

2008

Authoring and Sharing of Programming Exercises

Somyajit Jena
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Jena, Somyajit, "Authoring and Sharing of Programming Exercises" (2008). *Master's Projects*. 19.
DOI: <https://doi.org/10.31979/etd.gevj-k88n>
https://scholarworks.sjsu.edu/etd_projects/19

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

AUTHORING AND SHARING OF PROGRAMMING EXERCISES

CS 298 Project Report
Presented to
Computer Science Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Computer Science

By
Somyajit Jena
May 2008

ABSTRACT

Authoring and Sharing of Programming Exercises
By Somyajit Jena

In recent years, a number of exercises have been developed and published for educating students in the field of Computer Science. But these exercises exist in their own silos. There is no apparent mechanism to share these exercises among researchers and instructors in an effective and efficient manner. Moreover, the developers of these programming exercises are generally using a proprietary system for automatic submission and grading of these exercises. Each of these systems dictates the persistent format of an exercise that may not be inter-operable with other automatic submission and grading systems.

This project provides a solution to this problem by modeling a programming exercise into a Learning Object metadata definition. This metadata definition describes the learning resource in terms of its contents, classifications, lifecycle and several other relevant properties. A learning Object (LO) is persisted in a repository along with its metadata. This repository supports simple and advanced queries to retrieve LO's and export them to various commercially available or home-grown e-learning systems. In a simple query, keywords given by the user are matched against a number of metadata elements whereas an advanced query allows a user to specify values for specific metadata elements.

Acknowledgments

I am deeply indebted to Professor Cay Horstmann for his invaluable comments and assistance in the preparation of this study.

Table of Contents

1.1	Problem Domain.....	5
1.2	Defining Metadata	9
1.3	LOM Structure	10
2.1	Web-CAT.....	13
2.1.1	Setting Up A Course Offering.....	13
2.1.2	Setting Up An Assignment.....	15
2.1.3	Web-CAT assignment submission	23
2.2	Labrat.....	25
2.2.1	Metadata Structure.....	25
2.2.2	Directory Layout	26
3.1	Java Server Faces (JSF) Framework	29
3.1.1	JSF Programming Model	31
3.2	Java Persistence Architecture.....	34
3.2.1	Entities	35
3.2.2	Object Relational (OR) Mapping	36
3.2.3	Query	40
3.3	Web-Tier Programming Model – JSF, EJB 3.0 and JPA.....	41
.....	41
3.3.1	Entity Management.....	44
.....	44
4.1	CollabX Repository Overview	48
4.2	Integration with External Systems	49
4.2.1	Web-CAT Interoperability	49
4.2.2	Labrat Interoperability	56
4.2.3	Labrat Feature Extension	56
4.3	CollabX Solution Use Cases	56
4.3.1	Use Case – Upload	56
4.3.2	Use Case – Search	62
4.3.3	Use Case – Fork.....	65
4.3.4	Use Case – Download.....	71
4.3.5	Use Case – Critique	74
4.3.6	Use Case – Categorize	76

1. Introduction

One of the many challenges facing e-learning systems is that the course wares that are developed, exists in their own proprietary format in their respective repositories. There are no mechanisms of sharing this repository of knowledge and learning, to the outside world. These course wares can be shared with a large number of students' worldwide thus enhancing instructional efficiency and re-usability of these learning resources. In this era of modern communication and collaboration, the internet as we know it, is a great delivery platform for sharing and authoring learning resources. This will enable new forms of collaboration between educational institutions increasing the overall effectiveness of the educational systems.

1.1 Problem Domain

Today, instructional designers and developers of course content can develop assignments which can be graded by one of the many auto submission and grading tools available in the market. At Virginia Tech, researchers and instructors use Web-CAT, the Web based center for automated testing, for several years to automatically grade students assignments. Web-CAT [9] has an internal scoring strategy that assesses the validity and completeness of the student's tests, which then become the primary indicator of correctness of student's work. A set of "reference" tests can also be provided by the instructor to evaluate a student's work.

Labrat is another such tool which was developed by Cay Horstmann for Wiley to automatically grade student assignments. These assignments are programming exercises that are selected from the student's textbooks published by Wiley. Listing 1 shows a programming assignment developed for this tool and assigned to students of a particular course offering. This programming assignment is presented to students as a task of implementing an instructor provided interface so that each of the student's submission can be tested by the same set of unit tests. Automatic grading of the student's submission requires the instructor to develop reference test cases that can be used to verify different segments of the assignment.

An assignment adapted to the Labrat format will require the following from the instructor:

- Description of the assignment
A write-up on the assignment as shown in Listing 1. It describes the assignment and has instructions for the students to submit the same.
- Describing the assignment
The programming assignment needs to be described in terms of metadata *defined* by Labrat. This metadata describes among other things – main class of the assignment, list of classes that the student must submit and test related data for the submission.

Exercise 7.10 Factoring of integers. Write a program that asks the user for an integer and then prints out all its factors in increasing order. For example, when the user enters 150, the program should print

2
3
5
5

Use a class **FactorGenerator** with a constructor **FactorGenerator(int numberToFactor)** and methods **nextFactor** and **hasMoreFactors** .

```
/**
 * This class generates all the factors of a number.
 */
public class FactorGenerator
{
    /**
     * Creates a FactorGenerator object used to determine the factor of
     * an input value.
     * @param aNum is the input value
     */
    public FactorGenerator(int aNum)
    {
        // TODO
    }
    /**
     * Determine whether or not there are more factors.
     * @return true there are more factors
     */
    public boolean hasMoreFactors()
    {
        // TODO
    }
    /**
     * Calculate the next factor of a value.
     * @return factor the next factor
     */
    public int nextFactor()
    {
        // TODO
    }
    // TODO: instance fields
}
```

Supply a class **FactorPrinter** whose main method reads a user input, constructs a **FactorGenerator** object, and prints the factors.

Listing 1. A typical assignment in Java

The instructor has to package the assignment into a single zip file structure with its own internal directory structure conforming to the format set by Labrat. The *metadata file*, which describes the assignment, needs to be in certain directories for Labrat to recognize it. The directory structure format for sample exercise shown in Listing 1 is shown in Figure 1.

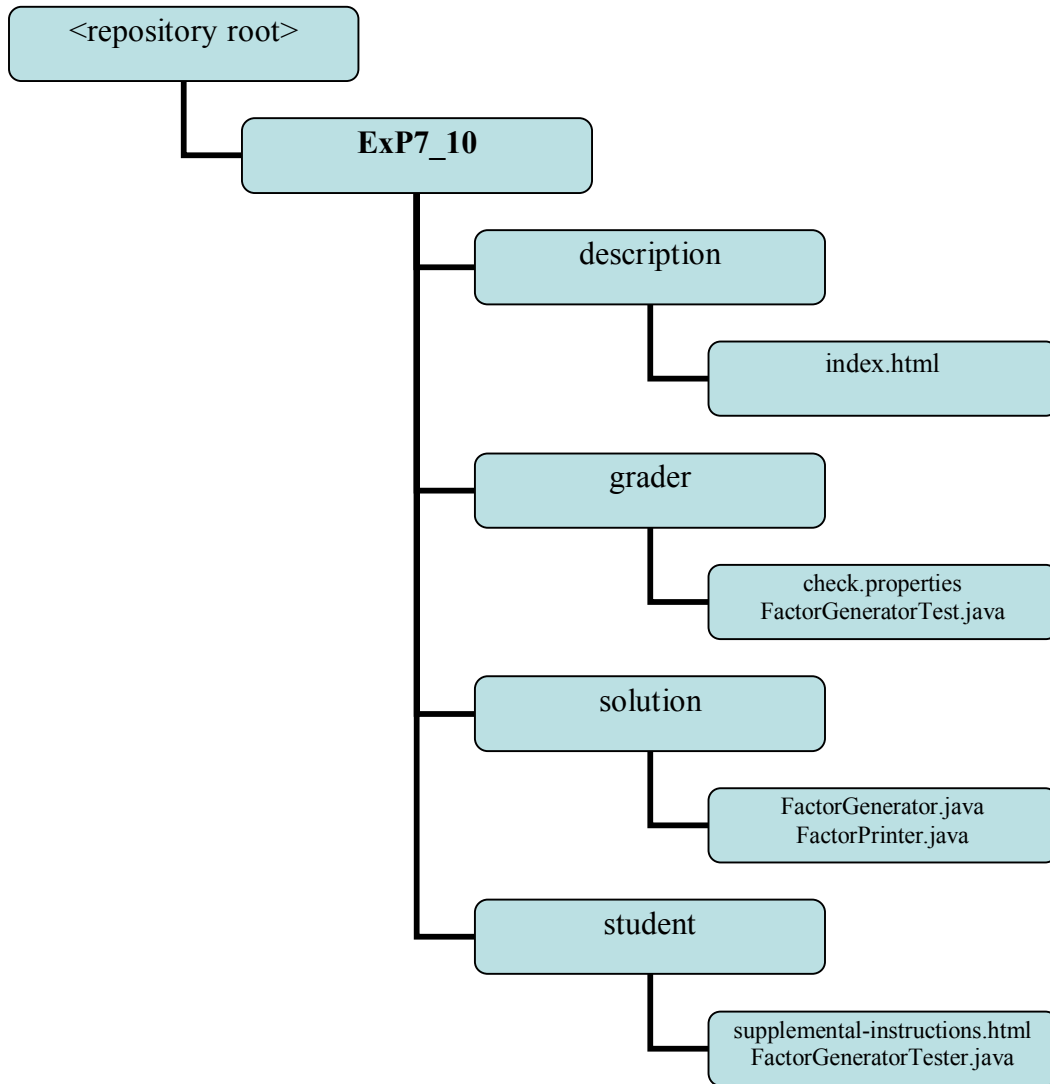
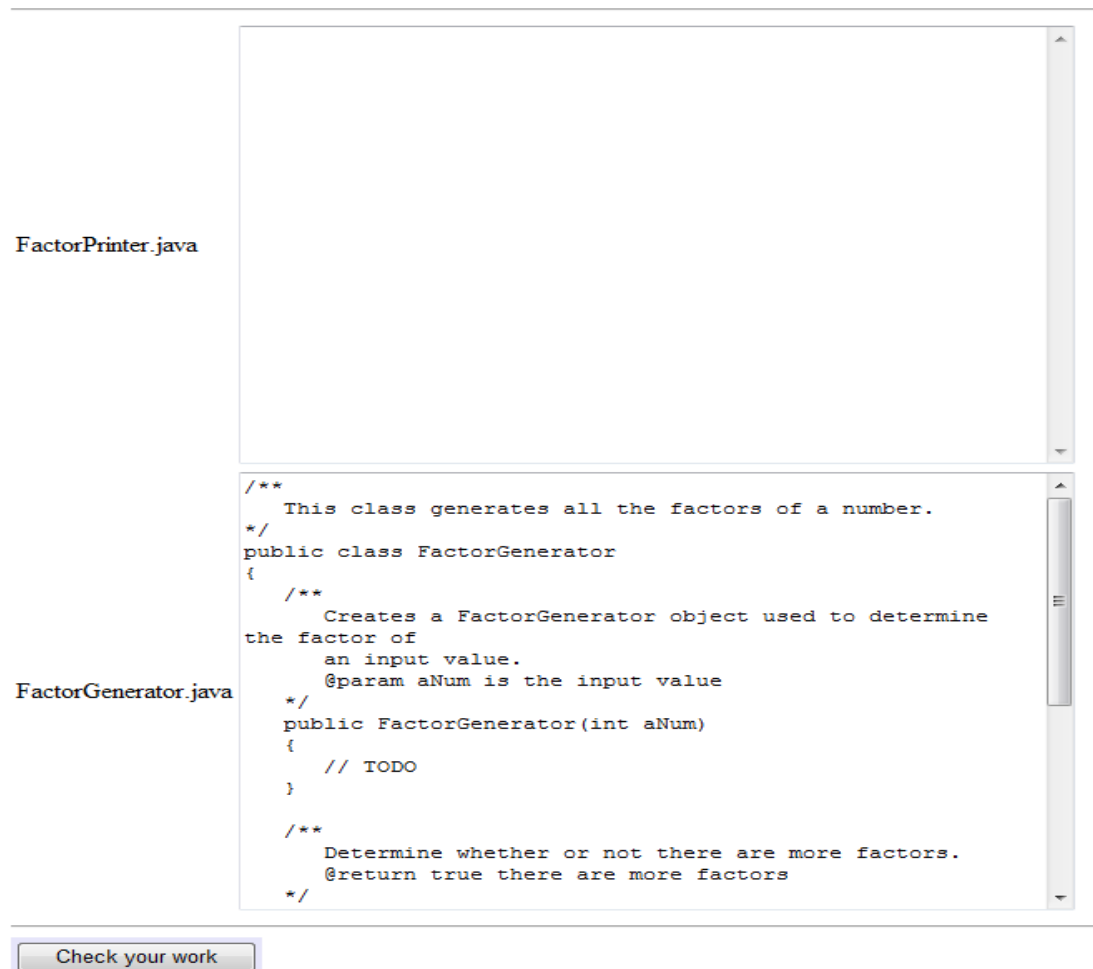


Figure 1 Directory structure of Labrat

The metadata file in Labrat is called `check.properties` and this file is present in the `grader` directory. Labrat will consult this file to prompt for the required files for submission and then automatically grade it in a manner defined by the instructor in the grading metadata.

Students can submit solutions to assignments as shown in Figure 2. Depending on the grading scheme setup in the metadata file, the submission page will prompt for the required files that must be included in the submission.

Submit files



The screenshot shows a submission interface with two text areas for code. The top area is labeled "FactorPrinter.java" and is currently empty. The bottom area is labeled "FactorGenerator.java" and contains the following code:

```
/**
 * This class generates all the factors of a number.
 */
public class FactorGenerator
{
    /**
     * Creates a FactorGenerator object used to determine
     the factor of
     an input value.
     @param aNum is the input value
     */
    public FactorGenerator(int aNum)
    {
        // TODO
    }

    /**
     * Determine whether or not there are more factors.
     @return true there are more factors
     */
}
```

Check your work

Figure 2 Labrat submission page

Once the student submits the required files, Labrat will execute reference test cases as shown in Listing 2 to verify the correctness of the submission and then reports a score to the student and instructor.

```
public class FactorGeneratorTester
{
    public static void main(String[] args)
    {
        FactorGenerator generator = new FactorGenerator(2 * 2 * 3 * 5);
        System.out.println(generator.hasMoreFactors());
        System.out.println("Expected: true");
        System.out.println(generator.nextFactor());
        System.out.println("Expected: 2");
        System.out.println(generator.hasMoreFactors());
        System.out.println("Expected: true");
        System.out.println(generator.nextFactor());
        System.out.println("Expected: 2");
    }
}
```

```

        System.out.println(generator.hasMoreFactors());
        System.out.println("Expected: true");
        System.out.println(generator.nextFactor());
        System.out.println("Expected: 3");
        System.out.println(generator.hasMoreFactors());
        System.out.println("Expected: true");
        System.out.println(generator.nextFactor());
        System.out.println("Expected: 5");
        System.out.println(generator.hasMoreFactors());
        System.out.println("Expected: false");
    }
}

```

Listing 2. Labrat instructor reference test case

Designing a programming assignment as described above shows that there is a substantial effort by a computer science instructor to develop an assignment for an automated grader tool. Specifically for Labrat, the following needs to be provided –

assignment write-up	A write-up on the assignment with instructions for the student on how to submit the assignment.
check.properties	Metadata file which has the grading metadata
Tester.java	Instructor reference test cases
test*.in, test*.out	Test data for inputs and outputs
Packaging	The assignment needs to be packaged as a zipped file archive with its own internal data-structures.

Even if the instructor does wish to share this assignment with the rest of the community, there are no widely used mechanisms that support instructors in finding and sharing such resources. At the same time, other instructors may have to spend a good amount of time and effort themselves in order to reshape a potential assignment into something that can be used in their own courses. The goal of this project is to prototype a shared repository of assignments defined in a common metadata format so that it can be shared with the rest of the community. This common metadata format should be extensible and flexible enough to handle assignments written for any programming language at any level of proficiency. The shared repository will make use of the common metadata format to export an assignment to a target automated grader tool so that it can be adapted or reused without any additional work from the instructor in their courses. As more and more instructors use automated grading tools to process student submissions, there is a growing need for a shared repository of assignments defined in an interchange format, which can lead to a community practice of sharing resources in a way that overcomes existing barriers to such reuse.

1.2 Defining Metadata

The basic step in sharing a learning resource is to tag it with a metadata definition. A metadata definition will describe the learning resource in terms of its contents, classifications, lifecycle and several other relevant properties. It will help an author in authoring or aggregating several learning resources in producing a new one. The two

main metadata schemas and standards used today by digital libraries of educational resources are Dublin Core (DC) and IEEE Learning Object [1] Metadata (LOM) (see Table 1). LOM was released as IEEE 1484.12.1 in June 2002 [1].

The present study will take a subset of data elements that are enumerated in LOM in describing learning resources that are uploaded to the repository. LOM is a structural, offering rich description of a learning resource in terms of data elements that grouped into different categories.

1.3 LOM Structure

Data elements describe a learning object and are grouped into categories. The LOMv1.0 Base Schema consists of nine such categories:

<i>General category</i>	groups the general information that describes the learning object as a whole.
<i>Meta-Metadata category</i>	groups information about the metadata instance itself
<i>Technical category</i>	groups the technical requirements and technical characteristics of the learning object.
<i>Educational category</i>	groups the educational and pedagogic characteristics of the learning object.
<i>Rights category</i>	groups the intellectual property rights and conditions of use for the learning object.
<i>Relation category</i>	groups features that define the relationship between the learning object and other related learning objects
<i>Annotation category</i>	provides comments on the educational use of the learning object and provides information on when and by whom the comments were created

<i>Classification category</i>	describes this learning object in relation to a particular classification system.
--------------------------------	---

Table 1 . LOM Metadata categories

Collectively, these categories as shown in Table 1 form the LOMv1.0 Base Schema. As stated before, this project takes a subset of categories as represented by the data elements i.e. General, Lifecycle, Educational, Rights, Relation and Annotation and creates a metadata representation of a learning resource. The metadata definition and the corresponding learning resources are persisted in the repository. Given below is an example of a learning resource as represented by its metadata definition using the above mentioned data elements-

```

<learning-resource-description>
  <resource-id> LO-78909087 </resource-id>
  <resource-ref-id> LO-098765 </resource-ref-id>
  <title>This is a lesson in Binary Search </title>
  <language> en </language>
  <description> Any description about this lesson </description>
  <keywords>
    <langstring lang="en">binary search tree</langstring>
  </keywords>
  <duration>60 minutes </duration>
  <version> 1.0 </version>
  <level> Easy </level>
  <owner> jdoe </owner>
  <resource-properties>
    <resource-property>
      <name> TextBook </name>
      <type> String </type>
      <value> Introduction to Data Structures in Java </value>
    <resource-property>
    <resource-property>
      <name> Chapter </name>
      <type> String </type>
      <value> Search Algorithms</value>
    </resource-property>
  </resource-properties>
  <resource-collection>
    <resource>
      <resource-id> folder-student-123 </resource-id>
      <parent-ref> folder-ex-4.2 </parent-ref>
      <name> student </name>
      <type> folder </type>
      <iscontainer> yes </iscontainer>
    <resource>
    <resource>
      <resource-id> file-java-123 </resource-id>
      <parent-ref> folder-student-123 </parent-ref>
      <name> DataSet.java </name>
      <type> file </type>
      <iscontainer> no</iscontainer>
    </resource>
  </resource-collection>

```

```
</resource-collection>  
</learning-resource-description>
```

Listing 3. XML representation of an assignment

The above XML listing lays down the metadata information about the uploaded content including its layout and relationships between its contents. This metadata information along with the relationship information will help in the assembly of a Learning Object persisted in the repository.

2. Related and Prior Work

This chapter covers study of the existing systems.

2.1 Web-CAT

Web-CAT is an automatic grading and submission tool developed at Virginia Tech. Course developers or instructors can develop courses in Web-CAT and publish them to their students. Web-CAT automatically grades submitted assignments against an instructor provided solution. Typically, the instructor solution is executed through a test harness, which is a collection of test cases provided by the instructor. These test cases compare the output of a student's submitted assignment with the expected output of the solution provided by the instructor.

Web-CAT has two distinct views for an instructor and students. An instructor has to set up a course offering from a selected list of courses offered by a department of an institution for a particular semester. Students who are enrolled in the course offering can submit their assignments by navigating to their enrolled courses. An instructor has to execute the following set of tasks to create a course offering and assignments for that course-

2.1.1 Setting Up A Course Offering

A new course offering can be created by an instructor choosing the Courses tab and then clicking on New sub-tab as shown in Figure 3.

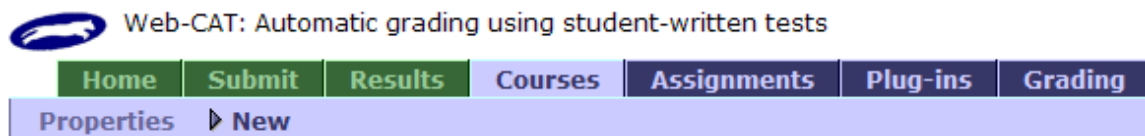


Figure 3 Web-CAT Course Tab

In Web-CAT terminology, the term course refers to the name of an academic course that is offered at a particular institution and is usually associated with an academic catalog's course descriptions. It has a name and a unique identifier, which helps in identifying the department that is offering the course. For example, a course having a name like "CS 40: Introduction to Java Programming" suggests that the name of the course is "Introduction to Java Programming" and is being offered by the Computer Science Department having a unique identifier as "CS 40".

While a course defines the name, course number, and department, a course offering is a specific offering of a given course during a specific semester/term, taught by a specific instructor. In other words, a course offering is a single instance of a course. One course, like "CS 40", might be offered once during the fall semester. Or there could be three separate sections that semester--three distinct course offerings. Those offerings might be taught by the same person, by different people, or even co-taught by a team.

So a course defines set of broad based features that are shared by all offerings. A single course offering defines the specifics for one section/offering of a course during a single semester/term--the instructor(s), the graders or TAs (if any), the student roster, etc.

Figure 4 shows the instructor has to select the course for which he wants to create a new course offering. The list of courses must have already been defined by the department of an institution.

Figure 4 Pick Course

Once a course has been selected, an instance of the course is defined through a set of properties. These properties as shown in Figure 5 are unique to a new course offering.

User ID	Instructor(s)	E-Mail	Action
jsomya	Somyajit Jena	jsomya@yahoo.com	<input type="button" value="Add"/>
<input type="button" value="Add another instructor"/>			

User ID	Graders/Teaching Assistant(s)	E-Mail	Action
<input type="button" value="Add"/>			
<input type="button" value="Add another grader/TA"/>			

Figure 5 New Course Offering

The description of the course properties are enumerated below:

- **Course** – This course is being offered by the department for a particular semester.

- **CRN** – CRN stands for course request number and is a unique identifier for the course offering, distinguishing it from all others. Different academic institutions may have different formats to describe this property, but usually it is an alphanumeric code. It cannot be blank and it has to be unique.
- **URL** – This is an optional property and is web link to the course offering's home site on the web. If there is value in this field, then references to the course offering on other pages will be hyperlinks to this destination. If it is left as blank, those references will just be plain text, not hyperlinks.

The other properties such as Moodle Id and Group Id are specific to courses offers at Virginia Tech and can be left blank.

Additional instructors or graders can be added to this new course offering by clicking on the Add button.

The list of students can be added to the new course offering as shown in Figure 6. They can be added individually or by uploading a CSV file.

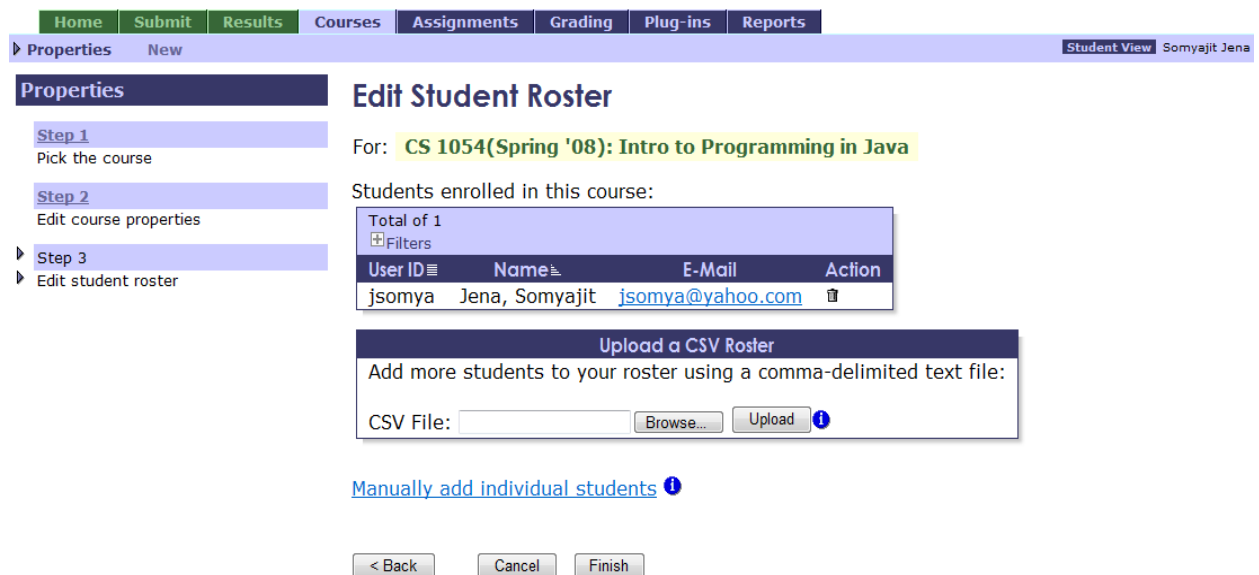


Figure 6 Course Student roster

The new course offering is saved to the database by clicking the Finish button. The new course offering forms its unique identity from the course name, CRN and the time-period for which it is offered at the institution.

2.1.2 Setting Up An Assignment

Once a course offering is set up, users with instructor permissions can create assignments for the enrolled students. An instructor will pick the course from the list of course offerings offered by a department for a particular semester as shown in Figure 7.

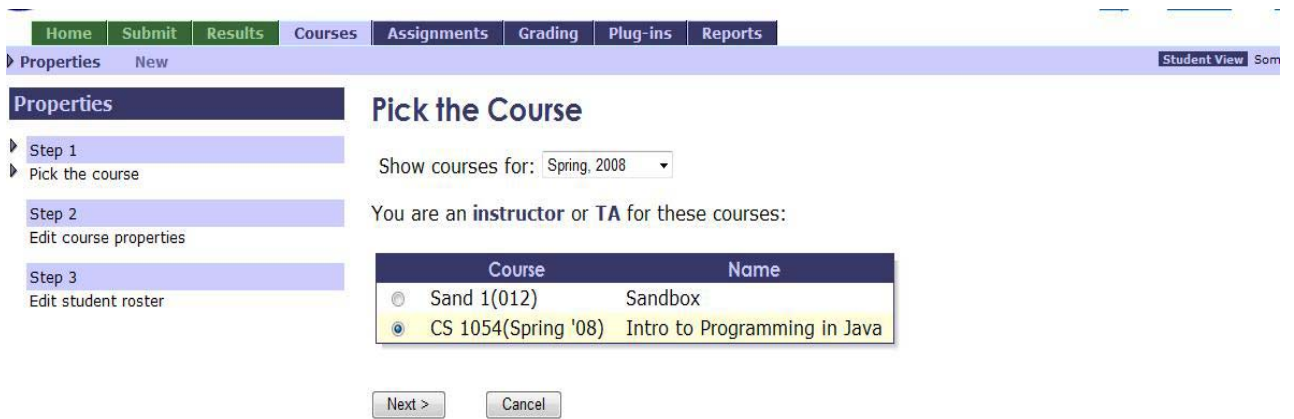


Figure 7 Pick Course For Assignment

Once the instructor has picked the course, he can see a list of assignments already created for the course offering as show in Figure 8.

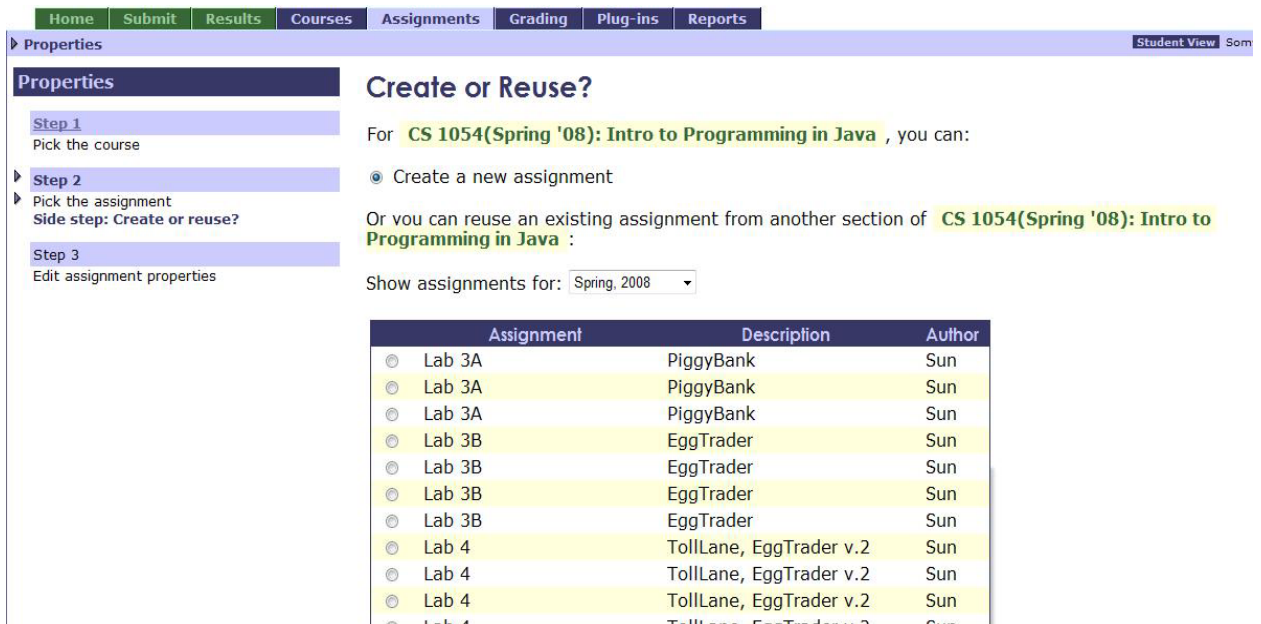


Figure 8 Create or Reuse Assignment

The instructor has the choice of creating a new assignment or re-using an existing assignment. By selecting an existing assignment, one can edit its properties to tailor it to the requirement on hand. If the instructor chose to create a new assignment, it can be a completely new assignment or derived from an existing assignment that had been created before for the course offering. In either case, the assignment specific properties need to be defined or edited.

The term assignment is used to refer to the name and basic properties of a programming assignment – the kinds of things one would associate with an assignment write-up or a program specification. For example, "Program 1: Hello World", might be an assignment used in a CS 101 course.

In Web-CAT, an assignment can be reused across course sections (that is, among many course offerings). The concept of "assignment" is associated with the reusable, shared properties that all offerings of an assignment have in common. Besides the assignment's name and a URL for the corresponding write-up or activity description, an assignment also has a submission policy (including whether or not late submissions are allowed, with or without associated penalties), and a grading scheme, which defines how submissions from students will be processed and scored.

The Edit assignment properties page is shown in Figure 9. This page shows both the shared properties for all instances of this assignment, as well as the properties unique to this course offering.

2.1.2.1 Properties For All Instances Of This Assignment

The following properties are shared by all instances of the assignment. For example if the same assignment is being offered across different sections of a graduate class, then these properties will be retained across all the different instances that were created.

Name	This property is an identifier that uniquely identifies this assignment from all others in this course offering.
Short Description	Optionally, enter a brief description of the assignment. For example Project 1 : Binary Search for a CS 46 course offering.
URL	Optionally, provide an URL which has elaborate information about the assignment. This URL will be hyperlinked with the Name of the assignment.
Upload Instructions	Optionally, enter any assignment-specific instructions that will be shown to students when they upload submissions through Web-CAT's web interface.

Edit Assignment Properties

For: **CS 1054(Spring '08): Intro to Programming in Java**

Properties for All Instances of This Assignment

Name: ⓘ

Title: ⓘ

URL: ⓘ

Moodle Id: ⓘ

Upload Instructions:

```
Supply a class FactorPrinter whose main method reads a user input, constructs a FactorGenerator object, and prints the factors.
```

 ⓘ

Grading Scheme for All Instances

Submission Rules: ⓘ ⓘ

Automatically grade using **these steps** in sequence:

Order	Plug-in	Time Limit (sec)	Move	Action
1	JavaTddPlugin	<input type="text"/>		

Add another step

Properties Unique to This Course Offering

Due: ⓘ [Show upcoming assignment deadlines](#)

Score Summary:

Moodle Id: (overrides shared value if set)

Publish it: ⓘ Suspend all grading: ⓘ

Figure 9 Web-CAT assignment properties

2.1.2.2 Grading Schemes for All Instances

The grading scheme consists of a policy defining the submission rules, and a series of one or more plug-ins that specify how student submissions are processed and scored.

2.1.2.2.1 Submission Rules

The submission rules define the policy that governs acceptance of student submissions, including the points available, limits on the number or size of submissions, the applicable late policy, and any penalties for late submissions (or bonuses for early submissions). A Submission rule for an assignment can be created by Clicking on the New button or can be assigned from a previously created one as shown in Figure 10.

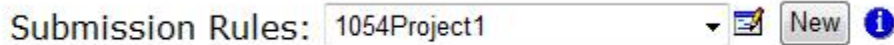


Figure 10 Submission Rules

A new submission profile consists of properties as shown in Figure 11. A submission profile can be configured to allocate different points from manual testing, static analysis tools and correctness testing.

Edit Submission Profile

Basic Properties	
Name:	San Jose State University
Points available:	<input type="text"/> from TA
	<input type="text"/> from static analysis tools
	<input type="text"/> from correctness/testing
	<input type="text"/> Total points available
Max submissions:	<input type="text"/> (blank for unlimited)
Max upload size:	<input type="text"/> (blank for system maximum)
Start accepting:	<input type="text"/> Day(s) before due date (blank for unlimited)
Stop accepting:	<input type="text"/> Month(s) after due date

Bonuses/Penalties	
<input checked="" type="checkbox"/>	Award bonus points for early submissions
	Award at most <input type="text"/> pts in increments of <input type="text"/> for every <input type="text"/> Day(s) early
<input type="checkbox"/>	Deduct penalty for late submissions
	Deduct at most <input type="text"/> pts in increments of <input type="text"/> for every <input type="text"/> Month(s) late

External Submission Engines/Plug-ins	
List assignment for submission using:	Not listed for external submission
Exclude file patterns:	<input type="text"/>
Include file patterns:	<input type="text"/>
Required file patterns:	<input type="text"/>
	(use commas to separate multiple patterns)

Figure 11 Submission Profile

The maximum number of submissions and size of each submission can also be specified in the submission profile. The instructor can also specify when Web-CAT should begin accepting submissions for the assignment – say, two hours before the due date for a

closed lab, or leave blank for unlimited. It can also be specified when Web-CAT should begin refusing submissions for an assignment – say, allowing submissions up to one week after the due date. A value of zero will deny all late submissions. Instructor can also specify that bonus points be awarded for early submissions, or penalty points be deducted for late submissions – say, a penalty of 10 points per day late, or a bonus of 10 points per day early.

2.1.2.2.2 Grading Scheme

The grading scheme configures the list of steps an assignment will be processed upon submission by a student. These lists of steps may include configuring language specific plug-in, static analysis tools, and code coverage tools for student submitted tests. An assignment that requires students to write-up a Java program, the instructor needs to add the `JavaTddPlugin` as shown in Figure 12. This will ensure that student submissions will be processed as per the Java guidelines set in this plug-in.

Automatically grade using these steps in sequence:





Order	Plug-in	Time Limit (sec)	Move	Action
1	JavaTddPlugin	<input type="text"/>	 	 
<input type="button" value="Add"/>		Add another step		

Figure 12 Plug-in Configuration

Various properties that need to be configured for this plug-in is show in Figure 13. The `JavaTddPlugin` plug-in is designed to provide full processing and feedback generation for Java assignments where students write their own JUnit test cases. It includes Ant-based compilation; JUnit processing of student-written tests, support for instructor-written reference tests, PMD and CheckStyle analysis, and Clover-based tracking of code coverage during student testing.

Configure Plug-in

Plug-in: **JavaTddPlugin**

For: **FactoringIntegers2: FactoringIntegers2**

Reusable Configuration Options

This plug-in supports additional option settings that can be placed in a named configuration profile that is shared among several assignments.

Reusable option set:

Assignment-Specific Options

JUnit Reference Test Class(es): SJSU/admin/Exp7_10/webcatbin/Exp7_10_29_TestCase.java

A Java source file (or directory of source files) containing JUnit tests to run against student code to assess completeness of problem coverage. If you select a single Java file, it must contain a JUnit test class declared in the default package. If you select a directory, it should contain JUnit test classes arranged in subdirectories according to their Java package declarations. If you make no selection, an empty set of instructor reference tests will be used instead.

Supplemental Classes for Assignment: SJSU/admin/Exp7_10/classes.jar

A jar file (or a directory of class files in subdirs reflecting their package structure, or a directory of multiple jar files) containing precompiled classes to add to the classpath when compiling and running submissions **for this assignment**. If you want to apply the same jar settings to many assignments, use the "Supplemental Classes" setting in the "Reusable Configuration Options" section instead. If you have multiple jars to provide, place them all in the same directory in your Web-CAT file space and then select the whole directory.

Data Files for Student: SJSU/admin/Exp7_10/webcatdata

A file (or a directory of files) to place in the student's current working directory when running his/her tests and when running reference tests. The file you select (or the entire contents of the directory you select) will be copied into the current working directory during grading so that student-written and instructor-written test cases can read and/or write to the file(s). The default is to copy no files.

Figure 13 JavaTDDPlugin Configuration

The assignment-specific options for the JavaTDDPlugin are shown in Table 2.

Configuration Parameter	Parameter Value	Description
JUnit Reference Test Class(es)	SJSU/admin/Exp7_10/webcatbin/Exp7_10_29_TestCase.java	A Java source file (or directory of source files) containing JUnit tests to run against student code to assess completeness of problem coverage. If a single Java file is selected, it must contain a JUnit test class declared in the default package. If a directory is selected, it should contain JUnit test classes arranged in subdirectories according to their Java package declarations.

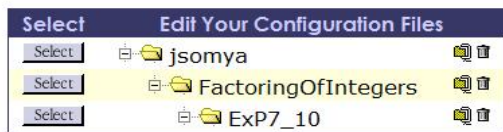
Supplemental Classes for Assignment	SJSU/admin/Exp7_10/classes.jar	A jar file (or a directory of class files in subdirs reflecting their package structure, or a directory of multiple jar files) containing precompiled classes to add to the classpath when compiling and running submissions for this assignment.
Data Files for Student	SJSU/admin/Exp7_10/webcatdata	A file (or a directory of files) to place in the student's current working directory when running his/her tests and when running reference tests. The file selected (or the entire contents of the selected directory) will be copied into the current working directory during grading so that student-written and instructor-written test cases can read and/or write to the file(s). The default is to copy no files.

Table 2 . JavaTddPlugin Configuration parameters

Each of the file system paths shown in Table 2 can be configured as shown in Figure 14.

Edit Your Configuration Files

Any file modifications made on this page will cause all pending changes to be applied. All file actions take effect immediately.



Create a New Subfolder

In: new folder name:

Upload a New File

In: upload:

unzip/expand archive in target folder

Replace This Entire Folder

Replace with contents of this zip or jar file:

Figure 14 Configuring Java Plug-in

Each parameter value shown in Table 2 is a relative path to the resource from the root folder of the assignment. When a submitted assignment is being graded, the

JavaTDDPlugin reads these configuration values to fetch the various resources from the respective file paths.

2.1.3 Web-CAT assignment submission

Students can submit assignments by logging in to Web-CAT and clicking on the Submit tab as shown in Figure 15. It has a list of assignments for a course that the student has enrolled in.

The screenshot shows the Web-CAT interface with a navigation bar at the top containing 'Home', 'Submit', and 'Results' tabs. Below the navigation bar, there are links for 'Status', 'My Profile', and 'Feedback'. The main heading is 'Your Web-CAT Status'. Underneath, there is a section titled 'Assignments Accepting Submissions' which contains a table with the following data:

Assignment	Due	Score Distribution	Action
CS 1054(Spring '08) Project 1: Factroing of Integers: Factoring of Integers	06/30/08 11:55AM		

Below the table is an 'Announcements' section with the following text:

If you are new to Web-CAT, you might want to watch this [movie of making a submission](#), which also explains how to interpret results.

An **Internet Explorer bug** may prevent you from downloading files from Web-CAT. You can find out the cause of this bug, as well as available workarounds, in this [article on the Web-CAT blog](#). Some Firefox bugs are also covered in other blog articles.

Daily down time is scheduled at **4:00AM each morning**. Web-CAT will automatically shut down for a few minutes for house cleaning, and should resume operation automatically within 5-15 minutes. Plan your actions accordingly.

Below the announcements is a 'System Status' table with the following data:

System Status	
Up since	05/31/08 04:03AM
Next scheduled down time	4:00AM
Current users	1
Queued jobs	0
Jobs processed	0
Most recent job wait	00 seconds
Average time per job	30 seconds
New submissions processed in about	30 seconds
Halted assignments	1
Stalled jobs	0

Figure 15 Listing of student's assignment

The student selects the assignment and clicks on the Submit icon as shown in Figure 16.

This screenshot is a close-up of the table from Figure 15, focusing on the 'Action' column. It shows the 'Submit to this assignment' button next to the assignment link.

Figure 16 Selecting assignment for submission

Submissions to Web-CAT are always in the form of a single file as shown in Figure 17. If the solution consists of multiple files, these need to be combined into a single archive. Web-CAT currently accepts zip, jar, tar, and tgz archives.

After browsing for and selecting the submission file, click "Next".

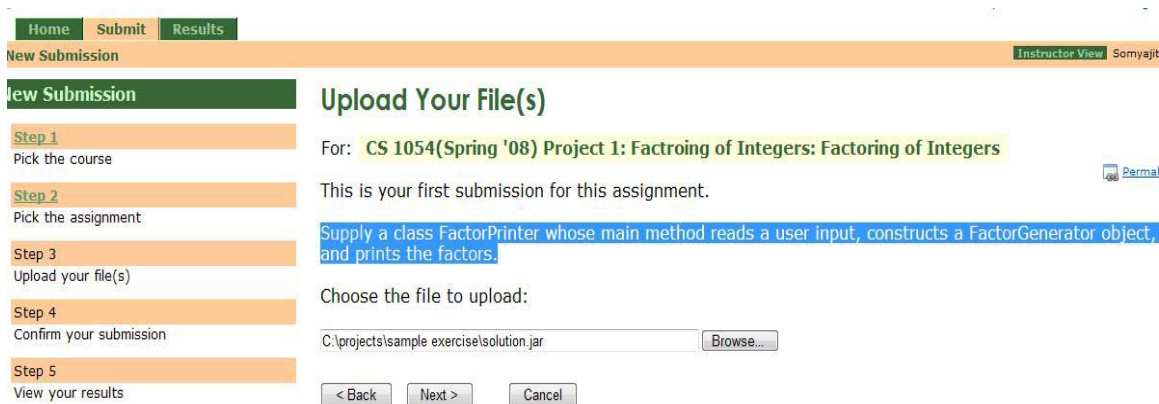


Figure 17 Upload files for submission

Web-CAT will summarize the information it knows about the file that has been uploaded, including its name and size. If an archive has been uploaded, it will summarize the contents of the archive so that the student can make sure that all relevant files included in the archive will be submitted. Click “Next” to confirm the submission. Once the files are submitted, each submission will be tagged by the timestamp of the submission. This will help in keeping track of multiple submissions by a student. Each submission status can be checked by the student from the Web-CAT home page. Once the submission has been graded, its results will be published as shown in Figure 18.

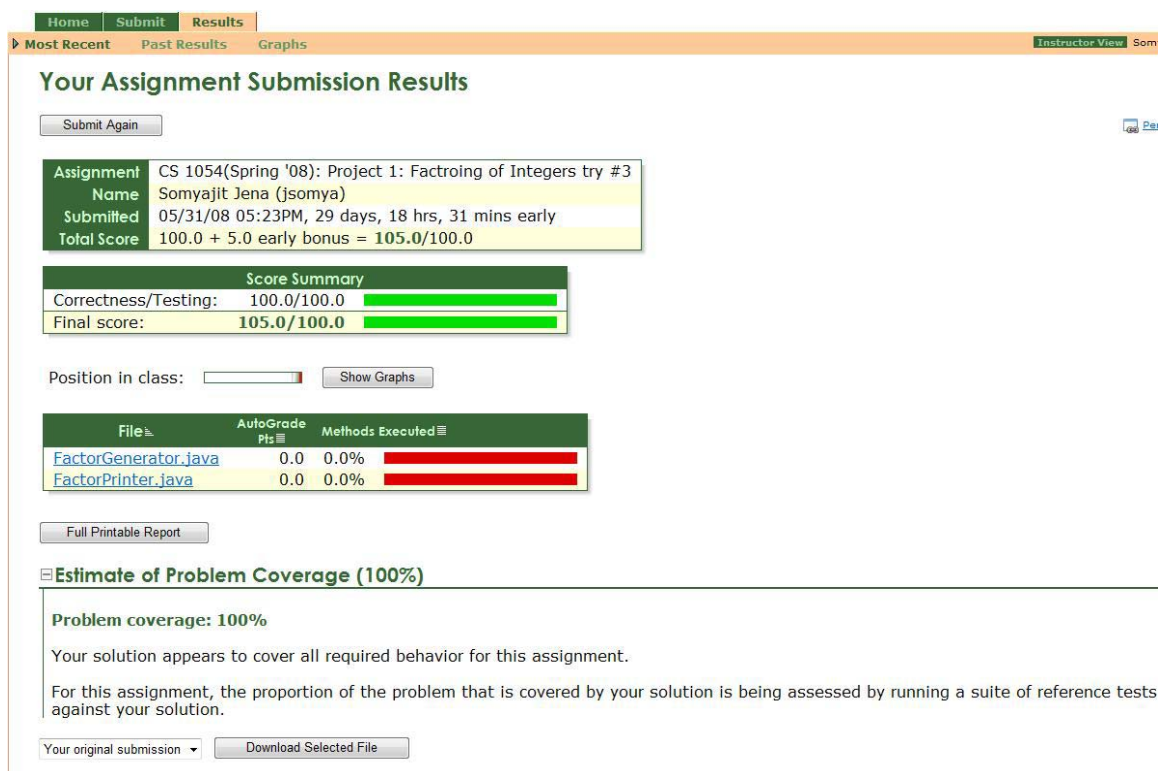


Figure 18 Web-CAT submission results

The results shown in Figure 18 are the result of running the instructor’s reference tests on the submitted assignments. The assignment was configured by the instructor to execute only the instructor provided reference tests. If it would have been configured to execute static analysis, check style and code coverage then these additional results would have been displayed in the submission results page.

2.2 Labrat

Labrat is an automatic submission and grading tool developed by Cay Horstmann. It is an extension of a popular open source program called Ant, which is used by developers for automating their software build process. Each lesson is defined by a metadata structure that is used in the student submission and grading process.

2.2.1 Metadata Structure

The metadata of the learning lessons defined by Labrat is in a property file called `check.properties`. Given below is an illustration of a subset of properties that are used to automate the submission and grading of a lesson –

check.properties	
<code>mainclass</code>	The main class of the assignment. This property is required
<code>requiredclasses</code>	comma-separated list of classes that the student must submit.
<code>optionalfiles</code>	If true then during submission of files, the student will be presented with more files dialog. If false then asking for more files will be suppressed other than the main class and required classes.
<code>test.token</code>	This property indicates the type of output comparison to be made with the expected value. This value can be one of the 4 types: line, word, number or a regular expression. The default value is line.
<code>test.tolerance</code>	The degree of tolerance to use when the output comparison type is number.

Table 3 . Sample Properties

A student while submitting a lesson has to upload the following files –

- `${mainclass}.java`
- All java files for `requiredclasses`.
- Any other `*.java` files if `optionalfiles` is not false.

Labrat reads the `check.properties` file of the lesson being submitted to verify that submission is in accordance with the values that are set in the properties file. The files that are submitted are stored in a persistent repository in a Labrat specified directory layout. This layout is important as the grading process will read the submitted files from a designated directory.

2.2.2 Directory Layout

The directory layout of a lesson in Labrat format is given in Figure 1. The submitted files for a lesson are stored in the `student` directory. This directory could also contain “Horstmann style” unit tests. These unit tests are Plain Old Java Objects(POJO) and have a very simple routine to execute the submitted assignments known methods and compare the return value with the expected value. A scoring strategy is used in Labrat that takes into account the failure rate of these comparisons and a score is assigned to the submission.

An example of a “Horstmann style” unit test is given below –

```
/**
 * This is a test driver class for DataSet class.
 */
public class DataSetTester
{
    public static void main(String[] args)
    {
        DataSet a = new DataSet();
        a.add(5);
        a.add(6);
        a.add(8);
        a.add(9);
        System.out.println("count: " + a.getCount());
        System.out.println("Expected: 4");
        System.out.println("average: " + a.getAverage());
        System.out.println("Expected: 7");
        System.out.println("standard deviation: " +
            a.getStandardDeviation());
        System.out.println("Expected: 1.83");
    }
}
```

Listing 4. Horstmann Style Unit Test

These style of unit tests have a file naming pattern like `*Tester*.java`. Labrat will execute these files to compare and evaluate the outcome of a submitted program by a student. In the above example it is assumed that the `$(mainclass).java` translates to the entry `DataSet.java` which this tester class tests.

The grading process begins by verifying that submission has at least the `$(mainclass).java` and all other java files mentioned in the `requiredclasses` property. After a successful compilation process, Labrat looks for Horstmann style unit tests files that has the pattern `*Tester*.java` in the student directory. This strategy of

a simple tester class probably gives the simplest test evaluation strategy for a submitted program. The grading automated script, which is part of Labrat, is given below –

```
<target name="tester" depends="compile" if="test.tester">
  <java classpathref="runclasspath"
    dir="{submit.dir}"
    classname="{mainclass}"
    failonerror="true"
    timeout="{test.timeout}"
    outputproperty="mainclass.out"
    errorproperty="mainclass.err"
    fork="true">
    <assertions>
      <enable/>
    </assertions>
  </java>
  <echo message="{mainclass.out}" />
  <echo message="{mainclass.err}" />
  <condition property="tester.fail" value="Output not as expected">
    <not>
      <asexpected value="{mainclass.out}"
        tolerance="{test.tolerance}" />
    </not>
  </condition>
</target>
```

Listing 5. Labrat Grading Script

The Ant target defined above compares the program output with the expected output. An error is flagged if the corresponding output mismatches.

The contents of an assignment designed for Labrat includes converting the set of artifacts that make up the lesson into a directory structure as shown in Figure 19, a `check.properties` file that defines the metadata of the lesson and a submission page for the students whose content is derived by introspecting the property file and a grading process which evaluates the submitted program.

As part of my project, I extended the web-interface of Labrat with the addition of the following features:

- Designed a new UI interface for Labrat
- Supported uploading of a lesson in the Labrat format by an instructor.
- Generation of `check.properties` file through a wizard style interface.

3. Architecture

The project implementation is a 3-tier JavaEE based enterprise application as shown in Figure 19. JavaEE defines a set of standards to break apart a monolithic application into multiple layers each defined by a set of standardized modular components handling many details of application behavior – such as security and multithreading – automatically.

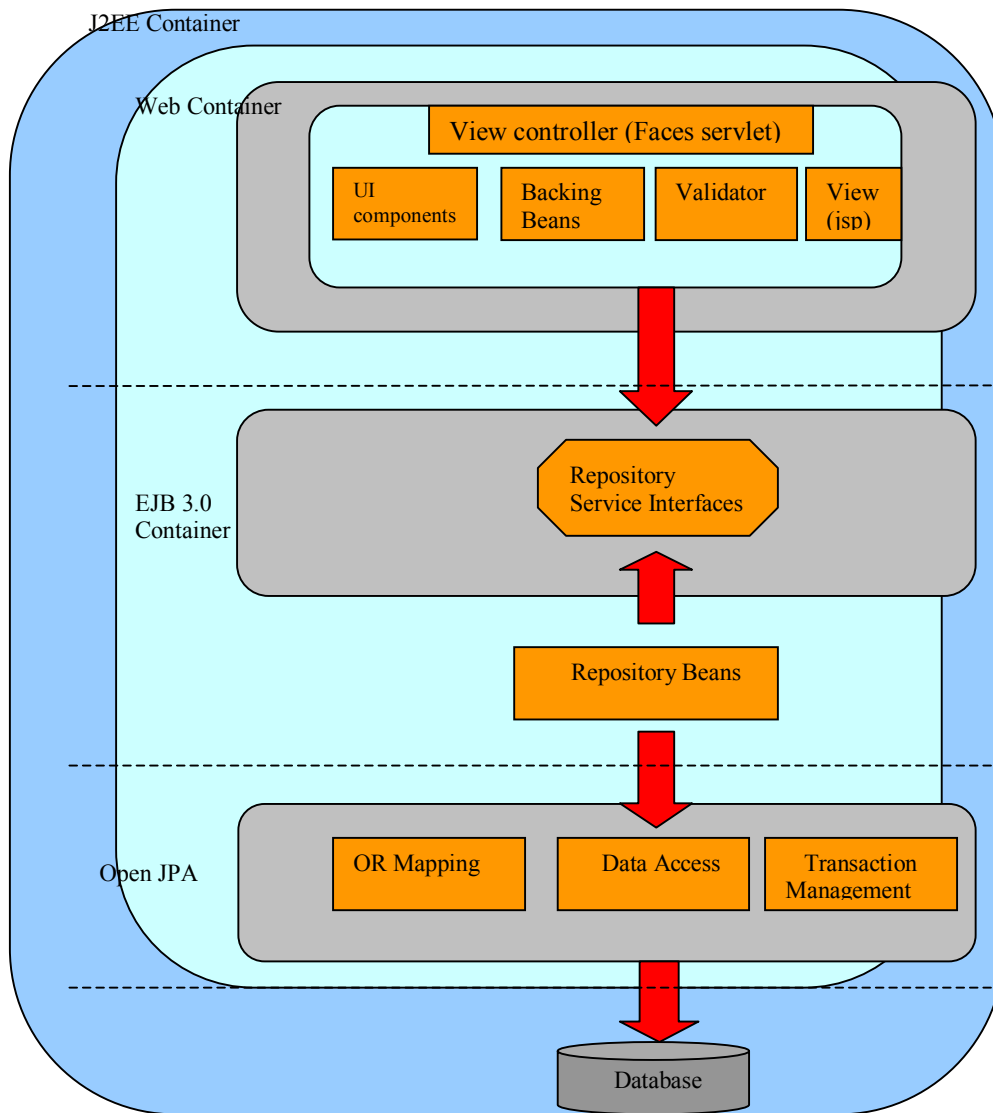


Figure 19 JavaEE Layered Architecture

A 3-tier application provides separate layers for each of the following services:

- Presentation: In a typical Web application, the presentation logic is built using JSPs, JSF, servlets, or XML (Extensible Markup Language) and XSL (Extensible Stylesheet Language).

- Business logic: Business logic is typically implemented in the EJB layer.
- Persistence and Data Access Layer: Data access is best implemented by a persistence framework, which can serialize and de-serialize entities into a persistent store.

The application is deployed on Sun Glassfish Application Server and is built on top of the following technologies (refer to the Figure 20 above):

- Java Server Faces Framework
- EJB 3.0 Framework
- Open JPA Data Access Framework

3.1 Java Server Faces (JSF) Framework

Java Server Faces is a standard web user interface framework for the JavaEE platform. It is based on component architecture and a rich MVC style infrastructure. It has a basic set of server side components and an event driven model to synchronize UI components with the application objects. Figure 20 gives a high level overview of the JSF architecture.

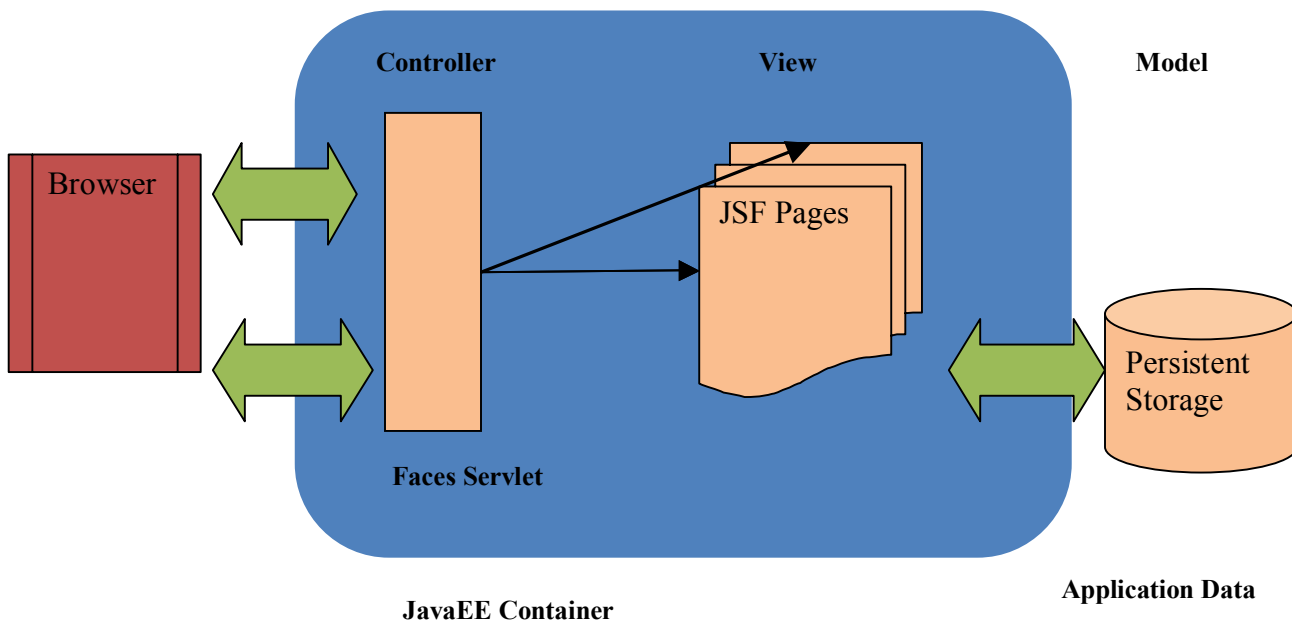


Figure 20 JSF Request Handling

The JSF framework connects the view with the model while rendering the web pages. In this implementation a view component can be wired to properties of a backing bean object, and buttons to event handlers and action methods. JSF acts as a controller that takes in the user input and routes the updates to the target bean object. The view implementation of JSF is a collection of UI components that are managed by the framework. Each UI component can maintain state and views are constructed from composition of different UI components.

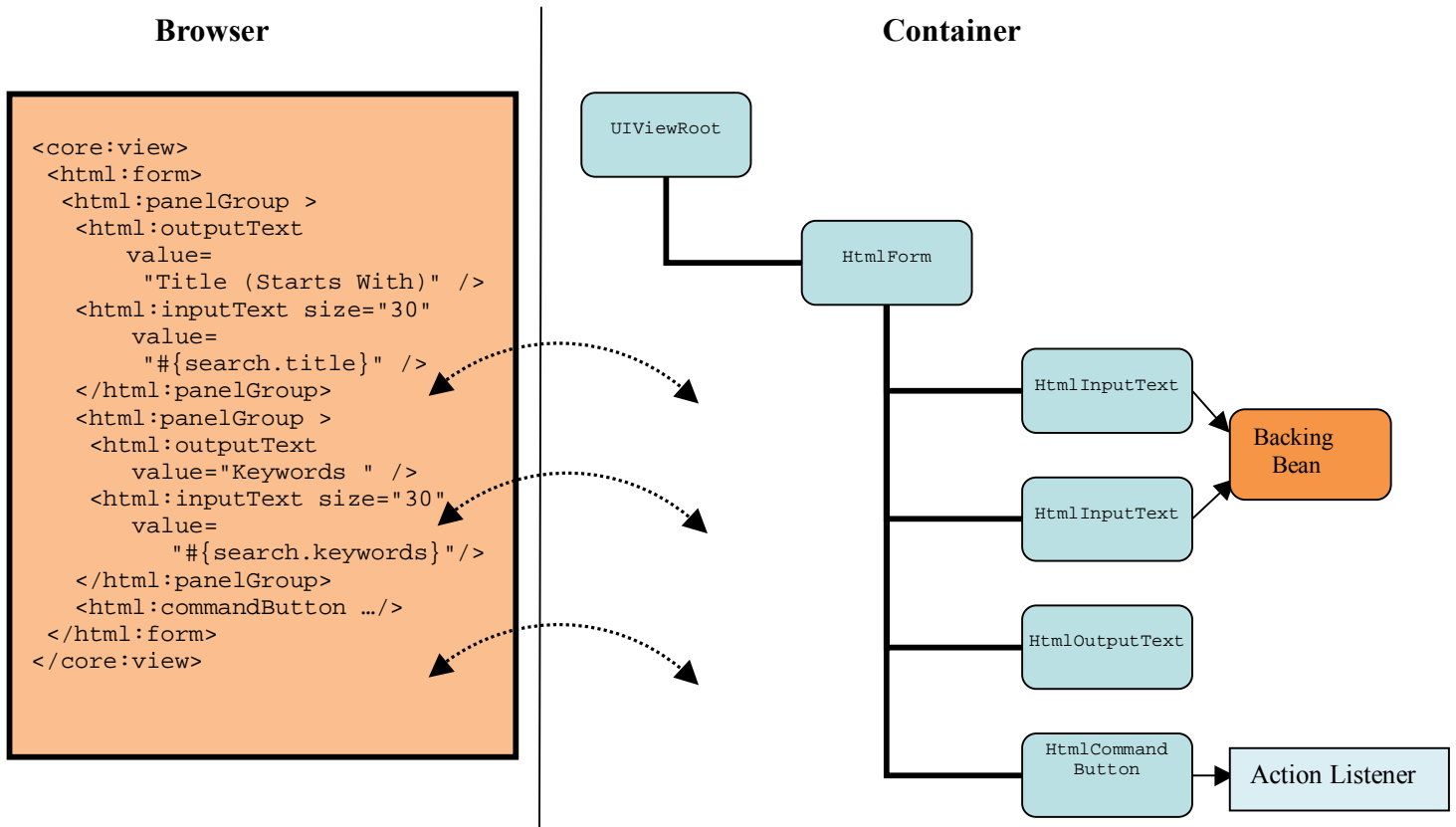


Figure 21 Rendering of a JSF view

Figure 21 shows the search JSF page of the CollabX application. This page has tags like `core:view` and `html:inputText`. Listing 6 shows a subset of tags that are contained in the page

```

<html:panelGroup >
  <html:outputText value="Title (Starts With)" />
  <html:inputText size="30" value="#{search.title}" />
</html:panelGroup>
<html:panelGroup >
  <html:outputText value="Keywords " />
  <html:inputText size="30" value="#{search.keywords}" />
</html:panelGroup>
  
```

Listing 6. JSF Tags

Each tag has an associated tag handler class. As the page is being rendered, the JSF tag handlers collaborate with each other to build a component tree. The component tree is a data structure that contains Java objects for all user interface elements. Each component has a renderer that produces HTML output. During the rendering phase, the renderer of each UI component requests a unique element ID and the current value of the binding expression. Once all the components of the view root have been rendered, the page is sent to the browser, and the browser displays as it would any other page. This marks the end of the rendering phase for a JSF page.

3.1.1 JSF Programming Model

In the JSF programming model, the backing bean is a mediator between the view and the model. The UI components are bound to the properties of the backing bean, and buttons to event handlers and action methods. An event change in the UI component is routed to the backing bean by the JSF framework.

The JavaServer Faces expression language (EL) syntax is used to bind UI component values and instances to backing bean properties or to reference backing bean methods from UI component tags. A JavaServer Faces expression can be a value-binding expression (for binding UI components or their values to external data sources) or a method-binding expression (for referencing backing bean methods).

An example of each binding expression is given below. A value-binding expression binds UI components to model tier data as shown in Listing 7.

```
<html:panelGroup >
  <html:outputText value="#{search.display.title}" />
  <html:inputText size="30" value="#{search.title}" />
</html:panelGroup>
```

Listing 7. Value-binding expression

On form submission, JSF framework will push back data into the model objects based on the value-binding expression.

A method binding expression binds UI components to “action” methods in the backing bean as shown in Listing 8. It is a convenient way of describing a method invocation that needs to be carried out when a component is activated.

```
<html:panelGrid styleClass="buttonPanel" columns="10">
  <html:commandButton value="#{msgs.search_button}"
    action="#{search.find}">
  </html:commandButton>
  <html:commandButton value="#{msgs.reset_button}"
    action="#{search.reset}" />
  <html:commandButton value="#{msgs.cancel_button}"
    action="#{search.cancel}" />
</html:panelGrid>
```

Listing 8. Method-binding expression

In the example shown in Listing 8, the command button component will call `search.find` and pass the returned string to the navigation handler. Binding a component's value to a bean property has the advantage that the backing bean has no dependencies on the JavaServer Faces API, allowing for greater separation of the presentation layer from the model layer and it can perform conversions on the data based on the type of the bean property value.

After developing the backing beans to be used in the JSF application, these beans needs to be configured in the `faces-config.xml` file, so that the JSF implementation can

construct a managed bean whenever it is first referenced in the application. The managed bean declarations for the CollabX application are given below:

```
<faces-config>
  <managed-bean>
    <managed-bean-name>search</managed-bean-name>
    <managed-bean-class>
      com.repository.web.beans.SearchMB
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>lob</managed-bean-name>
    <managed-bean-class>
      com.repository.web.beans.LearningObjectMB
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

A backing bean is defined with a managed-bean element inside the top-level faces-config element. The scope of the managed beans is `session`, which means that the managed bean is created once for each `session` and the lifetime of the bean spans across multiple requests. Once the bean's scope is `session`, the application developer has to factor in additional logic to clear out stale request data when handling a subsequent new request.

In the JSF framework, navigation among pages in a web application is determined by a set of rules. These rules are defined by the application developer in `faces-config.xml`. They determine the next page to be displayed after a button or hyperlink is clicked, as shown in Figure 22.

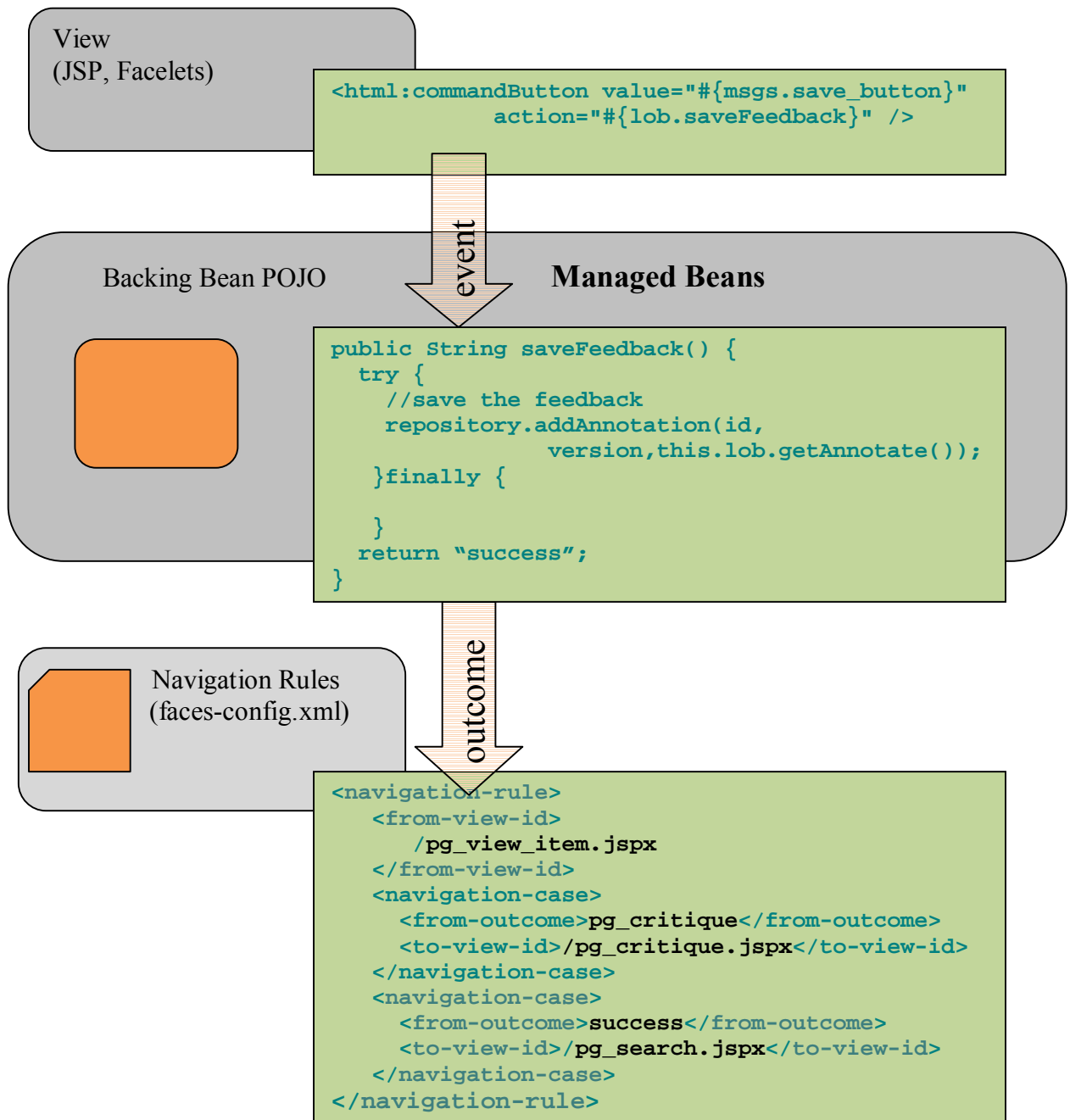


Figure 22 JSF programming model

In most web applications, navigation is not static. The navigation flow does not just depend on which button is clicked but also on the outcome of the triggered process. Each navigation rule in JSF can navigate from one source page to any of the target pages defined in the web application. Each navigation case within the navigation rule defines a target page and a logical outcome that triggers the navigation to the target page.

When a button or hyperlink is clicked, the component associated with it generates an action event. The framework handles this event and calls the `action` method referenced by the component that triggered the event. This action method is located in a backing bean and is provided by the application developer. It performs some processing and returns a logical outcome `String`, which describes the result of the processing. The framework then selects the page to display next by matching the result of the processing against the navigation rules in the application configuration resource file.

This is illustrated in Listing 9. When the command button is clicked, the method `lob.saveFeedback` is called by the JSF framework. Once the processing is done by the action method, the navigation rule defined for the view is now processed from the outcome of the action method `lob.saveFeedback`.

```
<html:commandButton value="{msgs.save_button}"
                    action="{lob.saveFeedback}" />

<navigation-rule>
  <from-view-id>/pg_view_item.jsp</from-view-id>
  <navigation-case>
    <from-outcome>pg_critique</from-outcome>
    <to-view-id>/pg_critique.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>success_feedback</from-outcome>
    <to-view-id>/pg_search.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Listing 9. Command button and navigation rule

The rule in Listing 9 states that when the button or hyperlink component on `pg_view_item.jsp` is activated, the application will navigate from the `pg_view_item.jsp` page to the `pg_search.jsp` page if the outcome referenced by the button component's tag is `success_feedback`.

3.2 Java Persistence Architecture

The Java Persistence Architecture is a key piece of the Java EE platform and provides an ease-of-use abstraction on top of JDBC so that the user code can be isolated from database, vendor peculiarities and optimizations. It can also be described as an ORM (object-to-relational mapping) engine, which means that a user can map Java objects to relational entities in a database through well-defined metadata annotations. In addition to object mappings, JPA also provides a query language that is very SQL-like but is tailored to work with Java objects rather than a relational schema. The primary features of the architecture are:

- POJOs
 - Entities are defined as “Plain Old Java classes” and not components.
- Support for enriched domain modeling such as inheritance, polymorphism etc.

- Standardized object/relational mapping
- Using annotations and/or XML
- Support for pluggable persistence providers
- JPQL-Java Persistence Query Language

3.2.1 Entities

Entities, in the Java Persistence specification, are plain old Java objects (POJOs). These entities are created using the `new` operator, in exactly the same manner in which an ordinary class is created. Unlike the past J2EE specifications standards, the entity objects don't have to implement any required interfaces. The container contract is abstracted out by means of annotations, special markers which are recognized by the container to manage the entity objects. Entities can also be serializable and they can be used as detached objects in the web tier of an application. This precludes the need for defining data transfer objects to transfer data between the web tier and the service layer. An example of an entity is given below:

```
@Entity
public class LearningObject implements Serializable {

    @Id
    private Long id;

    @Id
    private Long version
    ...
    //collection of feedbacks received for this lesson
    private Collection<LessonFeedback> feedbacks;

    public LearningObject() {
    }
    @OneToMany(cascade={CascadeType.ALL})
    public Collection<LessonFeedback> getFeedbacks() {
        return feedbacks;
    }
}
```

Each entity object has a persistence identity, which maps to a primary key in the database. A primary key can correspond to a simple type. This is represented by the following annotations:

- `@Id` – This will be a single field/property in an entity class
- `@GeneratedValue` – The persistent identity can be generated automatically

Relationships can exist between entities. A relationship is modeled between entities to reflect the relational model of the various tables in the database schema. There are four types of cardinality: one-to-one, one-to-many, many-to-one, and many-to-many. In addition to this, each relationship can be either unidirectional or bidirectional. The

relationship metadata is again expressed in terms of annotations. Here is an example of a one-to-many bi-directional relationship:

```
@Entity
public class LessonPropertyValue implements Serializable {
    @Id
    private Long id;
    protected LessonProperty property;
    @ManyToOne
    public LessonProperty getProperty() {return property;}
    public void setProperty(LessonProperty property) {
        this.property = property;
    }
}

@Entity
public class LessonProperty implements Serializable {
    @Id
    private Long id;
    private List<LessonPropertyValue> values =
        new ArrayList<LessonPropertyValue>();
    @OneToMany(mappedBy="property")
    public List<LessonPropertyValue> getPropertyValues() {
        return values;
    }
    public void addPropertyValue(LessonPropertyValue value) {
        getPropertyValues().add(value);
        value.setProperty(this);
    }
}
```

The relationship that is modeled between entities `LessonProperty` and `LessonPropertyValue` conforms to the One-to-Many Bidirectional relationship. A one-to-many bidirectional relationship occurs when one entity bean maintains a collection-based relationship property with another entity bean, and each entity bean referenced in the collection maintains a single reference back to its aggregating bean. From the above example the entity `LessonProperty` is the aggregation bean and maintains a collection based relationship property with the entity `LessonPropertyValue`. This is modeled by the relationship metadata annotation `@OneToMany` and `@ManyToOne`.

3.2.2 Object Relational (OR) Mapping

Entity beans represent data in the database and these beans have to model the relationship that exists between the data that they represent. The process of mapping persistent object state to database and mapping of relationships to other entities is known as Object Relational Mapping (ORM). The metadata for the mapping can be expressed as annotations or XML. These annotations can be categorized as given below:

- Logical Mapping — Used to define the object model (e.g., `@OneToMany`, `@Id`, `@Transient`)

- Physical Mapping — Used to map the object's persistent state to database tables and columns (e.g., @Table, @Column)

The ORM metadata in annotations can model a simple basic relational mapping or model a complex relationship between entities.

3.2.2.1 Basic Mapping

A simple mapping provides direct mapping of fields/properties of an entity to database columns. It maps any of the common simple Java types such as primitives, Date, Serializable, byte[] etc to its corresponding SQL types. It defaults to the type deemed most appropriate if no mapping annotation is present. It is used in conjunction with the @Column annotation type. An example of a basic mapping is given below –

```
@Entity
public class LessonFeedback implements Serializable {

    @Id
    private Long id;
    private String comments;

    @Column(name="comments", nullable=true)
    public String getComments() {
        return comments;
    }
}
```

It uses the following annotations-

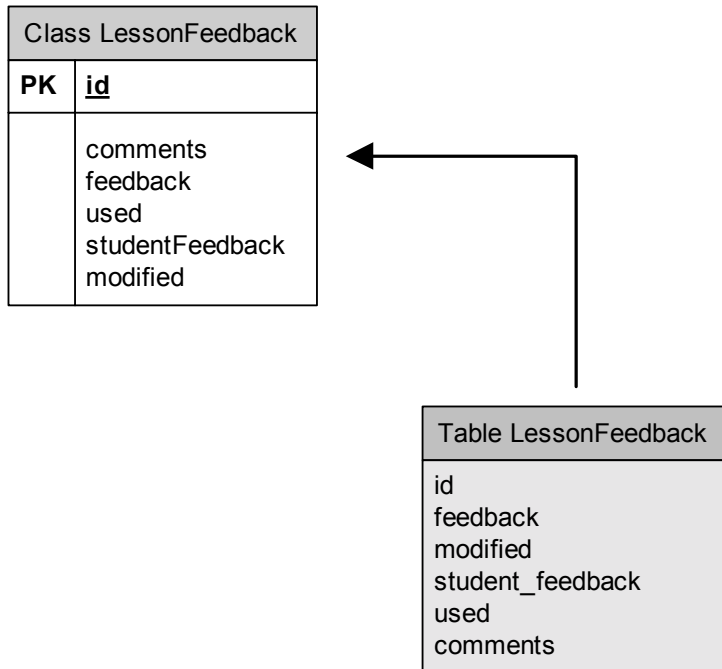
@Column

This annotation maps the entity beans property to a database column name.

@Id

Every entity bean that maps to a relational table must have an identity. The primary key is the identity of a given entity bean and it must be unique. This annotation identifies one or more properties of the entity bean that make up the primary key of the mapped table.

This class physical mapping to a table is given below:

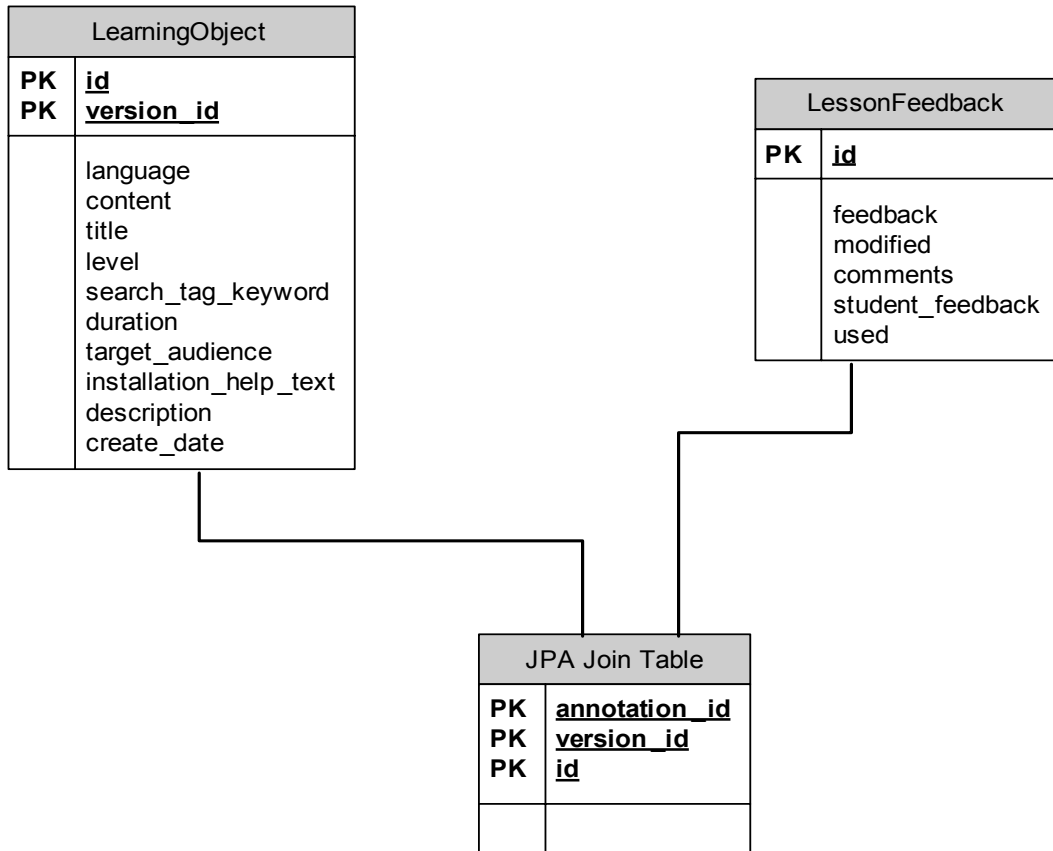


3.2.2.2 Relationship Mapping

The ORM metadata in annotations support relationship with single entity or multiplicity. The single entity relationship is supported by `@ManyToOne` and `@OneToOne` annotations, while relationship with multiplicity are supported by `@OneToMany` and `@ManyToMany` annotations. Every bidirectional relationship has an owning and inverse side. The owning side specifies the physical mapping with these annotations

- `@JoinColumn` – It specifies the foreign key column. If this column is omitted then the primary key of the target entity is taken as the foreign key.
- `@JoinTable` – Joins between entities can be done through an association table. It decouples the physical relationship mappings from the entity tables.

A one-to-many unidirectional relationship between entities `LearningObject` and `LessonFeedback` is given below:



The JPA provider generates the schema from the entity bean definitions. The relationship between the two entities is modeled as given below:

```

@Entity
@IdClass(ObjectKey.class)
public class LearningObject implements Serializable {
    @Id
    private Long id;
    //collection of feedbacks received for this lesson
    private Collection<LessonFeedback> feedbacks;
    @OneToMany(cascade={CascadeType.ALL})
    public Collection<LessonFeedback> getFeedback() {
        return feedbacks;
    }
    public void setFeedback(
        Collection<LessonFeedback> feedbacks) {
        this.feedbacks = feedbacks;
    }
}
@Entity
public class LessonFeedback implements Serializable {
    //this is the unqie key of the feedback for the
    //learning resource
    @Id
    private Long id;
    ...
}

```


The one-to-many relationship is declared using the `@OneToMany` annotation. The multiplicity in the relationship is represented by defining a relationship property that can point to many entity beans and annotating it with the `@OneToMany`. The data type is typically a `java.util.Collection` object that contains a homogenous group of entity object references.

TopLink, which is the default JPA provider in the glassfish container, implements a one-to-many relationship with a join table mapping. This is an association table that maintains two columns of foreign keys pointing to both the `LearningObject` and `LessonFeedback` records. The join table shown in the above figure is an association table, which is generated by the TopLink to implement the `@OneToMany` relationship annotation.

3.2.3 Query

The Java Persistence query language (JPQL) is used to define searches against persistent entities independent of the mechanism used to store those entities.. JPQL always references the properties and relationships of the entity beans rather than the underlying tables and columns these objects are mapped to. When a JPQL query is executed, the entity manager uses the information provided through annotations and translates it to a native SQL query. This generated query is then executed through a JDBC driver in the targeted database. As such, JPQL is "portable", and not constrained to any particular data store. It is an extension of the Enterprise JavaBeans query language, EJB QL, adding operations such as bulk deletes and updates, join operations, aggregates, projections, and sub queries.

Java Persistence has a query interface to execute JPQL and native SQL queries. This API gives methods for paging result set and passing java parameters to the query.

`EntityManager` is the factory for creating queries. Listing 4 shows how to retrieve a collection of `LearningObjects` from the CollabX repository. The factory method `createQuery()` of `EntityManager` is invoked to create a dynamic query to retrieve a collection of `LearningObject` instances. Listing 10 shows an example of a dynamic query creation in CollabX and retrieving the result set:

```
// Retrieve a collection
Query q = em.createQuery(
    "SELECT l from LearningObject as l
    WHERE l.title LIKE :title
    AND l.keywords LIKE :keywords
    ORDER BY l.creationTime");

q.setParameter("title", "Exercise 7.1");
q.setParameter("keywords", "factoring integer");
Collection departments = q.getResultList();
```

Listing 10. Query for retrieving a collection of `LearningObjects`

Here is an example below that shows how to retrieve a single instance of LearningObject.

```
// Retrieve a single instance
Query q = em.createQuery(
    "SELECT l from LearningObject as l
    WHERE l.id = :lessoned");
q.setParameter("lessoned", 1234);
//get the result
LearningObject lo = q.getSingleResult();
```

Listing 11. Query for retrieving a single instance of LearningObject

3.3 Web-Tier Programming Model – JSF, EJB 3.0 and JPA

The Java EE platform supports greatly simplified programming model for enterprise web applications. Java EE components are configured and deployed using annotations instead of bulky XML descriptors which needed to be packaged as part of the deployment archive. Annotations can be used to write specification and behavior of the components directly in the code. In the Java EE platform, dependency injection can be used for transparent creation and lookup of the resources to automatically insert references to other required components or resources using annotations.

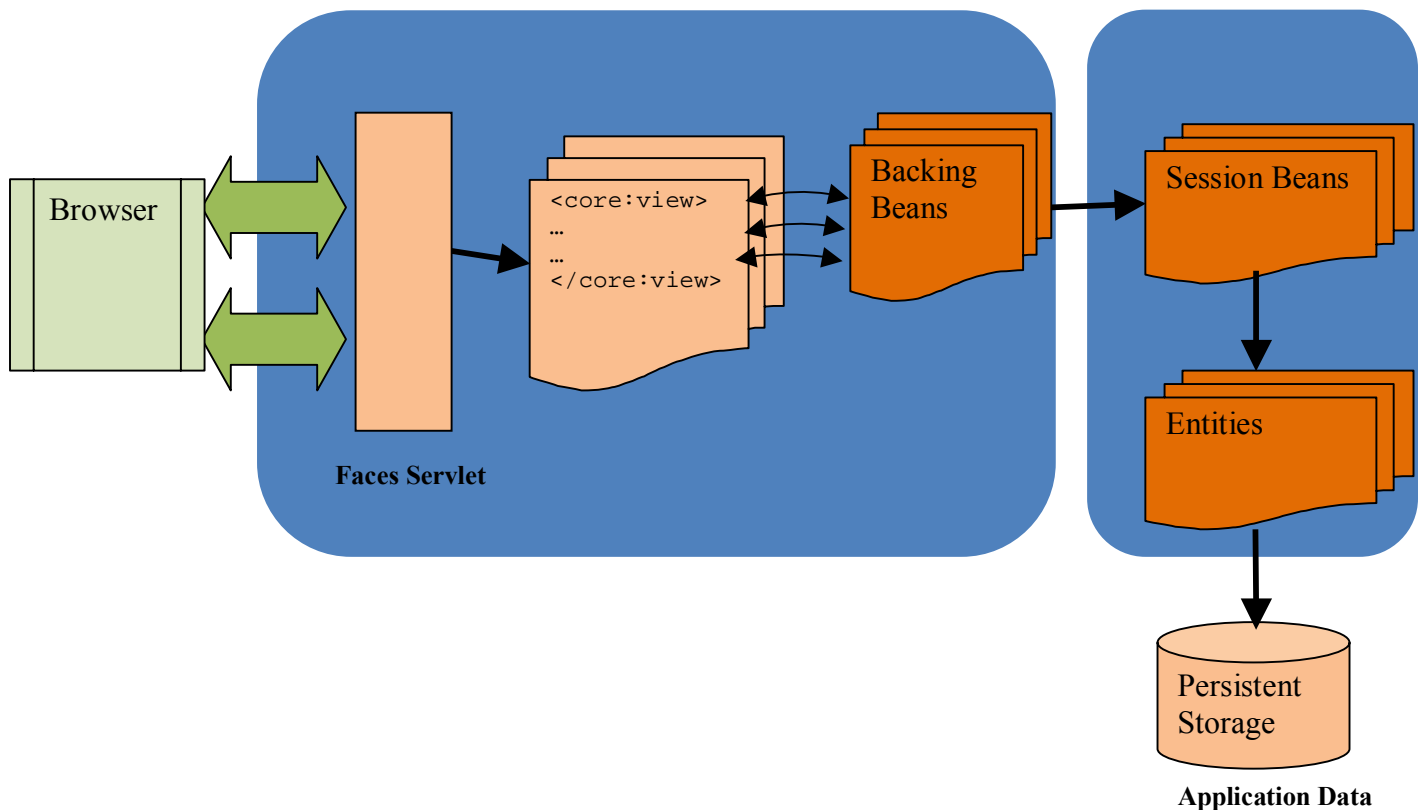


Figure 23 Web-Tier programming model

A logical diagram of the web programming model is given in Figure 23. A typical web request is first intercepted at Java Server Faces layer. The request data is then marshaled to the JSF page's backing bean through value binding expressions which wires the UI components to properties of the backing bean.

A JSF managed bean as shown in Listing 12 can make calls to EJBs by declaring a field of that EJB's type and annotating that field as a reference to a @EJB. The container will process the @EJB annotation during the deployment of the web application. It will look for an EJB reference that implements the business interface type as its local or remote interface. If more than one EJB uses the same business interface, it throws a deployment exception.

```
public class LearningObjectMB {
    @EJB
    private RepositoryServiceRemote repository;

    public void dispatch() throws Exception{

        //save the uploaded content in the database
        Properties metadata = new Properties();
        metadata.setProperty("title", this.getCurrentLO().getTitle());
        metadata.setProperty("language", getCurrentLO().getLanguage());
        metadata.setProperty("description",
            getCurrentLO().getDescription());
        metadata.setProperty("help_text",
            getCurrentLO().getHelp_text());

        metadata.setProperty("keywords", getCurrentLO().getKeyword());
        metadata.setProperty("audience", getCurrentLO().getAudience());
        metadata.setProperty("duration",
            Integer.toString(getCurrentLO().getDuration()));
        metadata.setProperty("level", getCurrentLO().getLevel());
        metadata.setProperty("context", "");

        //get the properties
        int suffixIndex = 1;
        for ( LOBProperty p : this.getCurrentLO().getProperties() ) {
            //check for null
            metadata.setProperty("property_name."+suffixIndex,
                Util.checkNotNull(p.getName(), "undefined"));

            metadata.setProperty("property_value."+suffixIndex,
                Util.checkNotNull(p.getValue(), "undefined"));
            suffixIndex++;
        }
        //call the method
        repository.createXObject(metadata, this.getCurrentUpload());
    }
}
```

Listing 12. Managed Bean with injected EJB reference

The injected remote EJB reference is a stateless session bean. Stateless session beans as the name suggests doesn't hold the client conversation state. As soon as it is finished servicing a method invocation it can be swapped to service request from another client. Since it doesn't maintain any client state, there is no overhead in swapping stateless session beans across multiple client invocations. Stateless session beans are also a prime candidate for implementation a façade pattern. This façade interface is shown in Listing 13. It centralizes requests to the domain and aggregates multiple calls to the `EntityManager`.

```
@Stateless
public class RepositoryServiceBean implements RepositoryServiceRemote {

    @PersistenceContext(unitName = "collabx")
    private EntityManager em;

    public long createXObject(Properties metadata,byte[] content);
    public PageData findLearningObject(long oid,long version)
    public PageData findLearningObject(long oid,long version,
                                       boolean load);
    public List<PageData> search(Properties metadata)
    public void addAnnotation(long oid, long version, Feedback critique);
    public void addMetadata(long oid,long version,List<String> keywords,
                           Map<String,List<Object>> metadata);
    public void addMetadata(long oid,long version,
                           Map<String,List<Object>> metadata);
}
```

Listing 13. Stateless session bean facade interface

An `EntityManager` is injected into an EJB by using the `@javax.persistence.PersistenceContext` annotation as shown in Listing 14. This annotation allows the EJB container to inject an `EntityManager` reference into the EJB. This is the most preferred method as the EJB container has full control over the life cycle of the underlying persistence context of the `EntityManager`. The application developer doesn't need to write any additional life cycle code to handle the cleanup of the `EntityManager` instance.

```
@Stateless
public class RepositoryServiceBean implements RepositoryServiceRemote {
    @PersistenceContext(unitName = "collabx")
    private EntityManager em;
}
```

Listing 14. EntityManager injection in a EJB

`EntityManager` is the primary interface to persistence runtime and manages the state of the entities that are attached to it as shown in Listing 15.

```
LearningObject exercise = new LearningObject();
//set the version
exercise.setVersion(Version.generateVersion());
```

```

exercise.setTitle(metadata.getProperty("title",""));
exercise.setContext(metadata.getProperty("context", ""));
exercise.setCreationTime(Calendar.getInstance().getTime());
em.persist(exercise);

//find an entity
ObjectKey key = new ObjectKey(oid,version);
LearningObject lob = em.find(LearningObject.class,key);

```

Listing 15. Usage of EntityManager API

When an entity is attached to an `EntityManager`, the manager tracks state changes to the entity and synchronizes those changes to the database whenever the entity manager decides to flush its state.

The entity manager manages the attached entities through a persistence context. The persistence context represents a collection of entities, whose persistence state is tracked by the entity manager for changes and updates. These changes are then flushed to the database. In a given persistence context, for each persistence identity there is a unique persistence instance. Once a persistence context is closed, all managed entities within that context become detached and are no longer managed by the entity manager. Once an object is detached from the persistence context, any state changes to this object instance will be not flushed to the database.

3.3.1 Entity Management

The state of the entity as it flows through the various tiers of an enterprise web application is shown in Figure 24.

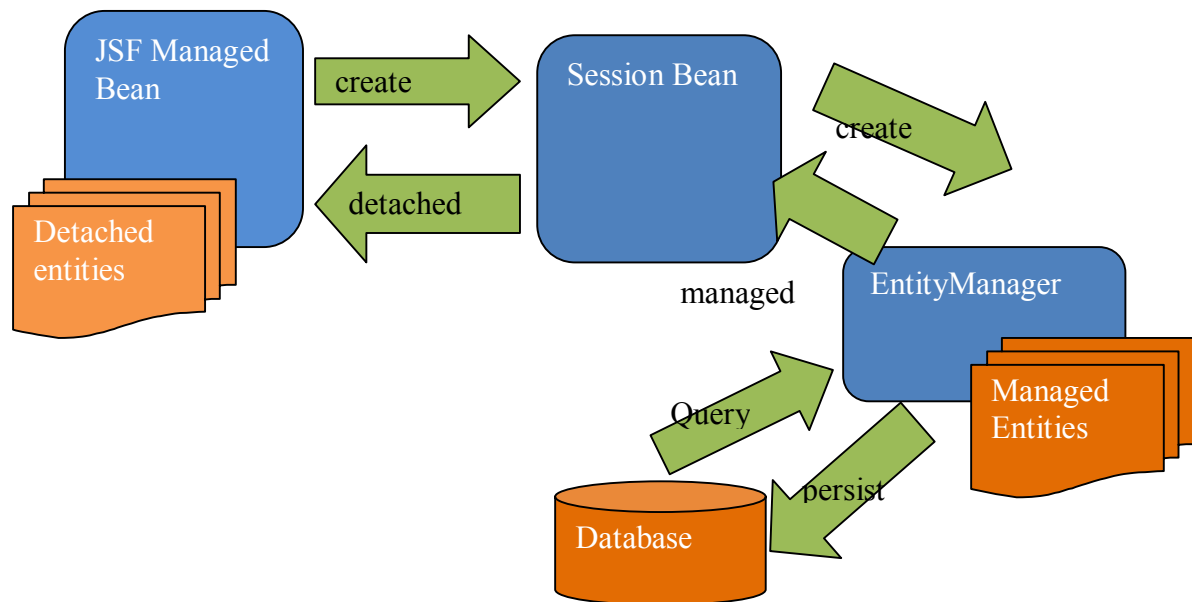


Figure 24 Entity Detach and merge

An entity can be created with the `new()` operator like any other ordinary POJO class. In this state entity is not yet managed or persistent. Code Listing 16 shows the relevant code for creating an instance of `LearningObject` entity.

```
LearningObject exercise = new LearningObject();
//set the version
exercise.setVersion(Version.generateVersion());
exercise.setTitle(metadata.getProperty("title",""));
exercise.setContext(metadata.getProperty("context", ""));
exercise.setCreationTime(Calendar.getInstance().getTime());
exercise.setDescription(metadata.getProperty("description",""));
//get the duration
String duration = metadata.getProperty("duration", "0");
exercise.setDuration(Integer.parseInt(duration));
```

Listing 16. Creating a New Entity

An entity is persisted in the database by invoking `EntityManager.persist()` method as shown in Listing 17. Once an entity is persisted in the database, it is *managed* by the `EntityManager` till the end of the `PersistenceContext`. If the entity has any relationships with other entities, these entities may also be created in the database if the appropriate cascade policies are mentioned in the relationship annotation.

```
@Entity
@IdClass(ObjectKey.class)
@SequenceGenerator(name="VERSIONKEY_SEQUENCE",sequenceName="VERSIONKEY_
SEQUENCE")
public class LearningObject implements Serializable {

    @Id
    private Long id;
    //collection of properties
    @OneToMany(cascade={CascadeType.ALL})
    private Collection<LessonProperty> properties;
}

@Stateless
public class RepositoryServiceBean implements RepositoryServiceRemote {

    @PersistenceContext(unitName = "collabx")
    private EntityManager em;

    public long createXObject(Properties metadata,byte[] content){

        try {
            LearningObject exercise = new LearningObject();
            //set the version
            exercise.setVersion(Version.generateVersion());
            //check if the HashMap contains
            //pairs of property_name.x and property_value.x
            for(String propertyName : metadata.stringPropertyNames() ) {
                if ( propertyName.startsWith("property_name")) {
                    //get the actual property name and values
                    String[] property = Util.getProperty(propertyName,
                        metadata);
```

```

        properties.add(new LessonProperty
                        (property[0],property[1],true));
    }
}
if ( properties.size() > 0 ) {
    exercise.setMetadata(properties);
}
//set the content
exercise.setXcontent(content);
em.persist(exercise);
return exercise.getId();
} catch (RuntimeException ex) {
}
}
}

```

Listing 17. Persisting Entities

Entities become detached and unmanaged when the persistence context or the transaction scope ends as shown in Figure 24. However, in the detached state, these detached entities can be serialized and sent over the wire to different tiers of the application. Each of these tiers can make changes to the state of the entity. The Java Persistence framework does allow to merge state changes made to a detached entity back into the persistence repository using the entity manager's `merge()` method as shown in Listing 18.

```

public void addAnnotation(long oid, long version,
                        Feedback critique) {
    long id = -1;
    try {
        //form the composite key
        ObjectKey key = new ObjectKey(oid,version);
        LearningObject lob = em.find(LearningObject.class, key);

        //add the feedback
        MetadataAnnotation m = new MetadataAnnotation();
        m.setComments(critique.getComments());
        m.setFeedback(critique.getFeedback());
        m.setStudentFeedback(critique.getStudentFeedback());
        m.setModified(critique.isModified());
        m.setUsed(critique.isUsed());

        lob.getAnnotation().add(m);

        em.merge(lob);
    }catch(RuntimeException rte) {

        Logger.getLogger(getClass().getPackage().getName())
                .log(Level.SEVERE, "", rte);

        throw rte;
    }
    return ;
}
}

```

Listing 18. Merging Entities

The most important thing to remember here is that detached entities which had been serialized to the web tier of an enterprise web application needs to be merged into the current persistence context to be managed by the EntityManager. And, if the entity is involved in bi-directional relationships, relationships need to be updated both on the owning side and dependent side.

4. Solution Implementation

4.1 CollabX Repository Overview

The solution proposes a central repository to which lessons created by different instructors can be uploaded. Each lesson uploaded to the repository is tagged with metadata data elements, which are a subset of definitions derived from the IEEE LOM specifications [1].

It addresses a set of use-cases, which are often valid for users of a shared repository, specifically

- Search (Advanced Search)
An end user would like to search for a particular lesson contained in the repository. Each lesson in the repository is tagged with keywords, which is used to identify and retrieve the lesson.
- Download
An end user can download a lesson from the repository. The format supports Labrat and a generic zipped version of the resources that make up the lesson.
- Upload
An end user can upload a zipped file containing resources that make up the lesson. The resources contained in the zipped archive could be any valid file arranged in any hierarchical layout.
- Fork (a variant of an existing lesson)
An end user can fork a variant from an already existing lesson in the repository. During the upload process, a user can indicate that the lesson is a variant of an existing lesson. This relationship will be maintained in the repository and a user can search for lessons which are derived from a particular type of lesson.
- Critique
An end user can search for a lesson and leave feedback about the quality of the lesson. This information will again be tagged with that particular lesson with an `Annotation` data element, which will be used as a search parameter while searching for a lesson in the repository.
- Categorize
An end user can categorize a lesson by changing properties associated with the lesson. These properties may include Text Book Name, Chapter Name, Section Name, level of difficulty, context etc.

This shared repository implementation introduces collaborative features like "Fork", "Critique" and "Categorize", which is lacking in other similar repositories. This shared repository henceforward will be named as CollabX.

4.2 Integration with External Systems

CollabX is built on the premise of a shareable and collaboration platform across global communities. These individual communities might be using a set of grading systems that evaluates a student's submission based on pre-defined set of configuration attributes that is not inter-operable with each other. CollabX tries to bring a common ground in defining a consistent vocabulary in developing courseware for Computer Science students. This common vocabulary takes the shape of a set of metadata definitions that describes a Computer Science assignment in CollabX. This very set of definitions is then used to export an assignment to a target system such as Web-CAT or Labrat by using a plug-in architecture.

CollabX supports plugins for Web-CAT, Labrat and a generic zipped format. The plugin interface is given below –

```
package com.repository.plugin;

/**
 * This is an interface which defines the contract for exporting a
 * exercise in the repository into an external format
 *
 * @author Somyajit Jena
 */
public interface Plugin {

    public void export(byte[] lesson) throws Exception;
    public void export(byte[] lesson,FileEventListener listener)
                                   throws Exception;

    /**
     * A simple getter to return the identifier of the exported
     * lesson from the repository
     * @return - String - an identifier
     */
    public String getExportId() ;
}
```

This interface exports a lesson to the target system by taking the contents (byte []) of the persisted lesson in the repository and transforming the same to a deployable unit of the target system. It uses the metadata definitions that were tagged with the lesson to build a deployment archive with associated property definitions that matches the target system's requirements.

4.2.1 Web-CAT Interoperability

Course developers or instructors can develop courses in Web-CAT and publish them to their students. Web-CAT automatically grades submitted assignments against an

instructor provided solution. Typically, the instructor solution is executed through a *test harness*, which is a collection of test cases provided by the instructor. These test cases compare the output of a student’s submitted assignment with the expected output of the solution provided by the instructor.

While creating an assignment in Web-CAT, an instructor can provide reference test cases in the form of a JUnit test case. These instructors provided test cases are executed by a `JavaTddPlugin` when a student submits a solution for the assignment. The configuration of this plug-in is done as part of the overall set-up of the assignment in Web-CAT as shown in Fig 25.

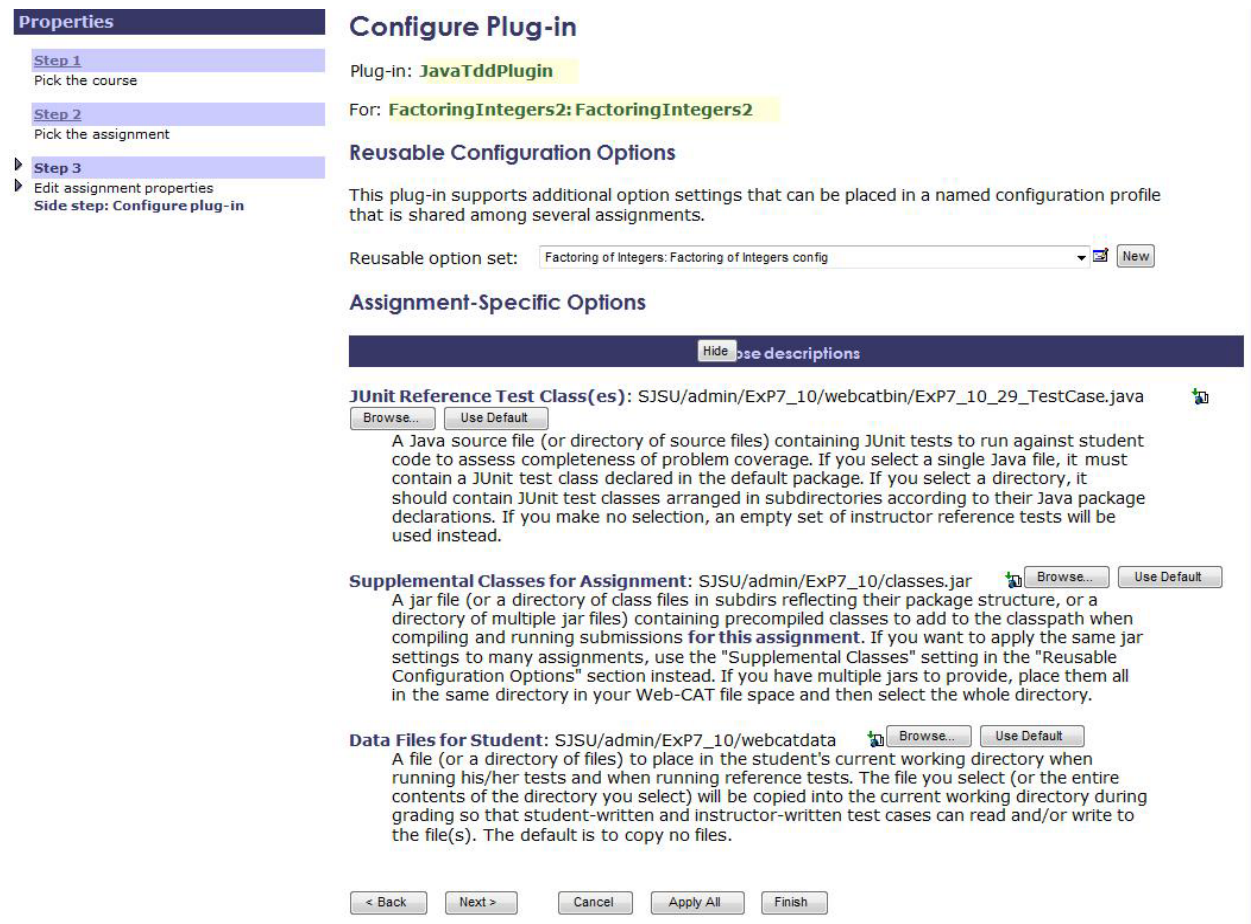


Figure 25 JavaTDDPlugin Configuration

An assignment created in Web-CAT should have adequate artifacts to configure the parameters of `JavaTDDPlugin` as shown in Table 4.

Configuration Parameter	Parameter Value	Description
JUnit Reference	SJSU/admin/Exp7_10/web	A Java source file (or directory of

Test Class(es)	catbin/Exp7_10_29_TestCase.java	source files) containing JUnit tests to run against student code to assess completeness of problem coverage. If a single Java file is selected, it must contain a JUnit test class declared in the default package. If a directory is selected, it should contain JUnit test classes arranged in subdirectories according to their Java package declarations.
Supplemental Classes for Assignment	SJSU/admin/Exp7_10/classes.jar	A jar file (or a directory of class files in subdirs reflecting their package structure, or a directory of multiple jar files) containing precompiled classes to add to the classpath when compiling and running submissions for this assignment.
Data Files for Student	SJSU/admin/Exp7_10/webcatdata	A file (or a directory of files) to place in the student's current working directory when running his/her tests and when running reference tests. The file selected (or the entire contents of the selected directory) will be copied into the current working directory during grading so that student-written and instructor-written test cases can read and/or write to the file(s). The default is to copy no files.

Table 4 . JavaTDDPlugin Configuration parameters

Each parameter value shown in Table 4 is a relative path to the resource from the root folder of the assignment. When a submitted assignment is being graded, the `JavaTDDPlugin` reads these configuration values to fetch the various resources from the respective file paths.

Since Web-CAT has support for executing JUnit test cases, the interoperability between CollabX and Web-CAT was done at a JUnit test case level. This interoperability support is only applicable for assignments whose content layout confirms to the Labrat format as show in Section 1.1 of Chapter 1. Each exported assignment from CollabX repository is tagged with a dynamically generated JUnit test case.

The Web-CAT plug-in of CollabX will generate two test methods – `testMainClassWithInputs` and `testWithCapturedOutput`.

4.2.1.1 Method testMainClassWithInputs

During export of an assignment to a Web-CAT grading system, the plug-in reads the Labrat metadata file `check.properties` and contents of directories to generate the code for this test function. It uses the following rules:

- It reads the value of the property `test.test-inputs` to determine if the main class of the submitted assignment needs to be loaded dynamically by the java classloader. A value "true" indicates that the main class of the assignment needs to be tested with instructor's reference test data. The code generated will read the property `mainclass`, dynamically load and invoke the java class.
- It scans the student and grader directories for instructor reference test data looking for files with names having patterns like `test*.in` or `test*.out`. The code is generated to make the program read data from the input test files and compare the program's output with instructor provided expected output data.

```
public void testMainClassWithInputs() throws Exception {
    //the input files are assumed to be
    //test.in and the output is test.out
    Properties properties = null;
    PrintWriterWithHistory saveOutputStream = null;
    try {
        //read the check.properties file
        properties = this.readProperties("check.properties");
        //check with check.properties if the Main class
        //of the assignment needs to be tested with supplied inputs
        boolean testMainClass =
            booleanProperty("test.test-inputs",properties,true);
        if ( testMainClass == false ) {
            //force a success evaluation
            assertTrue(testMainClass == false);
            return;
        }
        //set the input stream of the test case
        InputStream in = getClass().getClassLoader()
            .getResourceAsStream("test1.in");
        System.setIn(in);
        saveOutputStream = this.systemOut();
        //call the main class
        String mainClass = stringProperty("mainclass",properties,null);
        if ( mainClass == null ) {
            assertTrue(mainClass == null);
            return;
        }
        //invoke
        this.invokeMainClass(mainClass);
    }catch(Exception e) {
        System.out.println("Exception: Message " + e.getMessage());
        throw e;
    }
}
```

```

//evaluate the results
//get the CompareTask
CompareTask ct = new CompareTask();
try {
    //set the parameters
    //this configuration is for knowing the comparison
    //mode to test the output comparison
    String outputCompareTokenType = stringProperty("test.token",
                                                    properties,"line");

    ct.setToken(outputCompareTokenType);
    //set the source
    String result = saveOutputStream.getHistory();
    StringReader source1 = new StringReader(result);
    //source2 which is pre-canned
    Reader source2 = new InputStreamReader(getClass.getClassLoader()
                                          .getResourceAsStream("test1.out"));

    ct.setSource1(source1);
    ct.setSource2(source2);
    //evaluate the output
    assertTrue(ct.compareOutput() == true);

}finally {
}
}
}

```

4.2.1.2 Method testMainClassWithInputs

This test function is a wrapper around Horstmann style tester classes, which is included in any assignment that confirms to the Labrat format. Horstmann style tester classes are Java classes having a naming pattern like `*(T|t)ester\\.java$`. If there are any instructor provided Horstmann style tester classes, it will call that tester program with the appropriate inputs and compare the program output with the instructor provided expected output. The comparison task reads the `test.token` and `test.tolerance` metadata properties from `check.properties` file to determine the type of comparison to make and the level of tolerance to use before flagging the result as an error:

- `test.token`
This property indicates the type of comparison to use while comparing the program's output with the instructor's reference expected output. The valid values are `line`, `word`, `number` or `regular expression`.
- `test.tolerance`
This property holds the value for the level of tolerance to use while comparing numbers.

The captured output of a Horstmann style tester class has the following format as given below:

```

true
Expected: true
2
Expected: 2

```

```

true
Expected: true
2
Expected: 2
true
Expected: true
3
Expected: 3
true
Expected: true
5
Expected: 5
false
Expected: false

```

The code is generated to compare each output line with the line having the "Expected" token.

```

public void testWithCapturedOutput() throws Exception {

    //check if there are any tester classes in any of the
    //directories, we need to attach the System.out and check the
    //expected pattern

    PrintStreamWithHistory saveOutputStream =this.systemOut();
    //call the any * tester class present in the student or grader
    //directory
    FactorGeneratorTester.main(null);

    //all program done. Parse the output
    //get the print history
    String result = saveOutputStream.getHistory();

    //create the compare task object
    CompareTask ct = new CompareTask();
    //get this from check.properties
    ct.setToken("number");
    //get the tokens
    StringReader reader = new StringReader(result);
    List<String> alltokens = ct.getTokens(reader);

    //nothing to compare
    assertNotSame(alltokens.size(), 0);
    ListIterator<String> iter = alltokens.listIterator();

    boolean equals = true;
    while ( iter.hasNext()) {
        //sourcel
        String token1 = iter.next();
        String token2 = iter.next();
        equals &= ct.compare(token1, token2);
    }
    assertTrue(equals == true);
}

```

The file naming convention of this generated test case is given below –

<title of the assignment>_<random number>_TestCase.java

Figure 26 shows the layout of the exported assignment when it is deployed in Web-CAT. The Web-CAT plug-in adds two new directories: webcatbin and webcatdata.

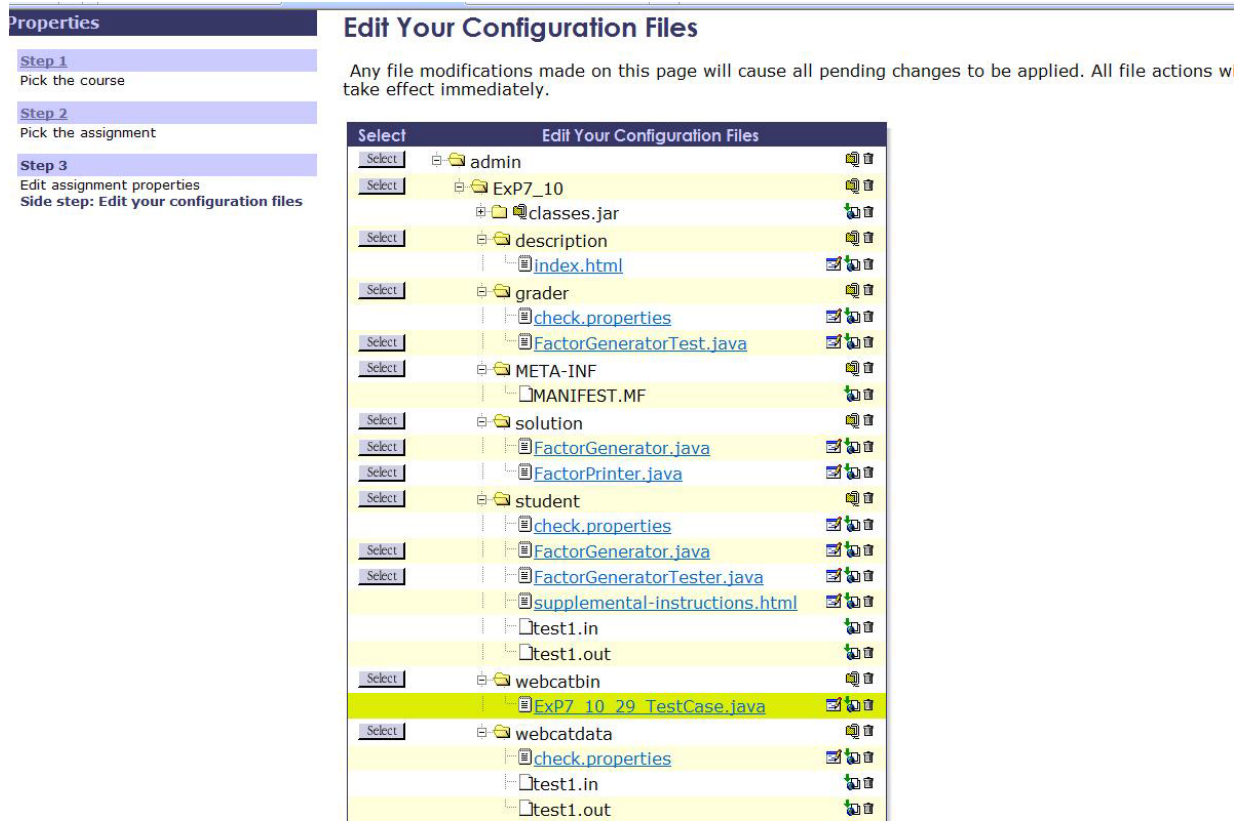


Figure 26 Layout of the exported assignment in Web-CAT

These new directories contains the following artifacts -

- webcatbin
This directory contains the generated JUnit test case for the exported assignment. This makes it easier to integrate with Web-CAT, since once can easily identify the folder and configure the test case to be a "Instructor reference test case"
- webcatdata
This directory contains all the test data files whose names match the patterns test*.in and test*.out.

As shown in Figure 26, file Exp7_10_29_TestCase.java has been configured to be the instructor reference test case for assignment titled "Exp7_10". The folder "webcatdata" has been configured as the test data file that needs to be copied over when Web-CAT is executing the instructor reference test in a temporary staging area. When a student submits a solution for an assignment, Web-CAT executes this reference test and the result is made available to the student.

4.2.2 Labrat Interoperability

An assignment confirming to the Labrat directory layout format is saved in the file system as shown in Figure 27. This is in addition to being persisted in the repository.

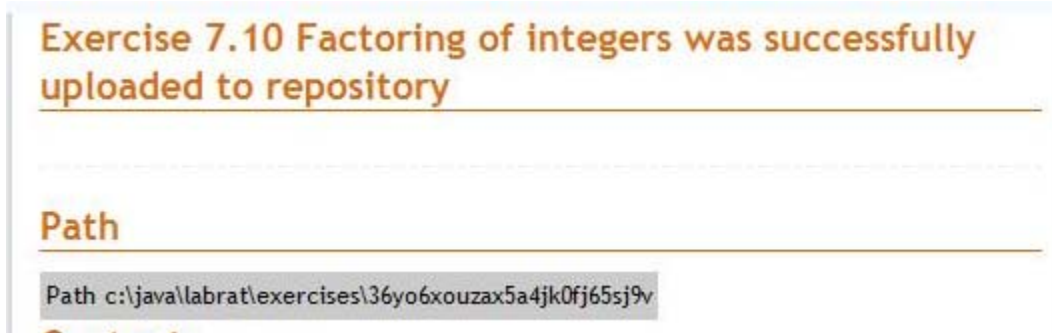


Figure 27 Labrat Interoperability

4.2.3 Labrat Feature Extension

Prior to these feature inclusions, an instructor had to prepare the lesson in the Labrat format and copy to the root of the repository. The web-interface was enhanced so that a zipped lesson in Labrat format would be uploaded to the repository. It also supported generation of the metadata of the uploaded lesson, if check.properties file was not included in the zipped file.

4.3 CollabX Solution Use Cases

4.3.1 Use Case – Upload

The table below enumerates a set of activities for uploading a lesson to the repository-

User action	System response
User enters the title and selects the zipped archive to upload	System uploads the zipped archive in to a temporary staging area. It reads the contents and tags metadata information to the contents. It then displays the properties page.

User enters the values for properties Text Book Name, Chapter, Section, Language, Installation Help Text, description, level of difficulty, Duration, Target Audience and any other additional properties that the use might think deem correct.	System associates these properties with the lesson and saves it to the repository. It then displays the page to associate search keyword with the lesson.
User enters the search keywords.	System saves the keywords for the lesson. It then displays to the user a result screen which has the contents of the zipped archive, properties and tagged search keywords. This marks the end of the upload process.

Table 5 . Use-Case of uploading a lesson to the repository

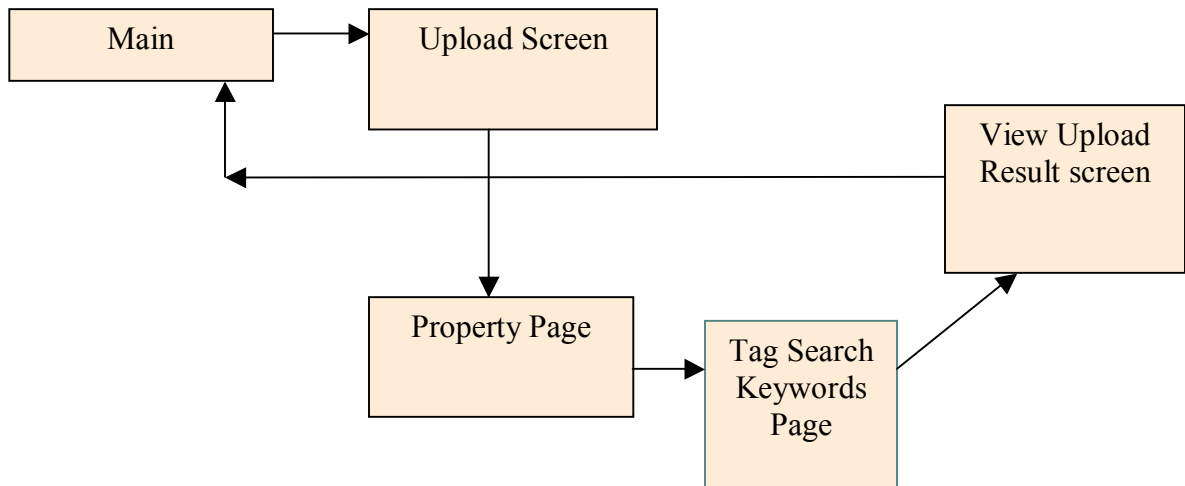


Figure 28 Flow Diagram of Uploading a Lesson to Repository

4.3.1.1 User Interface

This section has a set screen shots to illustrate the use case given above. Figure 29 shows the page for uploading a zipped archive into the repository. The user has to select the zipped archive in the file system. During the upload process, the application scans the zipped content and tag metadata information with the contents wherever appropriate. It then stages the uploaded zipped archive in a temporary storage area to gather additional information about the uploaded content before saving it into the repository. This additional information is gathered from the user through a wizard like interface as shown in Figure 30.

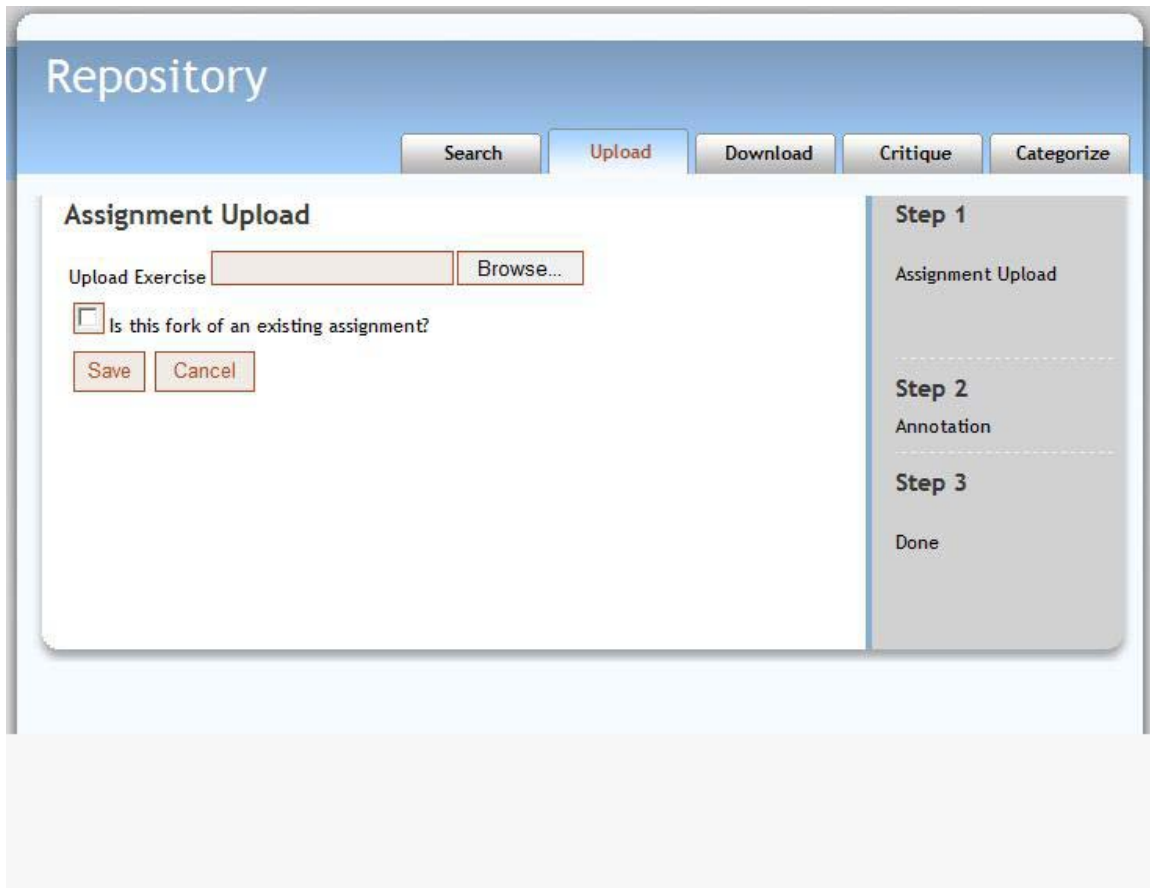


Figure 29 Screen shot – Repository Upload

Figure 30 shows the Properties page where the user has to add some contextual information to the uploaded lesson. This helps in describing the lesson and its usefulness in various environments of study as well as the targeted audience.

Search

Upload

Download

Critique

Categorize

Properties

Title

Language

Description

```
Write a program that asks the user
for an integer and then prints out
all its factors in increasing
order. For example, when the user
enters 150, the program should
print
```

Installation Help Text

```
Use a class FactorGenerator with a
constructor FactorGenerator(int
numberToFactor) and methods
nextFactor and hasMoreFactors .
```

Installation Help Text

Level

Slam Dunk

Easy

Somewhat Easy

Moderately Difficult

Difficult

Duration

Target Audience

Freshmen

Step 1

Assignment Upload

Step 2

Annotation

Step 3

Done

Easy
 Somewhat Easy
 Moderately Difficult
 Difficult

Duration

Target Audience

Freshmen
 Junior
 Sophomore
 Senior

Other Properties

Property Name

Property Value

Allow others to search for this assignment?

Figure 30 Screen shot – Properties Page

In the above screen shot, a user could also enter additional properties that may further describe the uploaded lesson for any special circumstances. They could also tag search keywords for a lesson that could be used in identifying the lesson in the repository as shown in Figure 31.

Repository

Keyword Tag

Step 1
 Assignment Upload

Step 2
 Annotation

Step 3
 Done

Figure 31 Screen shot – Repository upload

Figure 32 shows the result of a successful upload for a zipped archived lesson into the repository. At this point, the contents of the zipped archive along with the metadata information have been persisted in the repository.

The screenshot displays a web interface for a repository. At the top, there is a navigation bar with buttons for 'Search', 'Upload', 'Download', 'Critique', and 'Categorize'. The main content area is titled 'Repository' and features a prominent orange heading: 'Exercise 7.10 Factoring of integers was successfully uploaded to repository'. Below this heading, the interface is organized into several sections: 'Path', 'Contents', 'Properties', and 'Additional Properties'. The 'Path' section shows a file path: 'c:\java\labrat\exercises\36yo6xouzax5a4jk0fj65sj9v'. The 'Contents' section lists various files, including 'META-INF/MANIFEST.MF', 'description/index.html', 'grader/check.properties', and several Java source files like 'FactorPrinter.java' and 'FactorGenerator.java'. The 'Properties' section contains metadata such as 'Title: Exercise 7.10 Factoring of integers', 'Language: en_US', and a detailed 'Description' of the programming task. The 'Additional Properties' section is partially visible at the bottom. On the right side of the interface, a vertical sidebar shows a progress indicator with three steps: 'Step 1: Assignment Upload', 'Step 2: Annotation', and 'Step 3: Done', indicating the upload process is complete.

Figure 32 View uploaded assignment

4.3.2 Use Case – Search

This use case assists users of CollabX in searching for an assignment that has been persisted in the repository. The user interface exposes a set of search filters for targeted search of an assignment. Table 5 shows the various search filters that are enabled in the user interface –

Search Filter	Description
Title	Search based on title of the assignment. An assignment whose title starts with the search term is a match.
Keywords	Search is based on the “keywords” that was tagged with the assignment. This search filter finds keywords of an assignment that contains this term.
Property Name	Search is based on the user-defined properties that were associated with this assignment. This search filter will search for property names of an assignment which contains this term
Property Value	A value entered for this filter will search for property values that contains this term.

Table 6 . Search Filters

None of the search filters mentioned in Table 6 is mandatory. The default value is a wild-card search (%) for each of the above search terms. Listing 19 shows an example of a search query generated from the values entered for the search filters.

```
SELECT l from LearningObject as l JOIN l.metadata m
WHERE l.title LIKE 'Exercise 7.10 Factoring%'
AND l.keywords LIKE '%integer factorial%'
OR ( m.propertyName = 'Text Book' AND
    EXISTS (
        SELECT val from m.propertyValues val
        WHERE val.propertyValue LIKE '%Big Java volume 2%'
    )
)
ORDER BY l.creationTime
```

Listing 19. Generated EJB QL from Search Filters

The table below enumerates a set of activities to search for an assignment in CollabX.

User action	System response
User inputs the data based on which the search will be executed. The search filters are title, keywords and property names and values. These are the properties which are tagged with the assignment	System displays the search results matching the filter criteria.

when it was uploaded into the shared repository.	
User selects one of the assignments that is displayed in the search results page.	System displays the details of the assignment, which includes its contents, metadata definitions, keywords tagged for search and user defined properties.

Table 7 . Use Case - Search

4.3.2.1 User Interface

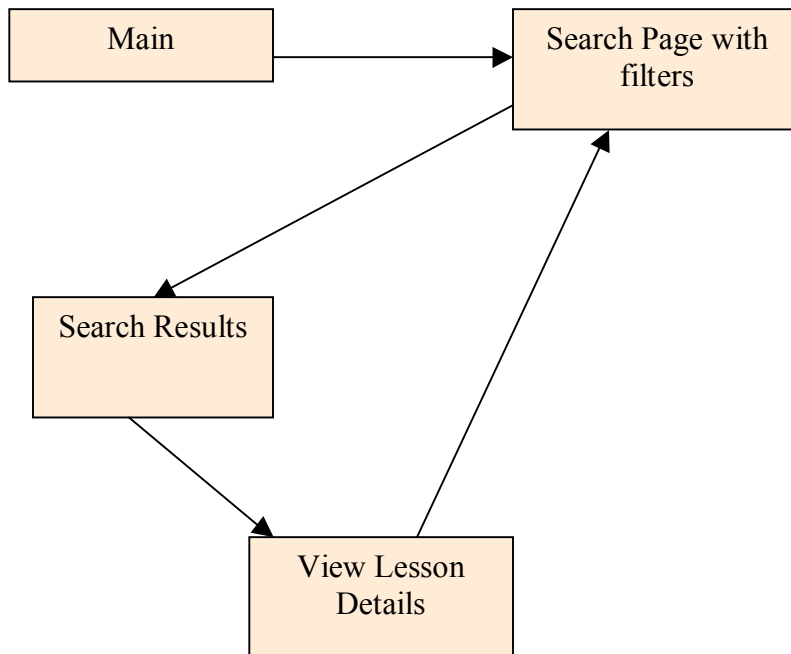


Figure 33 UI Flow Diagram of Search

This section has a set screen shots to illustrate the use case given above. Figure 34 shows the user interface for searching an assignment in the repository. It has the search filters as discussed in Table 6. If an assignment has multiple user-defined properties, these can be made part of the search filter by clicking on the “More Properties...” button. On clicking this button, another set of property name and value will be prompted to the user.

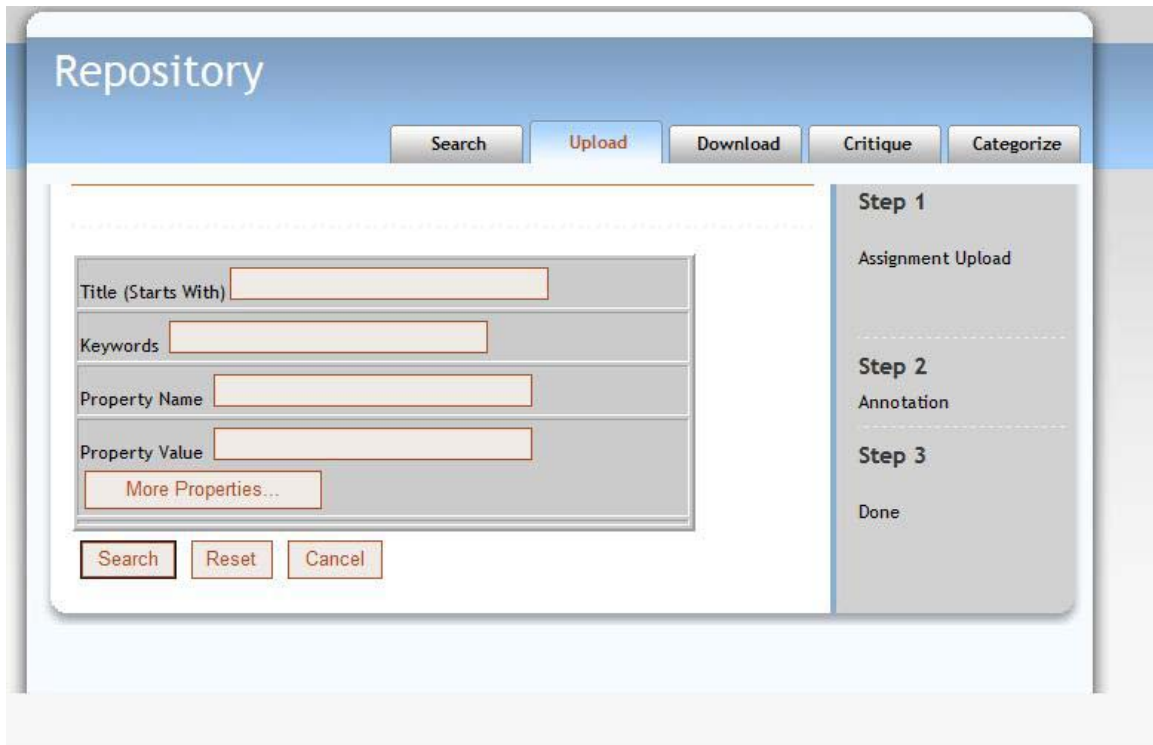


Figure 34 Search User Interface

Figure 35 shows the results for searching by the title of an assignment which starts with “Exercise 7.10”. Each search result displays the title and a brief description of the assignment.



Figure 35 Search Results

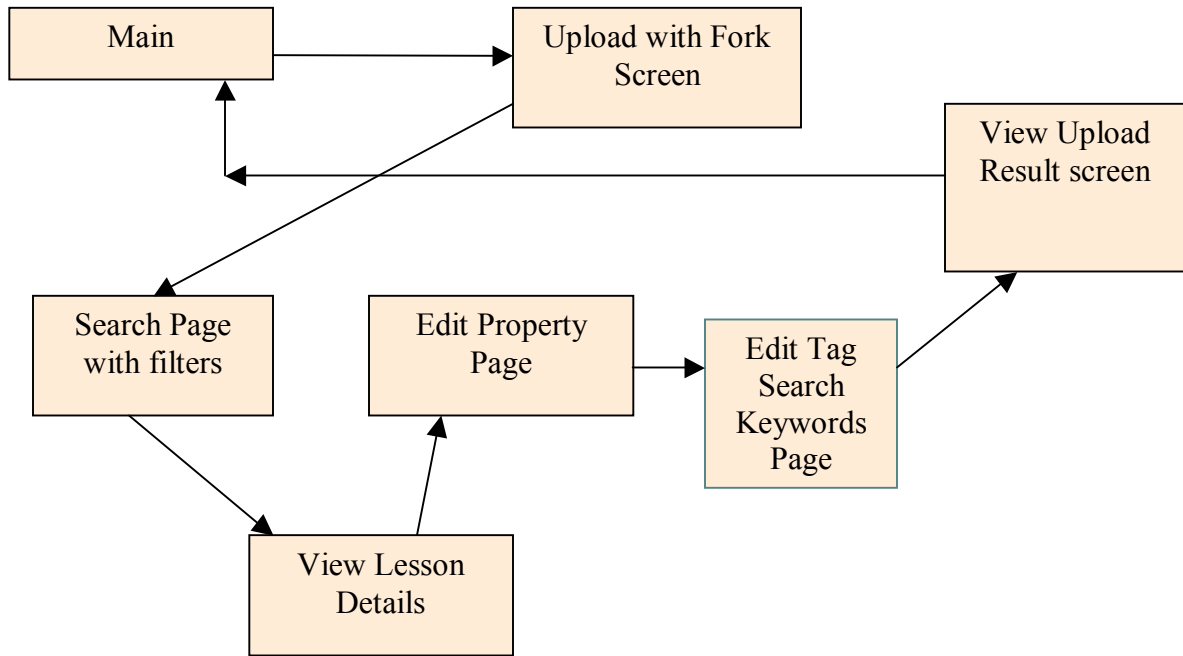
4.3.3 Use Case – Fork

This use case assists instructors in uploading a variant of a programming assignment that they may have downloaded and later modified it to suite their needs. A programming assignment fork helps the instructor in filling in the gaps in metadata definition of the altered programming assignment. They can choose to retain all of the original metadata definition and add new ones to reflect the contents of the altered assignment. Prior to forking an assignment, an instructor has to search for the original assignment and mark it as the parent from which the current assignment was created. CollabX will then load the metadata definition of the original assignment and associate that with the current assignment that is being uploaded to the shared repository.

The table below enumerates a set of activities to upload a forked assignment into CollabX.

User action	System response
User enters the title and selects the zipped archive to upload. The checkbox is marked to denote that the assignment that is being uploaded is a forked assignment from an existing one in CollabX.	System uploads the zipped archive in to a temporary staging area. It parses the contents and then redirects to a search page.
User inputs the data based on which the search will be executed. The search filters are title, keywords and property names and values. These are the properties which are tagged with the assignment whe it was uploaded into the shared repository.	System displays the search results matching the filter criteria.
User selects one of the assignments that is displayed in the search results page.	System displays the details of the assignment, which includes its contents, metadata definitions, keywords tagged for search and user defined properties.
User marks this assignment as the parent of the forked assignment that is being uploaded to the repository.	System shows an editable properties page which the user can edit metadata of the assignment.
User edits the values for properties Text Book Name, Chapter, Section, Language, Installation Help Text, description, level of difficulty, Duration, Target Audience. The end user can also delete or add additional properties that can aptly describe the modified assignment.	System associates these properties with the lesson and saves it to the repository. It then displays the page to associate search keyword with the lesson.
User can edit the tagged keywords for the assignment. These keywords are used as one of search filters to find an assignment.	System saves the keywords for the lesson. It then displays to the user a result screen which has the contents of the zipped archive, properties and tagged search keywords. This marks the end of the upload process of a forked assignment.

4.3.3.1 User Interface



This section has a set screen shots to illustrate the use case given above. Figure 36 shows the page for uploading a zipped archive into the repository.

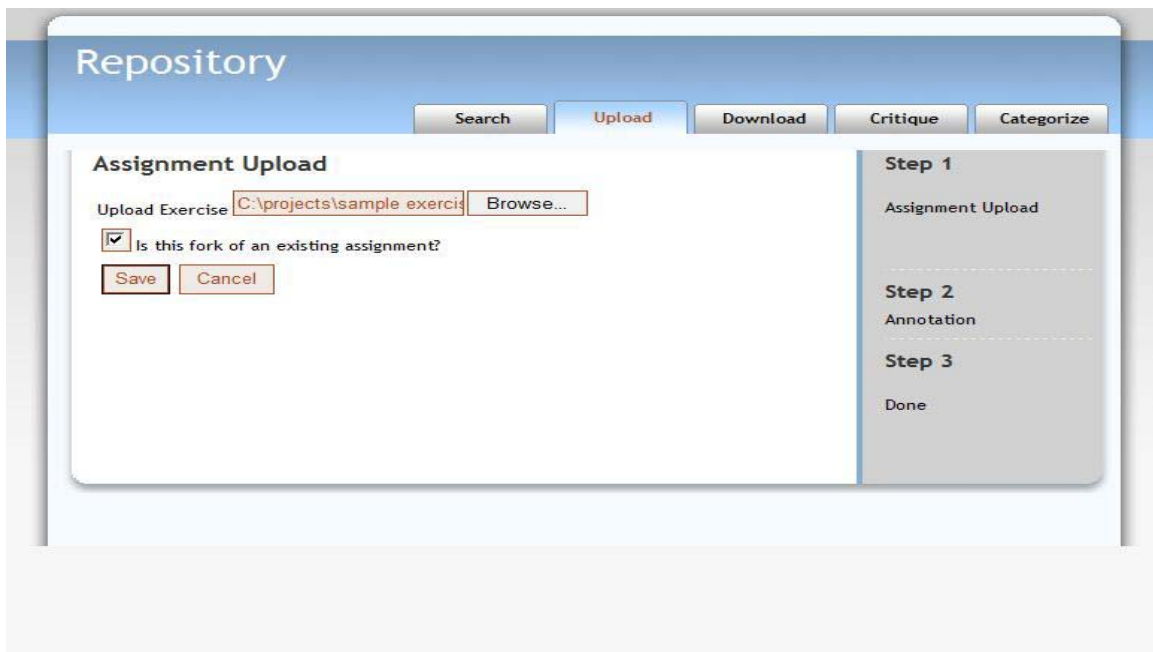


Figure 36 Screen shot – Repository Upload

The user has to locate the zipped archive in the file system and mark the checkbox denoting that the assignment being uploaded is a fork of an existing assignment in the repository. During the upload process, the application scans the zipped content and tag metadata information with the contents wherever appropriate. It then stages the uploaded zipped archive in a temporary storage area to gather additional information about the uploaded content; specifically the user has to provide the details of the original assignment by searching and selecting it. This additional information is gathered from the user through a search interface as shown in Figure 37.

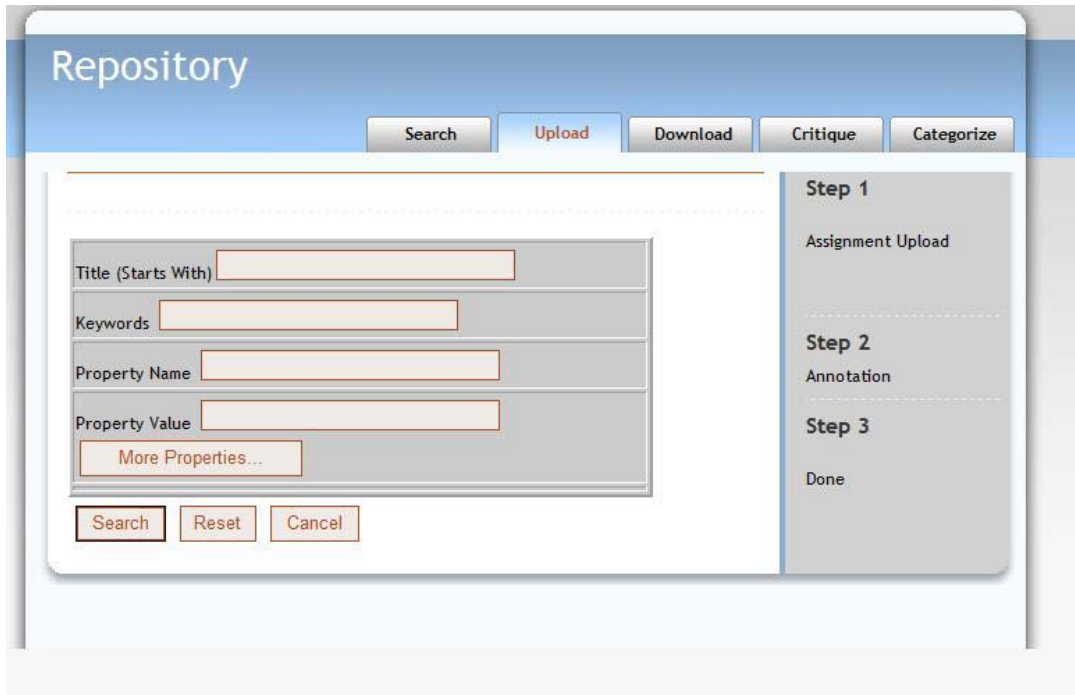


Figure 37 CollabX search interface

The user has to input the search filter criteria to narrow down the search for the original assignment. The search results are shown in Figure 38.

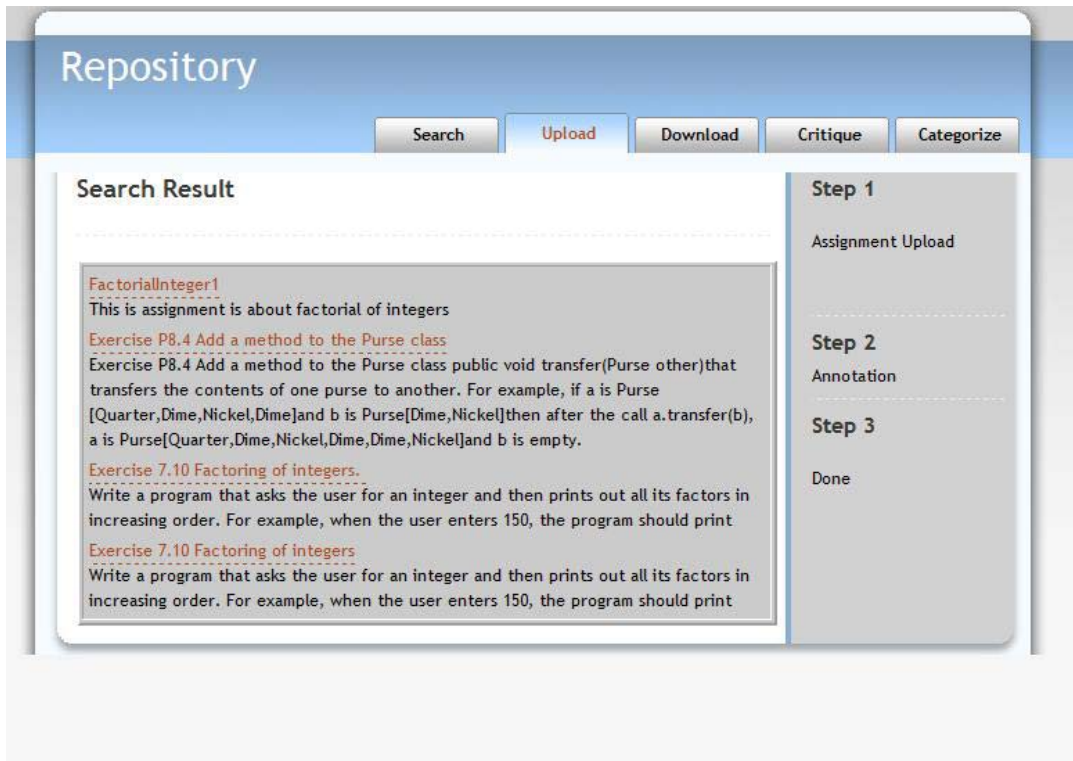


Figure 38 Search results page

Each search result shows the title of the assignment and a brief description about it. The original assignment from which the forked assignment has been derived can be selected by clicking on the title of the assignment.

Figure 39 given below shows the editable properties page of the selected assignment where the user can add, delete or edit the metadata definition.

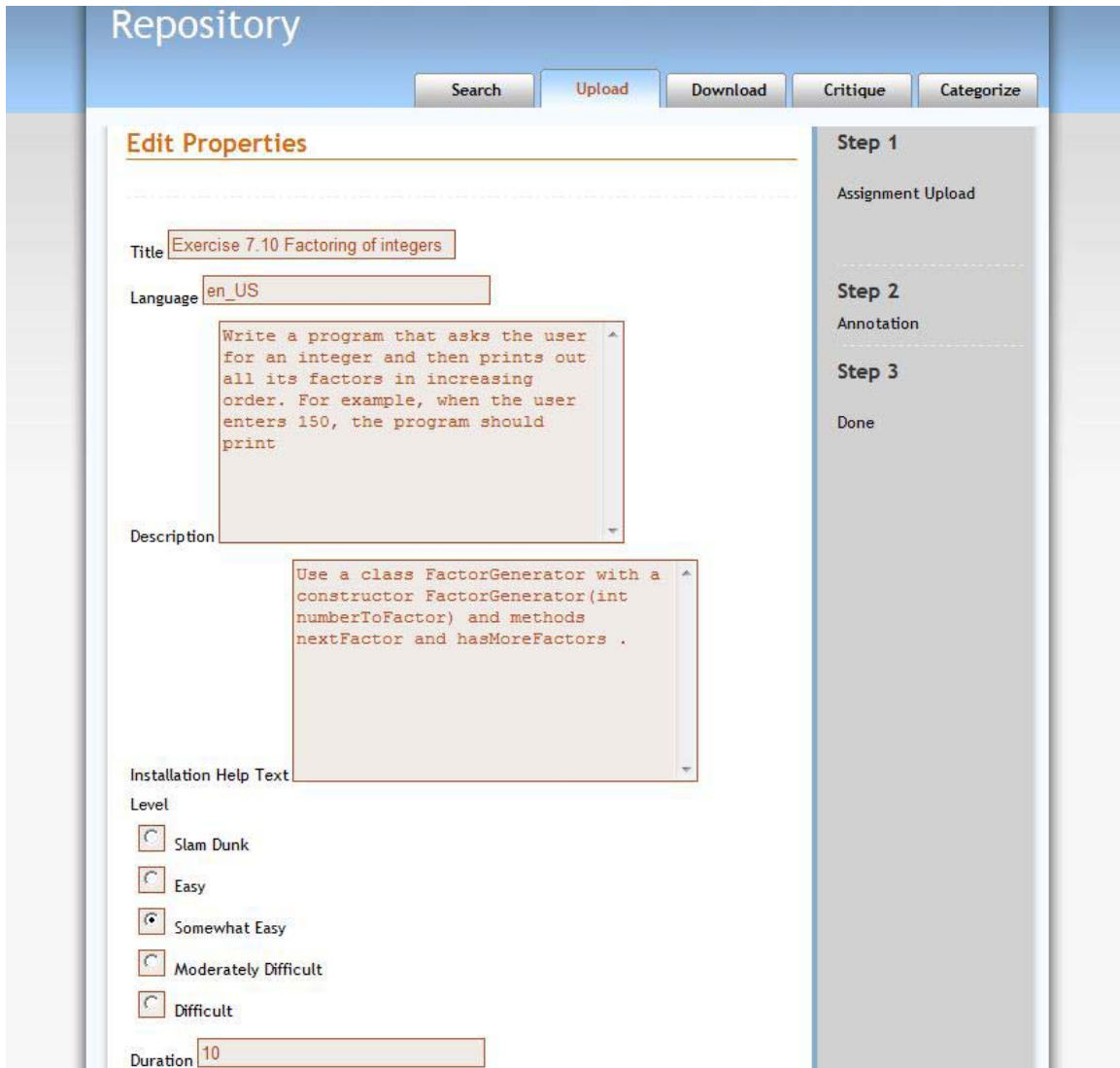


Figure 39 Edit assignment

In the above screen shot, a user could also enter additional properties or edit the existing ones that may further help in describing the altered assignment. They could also tag search keywords for a lesson that could be used in identifying the lesson in the repository as shown in Figure 40 below.

A user could edit the existing keywords list that may help in searching for the lesson in the shared repository. The keywords are delimited by a semi-colon (;).

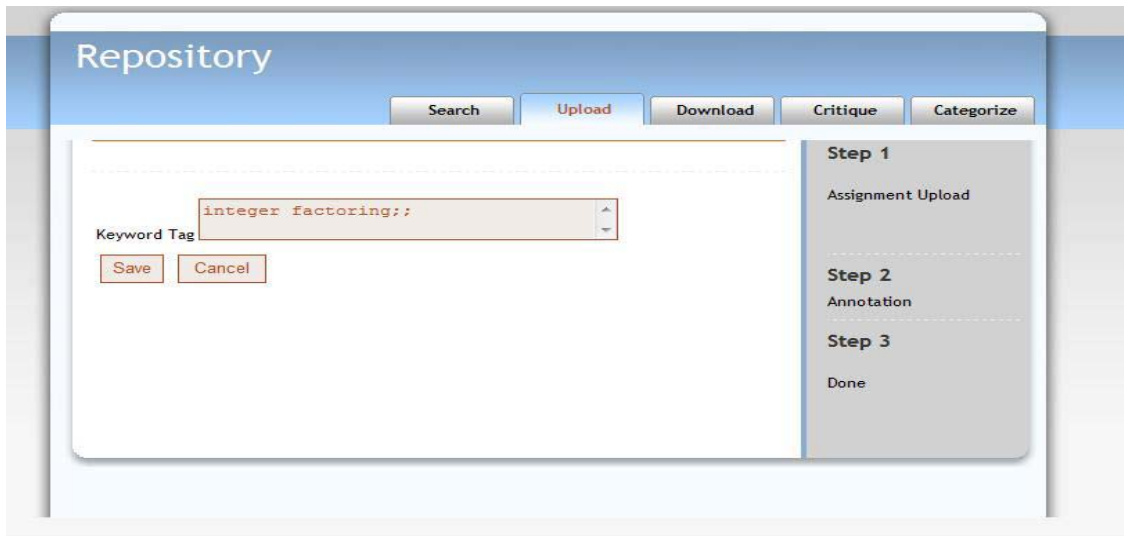


Figure 40 Edit Search Keyword Page

Figure 41 shows the result of a successful upload for a zipped archived lesson into the repository. At this point, the contents of the zipped archive along with the metadata information have been persisted in the repository.

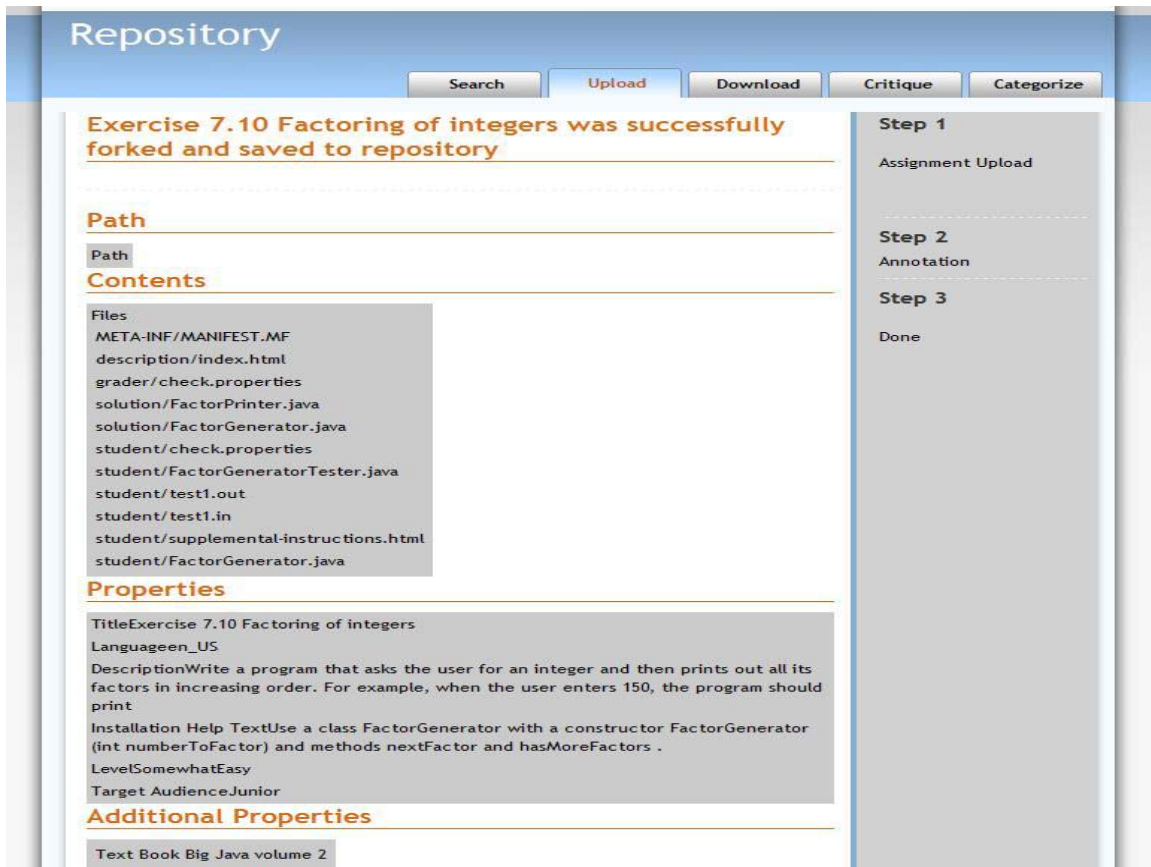


Figure 41 A Successful Save Page

4.3.4 Use Case – Download

This use case assists instructors, course developers and researchers in downloading an assignment from the CollabX repository. A user has to search for an assignment before being able to download it. The download options that are available are shown in Figure 42.

1. Generic zipped format

The selected assignment will be downloaded as a zip file. The entire contents of the same will be included in the archive. This option is useful in making offline changes to it and then uploading it as a fork of the original assignment.

2. Web-CAT format

This option will export the selected assignment as a deployable archive in Web-CAT.

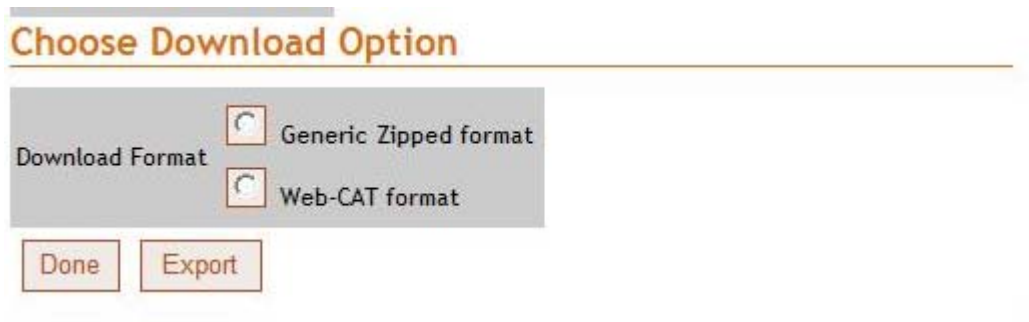


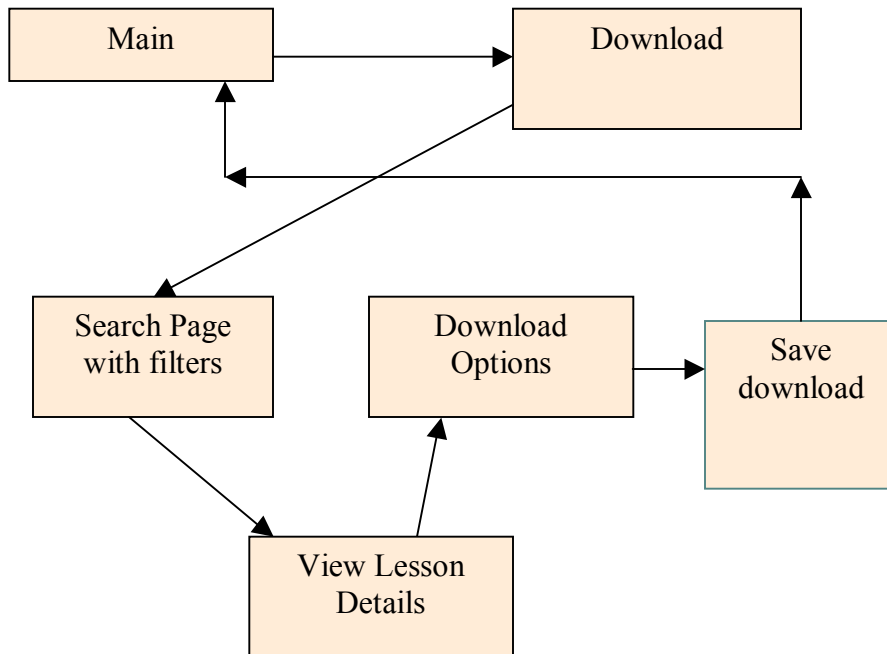
Figure 42 Download Options

Table 8 enumerates a set of activities to download an assignment from the repository.

User action	System response
User clicks on the Download tab.	System redirects to the search page.
User inputs the data based on which the search will be executed. .	System displays the search results matching the filter criteria.
User selects one of the assignments that are displayed in the search results page.	System displays the details of the assignment, which includes its contents, metadata definitions, keywords tagged for search and user defined properties. A download button is displayed in this page.
User clicks on the download button	System displays a page showing the details of the assignment selected and the download options as shown in Figure 46.
User selects one of the formats.	System downloads the assignment in that format.

Table 8 . Use Case - Download

4.3.4.1 User Interface



This section consists of a set of screen shots to illustrate the use case for downloading an assignment from the CollabX repository. As stated in Table 8, user has to search and select an assignment for download. Figure 43 shows an assignment that has been selected for download. It has the following details:

- **Properties**

It shows the properties that have been set for the specific assignment. The following properties are displayed – Title, Language, Description, Installation Help Text, Level, Duration, Target Audience.

View Object

Properties

Title Exercise 7.10 Factoring of integers.
 Language en_US
 Description Write a program that asks the user for an integer and then prints out all its factors in increasing order. For example, when the user enters 150, the program should print
 Installation Help Text Use a class FactorGenerator with a constructor FactorGenerator (int numberToFactor) and methods nextFactor and hasMoreFactors .
 Level Easy
 Duration 30
 Target Audience Junior

Contents

Files
 ExP7_10/description/index.html
 ExP7_10/grader/check.properties
 ExP7_10/grader/FactorGeneratorTest.java
 ExP7_10/META-INF/MANIFEST.MF
 ExP7_10/solution/FactorGenerator.java
 ExP7_10/solution/FactorPrinter.java
 ExP7_10/student/check.properties
 ExP7_10/student/FactorGenerator.java
 ExP7_10/student/FactorGeneratorTester.java
 ExP7_10/student/supplemental-instructions.html
 ExP7_10/student/test1.in
 ExP7_10/student/test1.out

Additional Properties

Text Book
 Chapter

Searchable: Yes

Keywords Tagged for Search factor;;

Done Critique Categorize Download

Step 1
 Assignment Upload

Step 2
 Annotation

Step 3
 Done

Figure 43 Details of a Selected Assignment

- **Contents**
 It shows the contents of the selected assignment
- **Additional Properties**
 It shows any additional user-defined metadata information that has been tagged with the assignment.
- **Keywords**
 It displays the specific words that help in searching for an assignment.

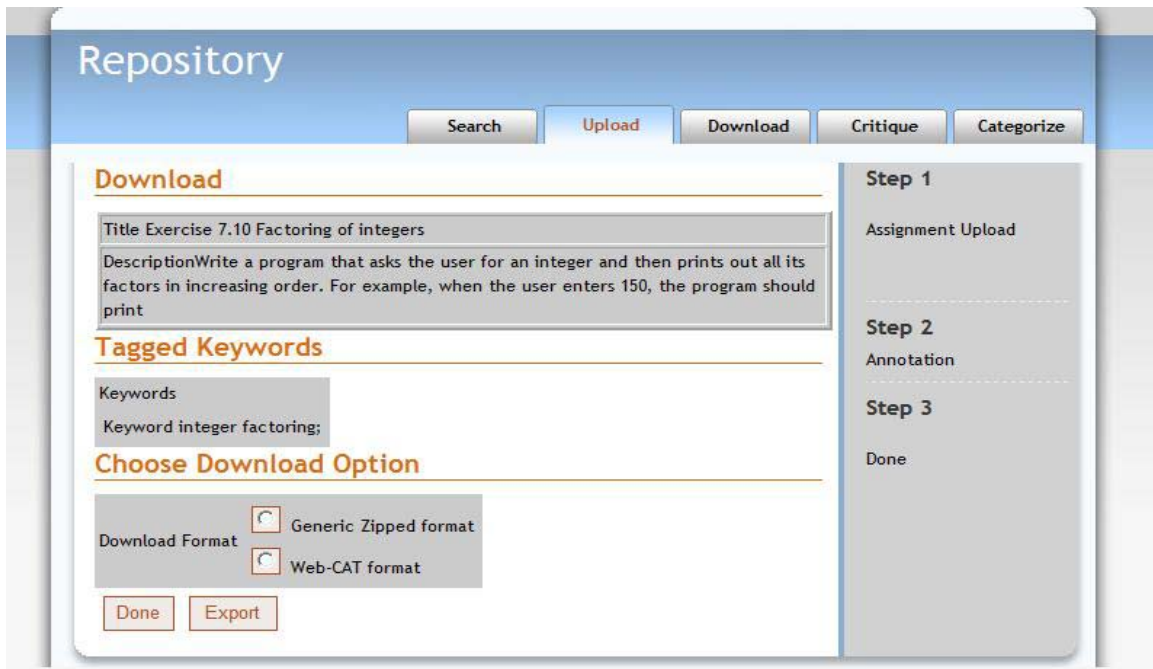


Figure 44 Download Assignment

A user can decide to download the assignment based on the information displayed above; the selected assignment can be downloaded by clicking on the Download button. Figure 44 shows the screen for downloading an assignment from the CollabX repository. User selects the zipped or Web-CAT format and click on the Export button. The Browser will then prompt to save the downloaded archive in the file system.

4.3.5 Use Case – Critique

This use case helps instructors, course developers and researchers in providing feedback for an assignment. These feedback comments are then persisted in the repository and made available to other users.

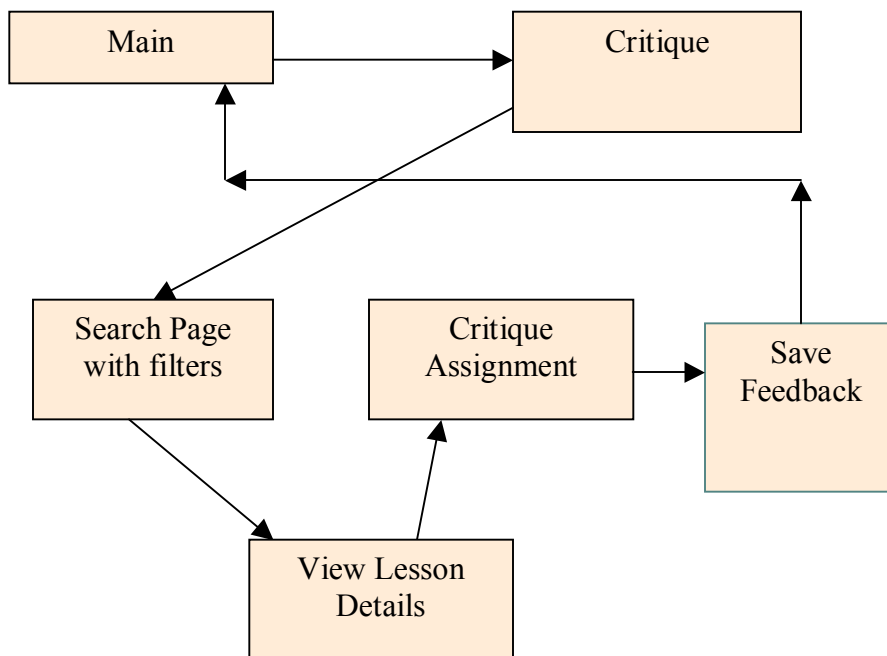
Table 9 enumerates a set of activities to provide feedback for an assignment in the repository.

User action	System response
User clicks on the Critique tab.	System redirects to the search page.
User inputs the data based on which the search will be executed. .	System displays the search results matching the filter criteria.

User selects one of the assignments that are displayed in the search results page.	System displays the details of the assignment, which includes its contents, metadata definitions, keywords tagged for search and user defined properties. A Critique button is displayed in this page.
User clicks on the Critique button	System displays a page showing a feedback questionnaire to the user.
User fills up the questionnaire and Clicks on the Save button	System saves the feedback for the assignment in the repository.

Table 9 . Use Case - Critique

4.3.5.1 User Interface



This section consists of a set of screen shots to illustrate the use-case for providing feedback for an assignment in the CollabX repository. As stated in Table 9, user has to search and select an assignment for providing feedback. Figure 45 shows feedback user interface for an assignment. User has to provide the following feedback:

- Did they like the assignment?
- Did their students like it?
- Did they modify it?
- Any other comments about the assignment.

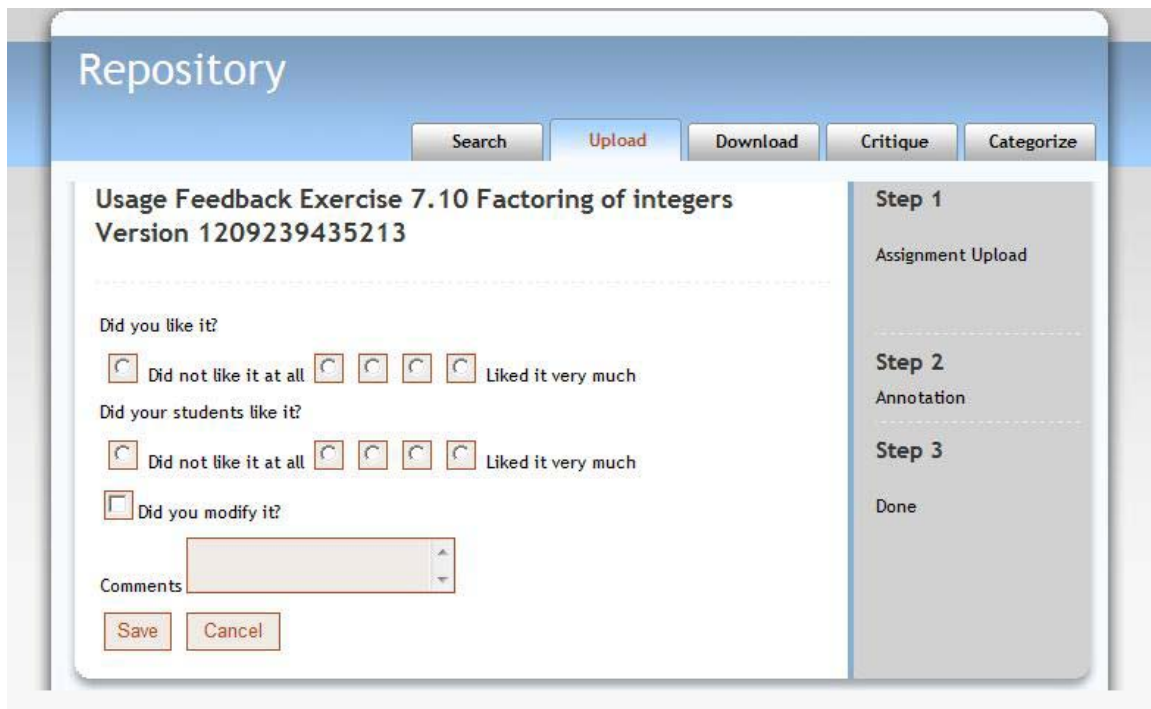


Figure 45 Feedback for a selected assignment

4.3.6 Use Case – Categorize

This use case helps instructors, course developers and researchers in categorizing an assignment. During the upload process, an assignment is assigned a specific set of properties and metadata definitions that best describes it. This categorization is persisted in the repository along with the contents of the assignment. But as more and more users start using this assignment for their needs there might be a need to add more metadata definition to accurately describe the assignment. This might be based on its usage in various research and academic fields.

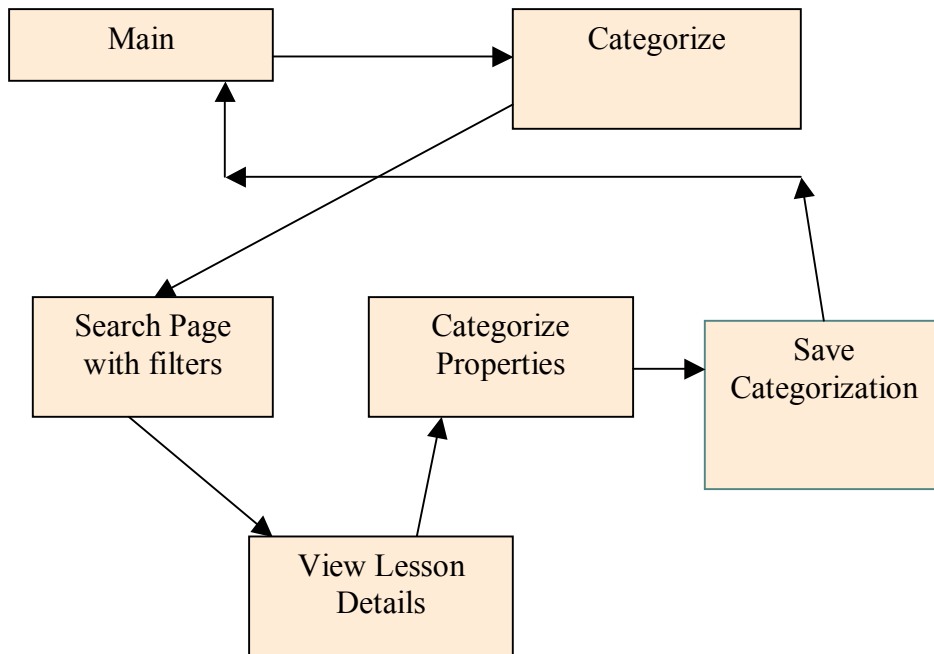
Table 10 enumerates a set of activities to categorize an assignment in the repository.

User action	System response
User clicks on the Categorize tab.	System redirects to the search page.
User inputs the data based on which the search will be executed.	System displays the search results matching the filter criteria.
User selects one of the assignments that are displayed in the search results page.	System displays the details of the assignment, which includes its contents, metadata definitions, keywords tagged for search and user defined properties.

	A Critique button is displayed in this page.
User clicks on the Categorize button	System displays a page to the user with edit controls to add, edit or remove properties.
User re-categorizes the assignment and clicks on the Save button.	System saves the categorization of the assignment in the repository.

Table 10 . Use Case - Categorize

4.3.6.1 User Interface



This section consists of a set of screen shots to illustrate the use-case for categorizing an assignment in the CollabX repository. As stated in Table 10, user has to search and select an assignment for categorization. Figure 46 shows the user interface for categorizing a selected assignment.



Figure 46 Categorization of Assignment

In this page, user can perform the following for re-categorization of the assignment-

- Add additional properties to the assignment by clicking on “More Properties...” button.
- Add multiple values to a property by clicking on the “More Values...” button.
- Deleting a property
- Deleting one of the existing values for a property.
- Edit the list of keywords for the assignment

5 Conclusion

This present study focused on the problem of isolated repositories of programming exercises. These repositories exist within their own eco-system and can hardly collaborate with the global communities of computer science researchers, course developers and instructors. Lack of a common vocabulary and inconsistent terminologies used across the existing systems, inevitably leads to re-inventing the wheel. Without knowledge of other people’s work, these exercises developed in isolation get lost due to lack of collaboration, feedback and insights.

This project attempts to develop a platform for collaboration of programming exercises among Computer Science instructors, researchers and course developers. The central point to this platform is the definition and persistence of the metadata in a repository that describes a programming exercise [1]. There are existing systems like Marmoset [7] and Web-CAT which supports submission and automatic grading of exercises, but these exercises are specifically developed for these systems in a proprietary format. As with any other proprietary technology, the exercises developed for these systems are pinned to these platforms. The collaboration platform proposed in this study is shown in Figure 47.

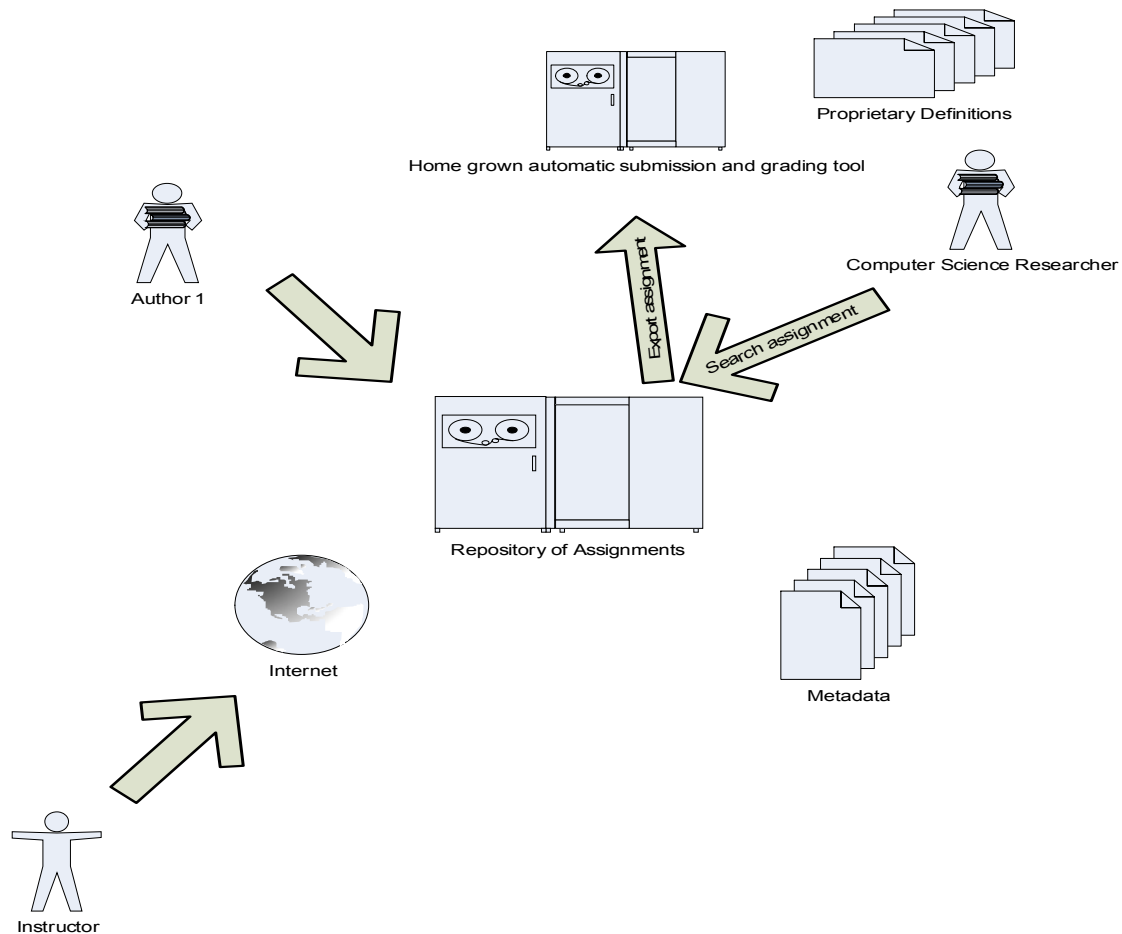


Figure 47 Repository of assignments used by different stakeholders

It has a repository, which is a place holder for programming exercises and its metadata definition. This persisted metadata will be introspected by the platform to collaborate and share a programming exercise among researchers and instructors.

The basic step in sharing a learning resource is to tag it with a metadata definition. This metadata definition is based on the IEEE LOM specification [1]. This specification leverages the power of XML to describe a learning object in several categories as shown in Table 11.

<i>General category</i>	groups the general information that describes the learning object as a whole.
<i>Meta-Metadata category</i>	groups information about the metadata instance itself
<i>Technical category</i>	groups the technical requirements and technical characteristics of the learning object.
<i>Educational category</i>	groups the educational and pedagogic characteristics of the learning object.
<i>Rights category</i>	groups the intellectual property rights and conditions of use for the learning object.
<i>Relation category</i>	groups features that define the relationship between the learning object and other related learning objects
<i>Annotation category</i>	provides comments on the educational use of the learning object and provides information on when and by whom the comments were created
<i>Classification category</i>	describes this learning object in relation to a particular classification system.

Table 11 . IEEE LOM Categories

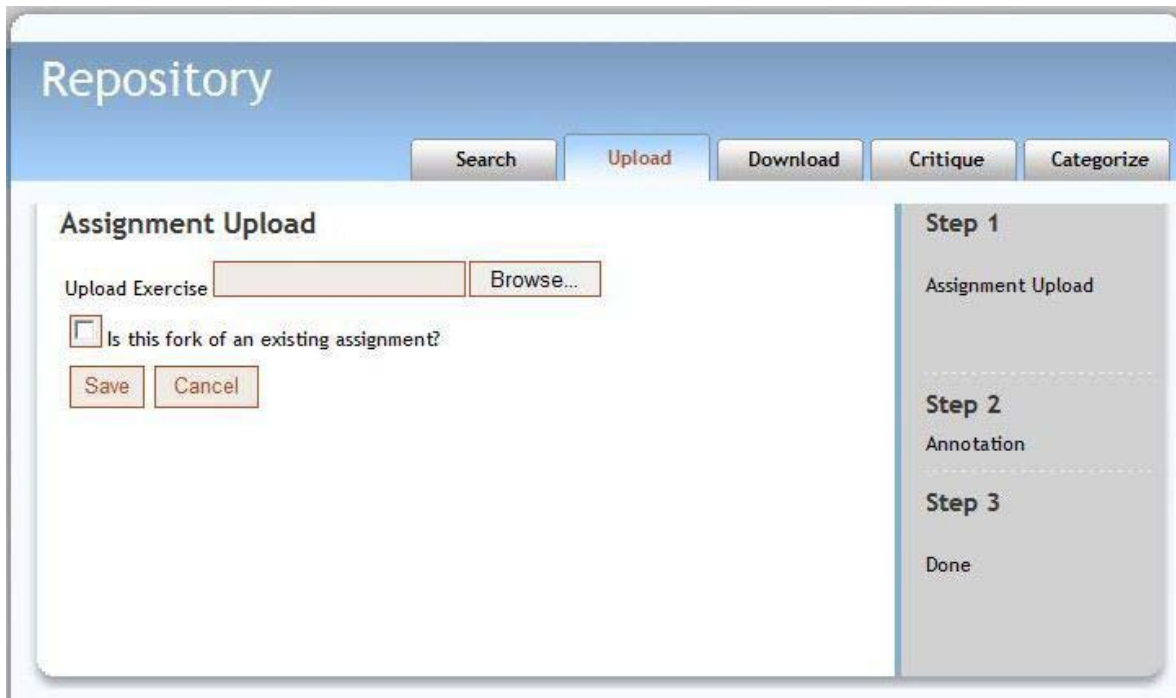


Figure 48 CollabX user interface

A programming exercise that is submitted to this repository is modeled in the form of a Learning Object (LO) [6] with its associated metadata. The metadata residing in the LO is persisted in the database [4] along with its content. Each LO can also be searched, critiqued, annotated and rated [8] according to its usefulness as shown in Figure 48. A feedback rating system and metadata definitions will enhance re-usability of the programming exercise ([2], [3]) and inter-operability with third party grading and submission systems [5].

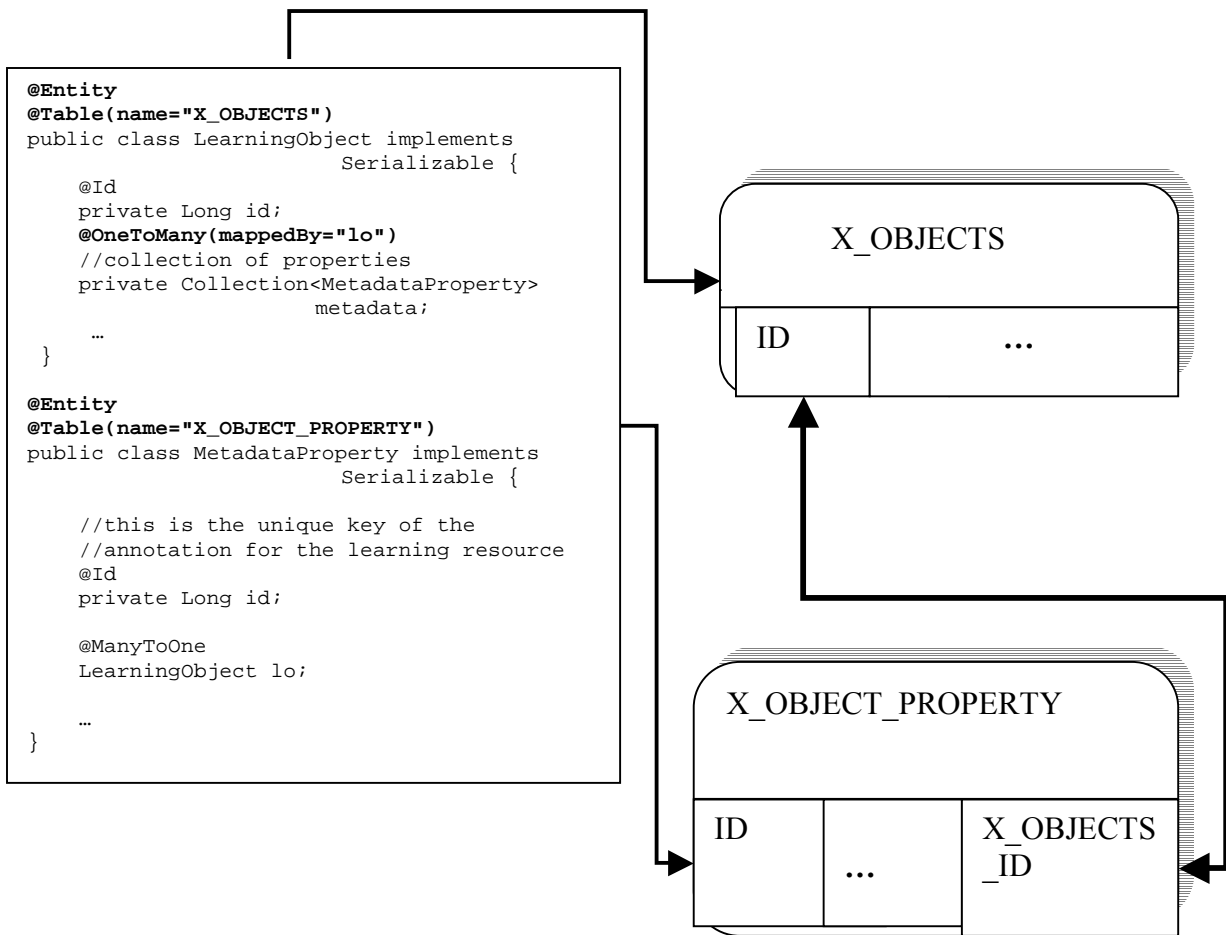


Figure 49 O/R Mapping

The prototype is 3-tier application built on Java EE platform. This platform has support for Java Server Faces, EJB 3.0 and JPA. The persistence layer is modeled as a set of lightweight domain objects called entities as shown in Figure 49. These entities have physical as well as relationship mapping defined by the JPA specification. JPA provider translates the physical and relationship mappings into a set of tables with referential constraints mimicking the relationships in the target database as shown in Figure 54.

This research study and implementation has the following achievements-

- The definition and persistence of the metadata in a repository that describes a programming exercise. The metadata definition was loosely modeled from IEEE Learning Object Metadata (LOM).
- In a relatively short course of time, built a prototype which can manage the lifecycle of an exercise in a shareable repository. Lifecycle of an exercise includes fork, feedback, critique and export.
- This implementation has collaborative features such as "Fork", "Critique" and "Categorize", which is unavailable in similar repositories.
- Integration with external systems such as Web-CAT and Labrat.

References

- [1]. Draft Standard for Learning Object Metadata Sponsored by the Learning Technology Standards Committee of the IEEE.

- [2]. Barbara P. Heath, David J. McArthur, Marilyn K. McClelland, Ronald J. Vetter. Metadata lessons from the iLumna digital library. *Communications of the ACM* Volume 48 , Issue 7 (July 2005)

- [3]. Reusable Learning Object Strategy: Designing and Developing Learning Objects for Multiple Learning Approaches. *A Cisco Systems white paper*.

- [4]. Filip Neven, Erik Duval, LU Leuven. Reusable Learning Objects: a Survey of LOM-Based Repositories. *Proceedings of the tenth ACM international conference on Multimedia, Juan-les-Pins, France*

- [5]. Michael T. Helmick. Interface based Programming Assignments and Automatic Grading of Java Programs. *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE'07)*

- [6]. Longzhuang Li, Hongchi Shi, Yi Shang, Su-Shing Chen. Open Learning Objects For Data Structure Course. *Journal of Computing Sciences in Colleges archive* Volume 18 , Issue 4

- [7]. Jaime Spacco, David Hovemeyer, William Pugh, Jeff Hollingsworth, Nelson Padua-Perez, and Fawzi Emad. Experiences with marmoset: Designing and using an advanced submission and testing system for programming courses. *ITiCSE '06: Proceedings of the 11th annual conference on Innovation and technology in computer science education*. ACM Press, 2006.

- [8]. Xavier Ochoa, Erik Duval. Use of contextualized attention metadata for ranking and recommending learning objects. *Conference on Information and Knowledge Management. Proceedings of the 1st international workshop on Contextualized attention metadata: collecting, managing and exploiting of rich usage information*

- [9]. Stephen H. Edwards, Manuel A. Pérez-Quiñones. Experiences using test-driven development with an automated grader. *Journal of Computing Sciences in Colleges archive* Volume 22 , Issue 3 (January 2007)

- [10]. Stephen H. Edwards, Manuel A. Pérez-Quiñones. Web-CAT: automatically grading programming assignments. *Proceedings of the 13th annual conference on Innovation and technology in computer science education*.
- [11]. Ruby on Rails Project, <http://www.rubyonrails.org>
- [12]. Web-CAT assignment interchange format. <http://web-cat.org/assignment-packaging/>
- [13]. WG6 - Developing a Common Format for Sharing Assignments.
<http://www.iticse08.fi.upm.es/WG6.htm>. *ITiCSE 2008 The 13th Annual Conference on Innovation and Technology in Computer Science Education*