Next-Generation Programming Learning Platform: Architecture and Challenges

Yutaka Watanobe[1,*], *Chowdhury* Intisar[1,**], *Ruth* Cortez[1,***], and *Alexander* Vazhenin[1,****]

[1]Department of Computer Science and Engineering, University of Aizu

Abstract. With the rapid development of information technology, programming has become a vital skill. An online judge system can be used as a programming education platform, where the daily activities of users and judges are used to generate useful learning objects (e.g., tasks, solution codes, evaluations). Intelligent software agents can utilize such objects to create an ecosystem. To implement such an ecosystem, a generic architecture that covers the whole lifecycle of data on the platform and the functionalities of an e-learning system should take into account the particularities of the online judge system. In this paper, an architecture that implements such an ecosystem based on an online judge system is proposed. The potential benefits and research challenges are discussed.

1 Introduction

With the growing significance of technology and the ubiquitous availability of information, a number of initiatives in computer education have been promoted by various organizations. Programming has become a vital skill not only for information technology but also for industry and academia. In Japan, the Ministry of Education, Culture, Sports, Science and Technology (MEXT) has decided to make programming a compulsory subject in primary schools starting from fiscal year 2020. Many leading countries have already put great emphasis on programming education.

However, programming is a very difficult task that requires logical thinking and the ability to identify syntactical and conceptual errors. As a result, instructors must expend a lot of effort for creating tasks and evaluating and debugging code, and learners tend to become frustrated. Among educational tools for programming, online judge systems (OJSs) have become popular. An OJS refers to a web service that was originally designed for programming contests such as ACM-ICPC [1], but has become widely utilized for education since it provides reliable evaluation that reduces the workload of teachers and supports self-driven autonomous learning. An attractive feature of an OJS is that learners can repeatedly achieve success, which maintains motivation. OJSs have thus become an indispensable tool for programming education. An OJS provides a number of programming tasks at different levels and compiles and runs submitted code on its servers. Then, it determines whether the code is acceptable using strict test cases. One of the oldest major OJSs, UVa Online Judge, started providing service in 1995 [1]. A number of OJSs are currently operated by various organizations and related tools are actively researched [2].

Although learning materials have been dramatically improved, most major OJSs have shortcomings regarding their educational use. For example, black-box tests are a notable disadvantage. There is a need to improve learning efficiency, but the amount of accumulated data has grown exponentially. Fortunately, this is a great opportunity to make e-learning systems more intelligent as well as to bring research seeds not only for education but also for software engineering fields. Here, we propose an architecture for organizing an ecosystem for an educational programming platform that takes advantage of an OJS. The architecture includes the management of learning objects (LOs) [3], such as tasks, solutions, verdicts, and logs, and a judge system. Most existing e-learning tools for programming cannot be organized into such ecosystems because of their monolithic architecture or localization, which hinders data accessibility, scalability, maintainability, and reusability. Besides, monolithic architecture can be a barrier to flexibly develop user interfaces (client systems) by the third parties.

This study proposes an architecture for a programming learning platform based on the Aizu Online Judge (AOJ) [4] [2], which is one of the oldest OJSs in Japan. Although AOJ has been operated for 15 years, its original core functions and architecture have still not been fully revealed [5]. The aim of the proposed architecture is to realize a programming learning platform ecosystem where data acquisition, reuse, feedback, and research are conducted by people and intelligent software agents. To design the architecture, we first analyzed the distinctive characteristics

*e-mail: yutaka@u-aizu.ac.jp
**e-mail: d8211106@u-aizu.ac.jp
***e-mail: rpatycr@gmail.com
****e-mail: vazhenin@u-aizu.ac.jp
[1]https://icpc.baylor.edu
[2]http://judge.u-aizu.ac.jp

of an OJS, focusing on the actors and LOs. The architecture consists of six layers, which are used to obtain LOs based on a resource-oriented architecture from various perspectives. In this study, an ecosystem is proposed and its generic architecture is described. The effectiveness of the proposed ecosystem is demonstrated through a concrete implementation and initial results of research and development employing the architecture. Potential research challenges for programming education are also given. Components of the proposed architecture are related to machine learning, software engineering, user interfaces, and e-learning systems. This work thus benefits users (e.g., students and teachers) as well as researchers.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 formulates the ecosystem and presents the characteristics of the OJS. Section 4 describes the proposed architecture, and Section 5 demonstrates its implementation. Section 6 presents the initial research results and identifies challenges. Section 7 concludes the paper.

2 Related Work

Traditional OJSs include UVa [3], SPOJ [4], and POJ [5], among others [2]. With an increasing number of problem sets from programming competitions, these OJSs continue to grow while contributing to programming education with advanced services such as integrated management interfaces (see for example [6]). A framework that considers feedback other than judge results has been proposed [7]. Jutge.org is an open online judge designed for students and instructors which integrates research and innovation such as formal verification methods, static code analysis and data-mining [8]. However, for most systems, system details are unavailable and data cannot be opened to the public because of architectural inflexibility.

Many companies hold regular programming contests. Codeforces [6], AtCoder [7] and TopCoder [8] are competition platforms with powerful and scalable judge systems that can handle a huge number of participants. They can be used for competition and human assessment and include high-quality problem sets. Although they have great potential as data resources and provide application programming interfaces (APIs) for third parties, their access policies are limited.

Systems that can be deployed in a local environment and provide functions to manage original tasks, evaluation, and grading are not only attractive, but essential for students to gain programming skills which requires deliberate practice and quality feedback [9]. CMS [9] and DOMjudge [10] are notable examples of active open-source projects for contest management that can also be applied for education. Efforts to integrate Automatic Judge to learning management system (LMS) for blended education are an attractive area, but the integration is challenging when standards are not applied, and the design of the applications are monolithic [10]. Enterprises have also released similar e-learning tools for induction courses and personal assessment. Although these closed systems provide a secure and robust environment and allow flexible task management, they are unsuitable for knowledge sharing.

In the consumer web space, some comprehensive architectures for ecosystems and data analysis have been developed [11–13]. Based on the concept of an ecosystem, many enterprises operate a data analysis system that manages data to provide additional services for end users. For example, batch and real-time data processing for Twitter [14] and the use of log data to provide additional information by a Hadoop ecosystem for LinkedIn [15] have been considered.

The concept of an ecosystem can also be applied to an e-learning system, where data is the core of services for education [16]. Massive open online courses (MOOCs) on platforms such as edX and Coursera are an attractive resource for learners and teachers. Such platforms are organized considering data processing and learning resource organization. For example, the base Open edX architecture consists of core applications (e.g., learning management systems) and client tools, which are supplemented by independently deployed applications and persistence systems. From a technical point of view, cloud computing has become a standard solution for dynamic resource allocation, virtualization, and dependable data storage.

Although a number of institutions are considering MOOC, the characteristics of data analysis in a general e-learning system are different from those for programming education, where the utilization of a trial-and-error process is much more valuable to assess the learning outcomes [17]. An example of the analysis is presented in [18], where code snapshots and modification sequence are used to discover patterns to predict students outcomes on exams. Similarly, the study in [19] focuses on understanding the variations of coding solutions to provide appropriate feedback of students learning at scale. However, to take greater advantage of these data, the systems should be able to interact under a common ecosystem.

Although many theoretical works and proposals of architectures for e-learning ecosystems have been presented [20–22], no specific architecture and a corresponding implementation for a programming learning platform have been proposed. Generally, an architecture adopted to a specific domain and specific activities can add value. With increasing demand for programming education and the potential benefits of OJSs, an architecture specifically devoted to LOs for programming education is necessary.

3 Concept of Ecosystem

The data in an e-learning system can be transformed, replicated, integrated, and shared, making them potentially

[3] https://uva.onlinejudge.org/
[4] https://www.spoj.com/
[5] http://poj.org/
[6] http://codeforces.com/
[7] https://atcoder.jp
[8] https://www.topcoder.com/
[9] https://cms-dev.github.io/
[10] https://www.domjudge.org/

reusable. Data can be used to make the system more intelligent. However, the data must be properly collected, classified, processed, and stored. Here, we review the lifecycle of data to help us design an architecture for the ecosystem. In this section, we formulate the actors and LOs in the ecosystem.

3.1 Actors

Figure 1. Concept of proposed ecosystem for programming education

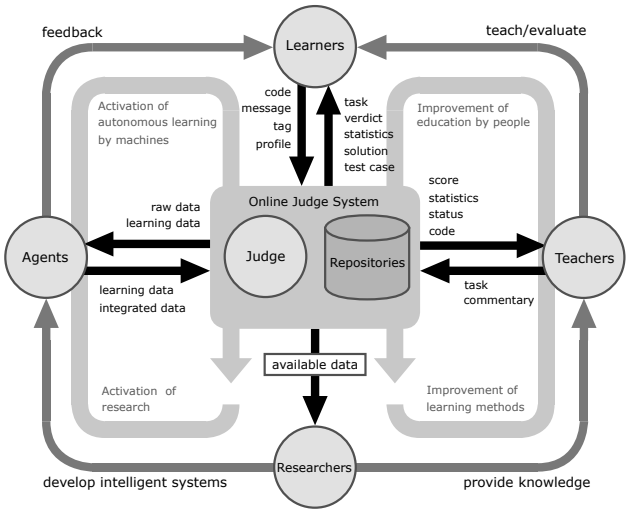


Figure 1 illustrates the concept of an ecosystem with five kinds of actor and OJS data repositories.

Learners are users (e.g., students) who want to learn programming. They search and browse problems (tasks) on the OJS and then try to solve them by coding and submitting the source code. They can also attach their own input/output data for a test submission to the judge. For clarification and discussion, they can post questions to message boards. Voting and tagging are supported. Learners receive a verdict from the judge and can view solutions offered by other uses as well as test cases (judge data) for each problem. Statistics related to problems and learners are available to users.

Teachers are users (e.g., instructors) who support learners in a classroom or group. They create and register tasks, exercises, and contests with special authorization (if needed). Comments regarding various programming languages can be also attached to a task. Teachers can observe learners' activities and solutions. They can instruct and evaluate learners based on the statistics available from the OJS.

Judge is a special software agent that receives programs from learners, makes decisions, and gives verdicts. It generates messages and status related to the decision. The judge manages the queue of submissions and handles them on back-end servers.

Agents are software agents that transform, clean, and integrate the data on an OJS. They provide intelligent feedback to learners on behalf of teachers by utilizing the learning data from the OJS. Agents are developed by researchers and deployed to facilitate development and autonomous learning.

Researchers analyze and utilize the data from the ecosystem to develop intelligent systems. They can input the research results back into the ecosystem by deploying the corresponding agents. They can also share the obtained results with teachers to improve the quality of education.

Agents and researchers can improve education by enhancing the use cases of an OJS, which conventionally involve only learners, the judge, and teachers. The advantage of the ecosystem concept is that learners can obtain feedback from a number of agents in addition to the automatic verification by the judge and instruction from teachers. Researchers can apply their algorithms to the repositories to make the system more intelligent.

3.2 Learning Objects

In this paper, we define any data generated by an actor as an LO. LOs in the ecosystem can be classified based on multidimensional perspectives, as shown in Table 1. The perspectives consider when, how, and by whom LOs were generated, activated, consumed, and processed as well as some features of the data, including data format, size, and frequency.

Notifications are chunks of real-time information broadcast to systems connected to the service. The most common notification is that from the judge for the corresponding submitted code. This notification is used to update the status of the judge queue, scoreboards, and status of other users in real time. Notifications are small but frequent.

Records are information accumulated for an activity by learners and the judge. The main objects created by learners are source code, which can be made public. The main objects created by the judge are messages regarding compilation errors, runtime errors, and program outputs as well as verdicts (Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, etc.) with accuracy and resource usage (CPU time and memory usage). Records can also include personal profiles, trial-and-error submission history, tagging data, and discussion on message boards. Access logs for each LO (e.g., problem description and public source code) are also considered as records.

Materials are learning materials, which are generally created and registered by teachers. They include problem descriptions, images, test cases, and comments. Although access frequency is relatively low, materials can be relatively large (for example, an input file for a test case can be several tens of megabytes).

Statistics are a secondary information resource derived from records and materials. They can be used for statistical analysis and visualization related to users' performance, learning history, problem status, and verdicts from the judge. They include information for the whole system, such as overall user rankings and statistics for each language, as well as individual statistics, such as the daily effort of a learner, their weekly achievement, and ability charts.

Table 1. Classification of learning objects from multidimensional perspectives

Perspectives	Who		What (examples of learning objects)	When/How (time)	Format	Size	Freq.
	Consumer	Creator					
Notification	Learner, Teacher	Judge	verdict, messages, etc.	Push All	Object	Small	High
Record	Learner, Teacher	Learner, Judge	source code, verdict, tagging, logs, etc.	Request	Object	Med	High
Material	Learner, Teacher	Teacher	description, test cases, etc.	Request	Object	Large	Med
Statistic	Learner, Teacher	Agent	ranking, score, statistics, etc.	Request	Object	Med	Med
Feedback	Learner	Agent	hints, debugging, recommendation, etc.	Push Personal	Object	Small	Med
Archive	Researcher	Agent	set of source codes, etc.	Request	File	Huge	Low

Feedback is a chunk of real-time information provided through the intelligent system to support learning, analysis, and other activities. Examples include hints for debugging and recommended problems to be solved considering the learning path as well as other personalized tips. Note that models and the corresponding learning data are required to provide feedback.

Archive is a large chunk of data for specific purposes. It can be used for data analysis, batch processing, and data mining. Examples include a set of submitted source code, judge data, and a set of problem descriptions from a specific source. Integrated data derived from available LOs can also be archived. Although access to an archive is rare, the size of a file can be huge.

4 Architecture

According to the characteristics described in the previous section, the architecture for the ecosystem was designed. We give an overview of the requirements and then present the details.

4.1 Overview

The main functional requirement of the architecture is to cover the whole lifecycle of data in an OJS as well as to provide methods for data acquisition and management for agents and researchers. In addition, the architecture should satisfy non-functional requirements for quality of service and for scaling up through continuous development considering the following observations.

- The amount of information in OJSs will grow exponentially as OJSs become increasingly popular. Therefore, requests should be distributed to provide data with low latency in a robust environment. For example, a server with a monolithic architecture may easily be put under heavy load, affecting other services. Although request processing can be sped up using load balancing through cloud computing and physical scaling up, classification, roles, and transformation of data should be clarified considering the features of each LO for efficient management and development.
- Technologies will change. Technologies for both client side (e.g., web and UI frameworks) and server side (e.g., web servers, programming paradigms, virtualization) are continuously changing, and thus the architecture should not depend on any given technology. A promising solution is to employ a standardized interface and technology-independent data/knowledge representation.
- Requirements are becoming increasingly individualized and diverse. Therefore, the architecture should be scalable so that many applications (agents) can easily be integrated by third parties.

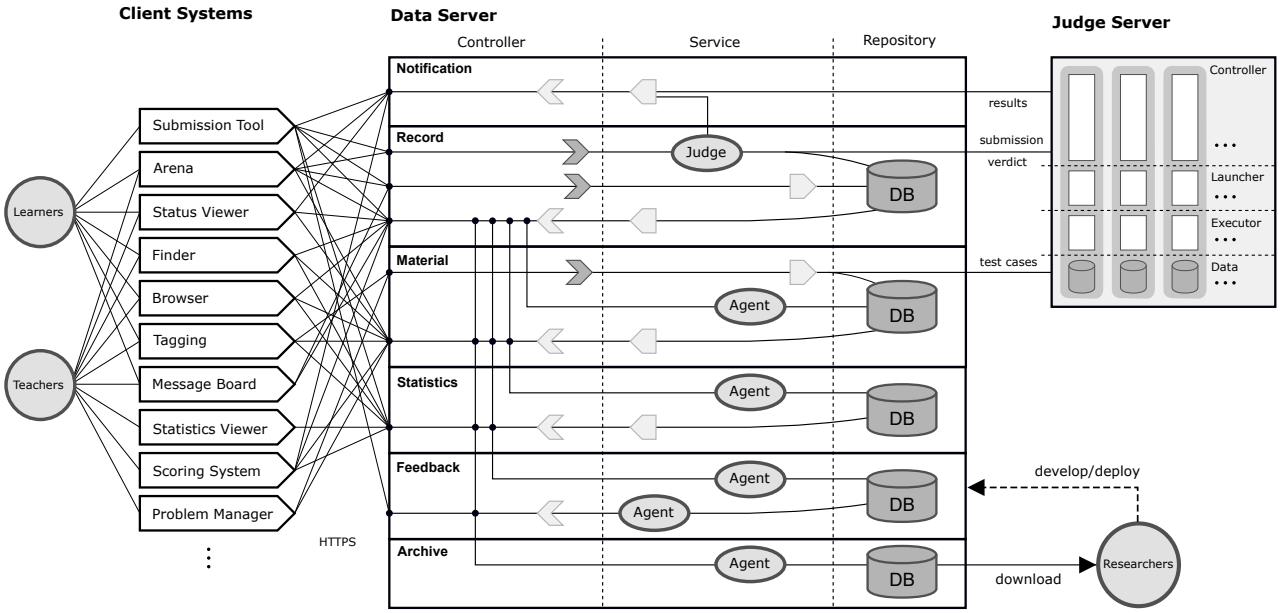
Figure 2 illustrates a generic architecture for the ecosystem. Data Server is used to access LOs and is connected to Client Systems and Judge Server. The main idea is that the proposed architecture is oriented around data, not views (user interfaces). The architecture is thus designed and implemented by applying concepts of resource-oriented architecture, where an application scenario is represented by a uniform resource identifier (URI). Each unique URI, with its associated parameters (keys), is a representation of an LO or a set of LOs (values for the keys). Data Server, the core part of the architecture, does not include any modules for creating views (e.g., HTML pages); instead, it provides data in a certain format (e.g., JSON and XML). The services are thus not tied to any specific operating system or programming language. Learners and teachers get and create data through client systems.

4.2 Data Server

Data Server consists of six layers for the corresponding perspectives, each of which is implemented by three modules. Controller is responsible for receiving and providing LOs as an interface between the client systems and the corresponding data. This is the entry point to get LOs or to post LOs through a URI with parameters. Repository is responsible for storing LOs as persistent data. An LO is mapped to a record in the database table, an object of a key-value store, or a file. An appropriate storage system should be employed for processing massive and structured data from various sources. Service is responsible for bridging the gap between a controller and a repository. Service can deploy transporters, as a bridge, or the judge and agents, which implement business logic for the corresponding layers.

Notification Layer is responsible for receiving information from the judge and broadcasting it to systems connected to the controller. The broadcast should be immediately and asynchronously activated when the corresponding LO becomes available. Real-time notifications make the system convenient to use. Generally, this layer does not need the repository because its role is to quickly forward information. To implement this layer, technologies

Figure 2. Proposed architecture



that can efficiently perform PUSH operations to a huge number of client systems at the same time should be employed.

Record Layer is responsible for processing all activities by the learner, teacher, and the judge and recording them as persistent data. It is considered as the main layer. It includes the judge and communicates with Judge Server. Activities and events from learners and Judge Server should be immediately forwarded to Notification Layer. Data should be cascaded by requests from services in the lower layers. This layer is the entry point for the main activities, so both an unstructured database for storing raw data and a structured database for CRUD (Create, Read, Update and Delete) operations should be deployed. An authentication function must be implemented for this layer so that private content can be accessed only by authorized actors.

Materials Layer is responsible for receiving available materials and forwarding and expanding them to Judge Server. The obtained data are stored in the database. Data should be cascaded by requests from services in the lower layers. Since this layer deals with relatively large data (e.g., judge data), the layer should be deployed considering scalability and latency. Materials are registered when a new LO is available from teachers. They can be obtained through request by any authorized actor.

Statistics Layer is responsible for generating and providing useful objects for analyzing data. Agents deployed on this layer regularly crawl the upper layers and create and update the analytic data as new LOs in the corresponding repositories. The frequency of crawling should be carefully specified so that it can provide the latest (not necessarily real-time) information (for example, statistics for each problem is updated every hour) without excessively loading the upper layers. An LO should be obtained through the transporter as is.

Feedback Layer is responsible for providing and broadcasting useful tips for an individual user. Agents crawl the upper layers and generate data (through a learning process, if any) for the corresponding model. The main difference compared to Statistics Layer is that another agent with business logic answers a request by utilizing the learning model (data) constructed by the crawler. The feedback should be activated when the computational results become available and delivered to the corresponding user in real time.

Archive Layer is responsible for acquiring LOs from all upper layers and providing chunks of LOs as useful archive files. Agents regularly crawl the upper layers to create an archive. For this layer, a relatively huge storage system and logging techniques are required. Content in Archive Layer could be downloaded or shared with researchers with proper authorization.

Record Layer and Materials Layer are crucial for implementing an OJS. The other layers are auxiliary layers used to make the OJS more convenient and intelligent. The six layers are loosely coupled channels that can be deployed on different hardware resources (micro services in the cloud). Required data can be transported and integrated by a cascading mechanism. The auxiliary layers are scalable and a number of systems (agents and the corresponding repositories) can be deployed on different hosts.

4.3 Judge Server

Judge Server should be isolated to allow only authorized processes to connect to it and prevent data leaks by malicious attacks and unintended operations. To improve the quality of service of judge processes as a whole, Judge Server consists of server clusters as a parallel machine, where each server can exclusively use hardware resources and execute a given program. The necessary environment (e.g., operating system, compilers, and virtual machines) should be set up for each judge server in advance. Each

server should be ready to judge all problems in the corresponding OJS. All available test cases and validators from Materials Layer are replicated in each judge server. This is one of the advantages of an OJS based on the proposed architecture; all required environmental settings, data, and validators are deployed and built in advance to realize quick decisions as well as to ensure robust environments. Therefore, only connections (ports) for sending and receiving limited objects should be allowed, including 1) source code (with accompanying data) from Record Layer when the server becomes available for submission, 2) a verdict to Record Layer and Notification Layer when the judge makes a decision, and 3) a set of test cases from Materials Layer when Judge Server is deployed or data are created.

The judge in Record Layer is a core actor that receives submissions and sends them to the queue of the corresponding judge server. The role of the judge is to select a judge server for a submission depending on the language, problem ID, and other parameters, as well as to scale up by adding available hardware resources. It also makes the verdict information from Judge Server persistent data that are linked to repositories. For broadcasting, the judge sends information to Notification Layer when 1) it receives a submission from a learner and pushes it to the queue, 2) the submission is sent to Judge Server from the queue, 3) it receives the corresponding verdict after the judge has finished deciding, and 4) the verdict is made persistent.

4.4 Client Systems

Client systems that connect to Data Server through the HTTPS protocol can be developed independently. Of note, third parties can freely develop client systems using the available URIs and the data on Data Server. Any application based on a character or graphical user interface that can deal with data as an object can be considered. Generally, a client system uses LOs from a set of layers and an integrated application can be constructed with several client views. In this way, we can separate client side and server side development, and thus technological changes on the server side do not affect the client side and vice versa.

In a data-oriented architecture, the implementation of core functions for a web system is made on the client side. Functions include page generation, page transition, management of URLs, and routing mechanisms, and can be implemented in JavaScript and related frameworks, which enable the development of rich and fast front-end interfaces. Although the client side is responsible for implementing many functions, it allows a division of labor, where the front-end and the back-end can focus on a rich UI and data management, respectively.

5 Implementation

In the previous section, a generic architecture that can be implemented using various technologies was presented. In this section, a concrete implementation is demonstrated through the development of AOJ, which consists of Data Server, Judge Server, and client applications.

5.1 Overview

AOJ, developed and operated since 2005, is one of the major OJSs [2]. More than 70,000 users have been registered and almost 4 million solutions have been judged. Almost 100 contests have been held, including Online Open Contests for ACM-ICPC Asia Regionals in Japan 2014-2019. The first submission to AOJ was done in 2005 when the system was internally released. AOJ was opened to the public in 2009. Before 2018, AOJ had a monolithic architecture with Judge Server. From 2018, all services on the server side were reconstructed as Data Server and now AOJ is operated based on the proposed architecture. The user interfaces are provided through a number of client applications.

5.2 Data Server

Table 2. Frameworks and technologies used for implementation of AOJ

Layer	Frameworks	Key Technologies
Notification	Akka, Play (Scala)	WebSocket
Record	Spring Boot, MySQL Hybernate	RESTful API
Material		
Statistics	Spring Boot, MongoDB	

Table 2 shows the key technologies used to implement AOJ for the main layers of Data Server. Each layer is deployed in a docker container on CentOS. The server for Notification Layer was developed based on the Akka toolkit [11] and its Actor model and broadcasting is realized using WebSocket. Akka provides a high-performance, robust, and scalable environment and ensures fault tolerance. Record Layer, Materials Layer, and Statistics Layer were developed based on Spring Boot [12] so that controllers can provide access to resources through a RESTful API [23]. MySQL was used for CRUD operations from repositories in Record Layer and Materials Layer. MongoDB was deployed for key-value operations to obtain objects from Statistics Layer. A total of 124 URIs, which cover most LOs required to implement an OJS, are available from the developer website [13]. Each LO obtained by a URI is represented as an object in JSON format, which can be easily and efficiently parsed in many programming languages (e.g., JavaScript and Python).

Some achievements and challenges for Feedback Layer are presented in Section 6.

In addition to the LOs available from the API, researchers can utilize archive files, presented in Table 3, from Archive Layer. For example, a set of source code from a certain range of submission IDs is available as a

[11] https://akka.io
[12] https://spring.io/projects/spring-boot
[13] http://developers.u-aizu.ac.jp

Table 3. Available archive files

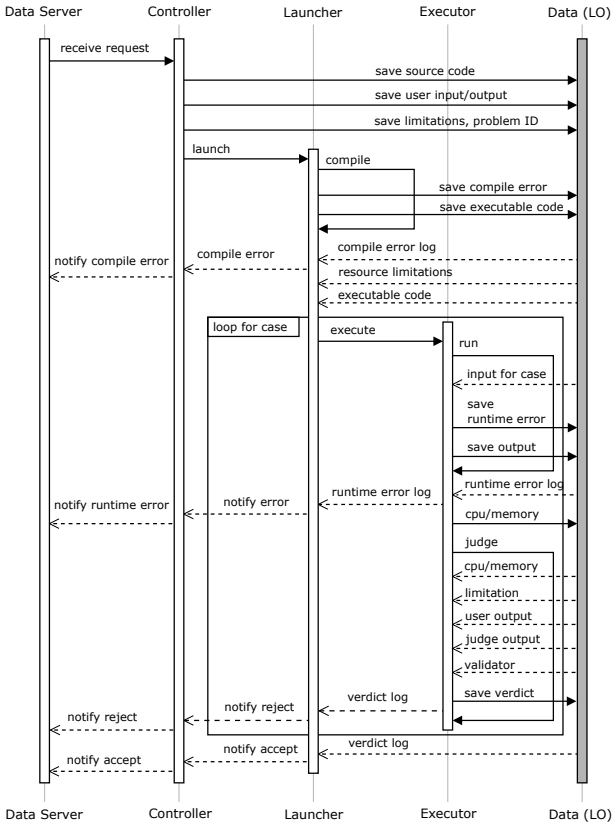
Available Data	Type
source codes in specific range	files in zip directory
test cases of specific problem	files in zip directory
descriptions of all problem	files in zip directory
verdict records in specific range	csv files

zip file from the developer website. We provide not only an archive of learning materials, but also an archive of source code with submission histories, including details of verdicts (for downloading private source codes and related files, special consent for research cooperation is required). Some agents have been implemented to crawl data based on a platform similar to that of Statistics Layer. On the other hand, logging technologies such as Apache Hadoop [14] and Fluentd [15] can also be employed for data proliferation.

5.3 Judge Server

Each judge system of a Judge Server cluster is mainly realized by processes of programs written in Perl on CentOS, where all necessary software and data have been set up. In addition, the necessary security settings are performed using iptables, user management, and resource limitations.

Figure 3. Sequence diagram of a judge system



[14] https://hadoop.apache.org/
[15] https://www.fluentd.org/

The programs are mature (stable versions available for more than 10 years). Figure 3 shows a sequence diagram of a judge system, focusing on what and when LOs are generated as well as communication with Data Server. Some technical details, such as security settings for the sandbox, performance, scaling, and the deployment process, are beyond the scope of this paper. Judge Server consists of three modules, namely Controller, Launcher, and Executor. Controller waits for a request from Data Server. When it receives a request, the solution code and accompanying data, including user-defined input/output (for test submissions), problem ID (pid), language, and resource limitations, are set as data files and Launcher is activated. Following the setup, Launcher tries to compile (if needed) the source code; if this fails, it generates error messages and the process returns back to Controller and then Data Server with the message. If the launch process is successful, according to the number of test cases, the generated code is executed by Executor within the resource limitations. Executor generates a runtime signal and messages if errors occur; otherwise, it generates an output file of the program for each test case. Executor terminates the process of the user program when resource limits are exceeded. After finishing each run, Executor decides the verdict (along with CPU time and memory usage) and sends the information to Data Server. When the solution is rejected or all test cases are passed, control is returned back to Controller with a summary of the verdict.

6 Initial Results and Challenges

The proposed architecture and the implemented Data Server have enabled the development of different systems by the third parties which can contribute to both education and software engineering. The systems include the agents based on machine learning approaches as well as client systems with advanced interfaces. In this section, we present the initial results from research and development based on Data Server and possible challenges for the ecosystem.

6.1 Initial Results

6.1.1 Client Systems

The AOJ system has been migrated to the current version which consists of Data Server with a number of accompanying client applications developed by the third parties in addition to the main interface [16]. First, a problem finder and a browser were implemented as a beta version of the new AOJ interfaces [17] based on a single-page application, which is compatible with the new architecture. Other core applications include an Arena system [18], where teachers (and admin) can set exercises and contests with selected tasks and settings through the special manager. Learners can use Arena to browse, code, grade, and manage their programs. Another major product is Virtual Judge

[16] http://judge.u-aizu.ac.jp
[17] https://onlinejudge.u-aizu.ac.jp
[18] https://onlinejudge.u-aizu.ac.jp/beta/arena.html

[19], which is a virtual contest system that connects to Data Server as a part of judge system. Intelligent Coding Editor (ICE) [20] is a special interface for coding as an online compiler/editor where users can test their program with specified test cases through Data Server.

6.1.2 Detection of Errors in Source Code

One useful function is automatic error indication in source code. Error detection is one of the most important criteria to support novice programmers. In programming practice a significant amount of time is spend in formulating the logic and solution. Thus, it is important to automate the process of logic error detection so that novice programmers can learn about the mistake instantly. Although, various syntax errors can be detected reliably by current state-of-the-art compilers and integrated development environments (IDEs), the task of detecting logic (algorithmic) errors in codes is still a challenge and open research. Various approaches and solutions have been proposed in order to automate the logic error detection. Researches [24–26] approach to solve the problem by pattern analysis of accumulated compiler error messages. In addition, students debugging activities have been taken into consideration. In our research [27], we have leveraged state-of-the-art neural network such as, recurrent neural network (RNN) with long short-term memory (LSTM). This learning model was directly trained on the compiled source code of expert programmers. Thus, during the inference the trained model shows the probabilities for a given sequence of source code. Our approach also extends to static code analysis using the intermediate representation in Abstract Syntax Tree [28]. The results show that the proposed algorithms can detect logic errors in compliable code with high frequency for some specific problems. We can further analyze these approaches to best utilize them and avoid mismatches to provide more intelligent and accurate feedback. Hybrid intelligence which is focusing on strong points of these approach has also been considered [29]. We believe that Archive can be a resource for developing state-of-the-art intelligent and adaptive learning algorithms for source code analysis [30] and intelligent coding editors.

6.1.3 Recommendation of Problems and Learning Path

A learning path is an important guideline for improving education results. We previously proposed a novel RNN-based recommendation system (RC) for recommending a set of tasks (materials, programming problems) to learners considering individual learners' strengths in each category and their preferences [31]. For training the RNN-based RC, we leveraged the accumulated data of AOJ, which contains a sequence of problems previously solved by expert users. The key assumption is that the expert programmers' learning path (in the AOJ database) is an ideal guide for novice programmers. The proposed RNN trained on these data has successfully predicted the next likely problem to be solved. In practice, this work is a big challenge because the process is different from that of conventional collaborative filtering. General collaborative filtering (for example, as used by several services such as Amazon) is not applicable since the recommendation of problems is not always based on a programmer's preferences. A pedagogical sequence based on a programmer's strengths has to be maintained when recommending programming problems. This issue has been solved in the aforementioned research through the clustering of programmers based on strengths.

6.1.4 Estimation and Classification of Problems

For possible preprocessing of the above algorithms, we developed many intelligent expert systems to estimate the difficulties and topics of programming problems. Previously (see for example [32]), we applied fuzzy theory to derive rules that can partition programming problems into several difficulty levels. We have also taken into consideration the problem statements to predict the topics and categories associated with the problems. In one study [33], various natural language processing algorithms, such as latent Dirichlet allocation and non-negative matrix factorization, were applied for topic extraction. In addition, to make the system more adaptive and pedagogical, we have also taken into consideration the strength estimation of programmers. Neural-network-based approaches, such as the self-organizing neural network, have been applied to this problem [34]. Classification of source codes and problems based on Convolutional Neural Network (CNN) has also been performed [35]. Similar work has been conducted for other online judge systems [36]. These achievements demonstrate that personalized search, recommendation, and feedback, as well as human assessment can be improved using machine learning algorithms.

6.2 Challenges

6.2.1 Coding Support

IDEs, which support error highlighting, code completion, and refactoring, are an indispensable tool for software development. The above functions can be realized using machine learning and big data (program code). The latest technology, powered by Visual Studio IntelliCode [21], has had a great impact on software engineering. An artificial intelligence system that utilizes a number of GitHub repositories for learning provides intelligent completion functions with suggestions that consider circumstances other than conventional variable/function lists. In this context, in addition to bug detection, machine learning approaches that employ the accumulated source code in an OJS can revolutionize programming education. Of note, the data in an OJS includes not only typical mistakes by novices but also exemplary code written by talented programmers.

[19] https://vjudge.net/
[20] http://developers.u-aizu.ac.jp/ice/index
[21] https://visualstudio.microsoft.com/

Personalized feedback will also be possible by considering user experience. As one of our research challenges, code completion based on RNN and solutions in an OJS has been considered for programming education [38].

6.2.2 Automatic Generation of Learning Objects

The automatic generation of LOs has became an important research area for intelligent autonomous learning. We can generate secondary LOs by reusing existing LOs in the ecosystem. For example, the automatic generation of pseudocode and comments for source code is an attractive challenge (see for example [37]). For this type of generation, problem descriptions, tags, and source code can be utilized. Another idea is the automatic annotation of source code and problem descriptions to support understanding. For creating programming tasks, automatic generation of fill-in-the-blank programming problems is one of promissing functions to support instructors [39]. The generation of test cases, solution code, and tasks (problem descriptions) are also big challenges. For example, automatic generation of a program code from another code which has the same function but in different programming languages, is attractive for many applications.

6.2.3 Integration of Services via Standard Protocol

The usefulness of the data goes beyond the programming platform itself because every LO can be accessed via a URI and applications can exchange data via REST, and according with the proposed architecture this task is handled by clients. In similar efforts such as MAGIC system, the integration of auto-graders at massive scale for Cloud utilize REST API [40]. As technology is evolving and adopting new standards, the clients (usually LMS, but not limited to) can also connect via Learning Tools Interoperability (LTI) standard. The LO can be seamlessly created and associated as part of the LTI client application resource. Since LTI is now widely adopted by modern LMS based on its protocol style, an OJS can become part of a larger ecosystem. CodeOcean project is an example of an auto-grader using LTI, and CodeHarbor project proposes LO organized as repository, both to reuse programming LO and integrate with other applications [41].

Based on the LTI standard, LOs and coding performance can be combined with material or study paths (in this case, in a guided environment) taken by the student on a MOOC platform such as Coursera or edX, or a LMS such as Moodle or Canvas. This would make it easier for researchers to cross reference user behaviors and interaction with specific content, the path in the OJS, the impact of their materials on coding performance. More accurate feedback could also be provided based on the data in the OJS, as each LO can be clearly identified. Other efforts integrating MOOCs and Automatic grading for improving feedback in programming are presented in [40] [42]. However, the integration and data cross-reference is still challenging due to the different architectures, and lack of ecosystem definition.

7 Concluding Remarks

An OJS can be organized into an ecosystem where learners, teachers, graders, and intelligent software agents are involved. The accumulated LOs allow effective e-learning for programming education. Such an environment gives learners significant advantages and reduces costs in terms of human resources.

In this paper, a generic architecture and the corresponding concrete implementation were presented based on the Aizu Online Judge. We demonstrated that the proposed architecture and implemented data servers have activated development of different interfaces for learning support as well as research for creating intelligent agants by the third parties. We developed some feedback systems related to bug detection, recommendation, and estimation using learning objects, which are open to the public. We also considered research challenges for the ecosystem. Although researchers can utilize data and develop their own agents, we should consider making a framework where they can easily define and deploy their actors to the cloud or the corresponding data server.

8 Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 16K16174.

References

[1] Miguel A. Revilla, Shahriar Manzoor, Rujia Liu. 2008. Competitive Learning in Informatics: The UVa Online Judge Experience, *Olympiads in Informatics* 2 (2008), 131 – 148.

[2] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, Tomasz Sternal. 2018. A Survey on Online Judge Systems and Their Applications, *ACM Computing Surveys* (*CSUR*) 51, 1, Article 3 (2018).

[3] IEEE Standard for Learning Object Metadata 2002. *IEEE Std* 1484.12.1-2002, 1–40 (2002).

[4] Yutaka Watanobe. Aizu Online Judge, https://onlinejudge.u-aizu.ac.jp/

[5] Yutaka Watanobe. 2015. Development and Operation of an Online Judge System, *IPSJ Magazine* 56, 10 (2015), 998 – 1005.

[6] Luisa M. Regueras, Elena Verdu, Juan P. de Castro, Maria A. Perez, Maria J. Verdu. 2009. A Proposal of User Interface for a Distributed Asynchronous Remote Evaluation System: An Evolution of the QUESTOURnament Tool. In *Proceedings of the 9th IEEE International Conference on Advanced Learning Technologies*, 75–77.

[7] Wenju Zhou, Yigong Pan, Yinghua Zhou, Guangzhong Sun. 2018. The framework of a new online judge system for programming education, In *Proceedings of ACM Turing Celebration Conference*, 9–14.

[8] Jordi Petit, Salvador Roura, Josep Carmona, Jordi Cortadella, Jordi Duch, Omer Gimenez, Anaga

Mani, Jan Mas, Enric Rodriguez-Carbonell, Enric Rubio, Enric de San Pedro, and Divya Venkataramani. 2018. *IEEE Transactions on Learning Technologies* 11, 3 (2018).

[9] Jose Paulo Leal, Fernando Silva. 2003. Mooshak: a Web! ›based multi! ›site programming contest system, *Software-Practice & Experience* 33, 6 (2003), 567–581. https://doi.org/10.1002/spe.522

[10] Katerina Georgouli, Pedro Guerreiro. 2010. Incorporating an Automatic Judge into Blended Learning Programming Activities, Lecture Notes in Computer Science, Vol. 6483, 81–90.

[11] David G. Messerschmitt, Clemens Szyperski. 2003. *Software Ecosystem: Understanding and Indispensable Technology and Industry*, 1st ed. The MIT Press.

[12] Konstantinos Manikas, Klaus Marius Hansen. 2013. Software ecosystems - a systematic literature review, *Journal of Systems and Software* 86, 5 (2013), 1294–1306.

[13] Desamparados Blazquez, Josep Domenech. 2018. Big Data sources and methods for social and economic analyses, *Technological Forecasting and Social Change* 130, 99–113 (2018).

[14] Jimmy Lin, Dmitriy Ryaboy. 2013. Scaling big data mining infrastructure: the twitter experience, *ACM SIGKDD Explorations Newsletter* 14 (2013), 6–19.

[15] Roshan Sumbaly, Jay Kreps, Sam Shah. 2013. The big data ecosystem at linkedin, In *Proceedings of International Conference on Management of data*, 1125–1134.

[16] Qinghua Zheng, Huan He, Tian Ma, Ni Xue, Bing Li, Bo Dong. 2014. Big Log Analysis for E-Learning Ecosystem, In *Proceedings of e-Business Engineering*, 258–263.

[17] Paulo Blikstein. 2011. Using learning analytics to assess students' behavior in open-ended programming tasks, In *Proceedings of International Conference on Learning Analytics and Knowledge*, 110–116.

[18] Paulo Blikstein, Marcelo Worsley, Chris Piech, Mehran Sahami, Steven Cooper, Daphne Koller. 2014. Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming, *Journal of the Learning Sciences* 23, 4 (2014), 561-599.

[19] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, Robert C. Miller. 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale, *ACM Transactions on Computer-Human Interaction* (*TOCHI*) - Special Issue on Online Learning at Scale, 22, 2, Article 7 (2015).

[20] Bo Dong, Qinghua Zheng, Jie Yang, Haifei Li, Mu Qiao. 2009. An E-learning Ecosystem Based on Cloud Computing Infrastructure,
In *Proceedings of 9th IEEE International Conference on Advanced Learning Technologies*, 125 – 127.

[21] Lorna Uden, Ince Trisnawaty Wangsa, Ernesto Damiani. 2007. The future of E-learning: E-learning ecosystem, In *Proceedings of Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, 113 – 117.

[22] Hemant Kumar Mehta, Manohar Chandwani, Priyesh Kanungo. 2010. Towards development of a distributed e-Learning ecosystem,
In *Proceedings of International Conference on Technology for Education*, 68 – 71.

[23] Roy T. Fielding, Richard N. Taylor. 2002. Principled design of the modern Web architecture, *ACM Transactions on Internet Technology* (*TOIT*) 2, 2 (2002), 115–150.

[24] Fitzgerald, Sue and McCauley, Renée and Hanks, Brian and Murphy, Laurie and Simon, Beth and Zander, Carol. 2009. Debugging from the student perspective, *IEEE Transactions on Education* 53, 3 (2009), 390–396.

[25] Ahmadzadeh, Marzieh and Elliman, Dave and Higgins, Colin. 2005. An analysis of patterns of debugging among novice computer science students, *Acm sigcse bulletin* 37, 3 (2005), 84–88.

[26] Fitzgerald, Sue and Lewandowski, Gary and McCauley, Renee and Murphy, Laurie and Simon, Beth and Thomas, Lynda and Zander, Carol. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers, *Computer Science Education* 18, 2 (2008), 93–116.

[27] Yunosuke Teshima, Yutaka Watanobe. 2018. Bug Detection based on LSTM Networks and Solution Codes, In *Proceedings of The 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC2018)*, 3531-3536.

[28] Yuto Yoshizawa, Yutaka Watanobe. 2019. Logic Error Detection System based on Structure Pattern and Error Degree, Advances in Science, *Technology and Engineering Systems Journal* 4, 5 (2019),1-15.

[29] Taku Matsumoto, Yutaka Watanobe. 2019. Hybrid intelligence for logic error detection, In *Proceedings of The 18th International Conference on Intelligent Software Methodologies, Tools, and Techniques (SOMET 2019)*, 120-131.

[30] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, Charles Sutton. 2018. A Survey of Machine Learning for Big Code and Naturalness, *ACM Computing Surveys* 51, 4, Article 81 (2018).

[31] Tomohiro Saito, Yutaka Watanobe. 2020. Learning Path Recommendation System for Programming Education based on Neural Networks, *International Journal of Distance Education Technologies* (*IJDET*) 18, 1, Article 4 (2019).

[32] Chowdhury Md Intisar, Yutaka Watanobe. 2018. Cluster Analysis to Estimate the Difficulty of Programming Problems, In *Proceedings of 3rd International Conference on Applications in Information Technology* (*ICAIT*), 23–28.

[33] Chowdhury Md Intisar, Yutaka Watanobe, Manoj Poudel, Subhash Bhalla. 2019. Classification of Programming Problems based on Topic Modeling, In *Proceedings of International Conference on Infor-*

mation and Education Technology (*ICIET*), 275-283.

[34] Chowdhury Md Intisar, Yutaka Watanobe. 2018. Classification of Online Judge Programmers based on Rule Extraction from Self Organizing Feature Map, In *Proceedings of 9th IEEE International Conference on Awareness Science and Technology* (*iCAST*), 308–313.

[35] Hiroki Ohashi, Yutaka Watanobe. 2019. Convolutional Neural Network for Classification of Source Codes, In *Proceedings of IEEE 13th International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC-2019)*, 194–200.

[36] Wayne Xin Zhao, Wenhui Zhang, Yulan He, Xing Xie, Ji-Rong Wen. 2018. Automatically Learning Topics and Difficulty Levels of Problems in Online Judge Systems, *ACM Transaction on Information Systems* (*TOIS*) 36, 3, Article 27 (2018).

[37] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation, In *Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), 574–584 (2015).

[38] Kenta Terada, Yutaka Watanobe. 2019. Code Completion for Programming Education based on Recurrent Neural Network, In *Proceedings of 2019 IEEE 11th International Workshop on Computational Intelligence and Applications (IWCIA)*, 109–114.

[39] Kenta Terada, Yutaka Watanobe. 2019. Automatic Generation of Fill-in-the-Blank Programming Problems, In *Proceedings of IEEE 13th International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC-2019)*, 187–193.

[40] Armando Fox, David A. Patterson, Samuel Joseph, Paul McCulloch. 2015. MAGIC: Massive Automated Grading in the Cloud, CHANGEE/WAPLA/HybridEd@EC-TEL (2015).

[41] Thomas Staubitz, Ralf Teusner, Christoph Meinel. 2017. Towards a repository for open auto-gradable programming exercises, In *Proceedings of IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering* (*TALE*), 66–73 (2017).

[42] Guillaume Derval, Anthony Gego, Pierre Reinbold, Benjamin Frantzen and Peter Van Roy. 2015. Automatic grading of programming exercises in a MOOC using the INGInious platform, In *Proceedings of European MOOC Stakeholder Summit 2015*, 86–91.