International Journal of
Computer & Software Engineering

Review Article | Open Access

Special Issue: Software Architecture

A Software Architecture for Java Programming Learning Assistant System

Nobuya Ishihara[1], Nobuo Funabiki[1*], Minoru Kuribayashi[1] and Wen-Chung Kao[2]

[1]*Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan*
[2]*Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan*

Abstract

For advancements of Java programming educations, we have developed a Web-based *Java Programming Learning System* (JPLAS). JPLAS provides four problems with different levels, namely, *element fill-in-blank problem, value trace problem, statement fill-in-blank problem,* and *code writing problem*, to cover students at different learning stages. Unfortunately, since JPLAS has been implemented by a number of students who studied in our group at different years, the code has become complex and redundant, which makes further extensions of JPLAS extremely hard. In this paper, we propose a software architecture for JPLAS to avoid redundancy. Based on the *MVC model*, our proposal uses Java for the model (M), HTML/CSS/JavaScript for the view (V), and JSP for the controller (C). Besides, adopting a design pattern, the marking functions of the four problems are implemented uniformly. For evaluations, after JPLAS is implemented with this architecture, the number of code files is compared with that of the previous implementation, and the number of additional files is examined for two new functions.

Publication History:

Received: April 24, 2017
Accepted: July 25, 2017
Published: July 27, 2017

Keywords:

JPLAS, Java programming education, Web application, MVC model, Software architecture

Introduction

As a reliable and portable object oriented programming language, Java has been extensively used in industries and educated in schools. To advance Java programming educations, we have developed a Web-based *Java Programming Learning Assistant System* (*JPLAS*) that can assist self-studies of students while reducing workloads of teachers [1-4].

JPLAS has four types of problems to accommodate a variety of students at different learning levels. The first one is the *element fill-in-blank problem* which requires students to fill in correct elements in the blanks in a given Java code. The second one is the *value trace problem* which requires students to answer the actual values of important variables in the code. The third one is the statement fill-in-blank problem which makes students write whole statements that are blank in the code. The last one is the *code writing problem* which requires students to write whole Java codes to satisfy the given specifications. In the former two problems, the answers of students are marked by string matching with correct ones, and in the latter two, they are marked by unit tests using test codes on JUnit. The difficulty level of the problems is designed to increase in this order of the four problems.

As a laboratory project, JPLAS has been continuously implemented and modified by a number of students who studied at different years in our group. Furthermore, several functions in JPLAS have been extended each year. Through this project, we expect that students have experienced programming for practical systems that have been used in Java programming courses in universities. Unfortunately, JPLAS has plenty of redundant classes and methods in the code that have substantially the same functions. Because most of the students did not have sufficient knowledge and experiences on Java programming, they implemented new functions by copying the whole existing code and modifying/adding the related part. They commented out or did not call unnecessary parts of the code that remain in the JPLAS code. As a result, a large number of clone codes [5] have been accumulated in the code of JPLAS, which make it long and redundant.

In this paper, we propose a *software architecture* for JPLAS that can avoid clone codes to the utmost, even when new students implement new functions in JPLAS. This architecture closely follows the *MVC model* as the common architecture for Web application systems including JPLAS. Our architecture basically uses *Java* for the *model (M), HTML/CSS/JavaScript* for the *view (V)*, and JSP for the *controller* (*C*). It is emphasized that Servlet is not used in this case to avoid the possible redundancy that could happen between Java codes and *Servlet* codes where the same functions may be implemented. More specifically, in the model, a design pattern called *responsibility chain* is adopted to handle marking functions in the problems, and the specific functions for database access are implemented such that the controller does not handle them. In the view, the user interface is dynamically controlled with *Ajax*, to reduce the number of JSP files.

For evaluations of our proposal, we implement JPLAS from scratch by following this software architecture. First, the number of code files is compared with the previous implementation, which shows that the number of code files becomes 25%. Then, the number of newly added code files is counted when two functions are added to JPLAS, which shows the number of new files is extremely small for either new function.

The rest of this paper is organized as follows: Section II reviews the outline of the current JPLAS and notes its problems. Sections III and IV present the software architecture for JPLAS and the implementation, respectively. Section V shows evaluations. Section VII concludes this paper with future works.

Review of Current JPLAS

In this section, we review the outline of current JPLAS.

***Corresponding Author:** Dr. Nobuo Funabiki, Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan; E-mail: funabiki@okayama-u.ac.jp

Citation: Ishihara N, Funabiki N, Kuribayashi M, Kao WC (2017) A Software Architecture for Java Programming Learning Assistant System. Int J Comput Softw Eng 2: 116. doi: https://doi.org/10.15344/2456-4451/2017/116

Software platform

In JPLAS, *Ubuntu* is adopted for OS of JPLAS running on *VMware*. *Tomcat* is used as a Web server using *JSP/Servlet*. JSP is a script with embedded Java code within the HTML code, where Tomcat could return a dynamically generated Web page by JSP to the client. *Servlet* is a Java code that will dynamically generate a Web page. *MySQL* is adopted as a database for managing the data.

Four problems in JPLAS

In JPLAS, the four types of problems are provided. For each problem, JPLAS implements various functions which not only for teachers to generate and register new problems but assist students in answering them.

1) *Element Fill-in-blank Problem*: This problem requires students to fill in the blank elements in a given Java code. The correctness of the answers are marked by comparing them with their original elements in the code that are stored in the server. The original elements are expected to be the unique correct answers for the blanks. To help a teacher generate a feasible element fill-in-blank problem, the blank element selection algorithm has been proposed [4].

2) *Value Trace Problem:* This problem requires students to trace the actual values of important variables in a code when it is executed [2]. The correctness of the answers are also marked by comparing them with their correct ones stored in the server.

3) *Statement Fill-in-blank Problem*: This problem asks students to fill in the blank statements in a code. The correctness of the code is marked by using the test code on JUnit that is an open source software for the test-driven development (TDD) method [6]. To help a teacher select blank statements from a code, the program dependency graph (PDG) has been used [3].

4) *Code Writing Problem*: This problem asks students to write a whole code from scratch that satisfies the specifications. The correctness of the code of students are also marked by the test code [1].

Support functions for teacher

JPLAS provides several support functions for a teacher to register a new Java code, generate a new problem using the corresponding algorithm, and register a new assignment by selecting problems and writing the problem statement.

Markers in database

In the database, each problem has been stored in a single text field as a string with the markers in Table 1:

Support Functions for Student

Likewise, JPLAS provides several support functions for students to answer problems. The answer from a student is processed at the JPLAS server by the following steps:

(1) When a student accesses to JPLAS, the list of the assigned problems to the student is displayed.
(2) When a problem is selected by the student, the corresponding problem text in the database is displayed using the marker in Table 1.
(3) The student writes the answers in the corresponding forms.
(4) The answers submitted by the student are marked in the server, and both the answer and the marking results will be saved in the database.
(5) JPLAS offers feeds back to the student.
(6) If necessary, the student could repeat the steps from (3).

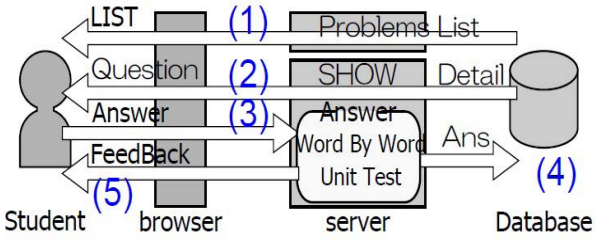


Figure 1: Overview of problem-answering procedure in JPLAS.

Marking Functions

Each problem possesses a unique marking function in the server.

1) *Element Fill-in-blank and Value Trace Problems*: In the element fill-in-blank and value trace problems, each answer from a student is compared with the corresponding correct word or number stored in the database.

2) *Statement Fill-in-blank Problem*: In the statement fillin- blank problem, first, the answer from a student is inserted into the blanks in the problem code, to make a complete code. Then, the two-step marking is applied to this code after creating a working folder on the server: 1) the syntax of the code is verified by compiling it, and 2) only when the compile succeeds, the logic or behavior of the code is tested by the unit test for *JUnit*. In 1), the syntax errors will be informed to the student if they are found. In 2), the error message from *JUnit* will be returned to the student if any test items in the test code fail.

3) Code Writing Problem: In the code writing problem, the same two-step marking is applied to the code of a student.

Marker	description
//@JPLAS answer	the following paragraph contains correct answers
//@JPLAS statement	the following paragraph contains the problem statement
//@JPLAS test code	the following paragraph contains the test code
//@JPLAS output	the following paragraph contains the output of the code
_(under line)	the corresponding element is blanked
//@JPLAS blank	the following statement is blanked
null	problem code

Table: Markers in problem text.

Drawbacks in implementation

As discussed in Section I, JPLAS has been implemented by plural students who studied in our group at different years. In addition, each of the three problems has been implemented by a different student. The code for the code writing problem was first implemented by a student who started the JPLAS project. Then, the code for the element fill-in-blank problem was implemented by another student through copying this existing code and modifying it. The code for the statement fill- in-blank problem was implemented by another student in the similar way.

Therefore, a considerable number of clone codes have been accumulated in the codes for JPLAS and made them long and redundant. Besides, the documents for the code implementations are not properly managed. Furthermore, these implementations did not follow the MVC model explicitly, and did not consider new Web technologies such as Ajax, HTML5, and CSS3. To be specific, Java was used for the model, JSP is of the view, and Servlet is for the controller as shown in Figure 2. Particular functions in the model controls the view directly. As a result, it could be a challenge for new students to make further extensions under the limited time and skills.

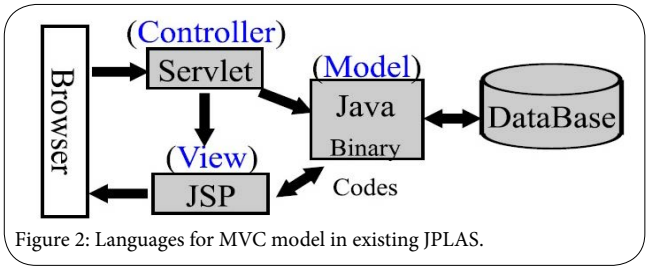


Figure 2: Languages for MVC model in existing JPLAS.

In JPLAS, numerous user interfaces have common items such as the menu. Unfortunately, in the existing codes for JPLAS, the corresponding code part of the common items will be always included in the JSP code for each interface, which causes the redundancy of codes. As well, students need sufficient time to learn the programming by Servlet and the setup of Tomcat for the use. Thus, it is better to implement JPLAS without Servlet.

Proposal of Software Architecture for IJPSL

In this section, we present the software architecture for JPLAS to solve the drawbacks in the current JPLAS implementation.

Adopted languages for MVC model

The proposed software architecture tries to follow the MVC model as strictly as possible as the standard architecture for a Web application system. As illustrated in Figure 3, Java is used for the model, HTML/CSS/JavaScript(JS) are in the view, and JSP is for the controller. Using HTML, CSS, and JavaScript for the view, advanced user interfaces can be implemented using animations and dynamic content changes. Using JSP only for the controller, students 'loads in learning Servlet and Tomcat configurations can be reduced.

Thus, by using suitable languages for each of the MVC model, it is expected to make the implementations of the corresponding codes easy, simple, and flexible. Particularly, by adopting JavaScript for the view, the common parts for several interfaces can be described in one HTML file, while the different parts are realized using Ajax, which can significantly reduce redundant descriptions of the same function called code clones.

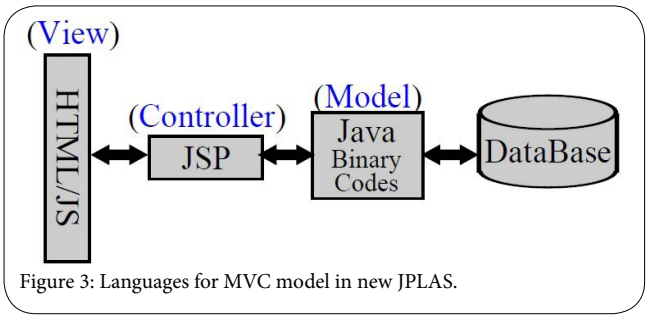


Figure 3: Languages for MVC model in new JPLAS.

Software architecture for model

In the proposed software architecture, the model implements the logic functions of JPLAS using Java. For the independence from the view and the controller, any input/output to/from the model uses a string or its array that does not contain HTML tags.

1) *Database Access:* In the MySQL database server adopted in JPLAS, plural databases can be defined, where each database can have plural tables. JPLAS uses one database to store the information on users, problems, and answers in the corresponding tables. To avoid the redundant codes for implementing common functions in database access in the upper classes, we prepare three classes to handle it. Figure 4 shows the relationships between them.

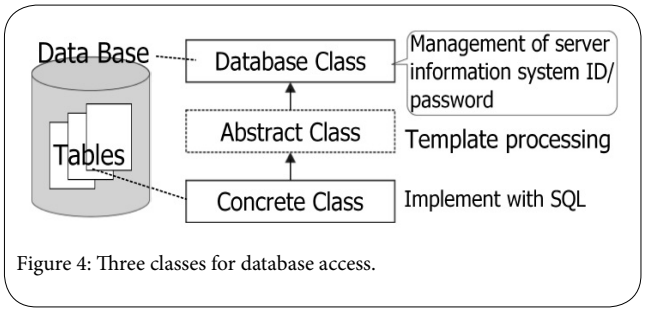


Figure 4: Three classes for database access.

The database class manages the information necessary for database access, such as the server IP address, the port number, the database name, the system ID that is used to connect to the database server in JPLAS, and the password. The system ID and the password are hidden in private variables in this class.

The table class makes a model of each table for users, problems, and answers so that it can be handled by Java. This class deal with the data access to/from the database using SQL commands. If the corresponding table does not exist, this class automatically generates the table using the initialization method.

The abstract class provides the common procedures for database access such as closing the data linkage, and unifying the data type for output. The table class implements this class.

2) *Overview of Marking*: JPLAS provides four different problems where each problem needs a different marking function. To automatically select the proper marking function to the answer from a student, the responsibility chain design pattern is adopted. Then, it is predicted that JPLAS will be easily extended by implementing new Java programming problems and new marking functions with combinations of existing marking functions.

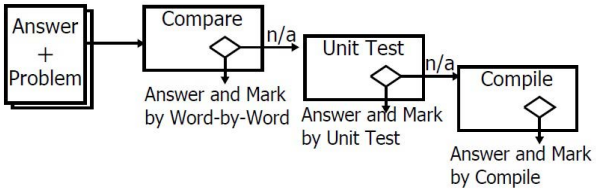


Figure 5: Marking functions by responsibility chain.

As displayed in Figure 5, the specified next marking process is automatically selected when the current process cannot handle the received data of the answer by using the responsibility chain. The class for each marking process has the method to check whether the answer can be handled or not, the method to mark the answer if it can be handled, and the method to specify the next marking process if not.

To give the integrated names to these methods, an abstract class is prepared such that any class for marking becomes its child class. Then, the methods in these marking classes have the names in the abstract class.

For each marking class, the answer data from a student and the correct answer/test code from the database are the inputs, and the strings and the score are the outputs. The output strings are the data coming from the marking class to offer students useful information to complete the answer. They include the compiler error messages, the JUnit output messages, and the marking results. The *word matching* test is specified as the first marking process.

3) *Word Matching Test*: When the data from the database contains the correct answer words of the problem, the *word matching test* is applied. In this marking, each answer is compared with the corresponding correct word in the database. When they are the same, it returns *correct*. Otherwise, it returns *wrong*. In this marking function, the grade can be given by the rate of the number of correct answer words to the total number of correct answer words. Then, the *unit test* is specified as the next marking process.

4) *Unit Test*: When the data from the database contains the test code for the problem, the unit test is applied. In this marking, each answer code is first saved on the disk of the server with the test code. Then, the required commands to the unit test are called.

In the unit test, the answer code is compiled by the Java compiler. If it is successfully compiled, the test code is compiled. After that, the unit test using JUnit is called by calling the command, and the standard output and the log file are obtained by strings. In this marking function, the grade will be given by the rate of the number of successful tests to the total number of tests in the test code. Then, the compiling test is specified as the next marking process.

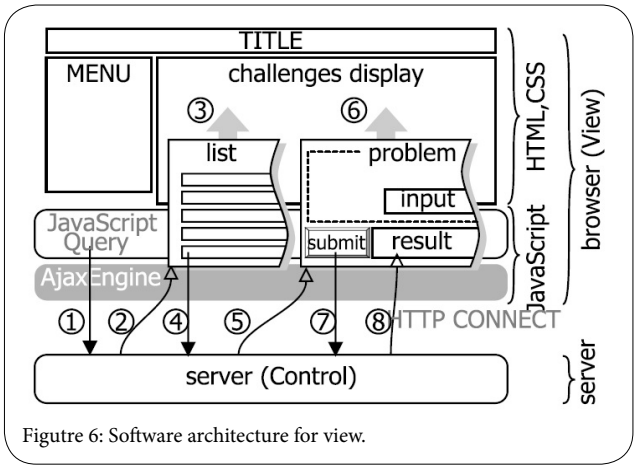
5) *Compiling Test*: When none of the previous tests were applied, the compiling test is applied. In this marking, each answer code is first saved on the disk of the server with the test code. Then, the command to compile the code is called. If it is successfully compiled, the score of one hundred can be given. Otherwise, the score will be zero. There is no next marking process.

Software architecture for view

The view implements the user interfaces of JPLAS by using a CSS framework to provide integrated interfaces using cascading style sheet (CSS) in the Web standard. In this paper, SkyBlue [8]is adopted in the CSS framework, because it can provide proper layouts for multi-size displays without JavaScript codes.

1) *Overview*: Figure 6 illustrates the overview of the software architecture for the view. In a Web page for the user interface in JPLAS, the layout is described by HTML and CSS, where it consists of the title, the menu, and the main body of the assignment. As the feature of this architecture, the fixed parts of the interface are made by HTML with CSS, while the changeable parts of the interface are realized by JavaScript for Ajax. By using different languages for the fixed parts and the changeable parts in the interface, this design can reduce the code size including code clones, and simplify the code architecture.

2) *Communications for Answer Interface*: The communications between the server and the browser for the assignment answer interface are as follows:



Figutre 6: Software architecture for view.

1) JavaScript in the browser requests the assignment list to JSP for the control in the server.

2) JSP on the server sends the list in the table to JavaScript.

3) JavaScript updates the assignment list in the interface.

4) JavaScript requests the details of the clicked assignment submitted by the student to JSP.

5) JSP sends the requested data after embedding the scripts for ”input field”, ”answer button,” and ”result display field”.

6) JavaScript updates the interface.

7) When the student clicks the ”answer button”, JavaScript collects the answers from the student, send them to the server, and requests marking them to JSP.

8) JSP marks the answers.

9) JavaScript receives the marking results and shows them in the interface.

Control

The control in JPLAS is implemented by JSP. When it receives a request from the view, it sends it to Java in the model and requests the corresponding process. When Java in the model returns the processing result by strings, the control changes the format for the view using HTML. The procedure is elaborated as follows:

1) To show the assignment list in the view, JSP in the control receives the list with strings in the two dimensional array, changes them into the table format in HTML, and sends them to JavaScript in the view.

2) To demonstrate the selected assignment in the view, JSP receives the details with strings, changes them into the table in HTML, and sends it to JavaScript.

3) To mark the answers from the student, JSP receives them from JavaScript in the view and sends them to Java in the model. After completing the marking in the model, JSP receives the marking results from Java, changes them into the table format in HTML, and sends it to JavaScript in the view.

Implementation of JPLAS by Proposal

In this section, we describe the implementation of JPLAS by following the proposed software architecture. In this paper, only the functions which allow students to use JPLAS are implemented. For example, the assignments were registered in the server database using SQL commands directly. Also, the implementation of the functions for teachers will be explored in future works.

Overview of implementation

Figure 7 exhibits the flow chart of data communication between the student, the user interface on the browser, the server, and the database.

(1) is executed by JavaScript after the browser receives the initial HTML file "index.html" from the server.

(2) is executed by JavaScript when a problem is chosen by the student. Detailed data are provided by the server.

(3) is executed by JavaScript when the answer is submitted by the student.

(4) and (5) are executed in the server by Java via JSP.

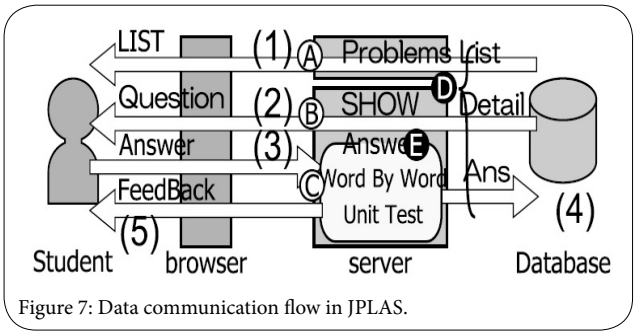


Figure 7: Data communication flow in JPLAS.

Implementation of database access

Figure 8 shows the set of classes to provide the database access functions corresponding to D) in Figure 7. In this figure, "Db.class" represents the parent class for the database access function, which corresponds to "Database class" in Figure 4. "TableDb.class" gives the template to implement each table class, which corresponds to "Abstract class". "UserTableDb.class" implements the table containing the information about the students, "QuestionTableDb.class" processes the table containing the assignments, and "AnswerTableDb. class" does the table containing the answers from the students, which correspond to "Concrete class".

1) *Problem Table and Group Table:* In the JPLAS database, all the registered problems in the four type problems are managed in the "problem table" where each problem is numbered with "ID". Subsequently, several problems among them in a group are given to students as one assignment of JPLAS in a Java programming course by the teacher. The problem groups are managed in the "group table" where each group is numbered with "tag". Each line in this table describes a pair of a tag and an ID to represent the problem IDs in each group tag.

2) Implementation of Table Update: The updates of these tables are implemented using the three classes for database access. First, a class named "TagTableDb.class" was implemented for the new table by implementing the template class "TableDb.class". In "TableDb.class", one method "createDBTabale()" is used for declaring the fields of the database table, and another method "appendRecord()" is for calling SQL commands to add the records.

"createDBTabale()" has the field definition of the database. If the accessed table does not exist, the corresponding table will be created according to this definition. Thus, it is not necessary for the programmer to directly access the database and manually define the table.

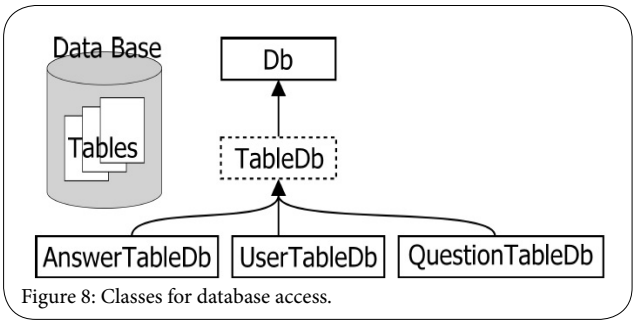


Figure 8: Classes for database access.

Implementation of marking function

Figure 9 revelas the set of classes to provide the marking functions corresponding to E) in Figure 7. Each of the three marking functions in JPLAS was implemented by a class that adopts the abstract class named "Mark.class". The word matching test was implemented in "BlankMark.class", the compiling test was in "CompileMark.class", and the *unit test was* in "TestMark.class". In "Mark.class", the method "canMark ()" judged the feasibility of the corresponding test, and the method "setNext ()" automatically selects the next test. The applying order of the three tests is *word matching test*, *unit test*, and *compiling test*.

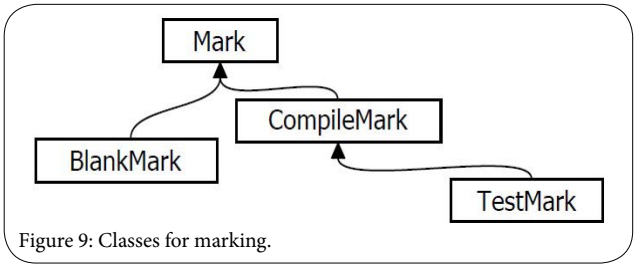


Figure 9: Classes for marking.

Implementation of Uploading/Downloading binary file

JPLAS offers functions to allow a user to download the problem statement, the problem code, the test code, and the correct answer to the PC, and to upload the answer and the marking results in the PC to the server. These data files are compressed into one binary file using the zip format before downloaded or uploaded.

In a database, a binary file is stored as BLOB [9]. To download the binary file, the JSP file needs to put the corresponding header on it. To upload the binary file, the "Commons FileUpload" package is used in the JSP file. IT transforms the binary file into the stream to Java classes [10]. To transmit the binary file on the fly (without storing in the file system) by using the stream as in Figure 10, the following procedure is applied:

1) A User selects the files for uploading/downloading with <input type = "file"> element in the HTML <form> element.

2) The files are transmitted with the POST method.

3) The JSP file describes the destination.

4) The JSP file receives the request and analyzes the original file name for each file.

5) The JSP file transforms the file into an object of "Input-Stream" to the Java binary code.

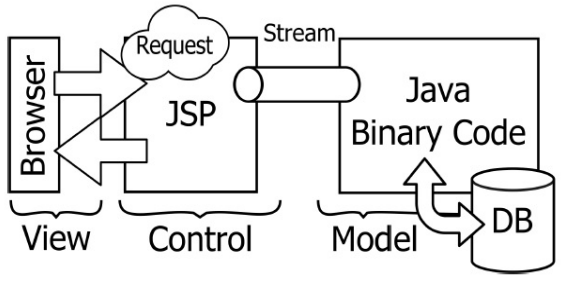


Figure 10: Binary files uploading/downloading.

Evaluation

In this section, we evaluate the proposed software architecture and implementation for JPLAS.

Number of program files in implementation

First, we compare the number of program files between the new implementation in the previous section and the original implementation of JPLAS. Table 2 shows the number of files in Java, JSP, CSS, JavaScript, and HTML, respectively. More specifically, the numbers of files for database access functions in Java and JSP are also presented.. To make accurate comparisons, the number of files is counted merely for the student functions in the previous implementation. This table manifests that the number of required files is reduced by 70% of our proposal in this paper. Only one file is necessary for the database functions in the proposal.

file extension	original	proposal
java database access	81 7	21 1
jsp database access	61 16	12 0
css	9	6
js	40	38
html	122	11

Table 2: Number of program files in two implimentations of JPLAS.

Number of program files for new functions

Next, we evaluate the number of additional program files required to implement two new functions into JPLAS.

1) *Coding Rule Learning Function*: As the first one, we evaluate the number of additional program files when we implemented the *coding rule learning function* [11]. This function will examine whether the code written by the student follows the coding rules or not by using open source software *Checkstyle* [9] and *PMD* [10]. To improve the readability and quality of the code, following the coding rules to write a code is considered as an effective method.

2) *Offline Answering Function*: Then, as the second one, we evaluate the number of additional program files when we implemented the offline answering function [12]. This function allows the students to answer the assignments for the element fill-in-blank and value trace problems using the browser even when they cannot connect to the Internet.

3) *Number of Additional Files for Two Functions*: Table 3 shows the number of additional program files required to implement the two functions in JPLAS. In both functions, no new program files are necessary for the database access. The implementation of the offline answering function needs only five JavaScript files additionally. However, the implementation of the coding rule learning function requires a number of additional program files. In future works, we will analyze the reason and consider the improvements of the proposed software architecture that can minimize the number of additional program files.

file extension	original	proposal
java	1	0
jsp	10	0
css	17	0
js	31	5
html	3	0

Table 3: Number of additional program files for two new functions.

Related Works

In this section, we show our short survey of Web application software structures using Java in a Web server.

In [13], the IBM Knowledge Center introduced the Struts framework and the model-view-controller design pattern. They suggest two JSP models, named *Model 1* and *Model 2*, that can separate content generations or business logics from content presentations using HTML files. *Model 1* uses only JSP pages and Java beans codes. *Model 2* is the MVC model, where Struts helps to develop application software architectures on it.

In [14], Tani et al. proposed a method of implementing the front-end of a Web application using XSLT. It is implemented as a framework, that is, a collection of Servlet classes tobecome components of the target Web software, where the overall logic is written with XSLT.

In [15], Nagao et al. offered *Web Distributed MVC* (*WD- MVC*) architecture for realizing real-time Web applications. *WD-MVC* uses Ajax to convey messages from the Web server to the browsers asynchronously.

In [16], Ree et al. proposed *EcoFW* for end-user-initiative development projects. *EcoFW* focuses on the controller- configuration in frameworks such as Struts, which not only separates the view and the model, but realizes the view by components using Ajax and the JSON-format.

Conclusion

In this paper, we presented the software architecture for Java Programming Learning Assistant System (JPLAS) that reduces redundancy in the codes and follows the MVC model using Java for the model (M), HTML/CSS/JavaScript for the view (V), and JSP for the controller (C), and showed the JPLAS implementation following this architecture. For evaluations, we compared the number of program files with the previous implementation and examined the number of additional files to implement new features/functions in JPLAS. In future works, we will improve the architecture to minimize the number of additional program files and apply it to implement new features/functions in JPLAS.

Competing Interests

The authors declare that they have no competing interests.

Competing Interests

This work is partially supported by JSPS KAKENHI (16K00127).

References

1. Funabiki N, Matsushima F, Nakanishi T, Watanabe K, Amano N (2013) A Java programming learning assistant system using test-driven development method. IAENG. Int J Computer Science 40: 38-46.
2. Zaw KK, Funabiki N, Kao WC (2015) A proposal of value trace problem for algorithm code reading in Java programming learning assistant system. Inform Eng Express 1: 9-18.
3. Ishihara N, Funabiki N, Kao WC (2015) A proposal of statement fill- in blank problem using program dependence graph in Java programming learning assistant system. Inform. Eng Express 3: 19-28.
4. Funabiki N, Tana, Zaw KK, Ishihara N, Kao WC () A graph- based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system. IAENG Int J Computer Science 44: 2.
5. Higo Y, Yoshida N (2011) An introduction to code clone refactoring. Computer Software 28: 43-56.
6. Beck K (2002) Test-driven development: by example, Addison-Wesley, 2002.
7. jQuery. Internet : http://jquery.com/ , Access April 1, 2017
8. SkyBlue. Internet https://stanko.github.io/skyblue/, Access April 1, 2017
9. The BLOB and TEXT Types. Internet https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/blob.html, Access April 1, 2017
10. Commons FileUpload. Internet https://commons.apache.org/proper/commons-fileupload/, Access April 1, 2017
11. Funabiki N, Ogawa T, Ishihara N, Kuribayashi M, Kao WC (2016) A proposal of coding rule learning function in Java programming learning assistant system. Proc. VENOA-2016, pp. 561-566.
12. Funabiki N, Masaoka H, Ishihara N, Lai IW, Kao WC (2016) Offline answering function for fill-in-blank problems in Java programming learning assistant system. Proc ICCE-TW, pp. 324-325.
13. Struts framework and model-view-controller design pattern. Internet : https://www.ibm.com/support/knowledgecenter/SSRTLW_6.0.1/com.ibm.etools.struts.doc/topics/cstrdoc001.html, Access April 1, 2017
14. Tani Y, Mitsuda N, Ajisaka T (2004) A modular method and framework for Web application development using XSLT. IPSJ Tech. Report 2003-SE-144, pp. 131-138.
15. Nagao T, Tsuchiya Y, Morimoto S, Chubachi Y (2007) Realtime distributed MVC architecture using Ajax. Trans IPSJ-PRO 48: 200-200.
16. Ree S, Takeshi C (2011) Web application framework for end-user- initiative development. Proc FIT2011 1: 271-274.

This article was originally published in a special issue:

Software Architecture

Handled by Editor(s):

Dr. Mohammad Alshayeb
Information and computer science Department
King Fahd University
Saudi Arabia