and any other LTI tool providers that OpenDSA might use content from). OpenDSA's interaction data also needs to be aggregated with data from other tools that might be used in a course besides OpenDSA so that researchers and analytics tools can see a holistic view of students' activities for that course. If we could have a single monolithic LRS that every learning application that could possibly used in any course at any university would send its data to, and if OpenDSA had access to the data in this LRS, then the problem of aggregating the data would essentially be solved. Such a thing is untenable since it would require the LRS to be capable of handling all the data from potentially hundreds or thousands of different applications used across a large number of universities, and it would require all of these universities to agree to store all of their data in a single place. Instead, suppose that the interaction data generated by all the tools used in a course, including OpenDSA and CodeWorkout, would be stored in an LRS managed by the organization offering the course. This alone would not be sufficient from OpenDSA's perspective. In order for OpenDSA to access the interaction data from CodeWorkout exercises used in an OpenDSA book, OpenDSA would require access to the LRS managed by the organization where the course using the OpenDSA book was offered, which may not be feasible in most cases. Even if OpenDSA was granted access to LRS managed by outside organizations in order to retrieve CodeWorkout data, all the data would still be split up among multiple LRS that are each managed by different organizations, which is not ideal.

A solution could be for OpenDSA and CodeWorkout to send their data to an LRS that is managed by OpenDSA, then that LRS could forward data to other LRS or other applications based on forwarding rules that would be set up. This could result in a sort of network of LRS, such that each LRS stores data that is relevant to the organization that the LRS belongs to, and if necessary, forwards that data to other relevant parties. So, an LRS managed by OpenDSA would store analytics data that is generated by OpenDSA, CodeWorkout, and any other LTI tool providers that OpenDSA might use. This would allow OpenDSA to retain access to the interaction data collected by LTI tool providers that OpenDSA uses content from. Forwarding rules could be set up in that LRS such that data generated by a user from a given university would be automatically forwarded to an LRS that is managed by that university. If other tools used in a given course were set up similarly, then the LRS managed by the university could act as a place to aggregate the data from the various tools used in a course. If a tool had no need for its own LRS, then that tool could instead send the data directly to the LRS of relevant organizations, rather than relying on its own LRS to forward the data. Figure 5.1 shows how this might look for the CS 3114 example described in Section 5.5.1.

5.6 Summary and Discussion

In this chapter we provided an overview and discussion of the problem of sharing analytics data among educational systems. In order to share analytics data, educational systems need to agree on a standard data format. The two prominent specifications for learning

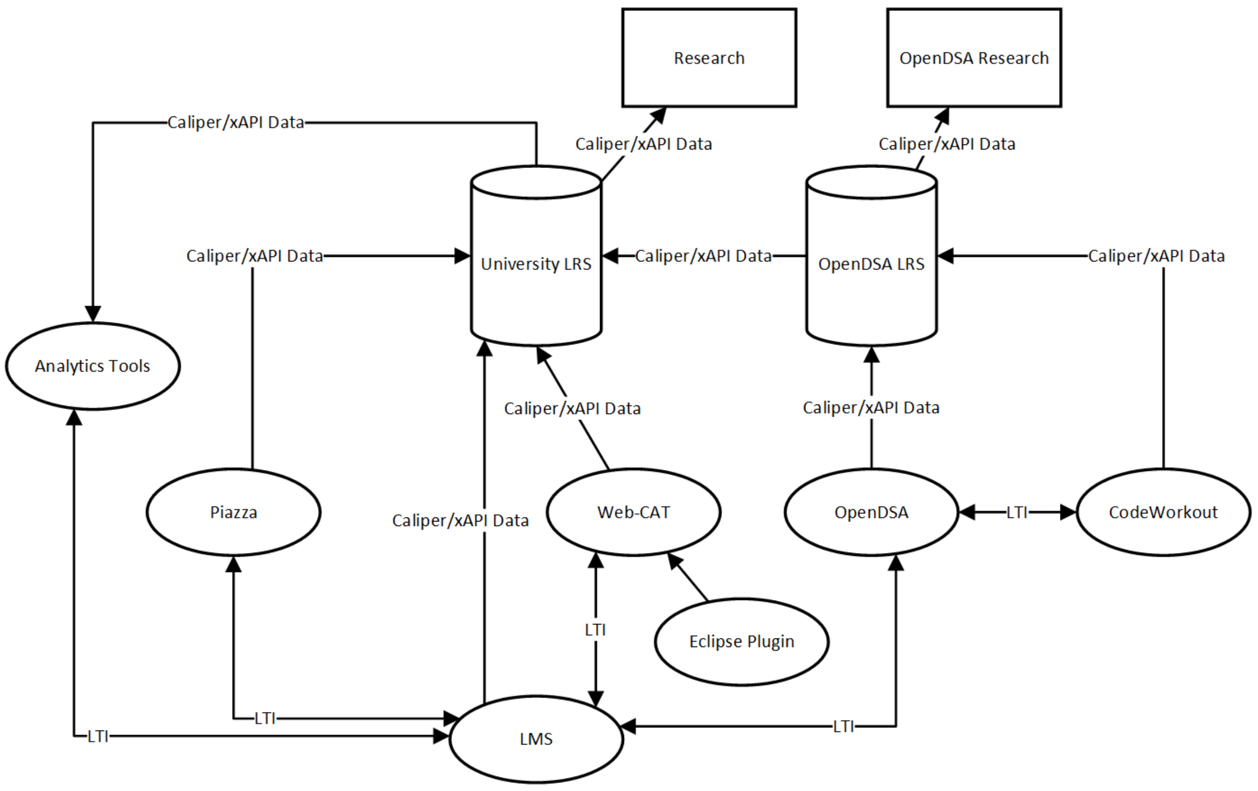


Figure 5.1: How a data sharing architecture for the CS 3114 example might look

data that exist today are xAPI [31] and Caliper [11]. These two specifications use similar event structures, but take different approaches. xAPI takes a more open approach in that it defines the core structure for capturing events, but leaves it up to the community to define the specific vocabularies used to describe different events. Caliper's approach is more controlled in that it defines both the core event structure and the vocabulary for capturing various types of events. In addition, the xAPI specification can be freely modified and redistributed, while modifications to the Caliper specification must be approved by IMS.

Regardless of whether xAPI or Caliper is used, there needs to be some place to store and aggregate the data from various systems. A Learning Record Store (LRS) is a system that is designed to serve such a purpose. Some existing LRS include the ability to forward data that they receive to other systems, such as other LRS.

Besides simply aggregating the data from separate systems, another issue is that there needs to be some way to tie together entities such as users and courses across systems. For example, since different systems might use different identifiers to refer to users that actually represent the same person, these systems either need to agree to use the same identifier, or there needs to be some way to group together user identifiers that refer to the same person. One approach that has been suggested is for learning tools to re-use UUIDs that some LMS use to identify users and courses [24], but this relies on all of the learning tools used in a course being accessed through the LMS, and relies on the LMS using UUIDs to identify users and courses. Other possible approaches might be to use the student's ID from the university's Student Information System (SIS), or the student's email. Both the email and the student's ID from the SIS have dedicated parameters in the LTI standard [5] and are already included in LTI launch requests by many LMS, but do not rely on a tool being accessed through the LMS.

There also needs to be an overlying architecture that describes how various systems (e.g. the LMS, learning tools, analytics tools, LRS) are connected. A broad goal is to be able to aggregate the interaction data from all of the various tools used in a course or at a university. OpenDSA's goal is to be able to aggregate its own interaction data with the interaction data generated by external content that is included in OpenDSA books, such as CodeWorkout exercises. Existing architectures that have been proposed [24, 50] for collecting interaction data from various systems describe the user of a central LRS, which would be adequate for allowing the data from various tools used in a course to be aggregated, but would not adequately satisfy OpenDSA's needs. We propose an architecture that would address both the broader goal of aggregating the data for all the tools used in a course or at a university, and the goal of allowing OpenDSA to aggregate and access the data from the tools that it uses content from. The architecture does this through the use of an LRS that would be managed by OpenDSA, and which could be set up to forward data to other LRS (e.g. the LRS of the university where a given student is enrolled) as appropriate.

In order to make the sharing of analytics data between tools and platforms a reality, there also needs to be strong support from the community. Namely, more LMS and learning tools need to implement sufficient support for Caliper and/or xAPI. One challenge is that the landscape for these two specifications is still changing and solidifying (particularly for Caliper), and it seems uncertain at this time whether changes will be made to align the xAPI and Caliper specifications, or whether these two specifications will continue to exist as distinct entities.

Another issue is that there currently seems to be a lack of robust LRS available for Caliper. Current LRS that support Caliper lack some important features, in particular they seem to lack data forwarding abilities, and most Caliper LRS lack rich querying interfaces. This is a significant issue, since the lack of robust LRS can hinder the adoption of Caliper, and hinder the development of analytics tools that might make use of Caliper data. This could potentially be attributed to the fact that the Caliper specification does not yet provide many details on how a Caliper LRS is expected to behave.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis we explored the problem of connecting and integrating educational systems by discussing interoperability problems, and designing, implementing, and discussing interoperability solutions in the context of the OpenDSA eTextbook system. Originally, third-party LTI content was integrated into OpenDSA textbooks by using the LMS API to set up separate assignments in the LMS, resulting in the LMS sending LTI launch requests directly to these third-party tool providers. However, this left OpenDSA unaware of when students accessed and completed third-party exercises, and unaware of what scores they received. We resolved this issue by turning OpenDSA into an LTI tool consumer, enabling OpenDSA to send LTI launch requests directly to the third-party tools that OpenDSA uses, and receive scores back from those tools. We also added support for having multiple graded exercises in a single LMS assignment.

Next, we took steps to improve the reusability of OpenDSA content by increasing the levels of granularity at which OpenDSA content can be accessed, and by making it easier to add OpenDSA content to various LMS. It used to be that OpenDSA content could only be accessed in the context of a textbook. We improved upon a prototype implementation for allowing instructors to add individual OpenDSA exercises and visualizations to their courses in the Canvas LMS by developing a more robust and permanent implementation.

The original prototype also used an extension to the LTI standard that was only supported by the Canvas LMS in order to allow instructors to add OpenDSA content to their LMS course through a GUI. We improved upon this by implementing support for the LTI Deep Linking standard in OpenDSA, thus allowing OpenDSA to provide a GUI for adding content in Canvas, Moodle, Blackboard Learn, and potentially other LMS. We also implemented functionality to allow for individual OpenDSA modules to be served over LTI, outside of the context of an OpenDSA textbook. In addition, we discussed issues and limitations we ran into related to LTI, including variations and non-conformance in LMS support for the Deep Linking standard, and we proposed a solution for a use case that the LTI standard does not currently support.

Next we described and discussed the problem of sharing analytics data among educational systems. We described the two prominent learning analytics specifications, xAPI and Caliper Analytics, and discussed the similarities and differences between them. We also described the concept of a Learning Record Store, as defined by the xAPI specification, as a place to store learning data generated by learning tools. We then discussed architectures for sharing learning data and mention two existing architectures. We described why these architectures do not adequately specify how to handle OpenDSA's use case, which requires that OpenDSA retain access to the learning data generated by the third-party tools OpenDSA uses content from, and we proposed an architecture that would work for OpenDSA's use case.

We also covered improvements that were made to the book configuration infrastructure of the OpenDSA system, and compared the OpenDSA book configuration system to the configuration systems of the zyBooks and Runestone Interactive eTextbook platforms. We developed an improved OpenDSA book configuration format that allows for default settings to be set for different exercise and visualization types, but also allows for the default settings to be overridden for specific exercises and visualizations. This new configuration format does not require an exercise, visualization, or module section to be explicitly listed in a configuration file unless the default settings for that exercise, visualization, or module section are being overridden, which can reduce the effort required to manually create and maintain configuration files. This new configuration format also results in book configuration files that are significantly smaller than the original format while still capturing the same information. We also developed a graphical web interface for creating and modifying OpenDSA book configurations. This interface reduces the barrier to entry for new OpenDSA instructors by making it easier for non-expert users to create and modify their own OpenDSA book configurations, without those instructors having to be familiar with the OpenDSA book configuration format.

6.2 Future Work

Building an ecosystem of tools and platforms that share standardized learning analytics data is something that requires strong support from the educational software community. A good first step for OpenDSA towards achieving the architecture for sharing and aggregating learning data described in Section 5.5.3 would be to implement support for generating and emitting Caliper and/or xAPI events in OpenDSA and CodeWorkout, and adopting an LRS to send that event data to. Since OpenDSA and CodeWorkout are both developed at Virginia Tech and already have some level of collaboration, this is something that could be achievable in the relatively near future. This could serve as a partial evaluation of the proposed architecture, and also potentially serve as an exemplar that others that develop educational tools could follow or draw inspiration from.

In April 2019 the IMS Global Learning Consortium released version 1.3 of the LTI standard [16], and began strongly encouraging that all LTI adopters upgrade to LTI 1.3 as soon as possible [17]. One major focus of LTI 1.3 is increased data privacy and security. Included in this focus is a switch from OAuth 1 to a new and improved security model using OpenID Connect, JSON Web Tokens (JWT), and OAuth 2. LTI 1.3 also renames many of the LTI launch parameters from LTI 1.1, and changes terminology from "tool provider" and "tool consumer" to "tool" and "platform" respectively. Tools like OpenDSA and CodeWorkout may want to start planning on upgrading to LTI 1.3. The upgrade for OpenDSA will be more complicated than for CodeWorkout, since OpenDSA acts as both a tool provider and tool consumer, while CodeWorkout acts only as a tool provider. IMS has also introduced a set of services that they call *LTI Advantage* [15]. These services currently include Deep Linking, Assignment and Grade Services, and Names and Role Provisioning Services. As described in Section 4.2.1, Deep Linking enables learning tools to provide a user interface for selecting resources to add to their LMS course. Assignment and Grade Services allows for the synchronization of grades, progress, and comments from learning tools into an LMS gradebook. Names and Role Provisioning Services enables LMS to securely share course enrollment information with learning tools, allowing learning tools to know who is enrolled in a course even if that person has not yet interacted with the tool.

As mentioned in Section 4.2.1, OpenDSA may be able to take advantage of LMS that support adding links for multiple resources at once through LTI Deep Linking to enable adding the links for an entire OpenDSA book. If OpenDSA was able to do this, then it could potentially eliminate the need for OpenDSA developers to create LMS-specific adapters for course generation for LMS that have functionality for adding multiple resources at once through Deep Linking. The Blackboard Learn LMS currently has such functionality and could serve as a good platform for exploring this.

Finally, as mentioned in Section 4.2.3, the infrastructure developed for serving stand-alone OpenDSA modules allows for the possibility of instructors customizing the settings for modules they add to their courses, but OpenDSA does not currently have a user interface to support this. As such, OpenDSA developers could create a user interface that allows instructors to customize the settings of a stand-alone module when they are adding one to their course, similar to how instructors can customize the settings of stand-alone exercises and visualizations. This customization would include customizing the settings for the exercises and visualizations in the module.

Bibliography

[1] Aneesha Bakharia, Kirsty Kitto, Abelardo Pardo, Dragan Gašević, and Shane Dawson. Recipe for Success: Lessons Learnt from Using xAPI within the Connected Learning Analytics Toolkit. In *Proceedings of the sixth international conference on learning analytics & knowledge*, pages 378–382. ACM, 2016.

[2] Alan Berg, Maren Scheffel, Hendrik Drachsler, Stefaan Ternier, and Marcus Specht. Dutch cooking with xAPI Recipes: The Good, the Bad, and the Consistent. In *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)*, pages 234–236. IEEE, 2016.

[3] Peter Berking. Choosing a Learning Record Store (LRS), Dec 2016.

[4] Daniel Aubrey Breakiron. Evaluating the Integration of Online, Interactive Tutorials into a Data Structures and Algorithms Course. Master's thesis, Virginia Tech, 2013.

[5] IMS Global Learning Consortium. Learning Tools Interoperability v1.1.1 Implementation Guide, Sep 2012. URL https://www.imsglobal.org/specs/ltiv1p1p1/implementation-guide.

[6] IMS Global Learning Consortium. IMS Caliper Analytics Implementation Guide Version 1.0, Oct 2015. URL https://www.imsglobal.org/caliper/caliperv1p0/ims-caliper-analytics-implementation-guide.

[7] IMS Global Learning Consortium. Learning Tools Interoperability Deep Linking, May 2016. URL https://www.imsglobal.org/specs/lticiv1p0/specification.

[8] IMS Global Learning Consortium. Deep Linking Information Model, May 2016. URL http://www.imsglobal.org/specs/lticiv1p0/specification-3.

[9] IMS Global Learning Consortium. IMS Caliper Certification Guide v1.1, Jan 2018. URL https://www.imsglobal.org/sites/default/files/caliper/v1p1/caliper-sensor-cert-guide-v1p1/caliper-sensor-cert-guide-v1p1.html.

[10] IMS Global Learning Consortium. Caliper Analytics v1.1 Introduction, Jan 2018. URL https://www.imsglobal.org/caliper-analytics-v11-introduction.

[11] IMS Global Learning Consortium. Caliper Analytics Specification Version 1.1, Jan 2018. URL https://www.imsglobal.org/sites/default/files/caliper/v1p1/caliper-spec-v1p1/caliper-spec-v1p1.html.

[12] IMS Global Learning Consortium. IMS Europe Summit 2018 Agenda, Nov 2018. URL https://www.imsglobal.org/ims-europe-2018-agenda.

[13] IMS Global Learning Consortium. Product Certifications, Jul 2019. URL https://site.imsglobal.org/certifications.

[14] IMS Global Learning Consortium. IMS Interoperability Standards, 2019. URL https://www.imsglobal.org/specifications.html.

[15] IMS Global Learning Consortium. Learning Tools Interoperability Advantage Implementation Guide 1.3 IMS Final Release, Apr 2019. URL https://www.imsglobal.org/spec/lti/v1p3/impl/.

[16] IMS Global Learning Consortium. Learning Tools Interoperability Core Specification 1.3, Apr 2019. URL https://www.imsglobal.org/spec/lti/v1p3/.

[17] IMS Global Learning Consortium. LTI Security Update, Jul 2019. URL https://www.imsglobal.org/spec/lti/security-update/v1p0.

[18] Rubén González Crespo, Oscar Sanjuán Martínez, Juan Manuel Cueva Lovelle, B. Cristina Pelayo García-Bustelo, José Emilio Labra Gayo, and Patricia Ordoñez De Pablos. Recommendation System based on user interaction data applied to intelligent electronic books. *Computers in Human Behavior*, 27(4):1445–1449, 2011.

[19] Declan Dagger, Alexander O'Connor, Seamus Lawless, Eddie Walsh, and Vincent P. Wade. Service-Oriented E-Learning Platforms: From Monolithic Systems to Flexible Services. *IEEE Internet Computing*, 11(3):28–35, 2007.

[20] Eden Dahlstrom, D. Christopher Brooks, and Jacqueline Bichsel. The Current Ecosystem of Learning Management Systems in Higher Education: Student, Faculty, and IT Perspectives. *EDUCAUSE Center for Analysis and Research*, 2014.

[21] Darina Dicheva, Christo Dichev, Gennady Agre, Galia Angelova, et al. Gamification in Education: A Systematic Mapping Study. *Educational Technology & Society*, 18(3): 75–88, 2015.

[22] Stephen H. Edwards and Krishnan Panamalai Murali. CodeWorkout: Short Programming Exercises with Built-in Data Collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 188–193. ACM, 2017.

[23] Stephen H. Edwards and Manuel A. Perez-Quinones. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, pages 328–328, 2008.

[24] Brendan Flanagan and Hiroaki Ogata. Integration of Learning Analytics Research and Production Systems While Protecting Privacy. In *The 25th International Conference on Computers in Education, Christchurch, New Zealand*, pages 333–338, 2017.

[25] Brendan Flanagan and Hiroaki Ogata. A Learning Analytics Platform Approach to Seamless Learning. In *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 370–373. IEEE, 2018.

[26] Eric Fouh, Ville Karavirta, Daniel A. Breakiron, Sally Hamouda, T. Simin Hall, Thomas L. Naps, and Clifford A. Shaffer. Design and architecture of an interactive eTextbook–The OpenDSA system. *Science of Computer Programming*, 88:22–40, 2014.

[27] Joseph D. Galanek, Dana C. Gierdowski, and D. Christopher Brooks. ECAR Study of Undergraduate Students and Information Technology. *EDUCAUSE Center for Analysis and Research*, Oct 2018.

[28] Dai Griffiths and Tore Hoel. Comparing xAPI and Caliper. *Learning Analytics Review*, 7, 2016.

[29] The Advanced Distributed Learning Initiative. xAPI: A Year In Review, 2016. URL `https://www.adlnet.gov/news/xapi-year-in-review`.

[30] The Advanced Distributed Learning Initiative. DoD Instruction 1322.26 ("Distributed Learning"), Oct 2017. URL `https://adlnet.gov/policy-dodi`.

[31] The Advanced Distributed Learning Initiative. xAPI Specification, Nov 2017. URL `https://github.com/adlnet/xAPI-Spec`.

[32] The Advanced Distributed Learning Initiative. xAPI Standardization, Apr 2018. URL `https://adlnet.gov/xapi-standardization`.

[33] Ville Karavirta and Clifford 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.

[34] Ayaan M. Kazerouni, Stephen H. Edwards, T. Simin Hall, and Clifford A. Shaffer. DevEventTracker: Tracking Development Events to Assess Incremental Development and Procrastination. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 104–109. ACM, 2017.

[35] Kyu Han Koh, Eric Fouh, Mohammed F. Farghally, Hossameldin Shahin, and Clifford A. Shaffer. Experience: Learner Analytics Data Quality for an eTextbook System. *Journal of Data and Information Quality (JDIQ)*, 9(2):10, 2018.

[36] Rustici Software LLC. Building a Learning Record Store, 2019. URL https://xapi.com/building-a-learning-record-store/.

[37] Rustici Software LLC. xAPI Adopters, Jul 2019. URL https://xapi.com/adopters/.

[38] Rustici Software LLC. xAPI Statements 101, 2019. URL https://xapi.com/statements-101/.

[39] Tomasz D. Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. Mastery Grids: An Open Source Social Educational Progress Visualization. In *European conference on technology enhanced learning*, pages 235–248. Springer, 2014.

[40] Joseph Abraham Luke. Continuously Collecting Software Development Event Data As Students Program. Master's thesis, Virginia Tech, 2015.

[41] Hamza Manzoor. Disseminating Learning Tools Interoperability Standards. Master's thesis, Virginia Tech, 2019.

[42] Bradley N. Miller and David L. Ranum. Beyond PDF and ePub: Toward an Interactive Textbook. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 150–155. ACM, 2012.

[43] Behrouz Minaei-Bidgoli, Deborah A. Kashy, Gerd Kortemeyer, and William F. Punch. Predicting Student Performance: an Application of Data Mining Methods with an Educational Web-Based System. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, volume 1, pages T2A–13. IEEE, 2003.

[44] Abelardo Pardo and George Siemens. Ethical and privacy principles for learning analytics. *British Journal of Educational Technology*, 45(3):438–450, 2014.

[45] Thomas Rabelo, Manuel Lama, Juan C. Vidal, and Ricardo Amorim. Comparative study of xAPI validation tools. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2017.

[46] Ángel Serrano-Laguna, Iván Martínez-Ortiz, Jason Haag, Damon Regan, Andy Johnson, and Baltasar Fernández-Manjón. Applying standards to systematize learning analytics in serious games. *Computer Standards & Interfaces*, 50:116–123, 2017.

[47] Clifford A. Shaffer, Ville Karavirta, Ari Korhonen, and Thomas 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.

[48] Hossameldin Latif Shahin. Design and Implementation of OpenDSA Interoperable Infrastructure. Master's thesis, Virginia Tech, 2017.

[49] Sharon Slade and Paul Prinsloo. Learning Analytics: Ethical Issues and Dilemmas. *American Behavioral Scientist*, 57(10):1510–1529, 2013.

[50] Brent Smith, P. Shane Gallagher, Sae Schatz, and Jennifer Vogel-Walcutt. Total Learning Architecture: Moving into the Future. In *Proceedings of the Interservice/Industry Trainging, Simulation, and Education Conference (I/ITSEC)*, 2018.

[51] Frank Vahid, Alex Edgcomb, Susan Lysecky, and Roman L Lysecky. New web-based interactive learning material for digital design. In *123rd ASEE Annual Conference and Exposition*. American Society for Engineering Education, 2016.