Beyond Curation: A Validation and Classification Infrastructure for an Educational Content Catalog

Adeyemi B. Aina

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

Master of Science
in
Computer Science and Applications

Clifford A. Shaffer, Chair
Edward A. Fox
Bob R. Edmison

December 18, 2024
Blacksburg, Virginia

Keywords: Catalog Organization, Metadata Validation, Ontology-Based Classification, Learning Tools Interoperability.

Beyond Curation: A Validation and Classification Infrastructure for an Educational Content Catalog

Adeyemi B. Aina

(ABSTRACT)

To address the challenge of discovering computer science learning resources, the Smart Learning Content (SLC) catalog is designed to simplify access to the growing body of educational content. As part of the Standards, Protocols, and Learning Infrastructure for Computing Education (SPLICE) research community's efforts, the catalog functions as a centralized platform supporting SPLICE's objectives of improving interoperability, enabling comprehensive data collection, and facilitating data analysis in computer science education. The SLC catalog stands out from previous catalogs with an approach that applies an ontology-based content organization and validation services. Additionally, it serves as a platform where educators can contribute, access, and share a wide range of resources—including slideshows, interactive exercises, programming tasks, and Learning Tools Interoperability (LTI)-integrated content from various learning tools. While the primary goal of the catalog is to disseminate high-quality learning materials, its extensive and varied content requires robust organization and validation mechanisms to ensure educators can efficiently locate and utilize resources. The catalog is designed to further support diverse content types, including both standalone resources and content bundles. For one of our key contributors, OpenDSA—an e-textbook system—we have adopted latest LTI 1.3 standard. This implementation enables the catalog to disseminate content in both LTI 1.1 and LTI 1.3 standards, ensuring compatibility. One key improvement in LTI 1.3 is its security features, incorporating robust authentication methods to ensure stronger protection of sensitive student information. This updated standard enables learning tools to meet the evolving demands of digital education, providing educators and learners with more secure, flexible, and effective resources.

Beyond Curation: A Validation and Classification Infrastructure for an Educational Content Catalog

Adeyemi B. Aina

(GENERAL AUDIENCE ABSTRACT)

Finding quality computer science learning materials online can be difficult. The Smart Learning Content catalog helps by collecting resources such as slideshows, exercises, and other learning content in a catalog. This catalog makes it easier for Instructors and students to find what they need quickly. This catalog is organized by set of keywords, so instructors can understand how different topics relate to each other. The catalog also verifies that materials meet high-quality standards. Since instructors rely on a variety of online tools, the catalog supports multiple content types and learning technology standards. In addition, the catalog aims to support multiple Learning Tools Interoperability (LTI) standards, including both older and newer versions. For one of our key contributors, OpenDSA—an e-textbook system—we have adopted the latest LTI 1.3 standard. This implementation enables the catalog to disseminate content in both LTI 1.1 and LTI 1.3 standards, ensuring compatibility. The LTI 1.3 standard enhances security in OpenDSA, protecting student data and maintaining privacy. By aligning with modern standards, the catalog provides pathway for discovering and sharing computer science learning materials.

Dedication

To Jesus and Anthony Olorunsola Aina.

Acknowledgments

These two years have been truly invaluable. I extend my heartfelt gratitude to my committee for their steadfast support and insightful guidance. Special thanks go to my advisor, Dr. Clifford A. Shaffer, for his continuous support, patience, detailed feedback, and his hands-on approach in actively contributing to the project's development. His dedication and willingness to delve into the details was instrumental in shaping this project. I am also grateful to Dr. Edward Fox for his contribution and exceptional Information Retrieval class in Fall 2023, which significantly enhanced my personal growth. Additionally, I thank Dr. Bob Edmison for his contributions and providing his teaching cluster to test various aspects of this project, and Dr. Stephen Edwards for his invaluable contributions. The insightful discussions in the lab with both Dr. Edmison and Dr. Shaffer have greatly impacted this work. My heartfelt thanks go to my lab mates—Arinjoy Basak, Riffat Sabbir Mansur, Alexandra Thompson, and especially Alex Hicks—for their assistance and collaborative spirit, which were essential to the project's success. I am eternally grateful to my family for their unconditional love and support. To my mother, Olabisi Kudirat, and my late father, Anthony Olorunsola Aina, thank you for believing in me. I also thank my siblings—Oluwaleke Aina, Ayodeji Aina, Adetutu Afonja, Adekemi Ogunsemowo, and Adeyinka Aina—for their constant encouragement. Lastly, I acknowledge all who contributed to my academic and personal growth during these two years. Your collective support has been the foundation of this thesis. Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background Information

The Standards, Protocols, and Learning Infrastructure for Computing Education (SPLICE) project is dedicated to advancing education through three primary goals. The first goal is improving interoperability, which involves connecting various quality learning tools to enable instructors to easily incorporate high-quality learning content into their Learning Management Systems (LMSs). For example, integrating learning tools like OpenDSA, a comprehensive e-textbook system for Data Structures and Algorithms, or CodeWorkout for programming exercises in a course, exemplify this interoperability.

The second goal focuses on comprehensive data collection, allowing instructors and researchers to leverage data to improve their course content, identify areas where students struggle, and implement targeted interventions. The final goal involves analyzing the data gathered from these integrated tools. The insights derived from this analysis contribute valuable knowledge to the computer science education community. Through this data-driven approach, SPLICE aims to promote continuous improvement in CS education practices.

By pursuing these three interconnected goals (interoperability, data collection, and data analysis), SPLICE strives to create robust, data-informed, and interoperable systems for

computer science education. This holistic approach benefits instructors and students directly and also contributes to the research and practice of CS education. Despite the advances made by the SPLICE project to improve interoperability, data collection, and analysis, instructors and researchers still face challenges in accessing a growing body of educational content in computer science. The absence of a centralized repository makes it difficult to locate high-quality learning materials, and the process of integrating various learning tools into a Learning Management System (LMS) is often time-consuming and complex in the setup procedures.

In response to these needs, the digital education research group at Virginia Tech developed the Live Catalog as an earlier part of the SPLICE project. The Live Catalog operated as a small, hand-curated collection of learning tools embedded within an LMS course. It was designed to unify diverse learning systems by leveraging Learning Tools Interoperability (LTI), enabling users to interact directly with tools and resources. The Live Catalog allowed users to actively engage with the learning tools within the LMS environment.

One primary limitation was its dependency on the Canvas LMS and its reliance on a small, hand-crafted collection of examples from a limited number of learning tools, rather than a broad catalog of numerous individual SLC items. The interoperability of learning tools within the Live Catalog was limited to the LMS environment, meaning that only users of a specific LMS could access the integrated resources. This confinement limited the catalog's reach and flexibility. It restricted access for educators and learners using different systems, reducing opportunities for widespread collaboration and resource sharing.

Another challenge was the complex integration process required to incorporate various learning tools into an LMS course. Potential contributors did not have the ability to independently add items to the Live Catalog. These difficulties may discourage educators from using the available resources, as the Live Catalog provides limited direct information about the content available in the cataloged systems. Furthermore, the need for technical proficiency and the additional workload posed barriers to entry, limiting the Live Catalog's usability and adoption among instructors who might otherwise benefit from its

offerings.

To address these challenges, we developed the Smart Learning Content (SLC) catalog as an improvement to the Live Catalog. The SLC catalog acts as a centralized repository, enabling instructors and researchers to contribute, access, and share a wide range of resources. These include computer science education datasets, learning tools, and smart learning objects such as interactive exercises, visualizations, and programming tasks—all without being restricted to a specific LMS. The SLC catalog aims to promote broader accessibility and encourage more educators to use and contribute to the repository, thereby enriching the pool of available resources [30].

However, as the volume and diversity of content in the catalog increase, new complexities and challenges emerge in managing these resources effectively [40]. A method of catalog management often relies on manual processes for validation and classification, which are not scalable or efficient for a large federated catalog. In addition, inconsistencies in metadata standards, invalid links, and unstructured content may hinder the discoverability and reuse of resources [11]. These challenges underscore the need for a validation infrastructure and a well-organized content classification framework to ensure that the Smart Learning Content (SLC) catalog is reliable and beneficial [63]. Validation support improves the quality of the catalog's content, ensuring consistency and reliability. In addition, the implementation of classification mechanisms organizes the content of various learning tools and improves the discoverability in the catalog.

As the field of computer science education continues to evolve, the role of learning content catalogs becomes essential in supporting innovative teaching methods and research initiatives [27]. By addressing the challenges of content management, we can create a more robust learning catalog that fosters collaboration and knowledge sharing.

1.2 Motivation

The Smart Learning Content catalog will thrive through contributions from educators and researchers offering a diverse range of quality learning content. This broad participation leads to a more comprehensive collection of learning materials that cover a wider spectrum of topics, teaching methods, and perspectives.

The contributions from numerous sources expand the catalog's content, but they also increase the likelihood of inaccuracies, inconsistencies, and the inclusion of inaccessible resources. Consequently, managing such a collaborative platform presents significant challenges, particularly in maintaining reliable and high-quality data. Without a robust quality control mechanism, the credibility of the catalog could be compromised. Inaccessible content risks reducing the overall utility for instructors and students.

Implementing validation support is essential to address these challenges. Manual validation of submissions is labor intensive, prone to errors, and incapable of scaling effectively for large catalogs. Additionally, manual processes cannot continuously monitor entries for ongoing compliance; for instance, a URL that was initially valid may later become inaccessible, rendering the resource unusable. Without robust automated validation, metadata standards may not be consistently enforced, and such issues can go unnoticed. This can result in a catalog populated with noncompliant entries, making it difficult for users to efficiently locate reliable resources.

A consistent and reusable classification system is essential for an effective Smart Learning Content (SLC) catalog. The lack of a clear and standardized approach leads to disorganized content, making resources difficult to locate in the catalog. Implementing an ontology-based classification ensures logical structuring, enabling users to navigate the catalog efficiently and find learning materials that align with their specific needs. By ensuring quality entries and organizing the learning content, this approach addresses many of the challenges inherent in developing the Smart Learning Content Catalog.

1.3 Contributions

The contributions of this work fall into two major categories:

- Development of a Smart Learning Content catalog infrastructure with validation support:
  - Implementation of services to validate content metadata against predefined standards for catalog quality.
  - Creation of a submission dashboard with feedback to the contributor.
  - Design of a well-organized classification system using an ontology to improve content discoverability.
- Upgrading OpenDSA's LTI support with LTI Advantage features:
  - Support LTI 1.3 Core functionality using OpenID Connect and OAuth 2.0 for enhanced security and authentication:
    - Implement JWT-based message signing for secure communication between OpenDSA and Learning Management Systems (LMS).
    - Integrate OpenID Connect for user authentication.
    - Use OAuth 2.0 for token-based secure authorization in API calls.
  - LTI Advantage services
    - Implement Name and Role Provisioning Services (NRPS) to streamline user and role management across integrated systems.
    - Integrate Assignment and Grade Services (AGS) to enable seamless grade passback and management between OpenDSA and LMS.

1.4 Objectives

The primary objectives of this work focus on enhancing the Smart Learning Content (SLC) catalog to strengthen its organization, reliability, and interoperability. The first objective focuses on the organization of content within the SLC catalog to facilitate efficient discovery and retrieval of curated learning content. By developing a custom ontology tailored to computer science education, we aim to effectively represent shared knowledge from various learning tools and resources. This involves leveraging keywords and the hierarchical structures from OpenDSA e-textbooks to develop the ontology. The result is a hierarchical classification framework that supports educators and researchers to locate relevant content with ease. Chapter 3 discusses the implementation and impact of the ontology-based framework for the SLC catalog.

The second objective focuses on ensuring the quality and integrity of the data in the content catalog by developing a validation service. The service is designed to ensure compliance with metadata standards, validate submitted Uniform Resource Locators (URLs), categorize validation errors, and provide constructive feedback to contributors. By automating the validation process, we improve the reliability of the catalog and reduce the administrative burden associated with manual validation. This service streamlines the submission process for contributors and maintains a high standard of content within the catalog. Chapter 4 provides a comprehensive overview of the development and functionalities of the validation service.

The final objective focuses on enhancing the interoperability of the OpenDSA e-textbook system through the implementation of Learning Tools Interoperability (LTI) version 1.3. This implementation includes handling JSON Web Tokens (JWT) and OAuth 2.0 authentication protocols, enabling secure and seamless integration with external educational tools. Supporting both LTI 1.3 and the existing LTI 1.1 standards ensures seamless integration of OpenDSA with diverse learning environments. The adoption of LTI 1.3 positions OpenDSA to better serve the evolving needs of digital education by facilitating

more flexible and secure tool integrations. A comprehensive discussion of the implementation process and its benefits is presented in Chapter 5.

Chapter 2

Review of Literature

2.1 Ontologies in Educational Resource Catalogs

Catalogs serve as repositories for educational resources, which may include datasets, learning content, and access to various tools. As the volume of content continues to increase [1], the need for retrieval and organization is essential. Educators can use catalogs to discover relevant materials that support their teaching and learning objectives. Early efforts in this field, such as the Multimedia Educational Resource for Learning and Online Teaching (MERLOT) catalog [56], demonstrated the value of centralizing resources and involving community contributions. MERLOT provides a peer-reviewed platform for higher education materials, and in the first release of the MERLOT catalog, discovering learning content was achieved through structured descriptions and user feedback.

Similarly, the Ensemble project featured an Algorithm Visualization Catalog [2] that aggregated algorithm visualizations and leveraged community tagging and user feedback to enhance resource discovery, making it easier for users to find and access relevant visualizations. These catalogs made strides in organizing and allowing instructors to discover learning content; they relied on keyword-based classification and community-driven tagging.

Building on these foundational efforts, rather than depending solely on keyword tagging

and user input, we applied an ontology-based content organization. The ontology-based approach provides a structured and semantically rich means of organizing educational content and harnessing the keywords to improve retrieval and navigation [26]. With this approach, we address the challenge of organizing content in the catalog. Otherwise, users may struggle to find relevant resources in the overwhelming volume of available content.

Ontologies have proven to be an effective tool for managing educational resources. In computer science, an ontology is defined as a formal and explicit specification of a shared conceptual framework [10]. In simpler terms, it is a structured representation of knowledge within a domain, defining key concepts and the relationships between them. By applying an ontology to catalogs, we can create hierarchical and relational structures that organize content meaningfully, facilitating efficient retrieval and navigation.

In the field of computer science education, implementing a domain ontology within a catalog offers several benefits. Firstly, it addresses the complex challenge of managing large volumes of learning content by providing a structured framework that organizes the learning content in the catalog. Secondly, it facilitates knowledge mapping by leveraging the taxonomy and explicit relationships between different pieces of content [26]. This helps the catalog users to understand how various resources relate to each other. This interconnected system of information makes it easier for users to navigate, comprehend, and use the available resources effectively.

The keywords in the ontology provide a common language and framework for describing learning content, which is essential for integrating resources from diverse sources [21]. This standardized approach improves the organization of the content and facilitates the discovery of learning resources, allowing educators and learners to find relevant materials more efficiently [28].

Existing ontologies, including the Computer Science Ontology (CSO) and the Association for Computing Machinery (ACM) Classification System, served as a useful starting point for establishing a shared vocabulary [7, 37]. However, these ontologies were often high-level and lacked the granularity needed for detailed content classification in the SPLICE

prototype catalog. As a result, their direct application proved limited for the diverse and nuanced resources in the catalog. To address this gap, we implemented a custom ontology capable of capturing broad and specialized concepts, ensuring a more precise semantic organization of learning materials.

However, developing a comprehensive ontology is a complex and time-consuming process that requires expertise in the domain and ontology engineering [10]. The challenges include accurately defining concepts, properties, and relationships, as well as managing synonyms and variations in terminology. For instance, different terms may refer to the same concept, such as "enrollment" and "registration", therefore, accounting for such variations is crucial to ensure consistency in retrieval and accuracy in the catalog [35]. Secondly, aligning ontologies with existing learning systems, such as federated catalogs, can also be difficult due to differences in data models and technologies [28]. These challenges require solutions that can integrate ontologies effectively into educational resource catalogs.

To address these issues, researchers and practitioners in computer science education have proposed several strategies. Semantic normalization techniques, such as the use of key-value pairs, help manage synonyms and aliases by mapping different terms to consistent keys, ensuring uniform interpretation across the system [35]. Modular ontology design is another approach in which ontologies are broken down into components that can be developed independently and later integrated [58]. The modular approach facilitates easier maintenance and scaling, especially when integrating ontologies from multiple learning tools into a federated catalog.

2.2 Validation Infrastructure in Educational Content Platforms

As these platforms grow, they often face challenges in managing vast amounts of content contributed by numerous educators and institutions. Ensuring that this content

adheres to predefined standards and remains accurate, accessible, and reliable is critical to maintaining the credibility and usefulness of the catalog for its users [43].

One of the primary challenges in catalog management is the risk of inconsistencies or errors. Ensuring the quality of content added to a catalog is critical, but manually reviewing each entry is both time-intensive and impractical for large-scale catalogs. This emphasizes the need for validation support for efficiency and reliability [59]. To address these challenges, a validation service is an essential component of catalogs. This service is designed to ensure that both the data and metadata associated with educational content are authentic and comply with established standards. In this context, a key example is DataShop, a data repository and analysis platform that provides a comprehensive framework for managing educational datasets throughout their life cycle, from collection and analysis to reuse. It also offers robust data validation and quality assurance features, which are essential to ensure the reliability of educational data [59].

A key element of the validation support is metadata management and validation. Metadata serves as a foundational layer of information for each piece of content, detailing attributes such as its source, description, associated URLs, and other essential properties [8]. The metadata schema validation involves verifying that the metadata for each content item conform to a predefined structure and set of rules. This approach ensures consistency across the platform, making it easier for users to locate and use resources effectively. The Informatica repository manager [36], a widely recognized tool for metadata validation within Informatica's product suite, exemplifies the practical application of metadata verification by enabling the efficient validation and management of metadata repositories. Metadata services are essential for maintaining data quality and minimizing errors when contributing content to a catalog.

In addition to metadata validation, URL validation is another integral component of a robust validation infrastructure. Learning tools often include numerous external links within catalog entries that point to resources. Implementing a URL validation service is crucial to ensure that all referenced content remains accessible. For example, HTTP

libraries such as Axios in JavaScript provide an efficient way to handle HTTP requests and validate URLs [9]. Axios generates a structured response object containing details such as status codes and headers, enabling real-time verification of URL validity and resource accessibility [4].

An effective validation infrastructure should include a robust feedback mechanism that aggregates and presents validation results to contributors through an accessible interface, such as a dashboard. This mechanism provides contributors with a clear summary of compliance status, detailed information on validation errors, and actionable recommendations to resolve issues [3]. By informing contributors of any problems with their submissions, the catalog with validation support encourages adherence to standards and fosters a collaborative environment focused on maintaining high-quality content.

2.3 Learning Tools Interoperability

Initial efforts to achieve interoperability were hindered by siloed learning environments that offered little to no cross-platform compatibility. The absence of standard protocols and proprietary restrictions forced custom development, making interoperability complicated. To address this challenge, the introduction of Learning Tools Interoperability (LTI) 1.1 by IMS Global [34] incorporated OAuth1.0a authentication as a framework to enable seamless integration of external learning tools with LMS platforms.

Learning Tool Interoperability (LTI) 1.1 emerged as an essential service in the development of accessible learning tools. It enables interoperability between learning tools and platforms, facilitating content sharing and user authentication in educational environments [18, 41]. LTI 1.1 has proven valuable for learning tools and platforms; however, there are some challenges such as LTI 1.1 relying on OAuth 1.0a for authentication, which had known security vulnerabilities and functional limitations [34].

LTI 1.1 lacked a standard way for tools to access user information and roles, complicating tasks such as course rosters or personalized content. As a result, learning tools often

relied on custom workarounds for roster management [32]. Also LTI 1.1's basic outcomes service combined with some LMSs limitations only provided a simple mechanism for associating a single gradebook column with each resource link. This lacked some flexibility to manage multiple gradable items within a resource [33, 64]. Many learning tools such as OpenDSA have adapted to this constraint and implemented workarounds such as aggregating module scores to present a unified grade within a resource and accessing the etextbook via the LMS to obtain user rosters [46].

IMS Global recently introduced LTI 1.3, an enhanced standard that includes the LTI Core specification and the LTI Advantage suite. This upgrade offers additional services such as Name and Role Provisioning (NRPS) and Assignment and Grade Services (AGS), along with improved security protocols through OpenID connect, JSON Web Tokens (JWTs) and OAuth 2.0. By extending interoperability, LTI 1.3 addresses the challenges of LTI 1.1 [33]. Like its predecessor, it features two primary components: the tool consumer, which is typically an LMS, and the tool provider, which is an external application. The Consumer initiates launches by securely transmitting information through claims and messages, specifying roles, assignments, and other aspects of the learning environment.

The LTI 1.3 security framework is based on OpenID Connect, JSON Web Tokens (JWTs) and OAuth 2.0 to enable encrypted authentication tokens, ensuring secure and standardized communication between platforms and tools [34]. The LTI Advantage suite extends these capabilities with services like Names and Role Provisioning Services (NRPS), which allow tools to access course rosters and identify user roles from the LMS. In addition, Assignment and Grade Services (AGS) improve grading workflows by managing assignment data and enabling detailed score updates [33]. As learning tools evolve, the emphasis on security and privacy intensifies. LTI 1.3 incorporates OAuth 2.0 and JWTs to ensure that sensitive student data is protected.

Chapter 3

Ontology Development

3.1 Building the Ontology

This chapter delves into the ontology development process for the Smart Learning Content (SLC) catalog. It covers the development of an ontology, the underlying schema of the database to support the hierarchical structure, and the classification mechanisms used to categorize the learning content in the catalog. The objective of the ontology is to provide contributors with a set of standard keywords to enrich their content. This facilitates connections among various learning resources, establish meaningful relationships, and effectively classify content within the Smart Learning Content (SLC) catalog [28]. The keywords of the ontology play a critical role in organizing and managing complex content. They establish an interconnected structure that simplifies content discovery as new learning materials are added [53].

3.1.1 Ontology Concepts - Classes and Properties

In building the ontology for the Smart Learning Content (SLC) catalog, an essential task was to define concepts, referred to as classes in the ontology terminology, and their associated properties. In the context of ontologies, a class represents a collection of entities

that share common characteristics within a domain [15]. To construct the initial draft for the classes, we leveraged content from OpenDSA, an e-textbook system for teaching Data Structures and Algorithms (DSA) [54]. OpenDSA provided a rich set of keywords and metadata associated with its learning materials, which were crucial in defining the ontology structure. Appendix A presents a table showcasing the keywords extracted from OpenDSA learning materials.

From OpenDSA's content, we extracted a hierarchical structure of topics and subtopics that cannot be represented as a tree, as some concepts logically belong to multiple higher-level categories. Instead, this structure is modeled as a Directed Acyclic Graph (DAG) with four levels, allowing for an overall top-down progression from broad topics to more specific details while accommodating overlapping relationships. For example, AVL Trees can belong to both the Binary Tree category (due to its structural morphology) and the Search Structure category (how it is used). This illustrates the necessity of a Directed Acyclic Graph (DAG) structure.

This interconnected multi-parent arrangement captures the complexity of relationships within the domain of computer science education, particularly in data structures and algorithms [16]. Figure 3.1 illustrates the hierarchical relationships between classes in the ontology, demonstrating how Disjoint Sets can belong to multiple categories. In this Directed Acyclic Graph (DAG), each node represents a class, and the edges define the subclass relationships.

The adoption of this hierarchical structure is crucial as it organizes content into a logical framework, allowing users to efficiently navigate from broad topics to specific details. This structured approach also facilitates discoverability by leveraging relationships between topics, enabling users to locate resources starting from general or related terms.

In the ontology, we distinguish between classes, subclasses (or subtypes), and properties to effectively categorize and describe educational content. Classes represent the primary categories or overarching concepts within the ontology, serving as the foundational nodes in the hierarchical structure—for example, broad topics such as Algorithms and Data

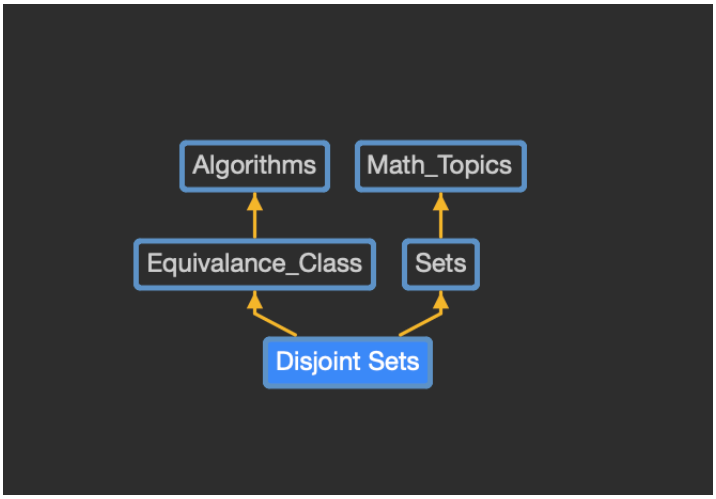

Figure 3.1: Directed Acyclic Graph (DAG) illustrating the hierarchical relationships between classes in the ontology.

Structures at higher levels. Subtypes, or specialized subclasses, refine these broader classes into more specific concepts, providing a granular breakdown of the parent class. These subtypes can, in turn, be further subdivided into even more detailed subcategories [50].

Properties represent attributes or characteristics associated with classes and subclasses, defining relationships and providing additional information about them [10]. Examples include "hasDifficultyLevel", "requiresPrerequisite", "isSubclassOf", or "isPartOf". The properties are linked to classes and subclasses to improve the relations. Understanding the differences among these elements is essential. Classes and subtypes define the hierarchical structure of the ontology, representing concepts ranging from general to specific. Properties, on the other hand, describe the attributes and relationships of these classes and subtypes, adding metadata that enriches and enhances the ontology.

3.1.2 Structure and Relationships

In developing the ontology for the Smart Learning Content (SLC) catalog, we use the terminology of classes and subclasses to represent the hierarchical organization of keywords. In this context, a class represents a group of entities or concepts that share common characteristics, serving as a general category within the domain [10]. A subclass

is a more specific category that inherits properties from its parent class, representing a specialization of the broader concept.

We primarily employ the "isSubclassOf" relationship to define hierarchical connections between topics and subtopics. This relationship is particularly suited to represent the progression from general concepts to more specific ones [52]. For example, the class Data Structures may have subclasses like Trees, which in turn may have further subclasses such as Binary Trees. Although ontologies can support various types of relationship such as hasPart, isRelatedTo, or prerequisiteFor, we deliberately focus on isSubclassOf for the following reasons:

- Hierarchical Clarity: The isSubclassOf relationship provides a clear and intuitive hierarchical structure, which aligns well with how computer science topics are typically organized. Figure 3.1 shows a SubclassOf relationship of some content related to recursion.
- Inheritance Properties: This relationship allows for the inheritance of properties from the parent class to the subclass, which is particularly useful in representing the shared characteristics of related computer science concepts.
- Simplicity and Scalability: By focusing on one primary relationship type, we have created a more straightforward and easily maintainable ontology structure. This simplicity helps to update the ontology as it grows to encompass more topics and to facilitate easy integration into the Smart Learning Content catalog.

For example, in the ontology:

- Binary Trees isSubclassOf Trees.
- Sorting Algorithms isSubclassOf Algorithms.

The use of the isSubclassOf relationship establishes a clear and navigable structure within the ontology. This approach ensures that content is classified with granularity and allows

scalable growth as new topics or subtopics are introduced [55]. It aligns with established ontology design principles, which highlight the importance of hierarchical relationships in representing domain knowledge [54].

Ontology Structure and Data Models for the Smart Learning Content Catalog

After establishing the importance of the isSubclassOf relationship in structuring our ontology, it is essential to define how we represent this structure within the Smart Learning Content (SLC) catalog. To effectively implement the ontology and integrate it seamlessly with the catalog, we developed data models to support the underlying relationships. These models define the entities and relationships that make up the ontology, providing a framework to organize content and facilitating navigation and search. The ontology is represented through three primary data models:

1. OntologyClasses (Class): This model represents the fundamental concepts or categories within the ontology. Each class corresponds to a topic or area of study in computer science education.
    - label: The name of the class or category (e.g., Binary Tree Traversal).
    - class_uri: A unique identifier for the class, often derived from a standardized ontology.
    - description: A detailed description of the class and its relevance to computer science education.
    - is_active: A flag indicating whether the class is active and being used for classification.
2. OntologyRelations (Relations): This model defines the relationships between classes, establishing the hierarchical structure of the ontology. Specifically, it captures the isSubclassOf relationships that connect more specific concepts (subclasses) to their broader categories (classes).
    - class_id: The ID of the main class (e.g., Binary Trees).

- subclass_of: The ID of the parent class from which this class inherits (e.g., Trees).
- relation_uri: A unique identifier for the relationship, often derived from a standardized ontology.
- description: A detailed description of the relationship, including its hierarchical connection (e.g., Binary Trees is a subclass of Trees).
- is_active: A flag indicating whether the relationship is active and being used within the ontology.

3. slc_item_catalog (Item Catalog): This model represents the educational resources within the catalog and links them to the appropriate classes in the ontology.
   - exercise_name: The name of the exercise or learning module (e.g., Quicksort Partition Proficiency Exercise).
   - keywords: Keywords associated with the item, often used for matching against the ontology (e.g., Sorting Algorithms).
   - url: The URL where the content is hosted.
   - description: A detailed description of the content item and its relevance to the subject matter.
   - platform_name: The name of the platform hosting the content (e.g., OpenDSA).
   - author: The author or creator of the content (e.g., Cliff Shaffer).
   - class_id: The ID linking the content item to a class in the ontology.

3.1.3 Ontology Representation Using OWL

The keywords are represented using the Web Ontology Language (OWL) to easily visualize relationships. OWL provides a robust framework for modeling complex relationships between concepts [10]. OWL can be serialized in several ways; we chose the most common serialization syntax that leverages both the Resource Description Framework (RDF) and

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
  <rdf:Description rdf:about="http://opendsa.org/ontology#Bucket_Hashing">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://opendsa.org/ontology#Hashing"/>
    <rdfs:label>Bucket Hashing</rdfs:label>
    <rdfs:comment>A type of hashing where multiple keys are placed in the same bucket.
</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://opendsa.org/ontology#Number_Problems">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://opendsa.org/ontology#Mathematical_Algorithms"/>
    <rdfs:label>Number Problems</rdfs:label>
    <rdfs:comment>Addressing mathematical problems related to numbers, such as prime
finding or factorization.</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:about="http://opendsa.org/ontology#Tree_Indexing">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://opendsa.org/ontology#Indexing"/>
    <rdfs:label>Tree Indexing</rdfs:label>
    <rdfs:comment>Using tree structures, such as B-trees or AVL trees, to index and access
data efficiently.</rdfs:comment>
```

Figure 3.2: RDF/XML representation of a Web Ontology Language (OWL) snippet.

the Extensible Markup Language (XML). In this approach, XML provides the structural foundation for serialization using tags to organize information systematically. Building upon this, RDF utilizes a graph-based model to represent the ontology's information, capturing the relationships and hierarchies between concepts as interconnected nodes and edges.

Figure 3.2 illustrates a snippet of the custom ontology represented in OWL format.

It uses the RDF syntax to define classes, properties, and relationships among various topics [29]. The RDF code snippet is written in OWL format. It defines a set of classes and uses the "rdfs:subClassOf:" property to indicate that one class is a subclass of the other.

Using OWL with RDF/XML serialization facilitates the representation of rich class hierarchies, detailed properties, and relationships in the SLC catalog. This approach also streamlines the process of updating keywords and allows for effective collaboration with

domain experts to review and refine relationships.

3.2 Ontology Aliases

In the process of building a robust and flexible ontology for the Smart Learning Content (SLC) catalog, we recognized the need to account for variations in terminology used across different sources of educational content. In the field of computer science education, particularly when addressing complex topics such as data structures and algorithms, individuals often use varied terms to refer to the same concept. This challenge is addressed through the implementation of ontology aliases, a well-established concept in ontology engineering [45].

3.2.1 Purpose of Ontology Aliases

Ontology aliases were introduced to manage alternative synonyms for concepts within the ontology. The primary objective was to eliminate inconsistencies in content classification by enabling the system to recognize and map different terms that refer to the same class or topic [45]. For example:

- "Hash Table" and "Hash Map" are different from those used in various programming languages and resources and might describe the same data structure.
- The terms "Binary Search Tree" and "BST" both refer to the same data structure. Although one term may be an acronym, aliases are a fallback option for terminologies that lack agreement on a common keyword.

3.2.2 Implementation of Ontology Aliases

To implement ontology aliases, the ontology structure is extended to include an additional property for each class: alias keywords. These alias keywords are stored alongside the

id	alias_name	ontology_class_id	created_at	updated_at
1	“Binary Search Tree”	1	2024-10-22	2024-10-22
2	“BST”	1	2024-10-22	2024-10-22
3	“Depth First Search”	2	2024-10-22	2024-10-22
4	“DFS”	2	2024-10-22	2024-10-22
5	“Breadth First Search”	3	2024-10-22	2024-10-22
6	“BFS”	3	2024-10-22	2024-10-22
7	“Dynamic Programming”	6	2024-10-22	2024-10-22
8	“DP”	6	2024-10-22	2024-10-22

Table 3.1: Overview of ontology aliases tables schema in the Smart Learning Content (SLC) catalog.

main class label and can be used interchangeably during the classification process to represent controlled vocabularies [47]. Each alias is directly mapped to its corresponding class in the ontology. When the system classifies a piece of content, it matches keywords with both the primary class and associated aliases [10]. An example of ontology aliases is the class “Binary Search Tree”, for which the following aliases are defined:

- Main Class Label: Binary Search Tree.
- Aliases: BST.

When the system encounters content containing any of these terms, the objective of the ontology can be set to classify those items with keywords that are not standardized. Table 3.1 provides an overview of how a set of aliases is assigned to their respective ontology classes in the scenario described.

Benefits of Using Ontology Aliases

The use of aliases provided several key advantages in the Smart Learning Content (SLC) catalog [48]:

1. Improved Flexibility: The catalog can handle variations in these terminologies, reducing the likelihood of misclassification due to different labels for the same concept.

Class ID	Class Name
1	Data Structures
2	Trees
3	Binary Trees
4	AVL Trees
5	Algorithms
6	Sorting
7	Graphs
8	Hashing
9	QuickSort
10	Depth-First Search
11	Search Structures

Table 3.2: List of ontology classes and assigned IDs.

2. Broader Keyword Matching: These related keywords can yield results that may not exactly match the keywords defined in the catalog.

3.3 Database Schema Design

The Smart Learning Content (SLC) catalog required a carefully designed relational database schema to store the ontology and enable dynamic content classification. This section outlines the key entities in the schema and their role in facilitating the classification mechanism [10].

3.3.1 Ontology Entities

The core entities in the database are designed to represent the classes and their relationships within the ontology. These entities include:

- **OntologyClasses**: This table outlines the classes and subclasses defined in the ontology, such as Data Structures and Algorithms, along with their descriptions [14], as discussed in Section 3.1.2. It serves as a foundational reference for Table 3.3, which defines the relationships between these classes. Table 3.2 offers a comprehensive overview of the classes represented in the ontology.

ID	Parent Class	Child Class
1	Data Structures	Trees
2	Trees	Binary Trees
3	Algorithms	Sorting
4	Binary Trees	AVL Trees
5	Data Structures	Graphs
6	Data Structures	Hashing
7	Sorting	QuickSort
8	Graphs	Depth-First Search
9	Data Structures	Search Structures
10	Search Structures	AVL Trees

Table 3.3: Ontology relations table illustrating isSubclassOf relationships.

- **Ontology Relations**: This table captures the relationships between classes, focusing specifically on the isSubclassOf relationship. Table 3.3 illustrates these relationships between ontology classes. For example, as described above, AVL trees belong to both the Binary Tree and Search Structure categories. While the original structure represents relationships using class IDs, the table offers a more accessible representation by using corresponding keywords.

3.3.2 Smart Learning Content Catalog

The Smart Learning Content (SLC) catalog features a primary table, the slc_item_catalog table, which stores metadata for all uploaded learning content, including exercises, slideshows, and other resources. Each entry includes fields such as the URL of the content, keywords, and associated metadata (e.g., author, platform). Table 3.4 provides a summary of an OpenDSA learning content exercise within the Smart Learning Content catalog, highlighting its various essential fields. The table shows an OpenDSA entry in the Smart Learning Content catalog for a binary search proficiency exercise from OpenDSA. The classification of this content is guided by the ontology, with keywords serving as properties that facilitate matching the content to specific classes within the ontology [39].

Field	Details
platform_name	OpenDSA
exercise_type	Proficiency Exercise
exercise_name	Binary Search Proficiency Exercise
description	A proficiency exercise on Binary Search algorithm, covering implementation and analysis.
language	N/A
url	`https://opendsax.cs.vt.edu/embed/BinarySearchProficiency`
author	Cliff Shaffer
institution	Virginia Tech
lti_url	`https://opendsax.cs.vt.edu/lti/launch?custom_ex_short_name=BinarySearchProficiency`
iframe_url	`https://opendsax.cs.vt.edu/embed/BinarySearchProficiency`
keywords	Binary Search, Searching Algorithms, Algorithm Proficiency

Table 3.4: Smart Learning Content (SLC) catalog: An OpenDSA entry for a binary search proficiency exercise from OpenDSA.

ID	Classified At	Item ID	Class ID
1	2024-10-21 00:12:52.003	2283	3
2	2024-10-21 01:35:42.407	2313	3
3	2024-10-21 02:00:22.193	2314	4
4	2024-10-21 02:15:37.892	2315	6
5	2024-10-21 02:45:52.734	2316	7
6	2024-10-21 03:01:12.340	2317	9

Table 3.5: Schema for item classification table representing classes and their relationships in a DAG.

3.3.3 Ontology Schema

In addition to the ontology and the SLC table, the schema includes an additional table to support classification. The ItemClassification table serves as a bridge, linking the content in the slc_item_catalog table to the classes defined in the OntologyClasses table. This table tracks the association between content and its corresponding class, facilitating updates as new entries are added to the catalog.

Structure of the Item Classification Table

The item classification schema is outlined in Table 3.5, which follows the principles of Ontology-Based Data Access (OBDA) [50].

The schema allows the system to classify new content based on the ontology and store the classification results for retrieval during user queries.

- ID: A unique identifier for each classification record.

- Classified At: The timestamp indicating when the content was classified.
- Item ID: A reference to the specific content item in the slc_item_catalog.
- Class ID: A reference to the corresponding class in the OntologyClasses table.

An example of classification is demonstrated through entries from the ItemClassification table, as presented in Table 3.5:

- ID 1: The content item with Item ID 2283 is classified under Class ID 3, which corresponds to the Binary Search Tree class from Table 3.2.
- ID 2: The content item with Item ID 2313 is also classified under Class ID 3, indicating another resource related to Binary Search Tree from Table 3.2.
- ID 3: The content item with Item ID 2314 is classified under Class ID 4, corresponding to the Graph Traversal class from Table 3.2.

These classifications ensure that all resources related to the Binary Search Tree are grouped under the same category, regardless of their individual labels. Figure 3.3 illustrates the complete ontology schema and its integration with the SLC content items.

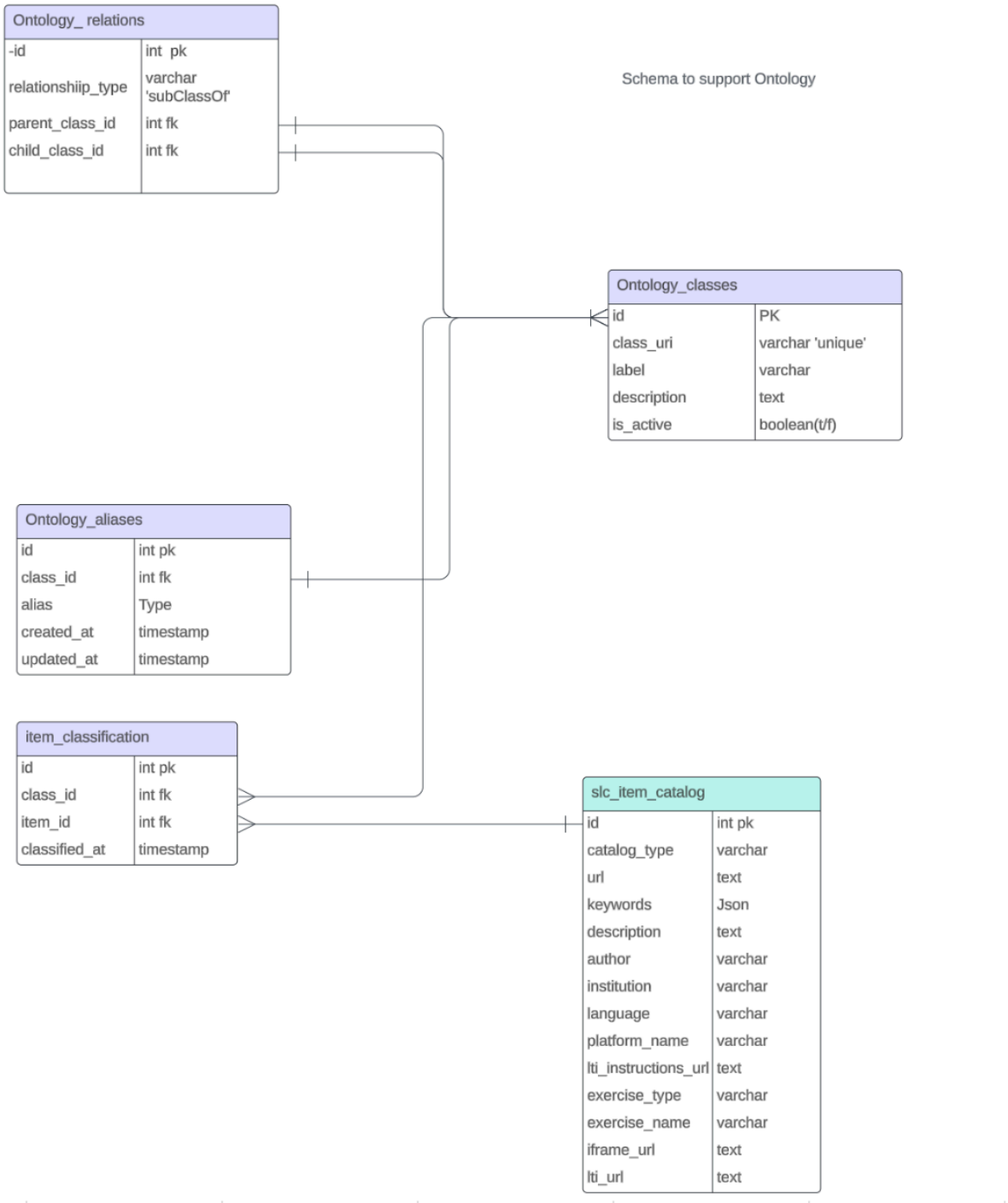

Figure 3.3: Detailed relationship diagram of the Smart Learning Content (SLC) catalog to support the classification and ontology.

3.4 Classification Mechanism

The classification mechanism is a straightforward process that systematically assigns each content item to the appropriate class within our ontology based on the keywords present in the contributor's content. The details of this classification process, including its steps, are described in the following section.

Classification Process Overview

The classification mechanism follows a series of steps to match keywords extracted from catalog entries with their corresponding classes in the ontology. This process ensures that new content is accurately classified at the time of entry. The classification process involves the following steps:

1. Keyword Extraction: Keywords are extracted from the metadata for each content entry in the SLC item catalog, as described in Section 3.1.2. These contributor-provided keywords represent the topics and concepts contained within the content item.

2. Keyword Matching: The extracted keywords are compared against those defined in the ontology to identify potential matches between the content item's keywords and the ontology's classes.

3. Ontology Lookup: The system queries the ontology classes table to identify the relevant class for the content based on the matched keywords.

4. Content Classification: Once the appropriate class is identified, an entry is created in the item classification table. This entry establishes the relationship between the content item and the ontology class.

Figure 3.4 provides a detailed illustration of the workflow for the item classification process described above.

Handling Unclassified Content

A critical aspect of the classification mechanism is managing content that does not directly match any existing ontology keywords. In such cases, the system assigns the content to a designated unclassified class. However, this does not impede access, as the keywords provided by contributors can still be utilized to retrieve these content types.

This allows for review and future refinement of the ontology to incorporate new terms and concepts as they emerge [62].

3.5 Challenges in Classification

Developing a robust classification system for the Smart Learning Content catalog involved several challenges, particularly interpreting and representing the ontology as a Directed Acyclic Graph (DAG), mapping catalog items to this structure, and effectively visualizing the resulting data model. Addressing these challenges was crucial to ensure accurate and efficient content classification.

1. **Interpreting and Developing the Ontology as a DAG**

   Representing the ontology as a DAG introduced complexity due to the inherent flexibility and multiple parent-child relationships it allows. Unlike a simple hierarchical tree structure, a DAG enables a single class to inherit properties from multiple parent classes, reflecting the multifaceted nature of computer science concepts.

   This flexibility, while beneficial for capturing intricate relationships, introduced challenges in maintaining consistency and avoiding redundancy in classification. For example, a concept like Graphs could logically belong to both Data Structures and Algorithms, necessitating careful consideration to ensure accurate representation without duplication. To address this, we assigned each class a unique URI, ensuring a singular, consistent identifier for all instances of a concept, thereby preventing duplication across categories.

   Additionally, we adopted best practices, including developing a relational database

model to represent class hierarchies clearly. Tools such as Web Protege were leveraged to support the creation and management of complex ontology structures, ensuring scalability and semantic accuracy [10].

2. **Mapping Catalog Items to the DAG Ontology**

   Mapping educational resources from the Smart Learning Content (SLC) catalog to the DAG-based ontology presented a significant challenge. Each catalog item contains metadata and keywords that must be matched accurately to the corresponding classes within the ontology. The complexity of the DAG, where classes can have multiple parent classes, required complex algorithms to ensure comprehensive and precise classifications for a single content item.

   To address this, we employed exact match techniques for straightforward keyword alignment, supplemented by more advanced algorithms for context-sensitive matching. These algorithms considered the semantic context of keywords to improve accuracy. Additionally, ontology aliases were used to handle synonyms and variations in terminology, ensuring consistency and reducing ambiguity. This multi-label classification approach enables resources to be categorized under all relevant classes, enhancing both discoverability and search accuracy within the catalog [61].

3. **Visualizing the DAG Ontology**

   An effective visualization of the DAG was essential for both developers and domain experts to understand and manage the ontology. Visualizing large and complex DAGs can lead to clutter and make it difficult to interpret the relationships between classes.

   To address this, we adopted a straightforward approach based on the Association for Computing Machinery (ACM) classification system [6]. This system employs a graph-based structure for tagging and research, enabling contributors to seamlessly navigate from broad keywords to more specific ones. This approach facilitates the creation of clear visualizations, effectively highlighting key relationships and hierarchical structures within the ontology. Figure 3.5 shows an interactive page of

the ontology.

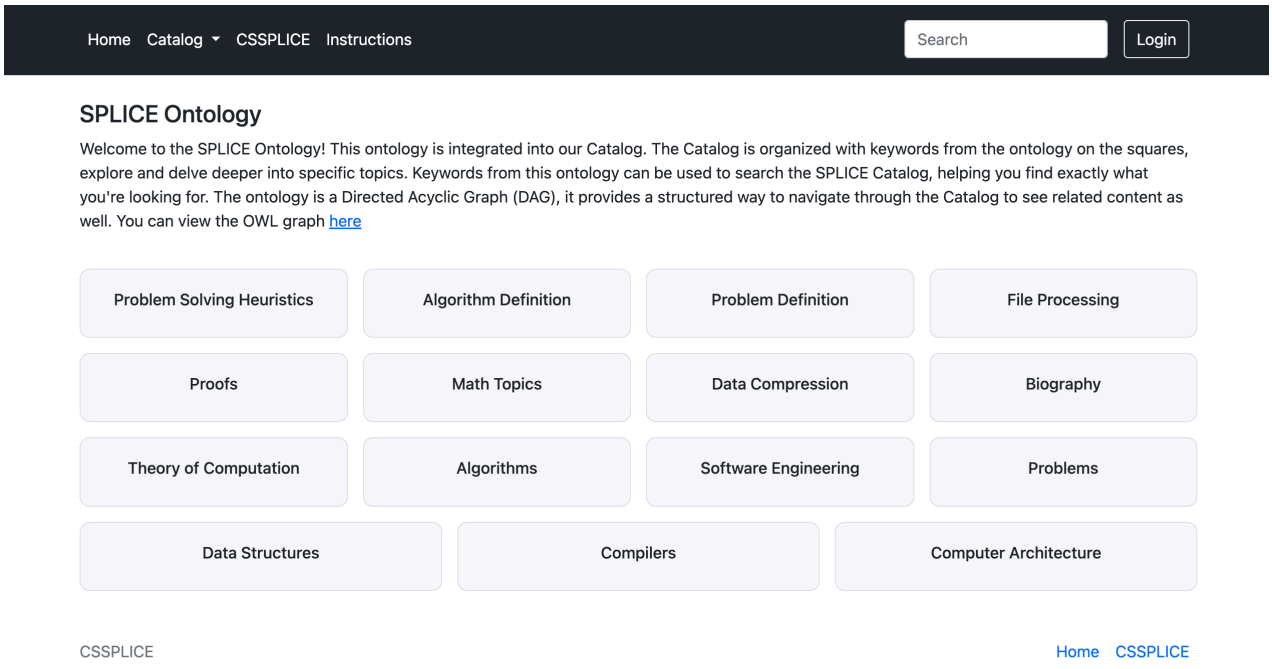

Figure 3.5: A page from the SLC catalog for interacting with the ontology.

4. **Managing Incremental Updates**

   Maintaining a balance between preserving a stable, well-defined ontology and allowing contributors to introduce new terms when necessary is another significant challenge. On one hand, it is important to guide educators toward using established terminology rather than continually adding synonyms. On the other hand, when contributors propose a new keyword not currently in the ontology, their suggestions can be reviewed by a small team of domain experts before being incorporated.

   To facilitate this process, a simple mechanism is employed. Contributors submit proposed updates to the ontology through an online form that mirrors its hierarchical structure. After submission, the updates are reviewed by maintainers to ensure alignment with the catalog's standards. Once approved, an automated script processes the changes, updating the ontology's classes and relationships in the database accordingly.

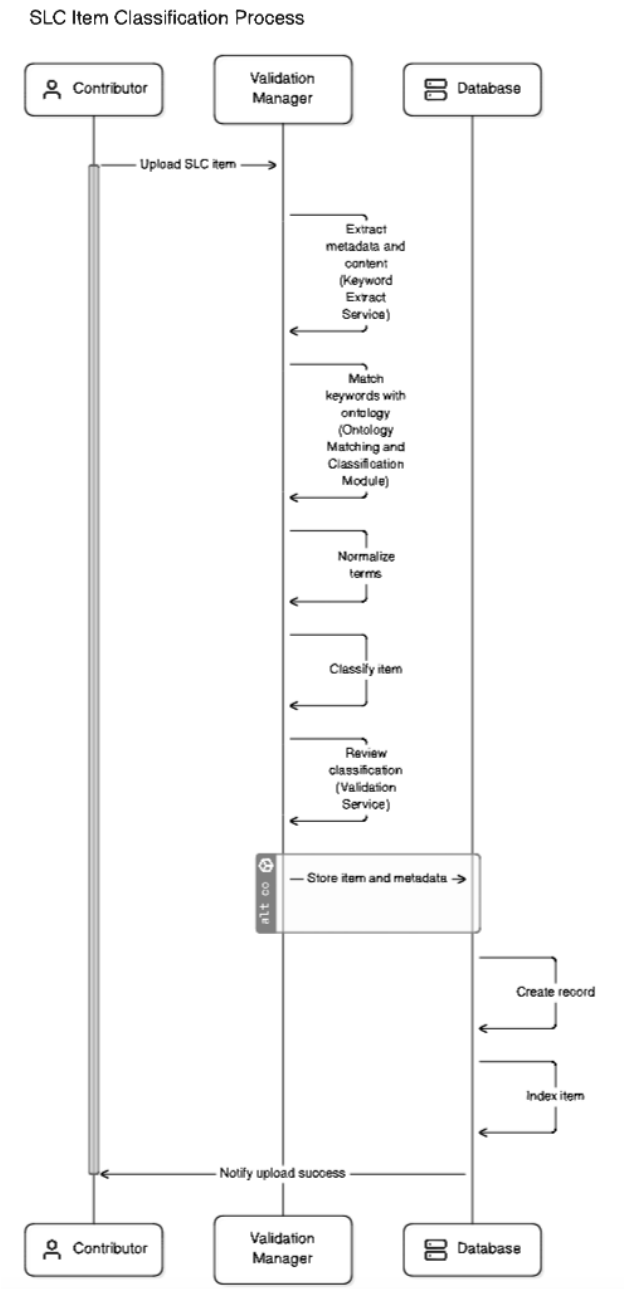

Figure 3.4: SLC upload workflow, detailing the sequential process for keyword extraction, ontology lookup, and classification.

Chapter 4

Infrastructure for the SPLICE Catalog

Metadata is essential for maintaining structured, organized, and searchable content in modern catalog systems. In this chapter, we describe the development of a validation system that ensures metadata compliance within an educational content-sharing platform. This system focuses on schema validation and URL verification. In addition, by categorizing errors and providing detailed feedback, this system supports high-quality content submissions while offering a smooth experience for contributors.

4.1 Smart Learning Content Catalog

4.1.1 Background

The SLC catalog serves as a centralized platform for the computer science education community, offering a repository of tools, learning materials, and datasets to support teaching and research. The content items include shareable and reusable learning objects. Shareable learning objects are created to be easily distributed and accessed by a diverse range of users across various platforms. These objects are typically accessible via URLs

and can be integrated into different educational systems. They are also reusable learning objects designed to be repurposed and incorporated into various educational contexts and curricula. For example, an iframe URL can be embedded directly into a course, or Learning Tools Interoperability (LTI) URLs can be used for deep linking. This enables instructors to integrate these objects into their teaching methods and course structures effectively. These objects are designed to be standalone and embedded [12], allowing integration into various learning environments. The SLC catalog infrastructure is built with the focus on facilitating content discoverability and reusability, serving the needs of instructors, course developers, and researchers.

At its core, the SLC catalog is organized as a collection of catalogs that host a wide variety of learning objects and tools from multiple external providers [42]. Some content providers currently include OpenDSA, CodeWorkout, ACOS-Server, CodeCheck, Codespec, RuneStone, Programming Course Resource System (PCRS), etc. The catalog is designed to expand continuously through collaboration with other content providers to promote a rich ecosystem of learning tools and content.

Shareable and Reusable Learning Objects

The SLC catalog is organized around three primary components:

- **Tools:** These are self-contained, applications designed for seamless integration into Learning Management Systems (LMSs) or platforms using Learning Tools Interoperability (LTI) or iframes [57]. Figure 4.1 presents a list of the current learning tools available in the SLC catalog. The tools serve various pedagogical purposes, such as assessments, animations, interactive exercises, and simulation environments.

- **Content Items:** This category encompasses a wide variety of learning materials, including exercises, worked examples, Parsons problems [22], slideshows and animations. Each content item is equipped with comprehensive metadata that facilitates search and categorization within the catalog. This metadata includes keywords. Figure 4.2 illustrates the variety of OpenDSA content items and exercises included

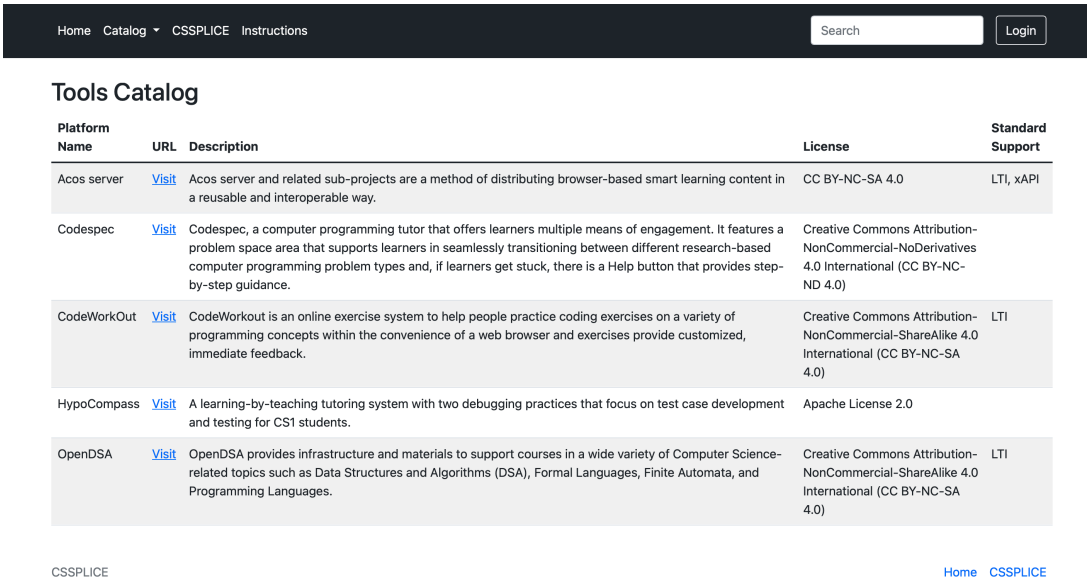

Figure 4.1: A view of the learning tools available in the Smart Learning Content (SLC) catalog.

in the SLC catalog. These exercises are available as iframes embedded within the SLC catalog, allowing users to interact with them directly on the platform without the need to access the learning tool separately.

- **Datasets:** These are collections of data used for teaching and learning purposes, including example datasets for algorithms, data structures, data analysis tasks, and machine learning projects. Datasets come with detailed descriptions and are linked to relevant content items and tools to support practical learning activities. Figure 4.3 provides an overview of the datasets currently available in the SLC catalog. The Datasets section includes metadata information about the datasets, including URLs linking to actual datasets hosted on various platforms, such as GitHub, DataShop, and others.

The infrastructure is built to support instructors and course developers in finding and embedding relevant content. For instructors, the catalog provides a Content Item Search feature, which allows them to search for specific learning materials based on keywords.

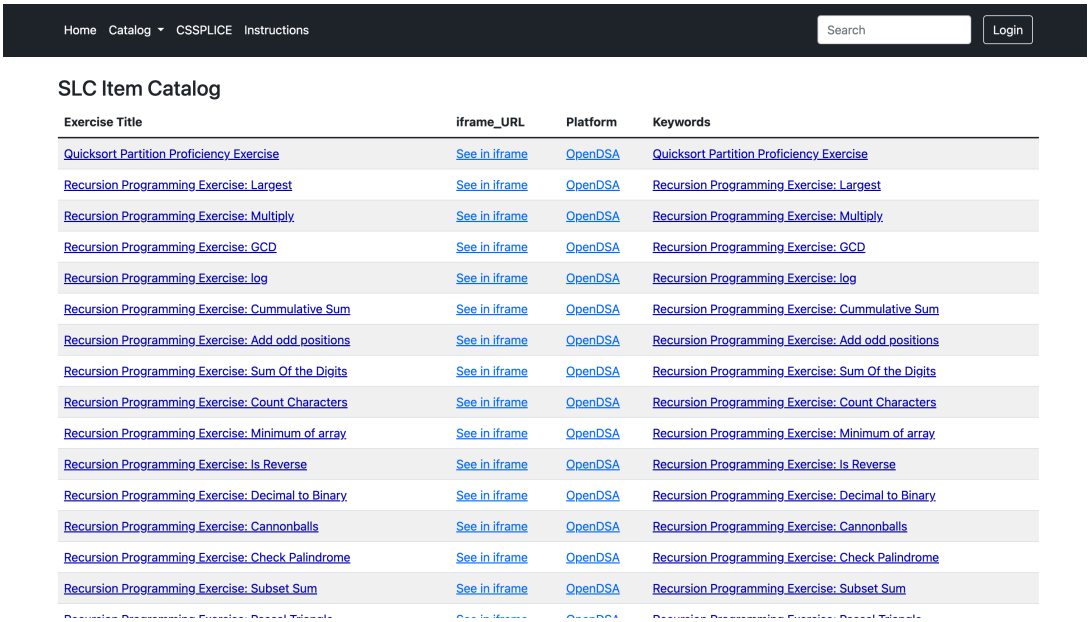

Figure 4.2: A collection of OpenDSA exercises in the Smart Learning Content (SLC) catalog.

4.1.2 Current State of the Catalog

As described in Sub-section 4.1.1, the Smart Learning Content (SLC) catalog supports three primary resource types: Tools, Content Items, and Datasets. Each entry, whether a self-contained tool, a learning exercise, or a data set, includes metadata that facilitate efficient search and organization within the catalog. This metadata specifies critical information such as title, keywords, and platform support, ensuring that users can discover and integrate relevant resources into their educational workflows.

As of this writing, the catalog comprises 5 Tools, 1,741 Content Items, and 34 Datasets. Each resource entry is accompanied by structured metadata such as title, author, keywords, and optional descriptors such as language or platform support that aid in both classification and retrieval. The foundation of the organization lies in the ontology consisting of 224 nodes (classes) and 256 links (relationships). These relationships form a Directed Acyclic Graph (DAG), enabling concepts and subconcepts to overlap without creating cycles.

Figure 4.2 shows the catalog's main interface, displaying a paginated list of all Smart

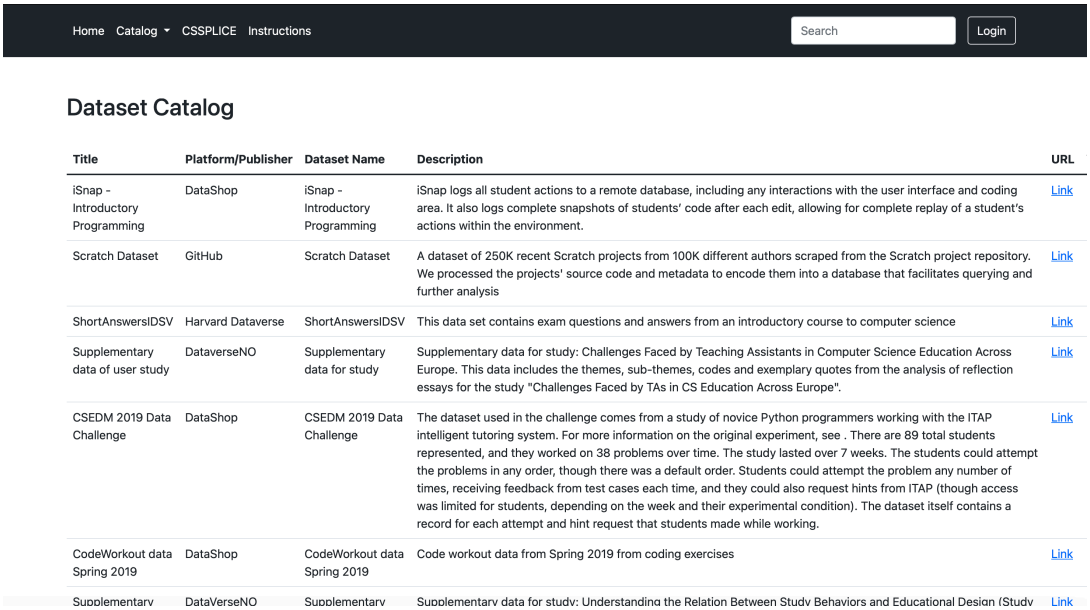

Figure 4.3: Collection of datasets featured in the Smart Learning Content (SLC) catalog.

Learning Content items. Users can browse through this list or conduct simple searches to quickly locate specific resources.

4.1.3 Metadata for Catalog Entries

The SLC catalog is designed as a repository for learning content, tools, and datasets. Each entry in the catalog is composed of structured metadata that aids in categorizing, searching, and understanding the content. This metadata is essential not only for accessing the content, but also for organizing and categorizing entries within the catalog, thereby enabling efficient search functionalities.

For the tools section of the catalog, each tool is represented with specific metadata fields that provide detailed information about the tool's characteristics and how it can be integrated into various educational platforms. The core fields of the SPLICE tools catalog include:

- **Platform Name**: The name of the platform hosting the tool.
- **URL**: The web address where the tool can be accessed.

- **Tool Description**: A detailed description of the tool and its functionalities.
- **License**: Information about the licensing of the tool.
- **Standard Support**: Details on the standards the tool supports, such as compliance with Learning Tools Interoperability (LTI) or other interoperability standards.
- **Keywords**: A list of terms that describe the tool, facilitating search and categorization.
- **Contact**: An email address for support or further inquiries about the tool.

Similarly, for content items in the catalog, each entry includes metadata fields that provide essential information about the learning materials. The core fields for content items are:

- **Exercise Name**: The name of the exercise or learning material.
- **Author**: The creator of the exercise.
- **URL**: The Web address where the exercise can be accessed.
- **Keywords**: A list of terms that describe the content.
- **Platform Name**: The platform hosting the exercise.
- **LTI and iFrame URLs**: URLs used for LTI (Learning Tools Interoperability) and iFrame integrations.
- **Description**: A textual description of the exercise content.
- **Institution**: The institution affiliated with the exercise (optional).
- **Exercise Type**: The type of learning content (e.g., assessment, quiz).

This metadata is essential for accessing content, organizing and categorizing catalog entries, and enabling efficient search functionalities.

Table 4.1: Validation rules for each Smart Learning Content item in the catalog.

Field Name	Type	Mandatory	Validation Rule
exercise_name	String	Yes	Not empty
author	String	Yes	Not empty
url	URL	Yes	Valid URL, Must be a syntactically valid URL and not empty
keywords	JSON Array	No	Optional, must be array
platform_name	String	Yes	Not empty
lti_url	URL	No	Valid URL or null
iframe_url	URL	Yes	Valid URL, not empty
description	String	Yes	Not empty
institution	String	No	Optional
exercise_type	String	No	Optional
catalog_type	String	Yes	Must match defined types

4.2 Validation Rules

The validation infrastructure enforces a strict set of rules to maintain consistency and correctness of content in the catalog. These rules are applied through Data Transfer Objects (DTOs) and Class-Validator Libraries in TypeScript, which ensure that content adheres to predefined metadata schemas. Table 4.1 summarizes the validation rules for each item in an SLC. They are categorized into three groups as follows:

- Required fields: Fields such as exercise_name, url, and author are mandatory, and missing values will trigger validation errors.
- Optional fields: Fields like keywords and institution are validated only if provided.
- Type validation: Validators ensure that field types (e.g., string, URL, array) are correct, using class-validator decorators like @IsString() and @IsUrl().

As shown in Table 4.1, each field has specific validation rules to ensure data integrity throughout the catalog. For URL fields, the validation checks that the URL is syntactically correct, which means that it follows the standard format with a valid protocol, domain, and path, if applicable. This helps prevent issues when users attempt to access

resources using these URLs. By enforcing these validation rules, we ensure that all entries in the catalog are consistent, accurate, and reliable, enhancing the overall user experience and maintaining the quality of the Smart Learning Content catalog.

4.3 Metadata Compliance

Ensuring metadata compliance is crucial to maintain consistency and adherence to specific standards across the catalog. The system utilizes Data Transfer Objects (DTOs) and the TypeScript Class Validator library to map the incoming metadata submissions and apply validation rules [3]. When a contributor submits a catalog entry, the system provides actionable feedback to address any underlying issues. Table 4.2 presents examples of feedback returned to the review dashboard for each Smart Learning Content (SLC) item. These issues do not prevent contributors from adding SLC items to the Smart Learning content catalog, unless they are classified as critical errors.

Errors encountered during validation are categorized into two types:

- Critical Errors: These are severe issues that prevent a catalog entry from being accepted into the system. For example, the absence of essential metadata fields such as exercise_name and author blocks the submission of the content [8].
- Minor Errors: These are minor issues, such as incorrect optional fields or inconsistent data formats, which can be corrected after submission. Minor errors are logged for later review but do not prevent the catalog entry from being added, allowing content to be enhanced post-submission.

Table 4.2 shows that each error type includes a description and is classified as critical or minor. By enforcing these validation rules, we ensure that all entries in the catalog are consistent, accurate, and reliable, enhancing the overall user experience and maintaining the quality of the Smart Learning Content catalog.

Table 4.2: Error categorization for Smart Learning Content catalog validation for each entry.

Error Type	Description	Category
Missing exercise_name	The exercise name is a required field.	Critical
Invalid URL	The provided URL is not valid or unreachable.	Critical
Missing platform_name	Platform name is required and cannot be empty.	Critical
Invalid keywords format	Keywords are expected to be in JSON array format.	Minor
Empty description	Description is mandatory and cannot be empty.	Critical
Optional fields missing	Optional fields such as institution are not filled.	Minor

4.4 URL Compliance and Validation

The URL validation process is essential to verify the accessibility of external resources linked within catalog entries. This process ensures that users can access the content referenced in the metadata, enhancing the overall reliability of the educational platform [44].

URL validation is crucial to ensure that the resources linked to the SLC catalog are accessible and usable. The system performs several checks:

- Basic URL Syntax Validation: Ensures that the URL conforms to standard URL formatting rules.
- Reachability Check: Using tools such as Axios, the system attempts to access the URL and checks if it returns a valid HTTP 200 response.

In cases where URLs fail to pass these checks, they are flagged and contributors are notified to provide valid, accessible URLs.

Validating Catalog URLs Using Axios

Axios [5] is a widely-used JavaScript HTTP client, it is utilized by the SLC catalog to send HTTP requests for validating the URLs provided in the metadata. Figure 4.4 shows an Axios implementation that validates URLs by checking their response status. If the request succeeds, the response is logged for feedback; otherwise, the error is handled and

```
import axios, { AxiosResponse, AxiosError } from 'axios';
axios.get(url)
   .then((response: AxiosResponse) => {
     // Handle success
     console.log(response);
   })
   .catch((error: AxiosError) => {
     // Handle error
     console.log(error);
   });
```

Figure 4.4: Code snippet demonstrating Axios implementation for handling HTTP requests in JavaScript.

logged. This approach ensures that submitted URLs are accessible and compliant with the catalog's standards. The response codes are handled as follows:

- 200 OK: The URL is considered valid and accessible.
- 4xx/5xx errors: URLs that return client or server errors are marked as invalid and flagged for review.

The validation process reduces the likelihood of broken links within the catalog, improving user experience and content quality.

4.5 Feedback Mechanism

The system includes a robust feedback mechanism for contributors, helping them identify and resolve errors before the content is approved. This feedback is crucial to maintain high-quality metadata and ensure the overall effectiveness of the educational content catalog.

Validation results are presented on the review dashboard in an accessible and user-friendly format, providing contributors with detailed feedback for each entry. Figure 4.5 displays

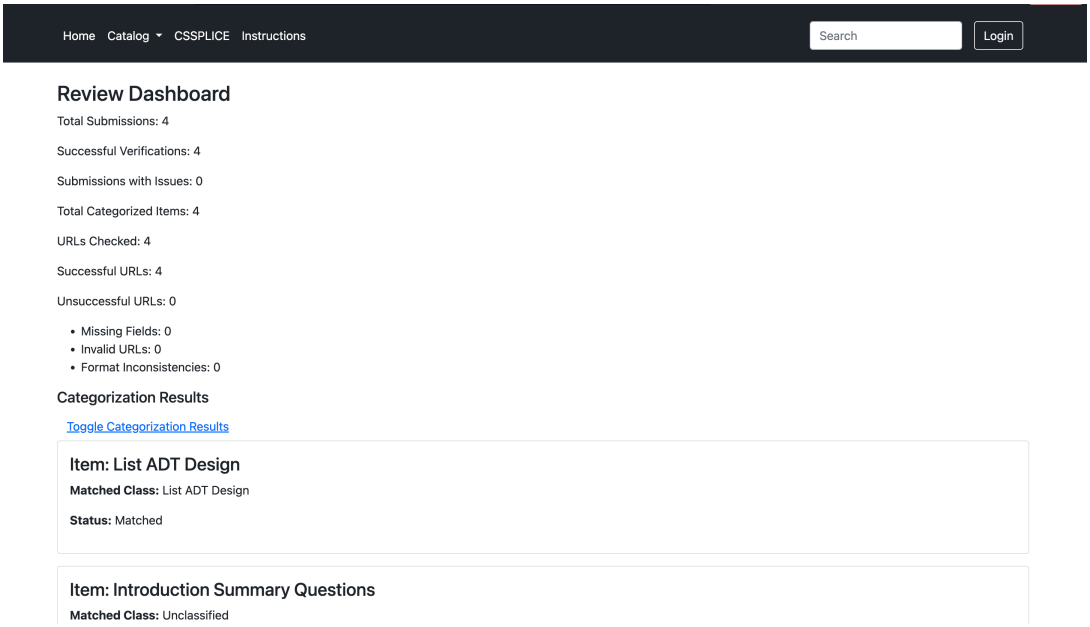

Figure 4.5: Review dashboard displaying validation results for a set of entries in the Smart Learning Content (SLC) catalog.

a screenshot of the validation results for several entries in the Smart Learning Content (SLC) catalog. The dashboard includes:

- Summary view: Displays a high-level overview of the validation process, including total submissions, successful verifications, and a count of critical and minor errors. This approach aligns with best practices in data quality feedback systems.
- Detailed view: The toggle entries show detailed information on validation issues, including specific field errors and recommended fixes. This granular feedback is essential for efficient problem resolution.

Pagination ensures that contributors can easily navigate through large numbers of submissions, adhering to user interface design principles for large datasets.

Chapter 5

Adding LTI 1.3 Support to OpenDSA

5.1 OpenDSA

OpenDSA is an interactive eTextbook platform offering comprehensive support for a wide range of computer science topics, including Data Structures, Algorithms, Formal Languages, Programming Languages, and more. As an open-source platform, OpenDSA operates as an LTI 1.1-powered tool provider, enabling integration with external tools. It facilitates seamless communication with Learning Management Systems (LMS), supporting features such as score reporting, progress tracking, and user authentication. OpenDSA offers a collection of exercise types, including Khan Academy-style exercises and algorithm visualizations developed using the JavaScript Algorithm Visualization (JSAV) library.

5.1.1 OpenDSA and LTI

Learning Tools Interoperability (LTI) is a standard developed by the 1EdTech Consortium (formerly IMS Global Learning Consortium) to enable the seamless integration of learning tools with Learning Management Systems (LMSs) such as Canvas, Moodle, and

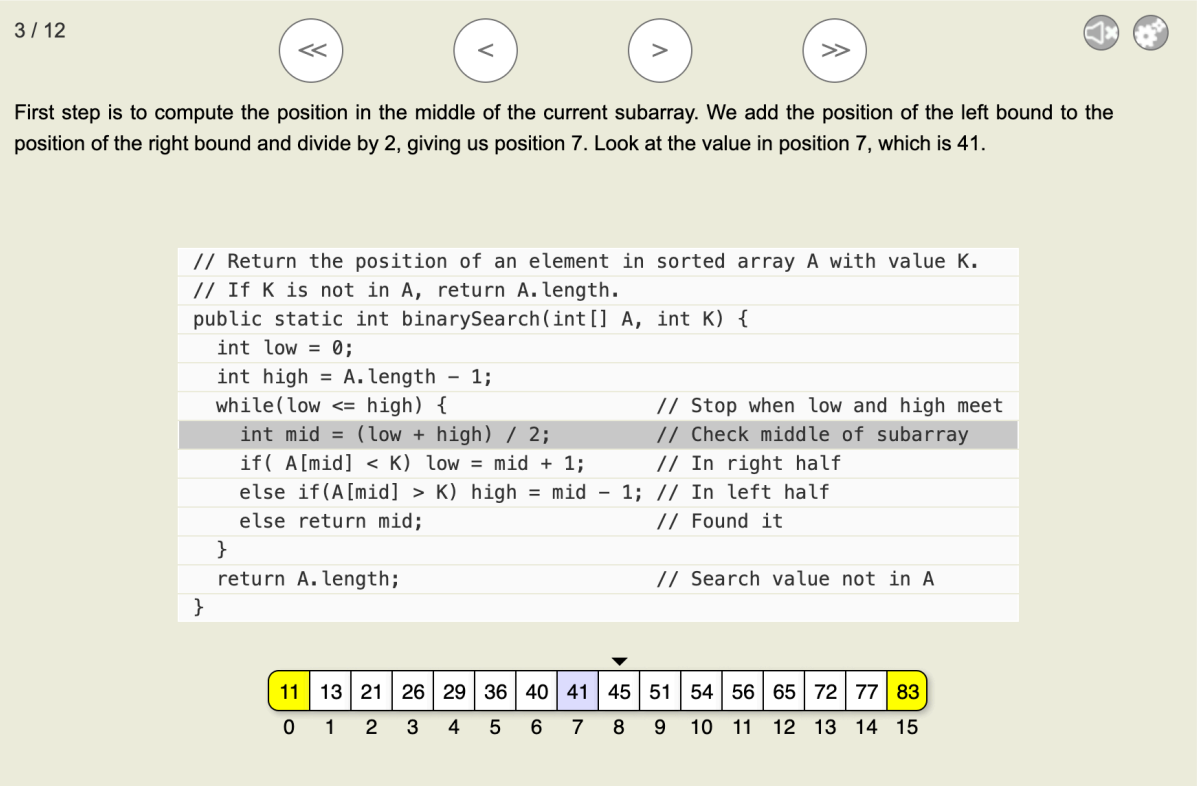

Figure 5.1: A JSAV-based OpenDSA slideshow illustrating the behavior of a binary search algorithm.

Blackboard [23, 49]. It provides a standardized way for third-party learning tools to interact with LMS platforms, ensuring secure data exchange and user management across these systems [60]. The evolution of LTI standards has been significant in terms of security, scalability, and flexibility. LTI 1.1, the prior iteration supported by OpenDSA, allows simple launches of external tools within an LMS using the OAuth 1.0a service; both the platform and tool relied on sharing a consumer key and having a consumer secret to establish a valid service. But LTI 1.1 had limitations in terms of authentication and robust user role management [49]. Many tools such as OpenDSA handled these limitations internally and have implemented workarounds to mitigate these role-management and authentication limitations.

5.2 LTI 1.3

The LTI 1.3 version resolves many of these issues by introducing OAuth 2.0 for authentication, JSON Web Tokens (JWT) for security, and expanded services such as Name and Role Provisioning Services (NRPS) for role management, and Assignment and Grade Services (AGS) for robust grade passback and targeted feedback [23].

Upgrading OpenDSA to support LTI 1.3 is crucial to enhance its interoperability with modern LMS services and to take advantage of the improved security and functionality offered by the new standard. The additional services provided by LTI 1.3, such as advanced role management through NRPS and more secure authentication mechanisms, provide a seamless and secure learning experience. By adopting the LTI 1.3 standard, OpenDSA enhances its integration capabilities, enabling instructors and students to interact with OpenDSA resources more seamlessly and effectively within their LMS environments. Furthermore, as Learning Management System (LMS) platforms increasingly adopt LTI 1.3 and move away from the deprecated LTI 1.1 standard, it is crucial for OpenDSA to update its integration capabilities to ensure compatibility and maintain relevance.

The remainder of this chapter discusses the process of upgrading OpenDSA to support LTI 1.3. We will explore the technical challenges encountered during the implementation, the solutions developed to address these challenges, and the benefits realized through the upgrade. This includes detailed explanations of how OpenDSA uses the new features of LTI 1.3, such as the OAuth 2.0 authentication flow, JWT security enhancements, NRPS for role management, and AGS for grade synchronization.

Figure 5.2 illustrates the relationships between platforms, tools, deployments, and LTI links, emphasizing the flow of information and interactions within the LTI 1.3 framework.

5.2.1 Components and Services in LTI 1.3

LTI 1.3 includes a number of components and services that work together to enable seamless integration between platforms and tools. This section highlights these core

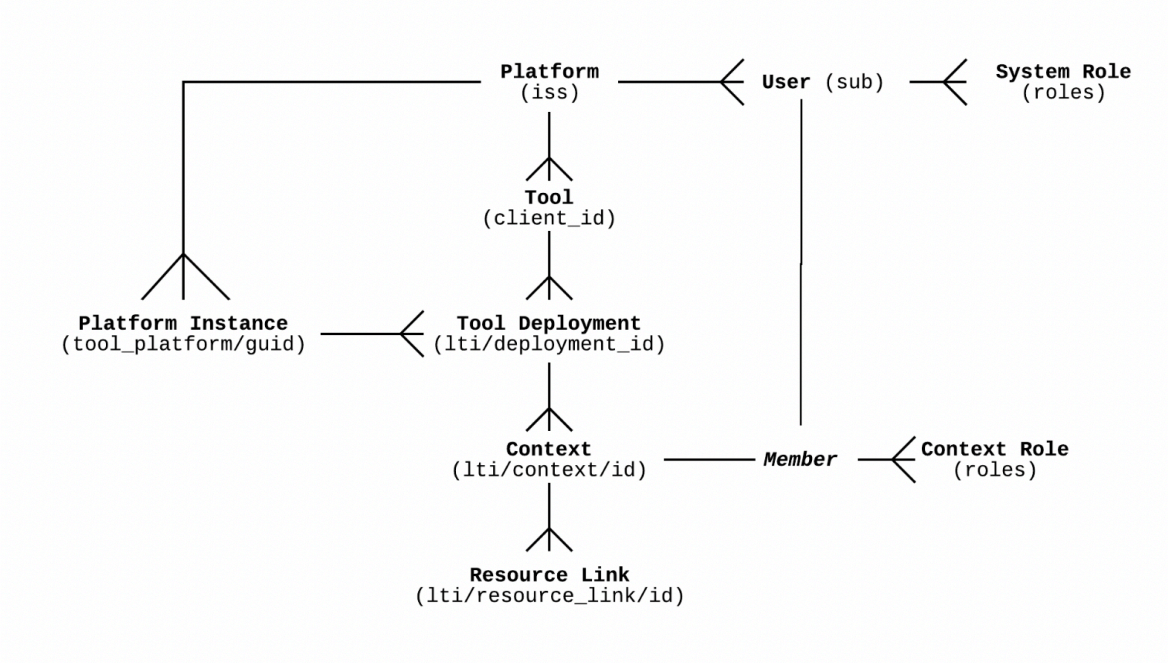

Figure 5.2: Diagram illustrating multiple main entities making the LTI 1.3 domain and their relationships [34].

components and their interactions.

1. **Platforms:**

   In the context of LTI, a platform (also known as an LTI Consumer) is the system that provides the learning environment—typically a Learning Management System (LMS) such as Canvas, Moodle, or Blackboard. The term "platform" is defined by the LTI standard to represent the LMS or any system that consumes LTI-compliant tools. Platforms orchestrate the delivery of content, modules, courses, and external tools to users within an educational context. To uniquely identify a platform or Learning Management System (LMS), LTI 1.3 introduces the iss (Issuer) parameter, typically represented as the platform's URL. This approach aligns with the standards used in OAuth 2.0 services. This ensures that every interaction between a platform and a tool is securely associated with the originating system. Additionally, the "sub" parameter represents the subject, which uniquely identifies the user in the context of the platform. This allows the tool to associate specific user actions with the correct platform account, providing a personalized and secure ex-

perience. platforms also manage key functionalities such as course context, roles, and deployment details that are essential for establishing a meaningful connection with external tools. These entities and their relationships are further depicted in Figure 5.2 illustrating how platforms, pools, and deployments interact within the LTI ecosystem.

2. **Tools:** A tool (or LTI provider) in the LTI 1.3 ecosystem is an external application integrated with a platform to provide specialized learning functionalities. These tools extend the capabilities of platforms by offering features such as interactive exercises, problem-solving environments, and automated grading systems. Tools rely on the client_id parameter to establish their unique identity within a platform's configuration. The client_id acts as a secure identifier that allows the platform to recognize and authenticate the tool during interactions, ensuring proper authorization and data exchange.

   By leveraging the client_id, tools can seamlessly integrate into the platform's environment while maintaining a high level of security and specificity in their operations. Tools also interact with other parameters, such as the sub (subject) passed from the platform, to personalize the learning experience for individual users. The relationships between platforms, tools, and deployments, including the role of client_id, are visualized in Figure 5.2, providing a comprehensive overview of the LTI architecture.

3. **LTI 1.3 Endpoints**: In the LTI 1.3 framework, the tool implements specific API endpoints that facilitate authentication and interaction flows between the platform and the tool. An "endpoint" refers to a URL exposed by the tool that the platform can call to perform various actions. These endpoints are crucial components of the LTI integration and include:

   - Initiate Login Endpoint: Used by the platform to start the OAuth 2.0 authentication flow with the tool.
   - Redirection URI: The endpoint on the tool where the platform redirects the

user after authentication, carrying authentication responses and parameters.

- Service Endpoints: Additional endpoints that support LTI Advantage Services like Names and Role Provisioning Services (NRPS) and Assignment and Grade Services (AGS).

These endpoints enable the platform to communicate with the tool, authenticate users, and utilize the services provided by the tool through standardized APIs defined by LTI 1.3.

4. **JSON Web Tokens (JWTs):** JWTs are compact, self-contained tokens used for secure transmission of user and context information between the platform and tool. They consist of three parts:

   - Header: The header contains metadata about the token type and hashing algorithm, which can be HS256 or RS256.
   - Payload: The payload carries claims about the user, roles, course, information about both applications and additional data relevant to both applications.
   - Signature: The signature ensures the integrity of the token by verifying that its content has not been tampered with. It is generated using the header and payload, which are signed with a secret key in symmetric encryption or a private key in asymmetric encryption.

   The JWTs are signed to prevent tampering and can be encrypted for added security. In LTI 1.3, JWTs are extensively used for passing launch parameters and user data securely between the platform and the tool.

5. **Security Framework:** The LTI 1.3 security model adopts industry-standard protocols to provide robust authentication and authorization mechanisms:

   - OAuth 2.0: A standard protocol that handles the authorization flow, allowing the tool to access resources on behalf of the user [51].

- OpenID Connect: An authentication layer built on top of OAuth 2.0, providing user authentication by verifying user identities.
- Private and Public Key Cryptography: The keys are used to sign and verify JWTs, adding an additional security layer to ensure the integrity and authenticity of the messages exchanged.
- State Parameter: The state parameter is an encoded JWT, which is a random string sent by the tool when starting the authentication request and validated by the platform when processing the response. It helps to protect against cross-site request forgery (CSRF) attacks by ensuring that the authentication response corresponds to the initial request made by the same user session.

This security framework ensures that only authorized tools can access user data and that user sessions are securely managed throughout the interaction between the platform and the tool.

6. **LTI Advantage Services:** LTI 1.3 introduces additional services, known as LTI Advantage Services, that extend the core functionality, providing robust roster management, grade passback, and content delivery:

   - Names and Role Provisioning Services (NRPS):
     - Allows tools to retrieve roster information from the platform.
     - Provides details about user roles and memberships status.
   - Assignment and Grade Services (AGS):
     - The AGS services enables tools to create, update, and read grade entries in the platform's gradebook.
     - Supports complex grading scenarios, including the use of rubrics and providing targeted feedback alongside grades.
     - It allows for seamless integration of external assessment tools with the platform's grading.
   - Deep Linking:

- Allows tools to add specific content items to courses such as text books, exercises, etc.
- Deep Linking also enables instructors to select and configure tool content directly within the LMS.
- Standardizes the method for embedding content, enhancing interoperability across different platforms without relying on platform-specific APIs.

5.2.2 LTI 1.3 Core

LTI 1.3 introduces a new security model based on OAuth 2.0 and OpenID Connect, which ensures a secure and standard authentication process [24, 34]. The key change is the use of JSON Web Tokens (JWTs), which are signed tokens that encapsulate user identity and other claims necessary for LTI launches [34]. JWTs ensure that each request made to the LMS is securely signed, providing authentication and authorization to users without exposing sensitive data [24]. This Figure 5.3 illustrates the interaction between an LMS, the LTI Core framework, and the OpenDSA system during an LTI launch process.

1. **Security Model**: LTI 1.3 enhances its security by using signed JWTs, OpenID Connect, and OAuth 2.0 flows for authentication. OpenID Connect functions as an identity provider. LTI 1.3 uses these services for robust authentication and authorization.

2. **OpenID Connect and OAuth 2.0**: OpenID handles user identity, as described above, by verifying the user's identity. This is accomplished through the platform acting as an OpenID provider. The platform authenticates the user and provides user data and roles to the tool as part of the launch process. OAuth 2.0 focuses on authorization, determining if a user has certain rights to access specific resources or functionalities, for example, an instructor and a student. In LTI 1.3 the tool requests an access token from the platform token endpoint, which is already provided during tool registration; then the tool includes a client assertion JWT in the

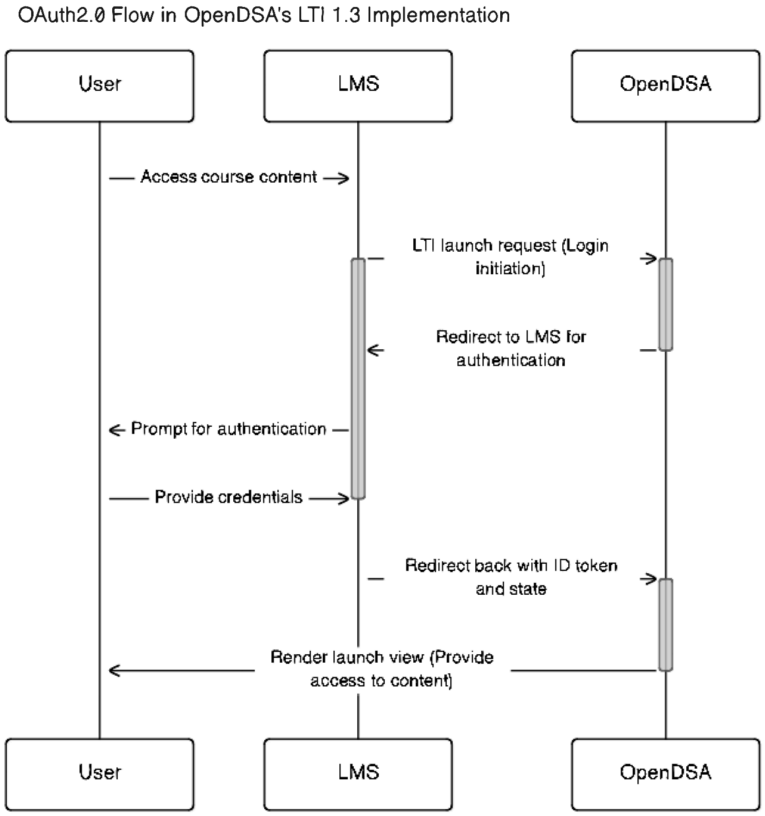

Figure 5.3: Overview of OpenDSA LTI 1.3 launch process with OAuth 2.0 flow.

token request. The JWT client assertion includes an issuer and subject known as "iss" and "sub", which is the OAuth2.0 client_id, which the platform issues during registration for the tool.

5.2.3 Names and Roles Provisioning Services (NRPS)

NRPS, as described above, is an LTI 1.3 Advantage service that allows the tool provider to request the course roster, which includes user names and roles [24]. This service enables the tool to identify user roles (e.g., instructor, learner, admin) and ensure access permissions are granted appropriately based on their role [24]. This service is essential in scenarios for courses in which the instructor must differentiate between students, teaching assistants, and other roles [24].

NRPS Process Flow

1. The tool initiates a request to the LMS's NRPS endpoint, often specified in the LTI launch message using this claim, `https://purl.imsglobal.org/spec/lti-nrps/claim/namesroleservice`. This claim contains the endpoint URL where the tool can retrieve user and role information, along with the associated scopes needed for access.

2. The LMS authenticates the request using LTI 1.3 JWT security mechanisms, which include launch claims and LTI messages for role request validation. The tool must include a valid access token in the request, which is obtained using the OAuth 2.0 client credentials flow. This ensures that only authorized tools with proper credentials can access the NRPS endpoint.

3. Upon successful authentication, the LMS responds with a structured data format, typically as a JSON object, containing the roster information. The `application/vnd.ims.lti-nrps.v2.membershipcontainer+json` media type is commonly used for this response.

    - The JSON response includes key details about the course roster, such as:
        - **User IDs**: Unique identifiers for each user in the course.
        - **Roles**: Information about the user's role within the course, such as `Instructor`, `Learner`, or `Administrator`.
        - **Names**: Full names of the users (if permissions allow).
        - **Email Addresses**: Email of the individual (optional and subject to privacy settings).
        - **Context Information**: Details about the course or group the user belongs to.

4. The tool processes the data to support its functionality, such as enrolling the user in a course offering and rendering the appropriate view based on their role.

5.2.4 Assignment and Grade Services (AGS)

Assignment and Grade Services (AGS) play a critical role in managing and synchronizing grade data between tools like OpenDSA and Learning Management Systems (LMSs). AGS was introduced in LTI 1.1, enabling basic grade passback functionality where tools could send grades to the LMS. However, the implementation of AGS in LTI 1.3 improves this functionality by introducing more features and better security data in the LTI workflows. Although LTI 1.1 AGS focused primarily on a single action, sending grades to the LMS, the LTI 1.3 AGS framework expands this capability to include more interactions, such as managing gradebook columns, sending detailed progress updates, and retrieving grading results [24, 34].

The AGS feature allows the LMS to pass back grades from the OpenDSA to the LMS. When a student completes an exercise or assignment in OpenDSA, AGS sends the grade to the LMS gradebook [24]. It uses the LTI 1.3 security model, which uses OAuth 2.0 and JSON Web Tokens (JWT) to ensure secure communication. For LTI 1.3 AGS features to work, the LTI Advantage service must be enabled during the learning tool registration phase, as shown in Figure 5.4.

Components of AGS

LTI 1.3 AGS introduces three services that collectively enhance the grade management process:

1. **Line Item Management:** The tool can create, read, update, and delete line items, which represent gradebook columns within the LMS. This flexibility allows tools to dynamically manage gradebook entries to align with course content and assessment needs.

2. **Score Reporting:** The tool can send score updates directly to the LMS gradebook. This includes not only the scores but also optional comments and progress indicators, providing detailed feedback to both instructors and students. For exam-

ple, OpenDSA uses this feature to record students' scores as they progress through assignments.

3. **Result Retrieval:** The tool can retrieve LMS grade data for specific users and line items. This enables the tool to generate reports based on a student's progress.

AGS Workflow for Proficiency-Based Exercises

OpenDSA uses AGS to submit scores for proficiency-based exercises, such as interactive exercises built using the Khan Academy exercise framework [46] or programming exercises created in CodeWorkout. This workflow ensures that student performance is automatically synchronized with the LMS gradebook, improving efficiency and accuracy. The AGS workflow in OpenDSA involves two major steps: creating assignments and their associated gradebook columns (line items) and submitting grade updates for completed exercises.

A critical aspect of this workflow is the use of scopes. In the context of LTI 1.3, a scope refers to a specific permission assigned to a tool during its registration with the LMS. Scopes define what actions the tool is authorized to perform, such as managing line items or submitting grades. For example, the scope `https://purl.imsglobal.org/spec/lti-ags/scope/lineitem` allows the tool to create and manage gradebook columns, while the scope `https://purl.imsglobal.org/spe c/lti-ags/scope/score` authorizes it to submit scores. This system generates line items for proficiency exercises and submits scores as students achieve proficiency. This streamlined process, combined with the enhanced security features of LTI 1.3, ensures that the grades are accurately recorded and transmitted securely.

The following subsections outline the detailed process OpenDSA follows to manage AGS interactions for proficiency-based exercises.

1. **Creating Assignments and Line items**: Proficiency exercises are associated with line items in the LMS gradebook. The process of creating these line items involves the following steps:

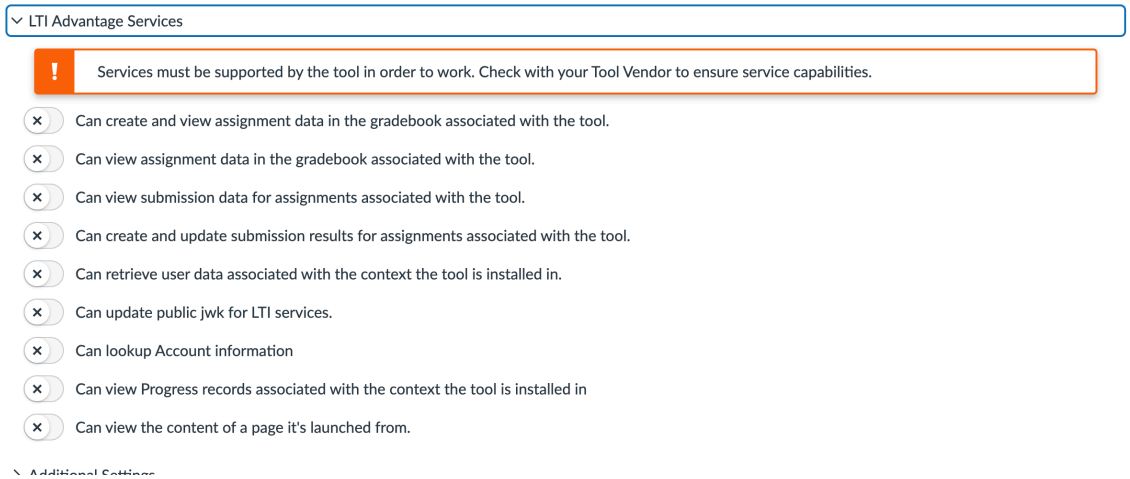

Figure 5.4: Canvas screen for learning tool registration with an option to enable LTI Advantage.

- **Authorization**: The tool's ability to manage line items is determined by the authorization scope granted during tool registration in the LMS. For example, the scope, `https://purl.imsglobal.org/spec/lti-ags/scope/lineitem` permits full management of line items. Figure 5.4 illustrates the configuration process during tool registration, where LTI Advantage services like AGS are enabled.
- **Requesting Line Item Endpoint URL**: The tool obtains the line item endpoint URL from the LTI launch claims under the `https://purl.imsglobal.org/ spec/lti-ags/claim/endpoint claim`. This URL is used to interact with the AGS line item service.
- **Constructing the Line Item Definition**: The tool creates a JSON object that defines the line item. The JSON object includes the following attributes:
  - **scoreMaximum**: This specifies the maximum score for the exercise.
  - **label**: Provides a descriptive label for the line item.
  - **resourceId**: An identifier for the resource within the tool associated with the line item.
  - **startDateTime and endDateTime (optional)**: Define the active period for submissions.

- **Sending a POST Request**: The tool sends a POST request to the line item endpoint URL. The body of the request contains the line item defined in the JSON above in the application/vnd.ims.lis.v2.lineitem+json media type.
- **LMS Response**: Upon successful creation, the LMS responds with an HTTP 201 (Created) status code, indicating that the line item has been added to the gradebook.

2. **Submitting Grade Updates** After students complete proficiency-based exercises, OpenDSA submits grade updates to the LMS through the AGS Score service. This process involves the following.

   - **Authorization**: The tool must have scope `https://purl.imsglobal.org/spec/lti-ags/scope/score`, which allows it to submit scores for existing line items.
   - **Constructing the Score Object**: The tool creates a JSON object that represents the score to be sent to the LMS. The structure of this score object includes:
     - **scoreGiven**: The actual score earned by the student.
     - **scoreMaximum**: The maximum possible score for the exercise.
     - **comment(optional)**: Additional feedback or notes for the student.
     - **activityProgress**: Indicates the status of the exercise (e.g., Completed).
     - **gradingProgress**: Indicates the grading status (e.g., FullyGraded).
   - **Sending a POST Request**: The tool sends an HTTP POST request to the score endpoint URL. This URL is derived by adding /scores to the line item ID URL. The body of the request contains the score object serialized as a JSON document with the application/vnd.ims.lis.v1.score+json media type.
   - **LMS Response**: If score submission is successful, the LMS responds with an HTTP 204 status code (no content), indicating that the score was accepted.

5.3 LTI 1.3 Integration in OpenDSA

Integrating LTI 1.3 into OpenDSA required a careful approach, taking into account OpenDSA's architecture, the existing implementation of LTI 1.1, and user and score management considerations. The first step was to build the endpoints and controllers specified by IMS global to support the authentication and launch request from the LMS, secondly include model to support registering an LMS and as an LTI 1.3 type. Finally, implement a secure LTI 1.3 core along with LTI 1.3 Advantage services [20].

5.3.1 LTI 1.3 Endpoints in OpenDSA

LTI 1.3 endpoints form the basis of the interaction between OpenDSA and Learning Management Systems (LMS). These endpoints facilitate authentication, content delivery, and secure communication, and support features such as role provisioning, grade management, and deep linking. Each endpoint serves a specific function in the LTI launch process, contributing to the overall workflow. This section provides an overview of the key endpoints implemented in OpenDSA to support LTI 1.3 functionality.

1. **Login Initiation Endpoint**: The login initiation endpoint facilitates the initiation of the LTI 1.3 launch process. This endpoint receives login requests from the LMS, validates them, and prepares the authentication flow. This serves as the entry point for any LTI interaction with OpenDSA and is accessed via a designated endpoint path: `/lti13/login_initiations`.
2. **Launch Endpoint**: The LTI 1.3 endpoint: `/lti13/launches` handles the actual LTI 1.3 launch request from the LMS. This is where the user is redirected after successful authentication and then OpenDSA receives the necessary information to render the requested content.
3. **Deep Linking Endpoint**: This endpoint is responsible for the selection of content elements and deep linking, handles the selection and return of specific resources, in which is essentially an OpenDSA book instance compiled into a course in the LMS.

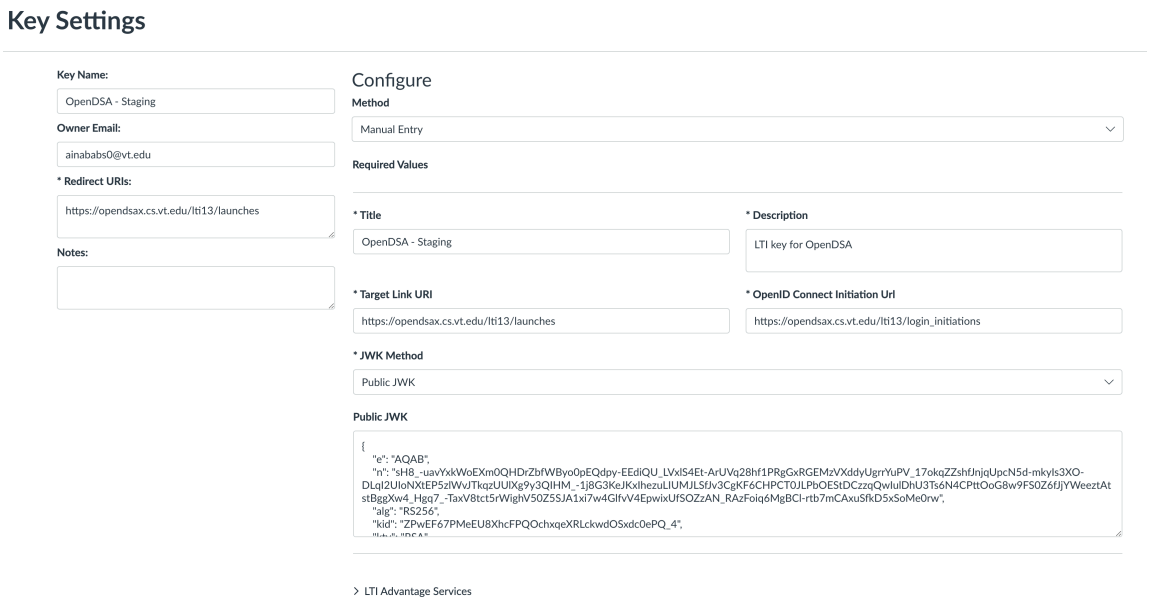

Figure 5.5: Adding OpenDSA as an LTI 1.3 tool to the LMS: tool registration form from Canvas's admin page.

The deep link launch endpoint is accessed through the path: `/lti13/deep_link_launches`.

4. **JWKS endpoint**: The endpoint provides the public keys required for JWT (JSON Web Token) validation. LMS platforms use this endpoint to verify the authenticity of tokens exchanged during the LTI launch process, ensuring secure communication. The endpoint is accessed through the path: `/.well-known/jwks`.

5.3.2 Tool Registration

The tool registration process requires the creation of a developer key in Canvas, which specifies the LTI 1.3 endpoints that the LMS will use to interact with OpenDSA. To successfully add tools with LTI 1.3 support, access to the LMS admin role is essential, as the tool registration process involves administrative privileges. Due to the lack of administrative access to the Canvas public instance, we deployed a dedicated instance of Canvas on Virginia Tech's Kubernetes cluster to address this limitation and enable testing and integration. Figure 5.5 illustrates the tool registration form in Canvas, showcasing the fields required to add an LTI 1.3-compliant tool.

Once the developer key is generated, it is used to authenticate OpenDSA during interactions with the LMS. This key allows OpenDSA to receive requests, launch content, and report back grades securely [46]. Additionally, the developer key can be applied across multiple courses, as the LMS generates unique deployment IDs for each course. Using these deployment IDs, OpenDSA tailors its content delivery based on the course and the user context [17].

Configuration Settings in OpenDSA

After OpenDSA is registered as an external tool provider with the LMS, the platform provides OpenDSA with a developer key, which is also a client ID. The tool provider (OpenDSA) must provide its client ID when requesting tokens from the LMS token endpoint. To complete the registration process, the LMS provides the tool with the following endpoints:

1. **Token Endpoint and Access Tokens**: OpenDSA uses this LMS token endpoint to request access tokens for authorized service requests.
2. **JSON Web Keys (JWKs) URL**: The JWKs URL provides an endpoint containing a list of public keys used by the LMS to sign the LTI 1.3 launches.
3. **OIDC Endpoint**: The OIDC authorization request endpoint is the URL that OpenDSA uses to redirect back to the LMS after the first leg of an LTI 1.3 launch.

5.3.3 Models and Database Schema

The transition to LTI 1.3 in OpenDSA required updates to the database design to accommodate its enhanced features and security requirements. As part of this process, a new table, lti_launches,was created to store launch-related data. This table holds critical information such as multiple ID tokens, the JWT state, and other LTI 1.3 claims. Specifically, the lti_launches table tracks data about each LTI 1.3 launch, including the user ID that initiated the launch, the course offering ID, and the LMS JSON Web Token

(JWT) associated with the launch (commonly referred to as the ID token). By capturing this information, OpenDSA can effectively manage user sessions and facilitate secure interactions with the LMS [20].

OpenDSA's database includes several existing tables that facilitate its interaction with LMS platforms and users. These tables facilitate access to OpenDSA content through the LMS. The lti_launches table is linked to key tables such as lms_instances, users, and course_offerings, ensuring that OpenDSA can track, validate, and manage LTI launches effectively. The following is an overview of these tables and their role in the system.

1. **LMS Instances Table**: The lms_instance table in OpenDSA stores information about registered LMS instances integrated with OpenDSA, supporting both LTI 1.1 and LTI 1.3 standards.

2. **Users Table**: The users table stores information about individuals interacting with OpenDSA course offering through the LMS.

3. **Course Offering Table**: The course_offering table maintains a reference to specific resources, such as an OpenDSA book instance, associated with a particular course within an LMS instance.

4. **LTI Launches Table**: The `lti_launches` table maintains a record of each LTI 1.3 launch and references key fields such as lms_instance_id, user_id, and course_offering_id. This table enables OpenDSA to manage individual launch sessions and verify the validity of the JWT (JSON Web Token) before processing user requests. The decoded_jwt field stores the decoded contents of the JWT for convenient access, while the kid (Key ID) field identifies the key used to sign the JWT, ensuring robust security verification [17, 20]. Table 5.1 provides an overview of the fields captured by the lti_launches table.

Table 5.1 shows the fields captured by the lti_launches table. The id_token field is designed as a temporary field with an associated timestamp, as ID tokens are short-lived and typically expire within 3600 seconds. This design ensures that expired or

Table 5.1: Schema of the LTI launches table in OpenDSA for LTI 1.3 launches.

Column Name	Data Type	Description
`id`	`bigint`	Primary key. Unique ID for each LTI launch.
`lms_instance_id`	`integer`	Foreign key referencing the `lms_instances` table.
`user_id`	`integer`	Foreign key referencing the `users` table.
`course_offering_id`	`integer`	Foreign key referencing the `course_offerings` table.
`id_token`	`text`	Stores ID token from LMS during launch.
`decoded_jwt`	`json`	Decoded JWT payload from ID token.
`kid`	`string`	Key ID from LMS's JSON Web Key Set (JWKS).
`expires_at`	`datetime`	Token expiration timestamp.
`created_at`	`datetime`	Launch creation timestamp.
`updated_at`	`datetime`	Last update timestamp.

invalid tokens are not stored in the database. Figure 5.6 illustrates an entity-relationship diagram representing the structure of the tables.

5.3.4 LTI 1.3 Core in OpenDSA

The implementation of LTI 1.3 Core in OpenDSA builds on the security and authentication framework outlined earlier in Section 5.2.2, leveraging OAuth 2.0 and OpenID Connect for secure communication between the LMS and OpenDSA.

1. **OpenID Connect ID Login Initiation**: The OpenID Connect ID (OIDC) Login Initiation is the initial step in launching OpenDSA as an LTI 1.3 tool. The LMS initiates the process by sending a login request to OpenDSA. This sets off a series of interactions between the LMS and OpenDSA which includes validation correct credentials such as the LMS instance, client ID, and other endpoints mentioned above during the tool registration and redirected to OpenDSA to launch as an LTI tool. Figure 5.8 illustrates a detailed look.

2. **Launch Request**: The LTI launch request is received by OpenDSA, the request contains a JSON Web Token (JWT) that encapsulates the user's identity, roles, and other related information such as the course ID and resource being requested [46]. The launch flow begins by decoding the JWT and validating the claims it con-

tains. In this context, a claim refers to the information contained within the JWT payload about the user, the LMS, or the context of the request. Validating these claims ensures that the request is legitimate and that the user has the appropriate permissions [46]. Figure 5.7 illustrates the code snippet responsible for decoding the platform JWT in the LTI 1.3 service.

3. **Resource Selection**: Once the JWT is validated, OpenDSA utilizes the claims within it to determine which resources (e.g., books or exercises) should be displayed. In this context, the resource is typically a book instance compiled for the LMS course [46]. The handle_resource_link_request method in the LTI 1.3 Controller identifies the correct material. It queries the OpenDSA server to retrieve the content based on the module and book path claims embedded in the JWT [46].

4. **User Enrollment and Course Mapping**: After retrieving the content, OpenDSA verifies the user's enrollment status within the course. This process follows two approaches. First, it queries the lms_instance and course_offering tables, which store detailed information about the LMS course and its corresponding OpenDSA course offering [46]. Using the data provided in the launch request, OpenDSA attempts to find the user with an existing record. If no matching record is found, OpenDSA proceeds to the second approach, utilizing the NRPS service.

   The NRPS service retrieves enrollment details directly from the LMS, including information about students already registered at the LMS level, and updates OpenDSA's records to enroll the user accordingly. [46]. The launch request includes details about the user's role in the LMS (e.g., Canvas). If the user is identified as an instructor for the Canvas course, they are enrolled as an instructor in the corresponding OpenDSA course offering. Otherwise, they are enrolled as a student [46]. Additionally, an entry is created in the lti_launches table, the schema of which is depicted in Table 5.1.

Figure 5.6 shows the database tables involved in the OpenDSA LTI 1.3 launch process, describing the flow from the initial launch request to the rendering of the content. Fur-

thermore, Figure 5.8 provides a comprehensive sequence diagram of the LTI 1.3 workflow, detailing each step from the initiation of the launch to the delivery of the selected content.

5.3.5 Names and Roles Provisioning Services (NRPS) in OpenDSA

The NRPS functionality allows OpenDSA to request user data (such as names and roles) from Canvas [19]. When a user launches OpenDSA from Canvas, the tool can request a list of all users in the course and their associated roles. This is done through the request_names_and_roles service, which OpenDSA calls upon receiving the launch request [33]. The decoded JWT contains role claims, which allow OpenDSA to identify whether a user is an instructor, student, or administrator [38]. Based on this role, OpenDSA determines the level of access the user has to course materials and tools.

Implementation of NRPS in OpenDSA:

1. **Endpoint Setup**: The endpoint setup for NRPS in OpenDSA establishes the communication pathways required to retrieve participant information, including names and roles, from the Learning Management System (LMS). Below are the endpoints for retrieving user roles and information:

    - **/lti13/names_roles**: This handles NRPS requests to retrieve names and roles used for NRPS. It is essentially a GET request /lti13/names_roles that retrieves the information of the participants from the LMS.

2. **JWT Validation and Access Token Retrieval**: Using the NPRS service, OpenDSA validates the JWTs (JSON Web Tokens) provided by the LMS for NRPS requests for the authenticity and integrity of the request. Then some information is extracted from the request, which includes the user information and the URL for accessing the LMS's NRPS endpoint (context_memberships_url). After validating the JWTs, OpenDSA sends a request to the LMS token endpoint to obtain an access token.

3. **Fetching and Parsing the Participant Data**: OpenDSA uses the access token obtained in the previous step from the LMS to fetch participant data from the LMS's NRPS endpoint. The request is an HTTP GET request to the context_memberships_url obtained from the decoded JWT, and includes the access token in the Authorization header. The LMS then responds with a membership container data structure.

4. **Storing and Using Participant Data**: The participant information defines users enrolled in the course at the LMS level. OpenDSA utilizes these data to enroll participants in the course on the OpenDSA server.

5.3.6 Assignment and Grade Services (AGS) in OpenDSA

The Assignment and Grade Services (AGS) in LTI 1.3 represent an improvement over their predecessor in LTI 1.1, offering improved grade management capabilities and security features. As described in Section 5.2.4, the AGS framework consists of three key services: line item management, score reporting, and result retrieval services. Together, these services facilitate seamless grade synchronization between OpenDSA and Learning Management Systems (LMSs). After a student completes an exercise, OpenDSA records the score on the server and then utilizes the AGS service to send the grade back to LMS via a POST request to the LMS AGS endpoint; Figure 5.9 illustrates the invocation of the AGS service.

The AGS service delivers an enhanced grade interactions, which includes a feature to set maximum points for each item [33]. The AGS comprises three subservices: Line item service, result service, and score service. [19]. The LineItem service allows for the management of grade objects, the result service enables retrieving current grades from the LMS's gradebook, and the score service facilitates the posting of scores by OpenDSA [19]. To enable these interactions, specific endpoints must be set up to handle requests between OpenDSA and the LMS securely and efficiently.

1. **Endpoints Setup**: The endpoint setup for AGS in OpenDSA establishes the communication channels required to interact with the LMS. These endpoints serve as interfaces for OpenDSA to manage line items, report scores, and retrieve results. Below are the specific endpoints and their roles in the AGS workflow:

    - **/lti13/line_items**: This service is used for managing line items, which represent gradebook columns. It enables OpenDSA to create, read, and update these entries. Line item information can be retrieved through GET requests to the /lti13/line_items endpoint.
    - **/lti13/results**: Retrieval of results (grades) from the platform's gradebook for a specific line item. This service is read-only and retrieves results via GET requests to /lti13/results.
    - **/lti13/scores**: This service enables OpenDSA to synchronize grades with the LMS by sending score updates. These updates are transmitted using POST requests to /lti13/scores and can include details such as scoreGiven, scoreMaximum, comment, and gradingProgress.

2. **JWT Validation and Access Token Retrieval**: The token retrieval process for the Assignment and Grade Services (AGS) follows the same procedure as that described for the NRPS service. OpenDSA validates the JWTs (JSON Web Tokens) provided by the LMS for authenticity and integrity of the request. Then some information is extracted from the request, which includes the user information and the URL for accessing the LMS's AGS endpoint (context_memberships_url). After JWT validation, OpenDSA requests an access token from the LMS token endpoint.

3. **Assignment Creation and Retrieval**:

    OpenDSA leverages the line item endpoint to interact with the Learning Management System's (LMS) Assignment and Grade Services (AGS) for creating and retrieving assignments. The `line_items#create` action in OpenDSA constructs and sends a JSON payload containing key assignment details such as scoreMaximum,

label, resourceId, tag, startDateTime, and endDateTime to the LMS. This payload is transmitted via an HTTP POST request to the line item container endpoint, which is derived from the JWT provided during the LTI launch.

4. **Grade Submission Process**: The OpenDSA post grade service is responsible for submitting grades to the LMS. It constructs a score object that includes details like timestamp, scoreGiven, scoreMaximum, comment, activityProgress, gradingProgress, userId, and scoringUserId. This score object is serialized into a JSON payload and sent as an HTTP POST request to the LMS's score endpoint for the corresponding line item. A successful submission would typically result in an HTTP 204 (No Content) response from the LMS. Figure 5.9 provides a simplified illustration of how the postgrad service is invoked to submit grades. Error handling mechanisms would be in place to manage situations like invalid access tokens, expired tokens, or issues with the score data. OpenDSA might retry failed submissions, log errors, or notify users if grade submission encounters problems.

5. **Storing and Managing Assignments/Grades in OpenDSA**: In OpenDSA, the process of transmitting scores back to the LMS through LTI 1.3's Assignment and Grade Services (AGS) begins by storing student scores in the odsa_module_progress table. This table tracks individual progress and scores for each module or exercise completed. To validate the score submission, OpenDSA retrieves the corresponding id_token from the lti_launches table, which logs each LTI session with details like user ID and course offering. Using the id_token, OpenDSA sends a secure request to the LMS's AGS post_score endpoint. This request includes the score data and a specific line item linked to the assignment or module. The LMS verifies the id_token, authenticates the request, and records the score within its gradebook, associating it with the appropriate student and assignment. This streamlined process ensures that scores are transmitted securely and in compliance with LTI 1.3 standards, ultimately creating a cohesive grading experience within the LMS.

5.4 Challenges and Solutions

Adding LTI 1.3 support to OpenDSA introduced several challenges due to the security protocols and the expanded functionality of the standard. These challenges primarily revolved around ensuring secure interactions, handling complex multi-course environments, managing dynamic course-specific data, and LTI version within OpenDSA. This section discusses key challenges encountered during implementation and the solutions employed to address them.

Security Issues

One primary challenge was to ensure that the implementation adhered to the security standards of OAuth 2.0 [51] and OpenID Connect specifications, also to securely manage JSON Web tokens (JWTs) and prevent unauthorized access to course materials [13]. The integration demanded secure management of JSON Web Tokens (JWTs), which are used to authenticate requests. Properly generating and signing JWTs was critical, as any discrepancies in token creation resulted in validation errors. Furthermore, the process of generating access tokens required careful handling of client credentials and secure interaction with the LMS authorization endpoints.

To address these challenges, the implementation of LTI 1.3 Core includes a robust token validation mechanism that ensure the JWTs are checked against essential claims, such as the issuer (iss), audience (aud) and expiration (exp). The LTI Core also incorporates standard encryption practices and secures JWTs by signing them with the RS256 algorithm. These measures ensure both the integrity and authenticity of each token. Access token exchanges are also streamlined through processes that interact securely with the authorization endpoints of the LMS, minimizing risks such as replay attacks or expired tokens. These measures collectively fortified the security of OpenDSA integration with LTI 1.3.

Handling Multiple Courses

Managing multiple courses in OpenDSA required a robust approach to ensure seamless integration and secure content delivery. To achieve this, OpenDSA leveraged deployment IDs, claims processing, and a carefully designed database schema to streamline operations and effectively support multiple LMS courses. The deployment IDs, assigned by the LMS during the tool registration process, serve as unique identifiers for each course. This approach eliminates the need for separate developer keys for individual courses, reducing manual configuration and potential errors. By associating these deployment IDs with course-specific data, OpenDSA can differentiate between courses while maintaining a unified tool registration. Then we retrieve the content for each course. Building on the existing functionality of the module and book path claims of LTI 1.1, OpenDSA continues to use these claims in LTI 1.3 to identify and compile course-specific resources. These claims were extracted from the JWT during the launch process.

The integration is further strengthened by OpenDSA's database design, which employs tables such as lms_instances, course_offering, and lti_launches. These tables work in harmony to establish and maintain relationships between LMS courses and their corresponding OpenDSA offerings.

Dealing with LTI versions

The handling of LTI 1.1 and LTI 1.3 within OpenDSA presented unique challenges, particularly in managing an LMS instance that supports both versions. While initial approaches considered splitting the lms_instances table into separate records for each LTI version, the current implementation follows a streamlined solution: maintaining a single record for both LTI versions of the same LMS instance. This approach accommodates the different attributes required by LTI 1.1 and LTI 1.3, such as OAuth 1.0 consumer keys for LTI 1.1 and public key sets for LTI 1.3.

To determine the LTI version in use, the course_offering table is updated with an LTI version type. The new field, course_offering_type, associates each course offering with its corresponding LTI version, enabling the system to infer the correct protocol and routing

logic. For example, courses registered with LTI 1.1 rely on oauth_consumer_key and lis_outcome_service_url for authentication and grade passback, while LTI 1.3 courses use ID tokens, deployment_id, and the Assignment and Grade Services (AGS) endpoints.

Backward compatibility in OpenDSA is ensured by carefully reviewing the scopes and claims provided in the LTI request. For example, scopes such as `https://purl.imsglobal.org/spe c/lti-ags/scope/lineitem` and claims such as `https://purl.imsglobal.org/spec/lti/ claim/deployment_id` serve as key indicators of LTI 1.3 requests. By analyzing these parameters, OpenDSA seamlessly supports both LTI 1.1 and LTI 1.3 versions.

Database Migrations and Scaling

The transition to support LTI 1.3 required updates to OpenDSA's database design, including the creation of the lti_launches table. The creation of the new table (lti_launches) required a lot of planning to avoid conflicts with the existing OpenDSA tables. One primary concern was ensuring that the new table could seamlessly integrate with the existing tables in OpenDSA while avoiding conflicts or redundancies. To facilitate connections between lti_launches and other key tables, such as lms_instances, users, and course_offerings, foreign key relationships were established that linked each LTI1.3 launch to its corresponding LMS instance, user, and course offering.

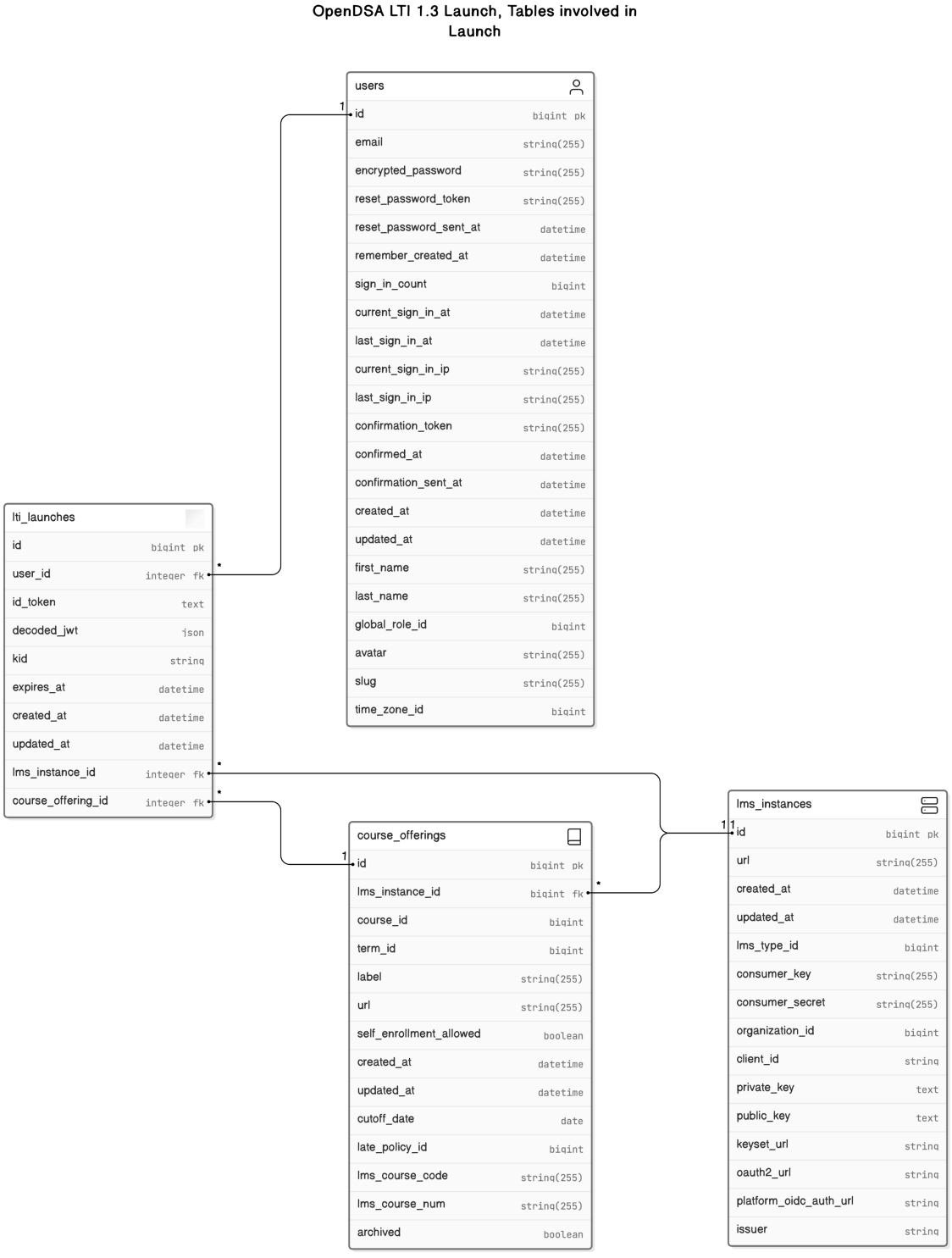

Figure 5.6: Entity-Relationship Diagram: Database structure of OpenDSA tables supporting LTI 1.3 launches.

```
#ruby
@decoded_jwt = Lti13Service::DecodePlatformJwt.new
(@lms_instance, params[:id_token], kid).call
```

Figure 5.7: A snippet from the LTI 1.3 controller demonstrating the process of decoding the JWT, allowing OpenDSA to extract and verify its claims.

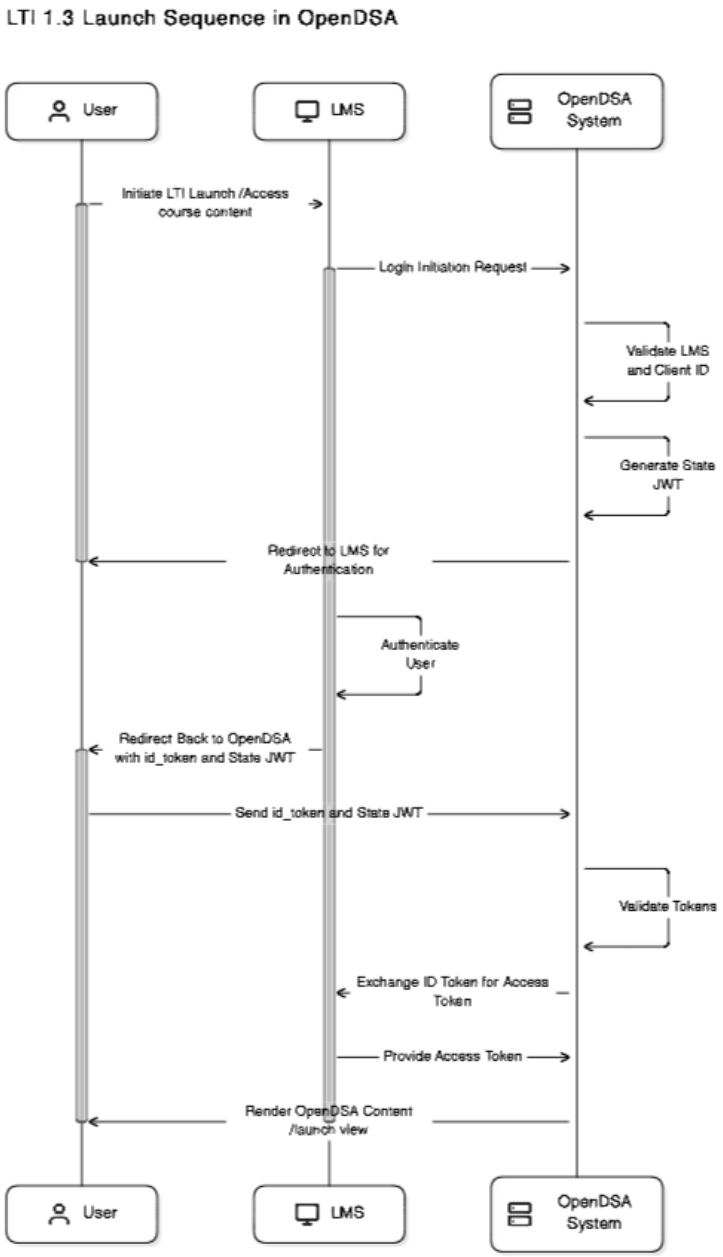

Figure 5.8: Detailed LTI 1.3 launch sequence between OpenDSA and LMS.

```
#ruby
Lti13Service::PostGrade.new(@lms_instance, grade).call
```

Figure 5.9: A snippet from the LTI 1.3 controller demonstrating the post grade service for grade submission.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we explored several key areas to improve access to the Smart Learning Content (SLC) catalog and the OpenDSA system. A well-structured catalog is valuable, as it centralizes learning resources and streamlines content management for educators. The integration of Learning Tools Interoperability (LTI) 1.3 into OpenDSA has improved interoperability and compliance with modern Learning Management System (LMS) standards. The LTI 1.3 support integrates OAuth 2.0 and JSON Web Tokens (JWTs), providing OpenDSA with enhanced security for learners and instructors accessing the system through Learning Management Systems (LMSs) such as Canvas. This change addresses key security concerns present in earlier versions and ensures that user authentication is managed according to best practice [34].

The development of the validation infrastructure for the SLC catalog has been another achievement. This infrastructure ensures accessibility, easy discovery of content with the ontology, and compliance with metadata across the platform. Validation rules include schema validation, URL verification, and LTI compliance, enhance the reliability and quality of content within the catalog [36].

Furthermore, the implementation of an ontology-driven classification system for the SLC

has demonstrated the effectiveness of organizing educational content using a structured ontology. Using an ontology-based approach improves content retrieval, reduces ambiguity, and provides a clear structure for understanding relationships between various topics, such as data structures and algorithms. Schema design, including the use of ontology aliases, item classification, and keyword matching, creates a flexible and scalable framework for ongoing content classification. Furthermore, the implementation of an ontology-driven classification system for the SLC aims to organize educational content using a structured ontology effectively. An ontology-based approach is expected to improve content retrieval, reduce ambiguity, and provide a clear structure to understand relationships between various topics. The Schema design, including the use of ontology aliases, item classification, and keyword matching, creates a flexible approach for more contributions to the catalog.

6.2 Future Work

While substantial progress has been made, there are several avenues for future improvement and research to build on this foundation. Firstly, with the adoption of LTI 1.3, future development should focus on the integration of LTI 1.3 with the LTI Caliper Analytics connector service, which enables learning tools to send Caliper events to the Learning Management System (LMS) and vice versa, providing a more comprehensive view of student interactions. By implementing the Caliper Analytics framework, learning tools can capture highly granular data consistently across various applications, establishing a universal vocabulary to describe learning events. This data collection could provide a deeper understanding of student performance, engagement, and learning outcomes. The standardization of interactions through Caliper's metrics profile allows easier analysis of student activities between different tools. Furthermore, extending Names and Role Provisioning Services (NRPS) to incorporate Caliper data could offer more granular insights based on role-specific interactions, supporting a more personalized educational experience. This combined approach of LTI and Caliper can lead to better interventions

[31].

As mentioned in Section 5.2.1 The LTI 1.3 service can be expanded by integrating LTI Advantage services, including deep linking, which could further enhance interoperability. Deep linking would enable instructors to seamlessly embed and manage external resources within their courses, creating a more flexible and content-rich learning environment. Although OpenDSA already supports an LTI 1.1 deep linking service, it requires custom implementation by the tool provider, relies on OAuth 1.0a for security, and offers a limited range of content selection options. LTI 1.3 Deep linking represents an advance in security, more expanded content selections like LTI links, static HTML, web links, etc. The foundation of the security has been established in Sub-section 5.2.2.

Automated metadata enrichment development mechanisms for automated metadata enrichment could reduce reliance on limited metadata provided by other learning tools. This enrichment can improve the classification accuracy for content with limited metadata. Machine learning algorithms could suggest missing metadata fields, such as keywords or descriptions, enhancing the completeness of content submissions [8]. Integrate LTI support to validate LTI-enabled content within the catalog, ensuring compatibility and streamlined interactions with learning management systems. Additionally, implementing a robust content verification mechanism would enhance the system's ability to verify whether the embedded content, such as exercises displayed in iframes, is rendered correctly with specific JavaScript protocols or lightweight APIs, such as potentially SPLICE protocols, upon loading the content. This approach would ensure that URLs are accessible and that the content they link to functions as intended.

The development of interactive ontology visualization tools could also improve usability and understanding of the relationships between various classes. Such tools would allow users to explore the ontology, facilitating better navigation and content discovery within the SLC [8, 36]. Implementing machine learning-based context-sensitive matching techniques could mitigate issues related to ambiguous or broad keywords in the ontology classification system. This would help to resolve cases where a keyword may refer

to multiple distinct concepts, improving classification precision and search. Further improvements to the ontology might also include expanding properties such as prerequisites and other common relationships, to capture more complex relational depth. Finally, evaluating the ontology with domain experts and applying data-driven methods for refinement would ensure that its structure remains robust, intuitive, and aligned with the evolving needs of computer science education. Another avenue involves comparing full-text search systems, such as Elasticsearch [25], a distributed search and analytics engine, to ontology-based search in smart catalogs. However, full-text search may excel in rapid retrieval and unstructured content. An ontology-based classification might provide more precise and contextually relevant results leveraging domain-specific knowledge representations. Some key research areas include comparing semantic richness and performance or exploring a hybrid approach.

Bibliography

[1] Monika Akbar, Weiguo Fan, Clifford A. Shaffer, Yinlin Chen, Lillian Cassel, Lois Delcambre, Daniel D. Garcia, Gregory W. Hislop, Frank Shipman, Richard Furuta, B. Stephen Carpenter, Haowei Hsieh, Bob Siegfried, and Edward A. Fox. Digital Library 2.0 for Educational Resources. In *Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries: Research and Advanced Technology for Digital Libraries*, pages 89–100, 2011. URL https://dl.acm.org/doi/10.5555/2042536.2042551.

[2] Alexander Joel D. Alon. The Algoviz Project: Building an Algorithm Visualization Web Community. Master's thesis, Virginia Tech, 2010. URL http://hdl.handle.net/10919/34246.

[3] Anomalo. How to Choose the Right Metadata Management Tool, 2024. URL https://www.anomalo.com/blog/how-to-choose-the-right-metadata-management-tool/. Accessed: 2024-10-07.

[4] ApiDog. Axios GET Request Parameters: A Comprehensive Guide. https://apidog.com/blog/params-axios-get-request/, 2024. Accessed: 2024-10-07.

[5] Apidog. Axios GET Request Parameters: A Comprehensive Guide, 2024. URL https://apidog.com/blog/params-axios-get-request/. Accessed: 2024-10-23.

[6] Association for Computing Machinery. The 2012 ACM Computing Classification System. https://www.acm.org/publications/class-2012, 2012. Accessed: 2023-09-21.

[7] Association for Computing Machinery. ACM Computing Classification System (CCS), 2024. URL https://dl.acm.org/ccs. Accessed: 2023-09-21.

[8] Atlan. Data Governance and Metadata Management: Understanding Their Synergy for Data-Driven Success, 2024. URL https://atlan.com/data-governance-and-metadata-management/. Accessed: 2024-10-22.

[9] Axios. Axios: Promise-based HTTP Client for the Browser and Node.js. https://axios-http.com/docs/intro, 2024. Accessed: 2024-10-07.

[10] Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. *W3C Recommendation*, 10(02), 2004. URL https://www.w3.org/TR/owl-ref.

[11] Jeremiah Blanchard, John R Hott, Vincent Berry, Rebecca Carroll, Bob Edmison, Richard Glassey, Oscar Karnalim, Brian Plancher, and Seán Russell. Leveraging Community Software in CS Education to Avoid Reinventing the Wheel. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2*, pages 580–581. ACM, July 2022. doi: 10.1145/3502717.3532169. URL https://doi.org/10.1145/3502717.3532169.

[12] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, et al. Increasing Adoption of Smart Learning Content for Computer Science Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference*, pages 31–57, 2014. URL https://dl.acm.org/doi/10.1145/2713609.2713611.

[13] Koen Buyens. OAuth 2.0 Security Cheat Sheet. https://github.com/koenbuyens/oauth-2.0-security-cheat-sheet, 2024. Accessed: 2024-10-03.

[14] Cambridge Semantics. OWL 101. https://cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/owl-101/, 2023. Accessed: 2024-10-21.

[15] Werner Ceusters, Barry Smith, and Jim Flanagan. Ontology and medical terminology: Why description logics are not enough. In *Towards an Electronic Patient Record (TEPR 2003)*, San Antonio, TX, May 2003. Medical Records Institute. URL https://philarchive.org/archive/CEUOAM. Boston, MA: Medical Records Institute (CD-ROM publication).

[16] Sridhar Chimalakonda and Kesav V Nori. A Survey of Semantic Technology and Ontology for e-Learning. *Semantic Web*, 10(6):1199–1222, 2019. URL https://api.semanticscholar.org/CorpusID:195323514.

[17] Codio Documentation. LTI 1.3 Integration. https://docs.codio.com/instructors/admin/integration/lti1-3.html, 2024. Accessed: 2024-10-03.

[18] CSSPLICE. LTI Use Case: Exposing OpenDSA Visualizations and Exercises. https://cssplice.org/lti/opendsa.html, 2024. Accessed: 2024-10-07.

[19] D2L. Assignments and Grades Services Extension with LTI 1.3. https://community.d2l.com/brightspace/kb/articles/23752-assignments-and-grades-services-extension-with-lti-1-3, 2024. Accessed: 2024-10-03.

[20] D2L Corporation. About LTI 1.3 launch and authentication. https://community.d2l.com/brightspace/kb/articles/23730-about-lti-1-3-launch-and-authentication, 2024. Accessed: 2024-10-03.

[21] Stefan Dietze, Hong Qing Yu, Daniela Giordano, Eleni Kaldoudi, Nikolas Dovrolis, and Davide Taibi. Linked Education: Interlinking Educational Resources and the Web of Data. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 366–371, 2013. URL https://dl.acm.org/doi/pdf/10.1145/2245276.2245347.

[22] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. A review of research on parsons problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*, ACE'20, page 195–202, New York, NY, USA, 2020. Association

for Computing Machinery. ISBN 9781450376860. doi: 10.1145/3373165.3373187. URL https://doi.org/10.1145/3373165.3373187.

[23] EdTech. LTI standards. https://www.1edtech.org/standards/lti, 2024. Accessed: 2024-10-03.

[24] EdTech. Why adopt LTI 1.3 and LTI advantage. https://www.1edtech.org/standards/lti/why-adopt-lti-1p3, 2024. Accessed: 2024-10-03.

[25] Elastic. Elasticsearch: The Official Distributed Search Analytics Engine, 2024. URL https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html. Accessed: 2024-12-30.

[26] Marcos André Gonçalves, Edward A. Fox, and Layne T. Watson. Towards a digital library theory: A formal digital library ontology. *International Journal on Digital Libraries*, 8(2):91–114, April 2008. doi: 10.1007/s00799-008-0033-1. URL https://doi.org/10.1007/s00799-008-0033-1. Published online: 11 March 2008.

[27] Wenge Guo. MOOC and SPOC, Which One is Better? *Eurasia Journal of Mathematics, Science and Technology Education*, 10(3):267–274, 2014. URL https://doi.org/10.12973/eurasia.2017.01044a.

[28] Amelie Gyrard, Antoine Zimmermann, and Amit Sheth. Building IoT-Based Applications for Smart Cities: How Can Ontology Catalogs Help? *IEEE Internet of Things Journal*, 5(5):3978–3990, 2018. doi: 10.1109/JIOT.2018.2854278. URL https://doi.org/10.1109/JIOT.2018.2854278.

[29] Jeff Heflin. An Introduction to the OWL Web Ontology Language. Technical report, Lehigh University, 2007. URL https://www.cse.lehigh.edu/~heflin/IntroToOWL.pdf. Accessed: 2024-10-21.

[30] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers,

et al. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. *Proceedings of the 2015 ITiCSE on Working Group Reports*, pages 41–63, 2015. URL https://doi.org/10.1145/2858796.2858798.

[31] IMS Global Learning Consortium. Caliper Analytics 1.2 Specification. Technical specification, IMS Global Learning Consortium, 2019. URL https://www.imsglobal.org/spec/caliper/v1p2. Accessed: 2024-10-23.

[32] IMS Global Learning Consortium. Learning Tools Interoperability (LTI) names and role provisioning services version 2.0. IMS final release, IMS Global Learning Consortium, April 2019. URL https://www.imsglobal.org/spec/lti-nrps/v2p0. Accessed: 2024-10-07.

[33] IMS Global Learning Consortium. Learning Tools Interoperability Assignment and Grade Services Version 2.0. Specification, IMS Global Learning Consortium, 2024. URL https://www.imsglobal.org/spec/lti-ags/v2p0. Accessed: 2024-10-03.

[34] IMS Global Learning Consortium. Learning Tools Interoperability (LTI) core specification. Technical Specification v1.3, IMS Global Learning Consortium, 2024. URL https://www.imsglobal.org/spec/lti/v1p3. Accessed: 2024-10-03.

[35] Inca Tools. Aliases and Synonyms. https://incatools.github.io/ontology-access-kit/guide/aliases.html, 2024. Accessed: 2024-10-07.

[36] Informatica. Repository Manager. https://docs.informatica.com/master-data-management/multidomain-mdm/10-4/overview-guide/informatica-mdm-hub-architecture/repository-manager.html, 2024. Accessed: 2024-10-07.

[37] Knowledge Media Institute. Computer Science Ontology (CSO), 2024. URL https://cso.kmi.open.ac.uk/topics/computer_science. Accessed: 2024-09-07.

[38] Instructure. Assignment Tools. https://canvas.instructure.com/doc/api/file.assignment_tools.html, 2024. Accessed: 2024-10-03.

[39] Ritu John. What is Document Classification? https://www.docsumo.com/blogs/ocr/document-classification, 2023. Accessed: 2024-10-21.

[40] A. M. Johnson, M. E. Jacovina, D. E. Russell, and C. M. Soto. Challenges and Solutions when Using Technologies in the Classroom. In S. A. Crossley and D. S. McNamara, editors, *Adaptive Educational Technologies for Literacy Instruction*, pages 13–29. Taylor & Francis, New York, 2016. doi: 10.4324/9781315647500-2. URL https://doi.org/10.4324/9781315647500. Published with acknowledgment of federal support.

[41] Hamza Manzoor. Disseminating Learning Tools Interoperability Standards. Master's thesis, Virginia Polytechnic Institute and State University, 2019. URL http://hdl.handle.net/10919/90772. Accessed: 2024-10-07.

[42] Hamza Manzoor, Kamil Akhuseyinoglu, Jackson Wonderly, Peter Brusilovsky, and Clifford A Shaffer. Crossing the Borders: Re-Use of Smart Learning Objects in Advanced Content Access Systems. *Future Internet*, 11(7):160, 2019. doi: 10.3390/fi11070160. URL https://doi.org/10.3390/fi11070160.

[43] MasterControl. A Guide for Performing and Maintaining a Validated IT Infrastructure. https://www.mastercontrol.com/gxp-lifeline/a-guide-for-performing-and-maintaining-a-validated-it-infrastructure/, 2019. Accessed: 2024-10-30.

[44] Carlos Mucuho. Understanding Axios GET Requests, 2024. URL https://blog.logrocket.com/understanding-axios-get-requests/. Last updated on 9 February 2024.

[45] Ontology Access Kit. The OAK Guide: Aliases and Synonyms. https://incatools.github.io/ontology-access-kit/guide/aliases.html, 2024. Accessed: 2024-10-21.

[46] OpenDSA. OpenDSA-LTI Implementation. https://opendsa.readthedocs.io/en/latest/OpenDSA-LTI-Implementation.html, 2024. Accessed: 2024-10-03.

[47] OpenSemanticSearch. Setup of Synonyms and Aliases. https://opensemanticsearch.org/doc/admin/config/synonyms/, 2024. Accessed: 2024-10-21.

[48] Oracle. Spatial and Graph RDF Semantic Graph Developer's Guide. https://docs.oracle.com/database/121/RDFRM/owl-concepts.htm#RDFRM200, 2024. Accessed: 2024-10-21.

[49] Overt Software. All about Learning Tools Interoperability: What is LTI? https://www.overtsoftware.com/all-about-learning-tools-interoperability/, 2024. Accessed: 2024-10-03.

[50] Marino O. Bejaoui R. Paquette, G. A New Competency Ontology for Learning Environments Personalization. *Smart Learning Environments*, 8(1):1–23, 2021. URL https://doi.org/10.1186/s40561-021-00160-z.

[51] Aaron Parecki. OAuth 2.0: The nuts & bolts of OAuth 2.0. https://oauth.net/2/, May 2018. Accessed: 2024-10-03.

[52] Goldin S. Ditcharoen N. Piriyapongpipat, P. An Alternative Approach to Ontology-Based Curriculum Development in Higher Education. *Smart Learning Environments*, 11(1):20, 2024. URL https://doi.org/10.1186/s40561-024-00307-8.

[53] Mar'ıa Poveda-Villal'on, Ra'ul Garc'ıa-Castro, and Asunci'on G'omez-P'erez. Building an Ontology Catalogue for Smart Cities. Technical report, Universidad Polit'ecnica de Madrid, 2014. URL https://oa.upm.es/36715/1/INVE_MEM_2014_193269.pdf.

[54] Monika Rani, Kumar Vaibhav Srivastava, and O. P. Vyas. An Ontological Learning Management System. *CoRR*, abs/1708.09475, 2017. URL https://doi.org/10.48550/arXiv.1708.09475.

[55] Restack. Hierarchical Ontology Learning Tools. https://www.restack.io/p/ai-ontology-creation-tools-knowledge-answer-hierarchical-ontology-learning-cat-ai, 2023. Accessed: 2024-10-21.

[56] George P. Schell and Max Burns. Merlot: A repository of e-learning objects for higher education. *e-Service Journal*, 1(2):53–64, 2002. URL https://www.jstor.org/stable/10.2979/esj.2002.1.2.53.

[57] Clifford A Shaffer, Monika Akbar, Ari J D Alon, Michael Stewart, and Stephen H Edwards. Getting Algorithm Visualizations into the Classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 129–134, 2011. URL https://dl.acm.org/doi/10.1145/1953163.1953204.

[58] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. Springer, Berlin, Heidelberg, 2 edition, 2009. ISBN 978-3-540-92673-3. doi: 10.1007/978-3-540-92673-3. URL https://link.springer.com/book/10.1007/978-3-540-92673-3.

[59] John C. Stamper, Kenneth R. Koedinger, Ryan S. J. d. Baker, Alida Skogsholm, Brett Leber, Sandy Demi, Shawnwen Yu, and Duncan Spencer. Managing the Educational Dataset Lifecycle with Datashop. In Gautam Biswas, Susan Bull, Judy Kay, and Antonija Mitrovic, editors, *Artificial Intelligence in Education*, pages 557–559, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-21869-9. URL https://doi.org/10.1007/978-3-642-21869-9_100.

[60] Training Industry. Learning Tools Interoperability. https://trainingindustry.com/wiki/learning-technologies/learning-tools-interoperability/, 2024. Accessed: 2024-10-03.

[61] Grigoris Tsoumakas and Ioannis Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007. URL http://dx.doi.org/10.4018/jdwm.2007070101.

[62] William Villegas-Ch and Joselin García-Ortiz. Enhancing learning personalization in educational environments through ontology-based knowledge representation. *Computers*, 12(10), 2023. ISSN 2073-431X. doi: 10.3390/computers12100199. URL https://doi.org/10.3390/computers12100199.

[63] David Wiley, TJ Bliss, and Mary McEwen. The RISE Framework: Using Learning Analytics to Automatically Identify Open Educational Resources for Continuous Improvement. *The International Review of Research in Open and Distributed Learning*, 15(5), 2014. URL http://dx.doi.org/10.19173/irrodl.v18i2.2952.

[64] Jackson D. Wonderly. Improving the Interoperability of the OpenDSA eTextbook System. Master's thesis, Virginia Polytechnic Institute and State University, 2019. URL http://hdl.handle.net/10919/94391. Accessed: 2024-10-07.

Appendices

Appendix A

SLC Catalog Keywords

This appendix presents the hierarchical structure of all currently available keywords in the ontology of the Smart Learning Content (SLC) catalog. Organized under the categories Class, Subclass, Sub-Subclass, and Instance, serving as a framework to classify and organize the content in the SLC catalog for search and retrieval.

Table A.1: Overview of the ontology structure and keywords.

Class	**Subclass**	**Sub-subclass**	**Instance**
Proofs	Lower Bounds Proofs		Adversary Arguments
Proofs	Lower Bounds Proofs		State Space Lower Bounds
Problems	Lower Bounds Proofs		Maximum Value
Problems	Lower Bounds Proofs		ith Best Element
Proofs	Lower Bounds Proofs		Search in Unsorted Lists
Proofs	Lower Bounds Proofs		Search in Sorted Lists

Class	**Subclass**	**Sub-subclass**	**Instance**
Algorithms	Sorting		Optimal Sorting Algorithms
Data Structures	Search Structures		Skip Lists
Algorithms		Randomized Algorithms	Skip Lists
Algorithms		Probabilistic Algorithms	Skip Lists
Data Structures		Probabilistic Data Structures	Skip Lists
Data Structures	Search Structures		Splay Trees
Data Structures	Search Structures		AVL Trees
Data Structures	Search Structures		Red-Black Trees
Data Structures	Search Structures		Balanced Binary Trees
Data Structures		Array Representations	Sparse Matrix
Math Topics			Logarithms
Math Topics			Summations
Math Topics	Sets		Disjoint Sets
Math Topics	Relations		Partial Order
Math Topics	Relations		Equivalence Relation
			Estimation

Class	**Subclass**	**Sub-subclass**	**Instance**
Algorithms			Factorial
Math Topics			Factorial
Math Topics			Permutations
ath Topics			Modulus
Math Topics			Math Background
Proofs			Induction Proofs
Proofs			Proof by Contradiction
Proofs			Deduction
Math Topics			Recurrence Relations
Biography			Francis Bacon
Biography			Carl Gauss
Software Engineering	Software Design	Abstract Data Type	Dictionary
Software Engineering	Software Design	Abstract Data Type	Container
Software Engineering			Unified Modeling Language
Data Structures		Record Keys	Record Comparison
Software Engineering	Object-Oriented Programming		
Data Structures	Lists		List ADT Design
Software Engineering	Software Design		Design Patterns

Class	**Subclass**	**Sub-subclass**	**Instance**
Problems			Fibonacci Sequence
Problems			Knapsack Problem
Algorithms	Dynamic Programming		Knapsack Problem
Theory of Computation	Complexity Theory	NP-completeness	Knapsack Problem
Problems			Prime Number Algorithms
Problems			Towers of Hanoi
Algorithms	Algorithm Analysis		Transforms
Algorithms	Computability		Unsolvable Problems
Problems	Mathematical Algorithms		Number Problems
Proofs	Lower Bounds Proofs		Amortized Analysis
Problems	Mathematical Algorithms		Fast Fourier Transform
			Problem Solving Heuristics
Algorithms	Algorithm Analysis		Upper Bound
Algorithms	Algorithm Analysis		Lower Bound
Algorithms	Algorithm Analysis		Growth Rate
Algorithms	Algorithm Analysis		Analyzing Programs
Algorithms	Algorithm Analysis		Analyzing Problems

Class	**Subclass**	**Sub-subclass**	**Instance**
Algorithms	Algorithm Analysis		Space Analysis
Algorithms	Algorithm Analysis		Big Theta
Algorithms	Algorithm Analysis		Big Omega
Algorithms	Algorithm Analysis		Big O
Algorithms	Algorithm Analysis		Best, Average, Worst Case
Algorithms	Algorithm Analysis		Multiple Parameters
Algorithms	Algorithm Analysis		Algorithm Analysis Misconceptions
Algorithms	Algorithm Analysis		Asymptotic Algorithm Analysis
Algorithms	String Matching		Rabin-Karp String Match Algorithm
Problems	String Matching		
Algorithms	String Matching		Boyer-Moore String Match Algorithm
Data Structures	General Trees		General Tree Implementations
Algorithms	String Matching		KMP String Match Algorithm
Data Structures	Search Structures		Bit Vectors
Problems			Edit Distance Problem

Class	**Subclass**	**Sub-subclass**	**Instance**
Data Structures	K-ary Trees		K-ary Tree Implementations
Algorithms	Equivalence Class	Disjoint Sets	Union/Find Algorithm
Data Structures	General Trees		Sequential Tree Representations
Algorithms	Search		Binary Search
Algorithms	Search		Self-Organizing Lists
			Problem Definition
			Algorithm Definition
Software Engineering	Software Design		Code Tuning
Problems	Graphs	Shortest Paths Problem	All-Pairs Shortest Paths Problem
Algorithms	Graphs	All-Pairs Shortest Paths Problem	Floyd's Algorithm
Problems	Graphs		Minimum-Cost Spanning Trees
Algorithms	Graphs		Graph Traversals
Data Structures	Graphs		Graph Representations
Algorithms	Graphs	Minimum-Cost Spanning Trees	Kruskal's Algorithm
Algorithms	Graphs	Shortest Paths Problem	Dijkstra's Algorithm

Class	**Subclass**	**Sub-subclass**	**Instance**
Problems	Graphs		Topological Sort
Algorithms	Binary Trees	Binary Tree Traversals	Preorder Traversal
Algorithms	Binary Trees	Binary Tree Traversals	Postorder Traversal
Algorithms	Binary Trees	Binary Tree Traversals	Inorder Traversal
Algorithms	Recursion		Recursive Tree Functions
Algorithms	Memory Management		Sequential Fit Memory Allocation
Algorithms	Memory Management		Buddy Method Memory Allocation
Algorithms	Memory Management		Garbage Collection
Algorithms	Memory Management		Dynamic Storage Allocation
Algorithms	Memory Management		Heap Memory
Algorithms	Memory Management		Garbage Disposal
Theory of Computation	Formal Languages	Context-Free Grammar	Context-Free Grammar Transformations
Theory of Computation	Formal Languages	Context-Free Language	

Class	**Subclass**	**Sub-subclass**	**Instance**
Theory of Computation	Complexity Theory	NP-completeness	
Theory of Computation	Formal Languages	Grammars	
Theory of Computation	Finite State Machines		Turing Machines
Theory of Computation	Finite State Machines		Non-Deterministic Finite Automata
Theory of Computation	Finite State Machines		Deterministic Finite Automata
Theory of Computation	Finite State Machines	Pushdown Automaton	Deterministic Pushdown Automaton
Theory of Computation	Finite State Machines	Pushdown Automaton	Non-Deterministic Pushdown Automaton
Theory of Computation	Formal Languages		Regular Language
Theory of Computation	Formal Languages		Regular Grammar
Theory of Computation	Formal Languages	Regular Language	Regular Expression
Proofs			Reductions
Theory of Computation	Formal Languages	Non-regular Language	Pumping Lemma Proofs
Compilers			Backus-Naur Form

Class	**Subclass**	**Sub-subclass**	**Instance**
Theory of Computation	Finite Automata		Deterministic Finite Acceptor
Math Topics	Relations		Closure Properties
Theory of Computation			Limits to Computing
Software Engineering	Testing		Code Coverage
Data Structures	Search Structures	Hashing	Perfect Hashing
Data Structures	Search Structures	Hashing	Hashing Cost Analysis
Data Structures	Search Structures	Hashing	Bucket Hashing
Data Structures	Search Structures	Hashing	Hash Functions
Data Structures	Search Structures	Hashing	Collision Resolution
Data Structures	Search Structures	Hashing	Hash Table Deletion
Data Structures	Search Structures	Hashing	Open Hashing
Data Structures	Linear Structures	Stacks	Array-based Stack Implementation
Data Structures	Linear Structures	Stacks	Linked Stack Implementation

Class	**Subclass**	**Sub-subclass**	**Instance**
Data Structures	Linear Structures	Stacks	Implementing Recursion with a Stack
Data Structures	Linear Structures	Queues	Linked Queues
Data Structures	Linear Structures	Queues	Array-based Queue Implementation
Data Structures	Linear Structures	Lists	List Element Implementation
Algorithms	Recursion		Implementing Recursion with a Stack
Data Structures	Linear Structures	Lists	Array-based List Implementation
Data Structures	Linear Structures	Lists	Doubly Linked List
Data Structures	Linear Structures	Lists	Freelists
Data Structures	Linear Structures	Lists	Linked List
Data Structures	Linear Structures	Lists	List ADT
Data Structures	Linear Structures	Lists	List Cost Analysis
Algorithms	Algorithm Analysis	Space Analysis	Overhead
File Processing			Random Access File

Class	**Subclass**	**Sub-subclass**	**Instance**
File Processing			External Sorting
Algorithms	Sorting		External Sorting
Computer Architecture			Disk Drive
Computer Architecture			Memory
Data Structures	Memory Management		Buffer Pool
Data Structures	Search Structures		Trie
Software Engineering	Software Design	Binary Trees	Binary Tree Node Implementation
Proofs		Binary Trees	Full Binary Tree Theorem
Algorithms	Algorithm Analysis	Binary Trees	Binary Tree Space Analysis
Data Structures	Search Structures	Binary Trees	Binary Search Tree
Data Structures	Binary Trees		Binary Tree Terminology
Data Structures	Binary Trees		Complete Binary Trees
Software Engineering	Software Design	Design Patterns	Composite Design
Data Structures	Tree Structures		Huffman Coding Tree

Class	**Subclass**	**Sub-subclass**	**Instance**
Data Compression			Huffman Coding Tree
Algorithms	Sorting		Bubble Sort
Algorithms	Sorting		Comparison of Sorting Algorithms
Algorithms	Sorting		Quicksort
Algorithms	Sorting		Radix Sort
Algorithms	Sorting		Non-comparison Sort
Algorithms	Sorting		Heapsort
Algorithms	Sorting		Exchange Sort
Algorithms	Sorting		N-squared Sorts
Algorithms	Sorting		Binsort
Algorithms	Sorting		Bucket Sort
Algorithms	Sorting		Shellsort
Algorithms	Sorting		Mergesort
Algorithms	Sorting		Selection Sort
Algorithms	Sorting		Insertion Sort
Software Engineering		Testing	JUnit Testing
Software Engineering		Testing	Mutation Testing
Software Engineering	Software Development	Debugging	Source Level Debugging
			Command Line Parameters
			Java Scanner Class

Class	**Subclass**	**Sub-subclass**	**Instance**
			Command Line Interface
Software Engineering	Software Development	Debugging	Debugging Methods
Data Structures	Search Structures	Indexing	Linear Indexing
Data Structures	Search Structures	Indexing	B-Trees
Data Structures	Search Structures	Indexing	ISAM
Data Structures	Search Structures	Indexing	Tree Indexing
Data Structures	Search Structures		2-3 Trees
Problems			Hamiltonian Cycle Problem
Theory of Computation	Complexity Theory	NP-completeness	Hamiltonian Cycle Problem
Problems			Traveling Salesman Problem
Theory of Computation	Complexity Theory	NP-completeness	Traveling Salesman Problem
Proofs	Complexity Theory	NP-completeness	NP-completeness Proofs
Problems			Independent Set Problem

Class	**Subclass**	**Sub-subclass**	**Instance**
Theory of Computation	Complexity Theory	NP-completeness	Independent Set Problem
Problems			Circuit Satisfiability Problem
Theory of Computation	Complexity Theory	NP-completeness	Circuit Satisfiability Problem
Problems			Clique Problem
Theory of Computation	Complexity Theory	NP-completeness	Clique Problem
Problems			Satisfiability Problem
Theory of Computation	Complexity Theory	NP-completeness	Satisfiability Problem
Problems			3-SAT Problem
Theory of Computation	Complexity Theory	NP-completeness	3-SAT Problem
Problems			Vertex Cover Problem
Theory of Computation	Complexity Theory	NP-completeness	Vertex Cover Problem
Data Structures	Search Structures	Spatial Data Structures	KD Tree
Data Structures	Search Structures	Spatial Data Structures	PR Quadtree
Data Structures	Search Structures	Spatial Data Structures	Bintree