NTNU

Norwegian University of
Science and Technology

Using Jexserver in TDT4100: Evaluation, Preparation and Integration

Frode Solheim

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The objective of the master thesis is to deploy, use and evaluate the automated JExercise grading server, Jexserver, in the programming course TDT4100. Jexserver is a grading system based on the JExercise framework for programming exercises.

Assignment given: 14. January 2008
Supervisor: Hallvard Trætteberg, IDI

Abstract

JExercise is a programming assistance tool used by students working on exercises for the programming course TDT4100. A prototype of Jexserver, a server for automatically grading students' submissions, was developed in fall 2007 to grade exercises for the course. The Jexserver prototype is now further developed into a production-quality system, tested on students attending TDT4100 with a range of exercises, and a simulated exam is staged to test the performance. The system performance of Jexserver is found to be sufficiently good to use it to host a digital exam, and the conclusion is that Jexserver is ready to be used as the primary exercise grading tool for TDT4100 and other programming courses. The JExercise client itself is extended to be able to submit exercises directly to Jexserver with zero configuration, improving student usability. Finally, Jexserver is successfully integrated with it's learning, the learning management system used by NTNU, by implementing the IMS Tool Interoperability Guidelines, and by specifically targeting the implementation at it's learning, the user interface is also seamlessly integrated, providing a coherent solution with JExercise, Jexserver and it's learning.

Table of Contents

1 Introduction

This chapter establishes the motivation and context for the project, and the outline for the thesis. A glossary with terms used in this document is located in appendix B.

1.1 Context of Master's Thesis

Before we discuss the motivation for this master's thesis, we should start by establishing the background context. This project ties in with an existing project at IDI; the JExercise project. JExercise is an open source project available from IDI's open source portal[1].

From the JExercise home page[2] (verbatim copy):

«JExercise is an Eclipse plug-in that supports a student (or anyone learning Java programming using Eclipse) in doing Java exercises. The goal of JExercise is to support the learning process, by breaking an exercise into a precise set of testable requirements and provide feedback to the student about the progress. By precisely specifying the requirements of classes and methods that the student must write, focus is put on two important aspect of programming: 1) you should know in advance what the code should do, and 2) the code has little value unless it implements the specification correctly. By specifying testable requirements, we make it possible to use Eclipse's integrated support for JUnit-testing to actually test the student's code. And by giving feedback on each step towards a completed exercise, the student will hopefully feel less insecure and more motivated.»

JExercise provides an editor for course staff to use for define the programming exercises. The project also establishes a model and XML-based file format for exercise specifications. But the most important focus of JExercise is to provide a complete environment in which the students can work with the programming exercises. It builds on Eclipse's powerful Java

1 http://www.opensource.idi.ntnu.no
2 http://www.opensource.idi.ntnu.no/homepages/jexercise/

programming support and adds an integrated exercise requirement browser, tools to guide students in creating the expected classes and methods, and sets of runnable tests to check student's code for problems and correctness before submission.

During semesters in both 2007 and 2008, JExercise was used to guide students through all programming exercises, alleviating some of the need to contact the course's staff [1]. Teaching assistants were still responsible for manually registering students' scores based on the JExercise score, after the students demonstrated their solution for the assistants.

JExercise, being a client-side solution, did not provide for a means to store students' submissions centrally, nor perform automatic and trustworthy grading of the exercises.

In autumn 2007, as part of the course Program and Information Systems, Specialization Project at NTNU, and a forerunner to this master's thesis, the author of this thesis designed a server-based system to automatically grade students' submissions. The system was named *Jexserver* and had a web-based interface where students could log in and upload a zip-file containing their submission. The students' would then be run, with focus on maintaining system security, on the server and a final score calculated -- ideally the same score as the student got in JExercise, but more trustworthy since the students' did not have access to the program grading the exercises.

1.2 Motivation

The Jexserver automated grading system designed in autumn 2007 as part of the preparing project for this master's thesis at NTNU culminated in a working prototype of Jexserver, tested with 52 submissions for an older mid-term exams.

The motivation for this master's thesis is evaluate integration opportunities with other course management systems, and promote the state of Jexserver from a working prototype to a production-ready system, by testing and evaluating its use in TDT4100 and identify and implementing changes needed to provide a good user experience for systems. Additionally, evaluating the use of Jexserver in real situations is important.

1.3 Report Outline

Chapter 2, Background and Prestudy, gives a more complete description of the project context and presents related work and background information.

Chapter 3, Research Agenda, presents the research questions, and what methodology is chosen to answer these questions.

Chapter 4 contains requirements for the software implementation, and chapter 5 and 6 describes the implemented system.

Chapter 7 contains presents the student survey performed towards the end of the semester.

Chapter 8 discusses the results, and the thesis ends with a conclusion and thoughts on further work in chapter 9.

2 Background and Prestudy

2.1 JExercise

This chapter expands on the information given in the introduction about JExercise. The description of JExercise is based on information from the earlier Jexserver project report written by the same author [2].

Three defined goals motivated the development of JExercise [1]:

1. Guide the students to writing correct code.
2. Continuously give feedback to the student while programming.
3. Reduce effort to grade the students' submissions.

These goals are met by creating an installable plug-in to the Eclipse development environment; JExercise. This plug-in runs in the Eclipse environment while the student is working and let the student at any time run an extensive set of JUnit[3] tests against the written code. This is made possible by enforcing a specific (minimal) structure to the students code. To further guide the student, JExercise will assist the student in creating this structure and warn about missing or incorrect elements.

From the students point of view, the process of programming for, and submitting an exercise is summarized in the following points:

1. The student imports a JExercise-compatible exercise into Eclipse, with JExercise already installed.
2. The JExercise plug-in displays a document containing the textual description of the

3 JUnit is a standard testing framework for Java. See http://www.junit.org/.

exercise. This document will be held open throughout the exercise for easy reference. The exercise is normally divided into several separate parts, independent of each other. The parts are in turn divided into sub-tasks. The document explains what structure (packages, classes, methods) in required, and how these elements must behave.

3. The structural requirements are supervised by JExercise and the student is continuously informed of missing elements from the required structure. Since the grading process is test-based, it is important that the expected structure is in place so that the tests can find and execute the students' code.

4. At the students request, JExercise will run the entire JUnit test suite against the code and a score will be calculated based on scoring information from the exercise. The student can see the results of every test run, improve the code where necessary and re-run the tests. This process can be repeated until the student is satisfied.

Illustration 1 shows a typical view of JExercise. The student has run the test suite, and the resulting score was 77 of 100 points. As indicated in the view, there was a problem with the brake method written by the student. A test associated with this method failed (see also illustration 2). On the right-hand side, you can see the requirements (and points) for the brake method.

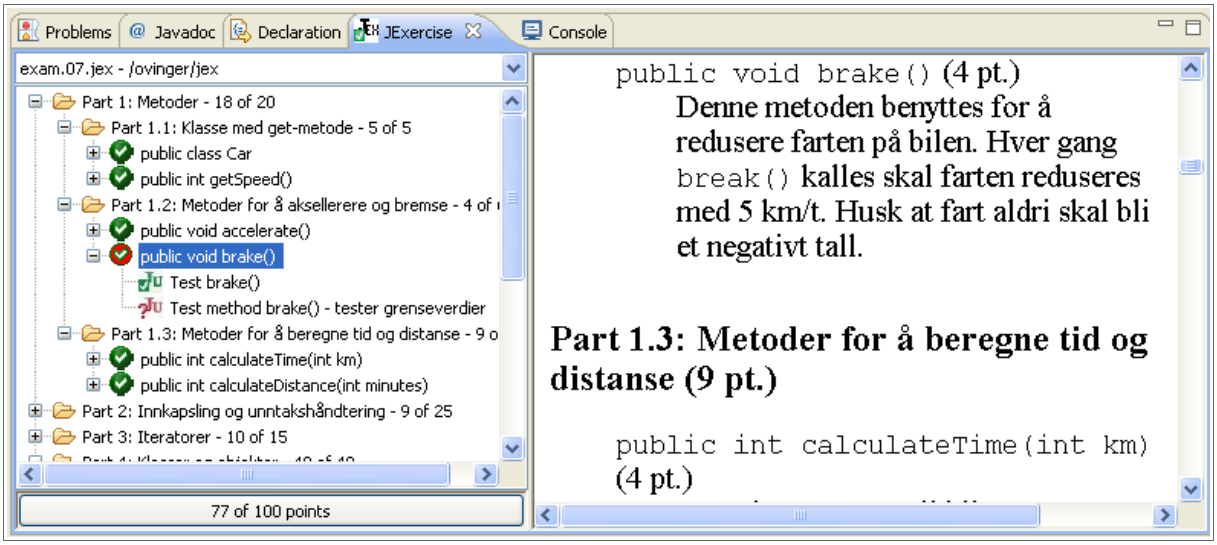

Illustration 1: The JExercise Eclipse view

The JUnit view (illustration 2) displays more detailed information about the failed tests. The

student has also access to the test code and can learn from this more about the nature of the problem. In this case, the test tries (and "succeeds") to brake the car from standstill, resulting in a negative car speed when of course nothing should have happened to the speed (should be 0, was found to be -5).

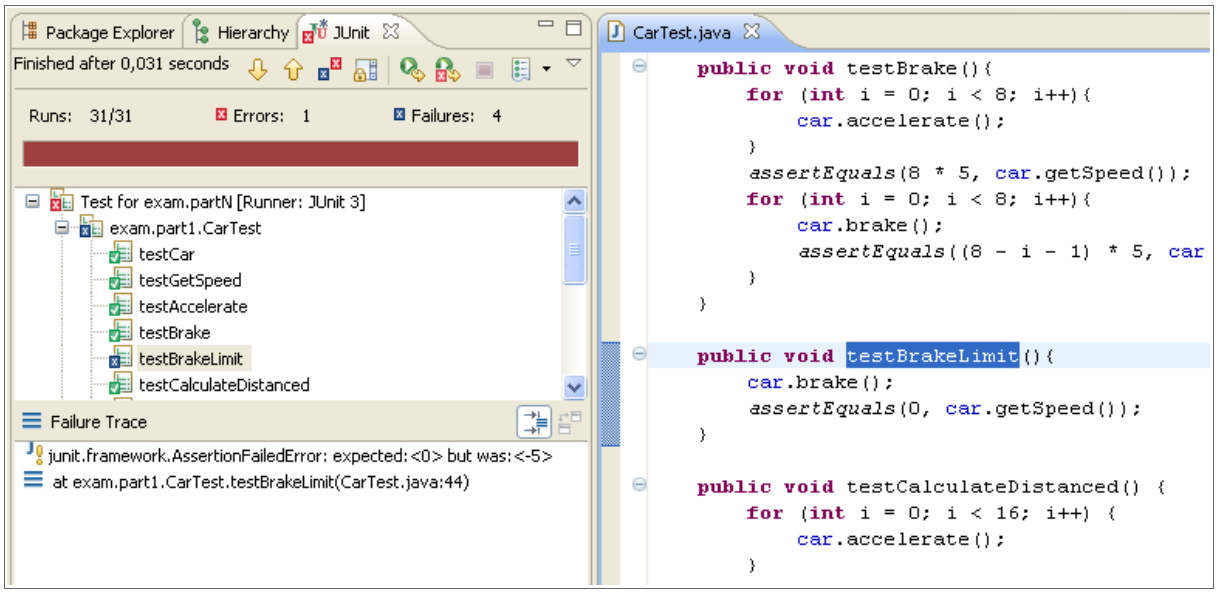

Illustration 2: The JUnit view and test source code

JExercise also includes a tool for creating exercises. First, a set of tests must be created to evaluate the student's code in sufficient detail. These tests depend on running specific methods written by the student, and information about these specific methods should be added to the exercise model as structural requirements. Secondly, a textual description must be written with enough details to enable the students to get a full score by fulfilling the requirements. The exercise model is editable in the JExercise editor. This editor displays the requirements visually as a hierarchical model of requirements and sub-requirements. The JUnit tests written must be linked into this model along with structural requirements.

JExercise was tested and used by 400-500 students in the course TDT4100 in 2006 and 2007, as well as being used for a voluntary exam-like mid-semester test in spring 2007 [1]. For the test exam, the students installed the JExercise plug-in for Eclipse and imported the JExercise test data for the exercise. During the test exam, the students could at any time compile their work and test the submission against the provided test code via the JExercise plug-in. The submissions were delivered as zip files containing the students' Java source code files.

2.2 Jexserver

The paper "Automated Grading of JExercise Programming Exercises and Exams" [2] was written as part of the course TDT4520 - Program and Information Systems Specialization Project during autumn 2007, by the same author writing this master's thesis. The described project, Jexserver, was implemented as an integral part of the research for the report.

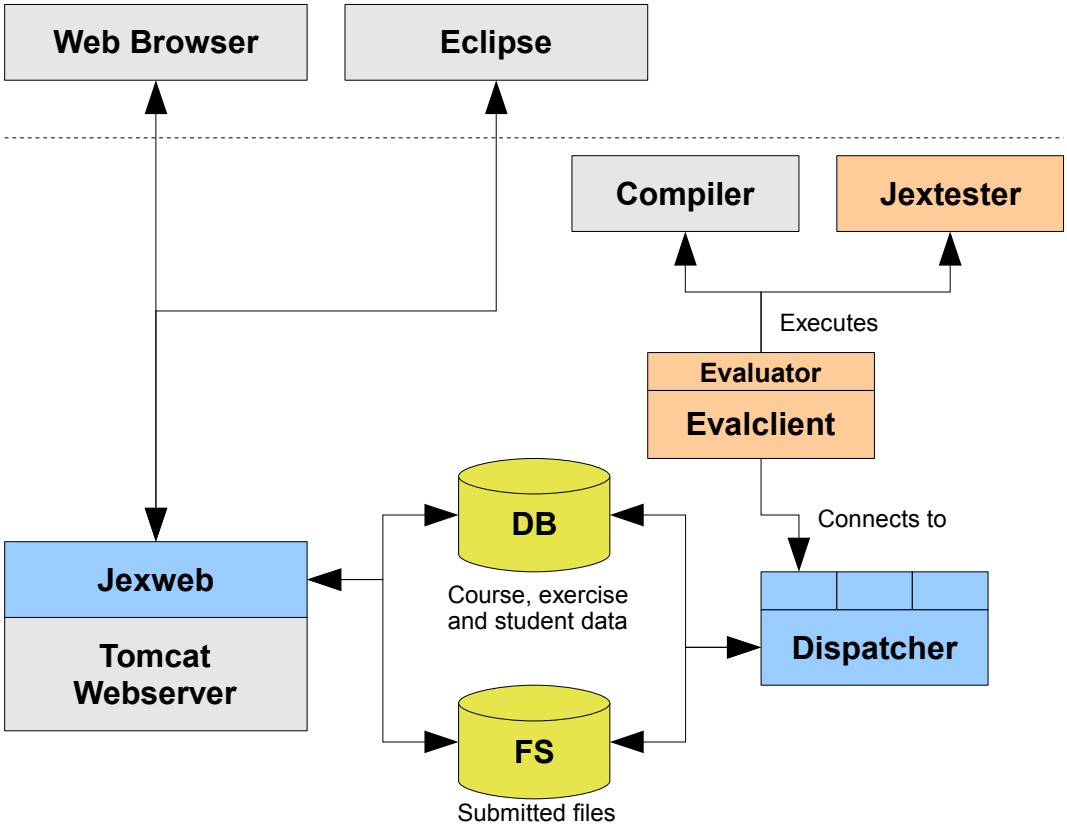

Figure 1: Components of Jexserver (illustration from project report [2]) .

Jexserver consists of the following components:

- A web interface for submission upload (by students) and administrative tasks (by course staff).
- A database and storage area to store all submissions.
- Evaluating client ("evalclient") running students' code and calculating a score based on passed tests.

- A dispatcher, distributing yet-to-be-graded exercises to connected "evalclients".

Jexserver was designed as a distributed system to allow the system to scale well by adding additional computers as needed. Focus areas during research and implementation was:

- Grading exercises by running students' code without risking system stability, facing various kinds of problematic code.
- Achieving grading results comparable to the embedded grading process in JExercise.
- Reading and validating exercise requirements as specified in JExercise's .jex files.
- Evaluate and implement security measures to contain risks of running code directly on university servers.

Illustrations 3 and 4 shows how the web interface looked like at the end of the implementation phase. Emphasis was on functionality and not on visually stimulating user interfaces.

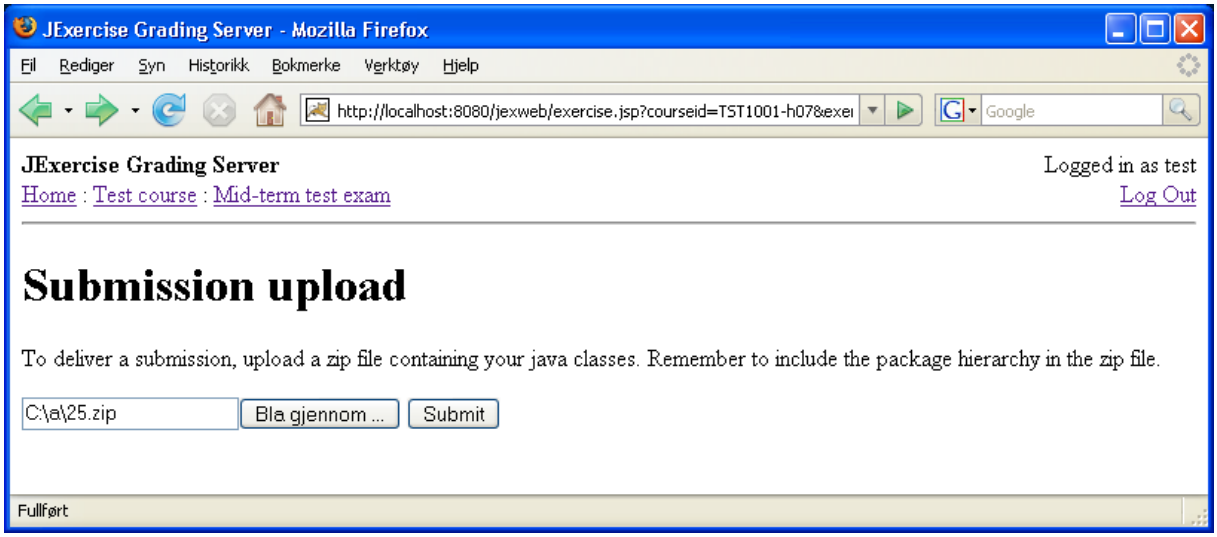

Illustration 3: Submission delivery page

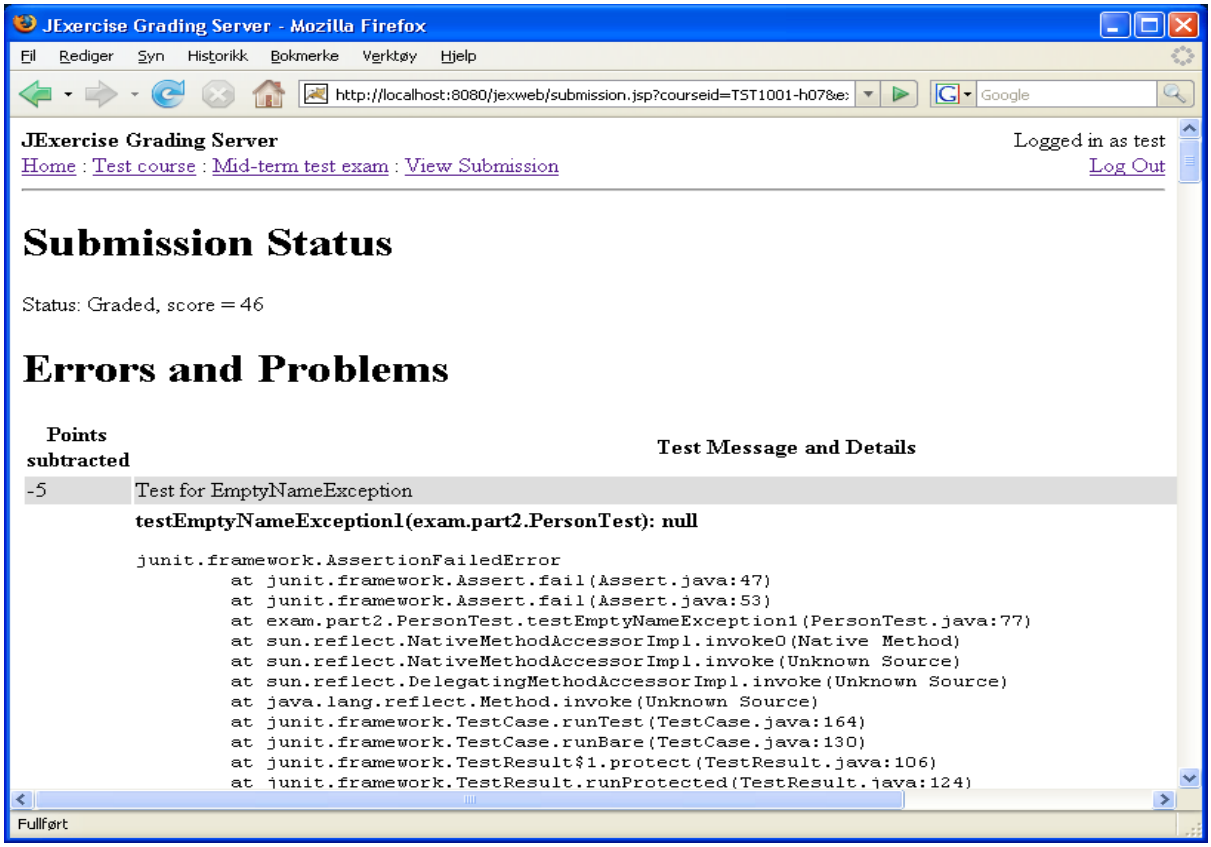

Illustration 4: Submission view with grading results

2.3 it's learning

it's learning is a Learning Management System (LMS) used by several learning institutions, including the Norwegian University of Science and Technology (NTNU).

it's learning supports multiple courses, with a wide range of tools teachers and students can use to interact [3]. The most basic feature is a messaging system where, for instance, a teacher can send announcements to students, comparable to a basic e-mail-like service. The advantage is that you can easily reach all students in a course without maintaining a separate mailing list. Content publishing is the other major basic function, where course staff can publish information about the course as well as downloadable content, including exercise resources or digital literature. Apart from basic content and messaging functionality, it's learning also supports many collaboration tools, such as project-specific collaboration areas and discussion forums. Of learning tools, it's learning has built in support for some assessment types,

including multiple choice exercises.

it's learning is used in the course TDT4100, "Object Oriented Programming", as the central place to find course information, receive announcements and download exercises, as well as providing a discussion forum where students can ask questions about exercises and the course in general.

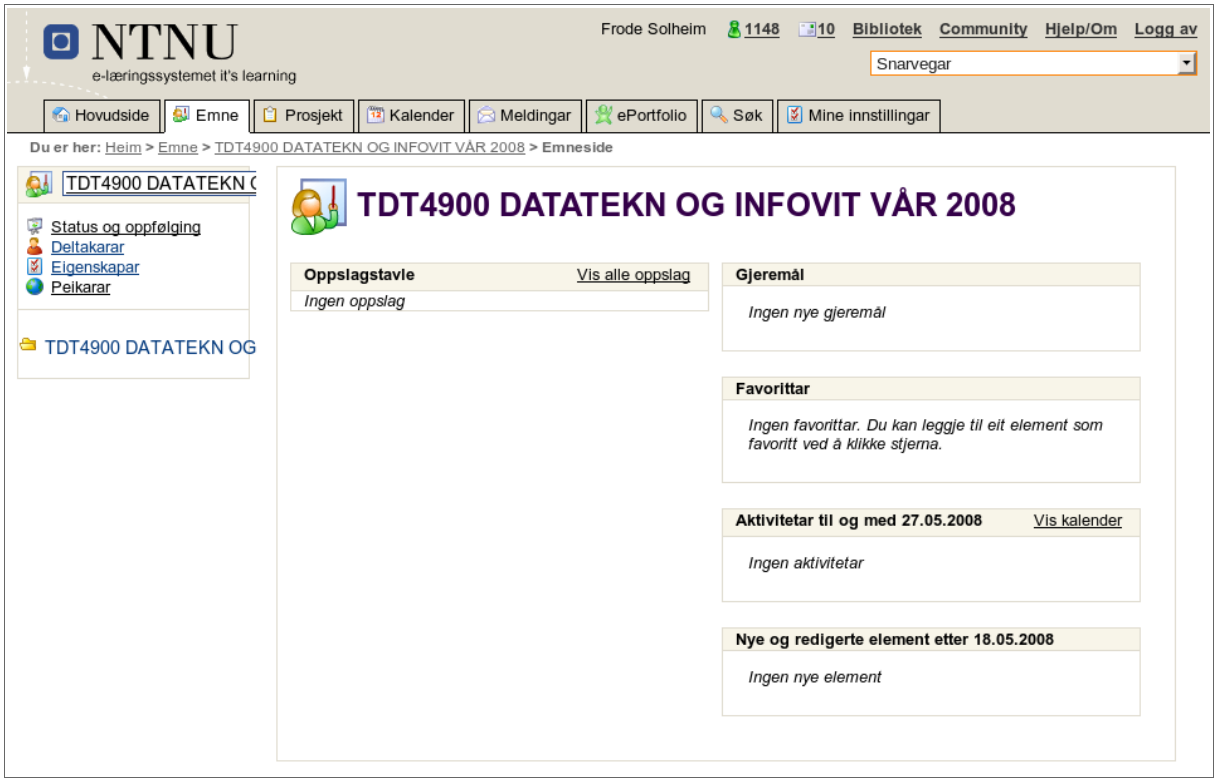

Illustration 5: The web interface of "it's learning" (here, in Norwegian language).

Illustration 5 shows the web interface of it's learning for a course with no published information.

2.3.1 IMS Tool Support

it's learning has announced its completion [12] of integrating IMS Tool Interoperability Guidelines [10] into its core product, making it possible to extend the features of it's learning with custom built learning tools.

The goal is to use the tool support to integrate Jexserver's grading of exercises as seamlessly as possible into it's learning, providing an easy to use interface for students consistent with it's

learning. Built-in support for generating assignment result reports exists in it's learning, and this can be used to aggregate and display results.

2.4 IMS Tool Interoperability Guidelines

The IMS Global Learning Consortium[4] has defined a standard for use by LMS providers and learning tool vendors. A standard such as this is beneficial to learning tool vendors, since it is possible to create a tool than can integrate with any compatible LMS solution. The benefit to LMS providers is that existing learning tools adhering to the standard can be used with any LMS system, including newly development ones, not requiring explicit support for tool vendors.

The standard is described in the document "IMS Tools Interoperability Guidelines" [10] and was issued on Feb 28. 2006. A brief overview of the guidelines is required to follow the discussion of the integration with it's learning

The guidelines provides many UML diagrams of data structures, as well as a illustration of the overall architecture. A simplified illustration of the architecture, relevant to this discussion is provided in figure 2, "LMS and tool interoperability architecture overview".

4 http://www.imsglobal.org/

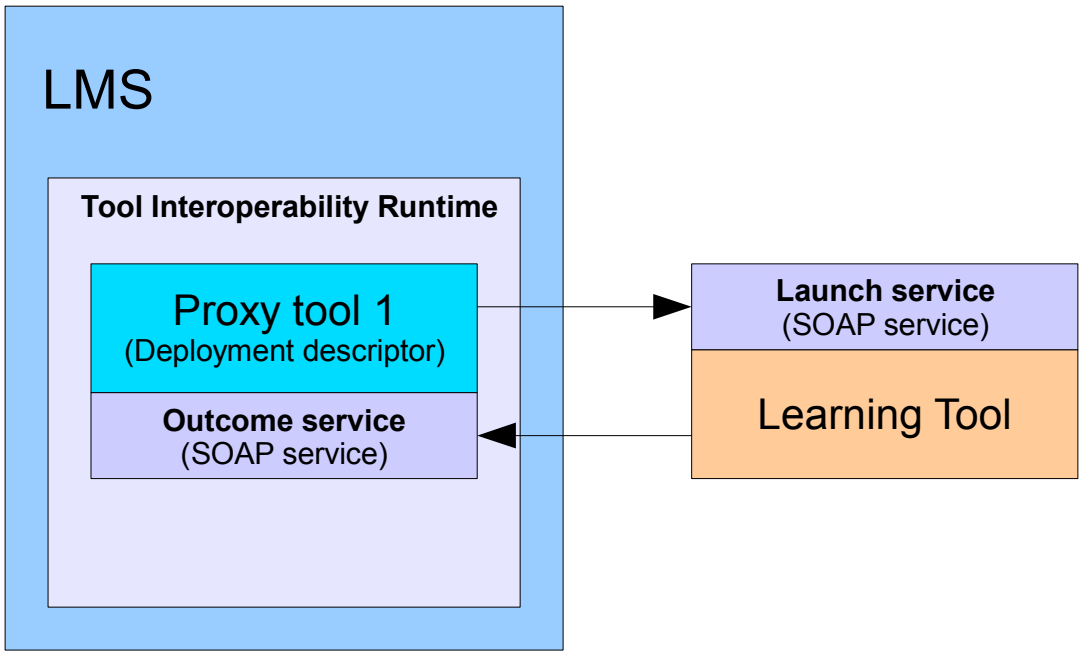

Figure 2: LMS and tool interoperability architecture overview

The relevant components of the system is:

- **Learning tool:** The learning tool (also referenced as "tool") to integrate with the learning management system. This must be a web-based tool, or at least have a web-interface/front-end from which a non-web-based tool can be launched. The interoperability guidelines assumes web-based tools.
- **Launch service:** This is a SOAP web service provided by the tool (or a system containing the tool). The LMS will connect to this web service to negotiate tool invocation when a student tries to access the tool from the LMS interface. How the launch service and the tool integrates with each other is an implementation detail.
- **LMS:** The learning management system that the tool will integrate with. When a student needs to access a learning tool, he or she will have to start the tool from the LMS interface (often, but not necessarily, a web-based interface).
- **Deployment descriptor:** The deployment descriptor is an XML file describing the tool, what settings it requires, and where is is located. More specifically, the XML file contains a textual description of the tool, a set of parameters specifying what

information the LMS must provide as part of the invocation, and an XML schema specifying settings that must be supplied when adding the descriptor to an LMS.

- **Outcome service:** If the LMS supports tools that report results back to the LMS itself (for instance a tool that provides exercises, and generates a score), it needs to implement an outcome service according to a WSDL file provided by IMS Global Learning Consortium. Currently, only simple scores are supported, though other types of result could be added in a later revision [10].

The guideline document frequently refers to a **proxy tool**, which is why this term is present in figure 2, but this is merely a conceptual entity living inside a "Tool Interoperability Runtime" inside the LMS, that communicates with the actual tool (based on information in the deployment descriptor). One could equally well say that the LMS communicates with the tool directly, using information from the deployment descriptor. The "Tool Interoperability Runtime" and "Proxy Tool" are really implementation-specific concepts, and these will not be used in the following discussion.

The guidelines are rich in abstractions and models, and having the following simplification in mind will help understand the document:

- A learning tool is a web-based application that also comes with a SOAP web-service implementing the *launch service* protocol. This allows the LMS to contact the web-service to set up a user session (providing user details along with the request), and the service sends back a URL which the LMS can display to the user.

- The LMS needs deployment descriptors (XML files) to know about a learning tool. The deployment descriptor contains, amongst other information, the URL to the launch service. The LMS optionally provides a web service of its own, the *outcome service*. The URL of this service, if any, will be sent to the tool on tool invocation.

Since communication is based on open and established extensible standards, such as XML, XML Schema, SOAP and WSDL, is is naturally extensible. However, the interoperability guidelines only allows for a quite restricted set of features without extending the standard (and thus breaking the interoperability part).

The feature set can be briefly summarized as:

- The LMS can send an invocation request to the tool, along with some contextual

information, such as the identify of the user accessing the service (for instance, a student), along with a set of tool settings configured in the LMS as part of the tool deployment (for instance, the name of an exercise, or the name of a resource known to the tool).

- The tool can respond either with an error message, a web URL to be displayed to the user, or just a simple text message. If a web URL is returned, the LMS is expected to display this URL, either in a new web browser window, or as a frame in a HTML frame set [11].
- When the student is done with the tool, the tool may report back a score (a single number) to the LMS via an *outcome service*. The URL to this service was sent by the LMS as part of the tool invocation request.

The communication can be summarized by the diagram in figure 3.

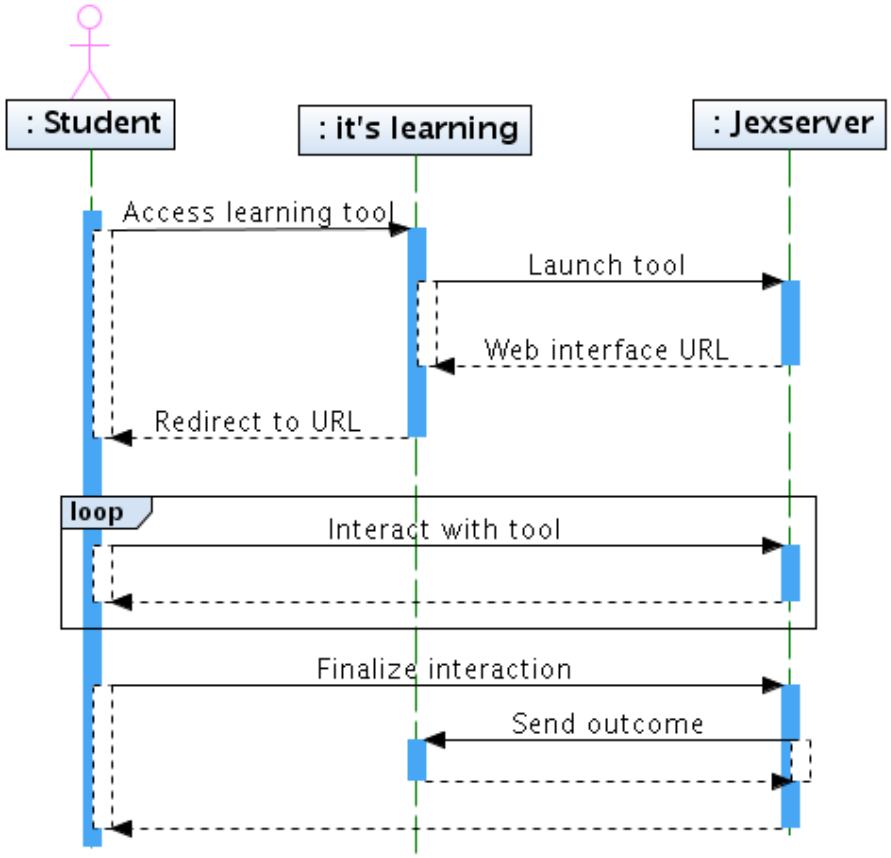

Figure 3: Sequence diagram for launching and interacting with a learning tool

The IMS Global Learning Consortium provides the following resources as part of the Tools Interoperability Guidelines:

- **The guidelines document itself** [10].
- **WSDL file for the launch service protocol.** The tool implements a launch service which is invoked by the LMS when a wants to access the tool via the LMS user interface.
- **WSDL file of the outcome service protocol.** The outcome service is implemented by the LMS, and is used by the tool to report "outcomes" of the tool invocation, e.g. reporting back an achieved score.
- **XML XSD schema** for the deployment descriptor files.
- **IMS Tools Interoperability Tool Test Harness:** This is a testing environment than can load tool deployment descriptors, launch the tool and provides a outcome service for the tool to use. The test harness can be used to test tool implementations.
- **IMS Tools Interoperability ProxyTool Test Harness:** This is a simple tool implementation, that can be used to test LMS implementations supporting IMS tools. The LMS should be able to load this tool (specifically, load the tool deployment descriptor), since this is the only reference implementation of a tool while also being referenced in the tool interoperability guidelines [10].

Both of these test harnesses are referenced in the tool interoperability guidelines [10].

While this is a published standard, little to no literature seems to be available on the subject of evaluation and practical use of the IMS Tool Interoperability Guidelines. This report will discuss it's learning's implementation of the guidelines and experiences designing to it.

2.5 Related Work

2.5.1 Web-CAT

Web-CAT is a server-based system designed to automatically process programming assignments. Web-CAT is an extensible plug-in-based framework for automatic testing. The

Web-CAT server itself does not process nor evaluate submissions, but is dependent on specific plug-ins to perform evaluation and grading [4].

A sub-project of Web-CAT is a submission plug-in for Eclipse, for submitting a zip-file with java files to a web server. JExercise integrates with this plug-in, and this plug-in could also be used to upload files to the automated grading system, as it is used for Web-CAT.

2.6 Related Background Material

This sub-chapter contains references to background material relevant for the implementation aspects of this thesis.

2.6.1 The Tomcat Java Servlet Container

Apache Tomcat is an *"implementation of the Java Servlet and JavaServer Pages technologies"* [8]. Java Servlets and JavaServer Pages (JSP) are the primary method of writing web sites in the Java programming language. Apache Tomcat can be run as a plug-in component for other more feature-rich general purpose web servers such as the Apache web server, or Microsoft's IIS, but Apache Tomcat can also be used as a standalone web server by itself.

For the Jexserver project, Apache Tomcat 6.0 is used to host the web component of Jexserver.

2.6.2 Web Services, SOAP, Apache AXIS

The IMS Tools Interoperability Guidelines is based on using web service technology for communication between the learning tools and the Learning Management System (LMS). Specifically, SOAP over HTTP is used as the messaging platform.

SOAP is an XML-based communication protocol (or service invocation protocol) which can be used to *"exchange structured and typed information between peers in a decentralized, distributed environment"* [9].

Apache Axis is an open-source and freely available implementation of the SOAP standard for Java[5], especially targeted at Java Servlet-based systems.

5 Apache Axis also provided a C++ implementation under the name Apache Axis C++

2.6.3 HTML, CSS and JavaScript

HTML is a markup language used to describe the content and structure of web documents. Traditionally, HTML tags[6] were used to specify both logical document structure as well as layout and presentation style. In later years, focus has been shifted to separate content from layout information, and this resulted in the creation of the Cascading Style Sheets (CSS) standards supported by all modern web browsers. CSS lets you specify layout and presentation rules to how the document content should be rendered thus allowing, to a certain degree, presentation-related markup to be removed from the document itself.

JavaScript, or ECMAScript[7], is a scripting language hosted by and available in all major web browsers. JavaScript lets you add dynamic content to web pages, and most interestingly, alter the Document Object Model of the document itself resulting in altering the document directly in the web browser.

The user interface of IMS learning tools must be written i HTML[8]. CSS and JavaScript can be used to supplement HTML and enrich the interface when designing the user interface for the tool.

6 Tags are markup elements in the HTML markup language
7 JavaScript and Microsoft's JScript are variations of the ECMASCript standard, but these technologies are commonly all referred to as JavaScript
8 Unless you just use HTML to host web-browser plug-ins such as Java Applets or Flash Objects

3 Research Agenda

A prototype implementation of Jexserver was developed for a project report preceding this master's thesis. An underlying goal is to, if the integration with it's learning is successful, move the state of the Jexserver prototype developed in the preceding project to a production-ready system available at the end of the semester of spring 2008 with tight integration between it's learning, Jexserver and JExercise.

The following research questions are to be answered in this thesis:

> RQ1: Can we achieve a sufficiently good integration between the course management system used in TDT4100, it's learning, with Jexserver, to improve user experience and system usability?”
>
> RQ2: How can we reduce or eliminate grading problems due to student errors in the submission process?”
>
> RQ3: Is the existing Jexserver implementation capable of grading actual exercises during the spring semester of TDT4100, and if not, what limitations must be imposed on exercises?”
>
> RQ4: Will the performance and scalability of a production-ready Jexserver be sufficient to perform a mid-term or final exam on computer using JExercise and Jexserver?

The following research methods will be employed for this thesis in order to answer the above research questions:

- Evaluate opportunities for integration with it's learning and implement a working integration component in Jexserver.
- Use Jexserver to grade student exercises during the spring semester of the course TDT4100 at NTNU, and evaluate the success of grading submissions for exercises created by course staff.

- Implement functionality in JExercise to assist with the exercise submission process and test this function in the programming course TDT4100.
- While there will not be staged a mid-term or final exam on computers in TDT4100 this semester, a system to test performance in this setting will be created and deployed, and results of this test will be evaluated.
- A student survey is commissioned to get answer to some usage patterns relevant for this thesis and for the system integration.

4 Requirements

This chapter contains requirements for the implementation part of the thesis, grouped by components and divided into functional and non-functional requirements. Because a significant part of this thesis represents a research project and not merely a software engineering project, a formal and detailed requirement specification is not developed. Still, a set of high-level requirements is developed to later be able to objectively assess the success of the project.

4.1 JExercise Requirements

The high-level requirements for JExercise where developed based on past experiences with JExercise [1] and [2], and by considering its role in the course TDT4100.

4.1.1 Functional Requirements

4.1.1.1 Submitting Exercises Directly to Jexserver

To reduce possibility of student error in submitting exercises, JExercise must be extended with the ability to submit exercises to Jexserver directly from the JExercise interface. This will replace the existing work flow where the student must manually create a zip archive and upload this through the Jexserver web interface.

4.1.1.2 Usage Data and Statistics Collection

JExercise should be extended with functions to collect student usage data. This can allow course staff to evaluate exercises and use the outcome to improvement future exercises. Depending on the data collected, student work pattern analysis can also be performed.

4.1.2 Non-Functional Requirements

4.1.2.1 Interface Should Be Consistent With JExercise

The grading report interface should be consistent with the JExercise interface for for usability reasons. This means trying to emulate both the look and behavior of the interface, as far as possible, including:

- The tree strucure of the exercise view
- Icons used to group elements and denote success and failure
- Display the user interface within Eclipse's own interface to make it an integral part of JExercise.

4.2 Jexserver Requirements

4.2.1 Functional Requirements

4.2.1.1 Receive Submission Directly From JExercise Clients.

Jexserver must provide an application programming interface (API) to receive submissions directly from JExercise clients.

4.2.1.2 Receive Statistics Data from JExercise

Jexserver must be able to receive statistics data from JExercise. As far as possible, this work should not reduce performance or interfere with normal submissions.

4.2.1.3 Successfully Grade Exercises for TDT4100

Jexserver must be able to grade submissions for exercises created for the course TDT4100 during the spring semester 2008, and encountered limitations, if any, must be identified and documented.

The project prototype build during the previous project successfully graded a set of 52 submissions for a mid-term exam, but a wider range of exercises and exercise types will put

Jexserver to the test.

4.2.2 Non-Functional Requirements

4.2.2.1 Performance Requirement

With all other functions implemented, the performance of the system must be sufficiently good to allow the Jexserver system to be used for digital mid-term and final exams.

The course TDT4100 has 250 students taking final exams, and the system must be able to receive submissions for all students during the last 15 minutes of the exam. The system should also grade the submissions within reasonable time to let student's verify that their submissions were properly submitted. We define reasonable time here to be 30 seconds or less.

4.3 Integration with it's learning

4.3.1 Functional Requirements

4.3.1.1 Implement IMS Tool Interoperability Guidelines

Jexserver must provide an implementation of the IMS Tool Interoperability Guidelines.

4.3.1.2 Successfully Integrate Into it's learning

The implemented tool must be able to be loaded into it's learning and function as intended when run from the it's learning environment.

- Students must be able to download exercises from Jexserver via it's learning
- Students must be able to submit and view grading status of their exercises via it's learning

4.3.2 Non-Functional Requirements

4.3.2.1 Interface Should Integrate With it's learning

To provide a seamless and easy-to-use system for students, the system should also achieve a good integration with the interface of it's learning, in addition to integrate well technically. A good indication of success is when the student cannot tell where it's learning ends and where the tool begins. Being consistent includes:

- Using similar icons and visual elements
- Layout information in a similar manner

5 User Interface and Interactions

This chapter discusses the user interface developed for JExercise and for the Jexserver integration with it's learning.

5.1 JExercise Submission Interface

Jexserver, the grading server developed in autumn 2007 [2] required that students log in to the Jexserver web interface and upload a zip[9] file containing their exercise submission, i.e. a set of .Java source code files.

The complete work flow for a student using JExercise to work with the exercise and then submit the result to Jexserver is:

1. Create a new project in Eclipse to host the code and resources.
2. Import a zip-file into the project containing base code and tests provided by course staff.
3. Implement requirements for the exercise, validating the results with JExercise and its JUnit testing and scoring feature.
4. When ready for submission, navigate to the directory on the computer containing the source folder for the Java classes, using a file manager (or a console).
5. Use an archiving tool supporting Zip files to add the entire contents of the source folder to a new Zip file archive.
6. Log in to the student's account on the web interface to Jexserver with a username and password.

9 A file format with lossless data compression, containing one or more files.

7. Navigate to the correct course and exercise on the web interface, and upload the previously created file.

While technically, this solution worked very well, there are some usability problems associated with this approach, one being a quite large number of steps required to submit an exercise, and another problem is caused by requirements to the structure of the zip file

5.1.1 Submit Function in JExercise

Java classes are arranged in a hierarchical structure, called packages (the packages also functions as name spaces to reduce risks of name collisions), and this hierarchical structure must be reflected in the file system. For instance, a class *Demo* in the package *org.example* must be located in a file *Demo.class* in sub-folders *org/example/* located on the Java class path[10]. During earlier tests with JExercise, where students manually created and submitted zip files to course staff, a common problem was that they did not include the expected folder structure into the zip file [2].

Jexserver was designed with this problem in mind, and was successful in eliminating most of these problematic occurrences by being quite flexible when searching for Java classes [2]. Still, it was required that the complete package folder structure was located *somewhere* in the zip file, so a student creating a zip file with no regard to the package structure would risk not having the submission successfully graded.

This is largely a usability problem, since the system works technically well provided that instructions given to students a followed by the letter, but a desirable solution is to remove this dependency on the student getting it right.

One solution to this problem is to convert submission from a manual process to an automatic one. JExercise is a plug-in to the extensible Eclipse program, a powerful integrated development environment (IDE) for Java[11], which means that Eclipse has knowledge of all the source code files (and package structure) used by the student in a project. It is therefore feasible to extend JExercise to bundle files together for submission, making sure the organization of files into a zip file archive preserves the required folder structure.

A button was added to the existing user interface of JExercise by using the appropriate button

10 A list of directories and locations to load classes from.

11 Eclipse is also used as the basis for many different types of development environments, as it is highly extensible due to its plug-in architecture based on the OSGi framework.

component and layout size mechanisms, the result which can be seen in illustration 6.

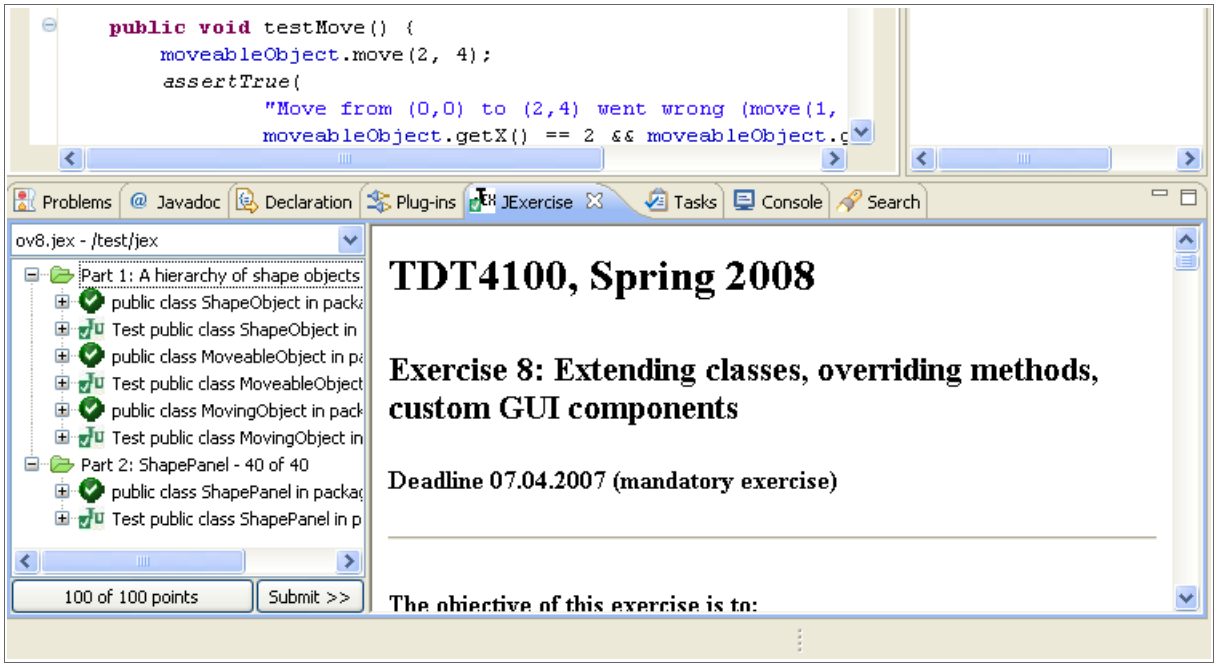

Illustration 6: JExercise View, extended with a "Submit >>>" button (in the lower left corner)

When the user clicks the submit button, all the required files for the submission is automatically bundled into a zip file and uploaded to the server. Details of this process is explained later in the implementation chapter. The only configuration needed by the student is to enter the username and password to the Jexserver account in the preferences dialog for JExercise.

5.1.2 Submission and Grading Progress

Indicating to the user, in a visual way, that things are happening in response to a user action is a good usability guideline. Especially in this case, since grading the exercise could take 1 second, or 5 minutes[12] depending on the circumstances. Eclipse itself displays a visual indication that the job is running by displaying a progress bar while the job is active. That gives the user the immediate feedback from clicking the button (i.e. something happened as a response to the click).

Instead of letting the submission function block the interface while the exercise is being graded, a message is displayed in the JExercise HTML view (illustration 7) until the results

12 Usually, the grading process is quite fast, 1-3 seconds [2], but not necessarily under all circumstances.

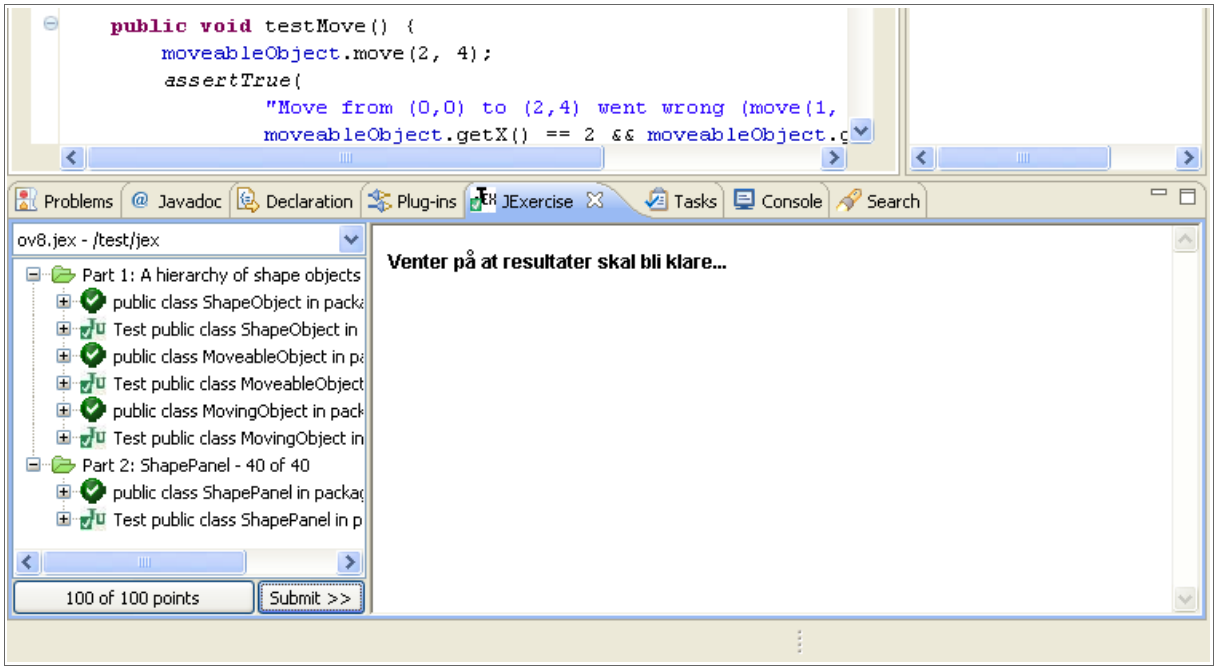

Illustration 7: Submit button pressed, waiting for grading results

are ready. This is implemented in an asynchronous manner, so the user is free to continue working on the exercise if he/she wants to, while the submission is being graded on the server.

5.1.3 Results View

While the already existing result web page (see illustration 4)developed for Jexserver could be used, a new result visualization was developed in order to provide a more consistent user experience. A tree view similar to the exercise requirement tree view already in JExercise.

All other elements other than the top elements are hidden by default. This can be seen in illustration 8 where "A hierarchy of shape objects" and "ShapePanel" are the top elements. The student can use the (+) icons to expand the tree and reveal more information. The tree structure comes from exercise requirements in the .jex file, which structures requirements in a hierarchical manner.

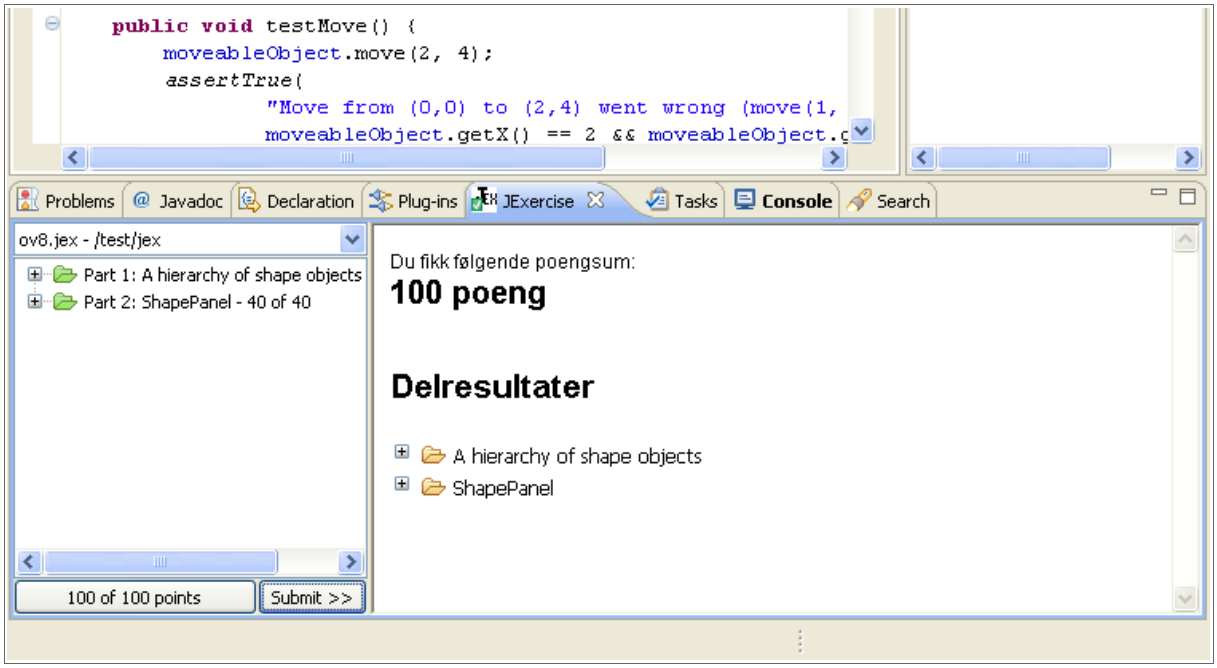

Illustration 8: Grading results are in. Any errors would have been highlighted in the partial results section.

If exercise requirements are not fulfilled, these are clearly marked as failed in the result view, along with the number of points subtracted from the score. All nodes required to display all the failing requirements are automatically expanded when the result page is generated, which gives the student a complete overview of problem without cluttering the interface with details about successfully implemented requirements. An example of a result page with errors can be seen in illustration 9. The nodes representing failed requirements can be expanded to get more details about the problem, In most cases, this will be error messages generated by JUnit tests associated with the requirements.

The user interface will revert to displaying exercise requirement descriptions when the user selects a requirement in the left hand tree view. If the user does this by accident, and wishes to view the results of the server grading, he/she can simply submit the exercise again, as there are no negative consequences of submitting several times.

Illustration 9: Jexserver results page embedded in JExercise, displaying problems

5.2 Integration with it's learning

it's learning is a learning management system (LMS) developed by the Norwegian company it's learning as[13]. The LMS system is used by NTNU for several courses, and is specifically used in the course TDT4100, "Object Oriented Programming". In the following discussion, *it's learning* will be used to refer to the product, not the company, unless otherwise noted.

On their company website, it's learning claims support for learning tools compatible with the IMS Tools Interoperability guidelines [12]. This also seems to be the only way to integrate external tools with the it's learning LMS.

13 http://www.itsolutions.no/

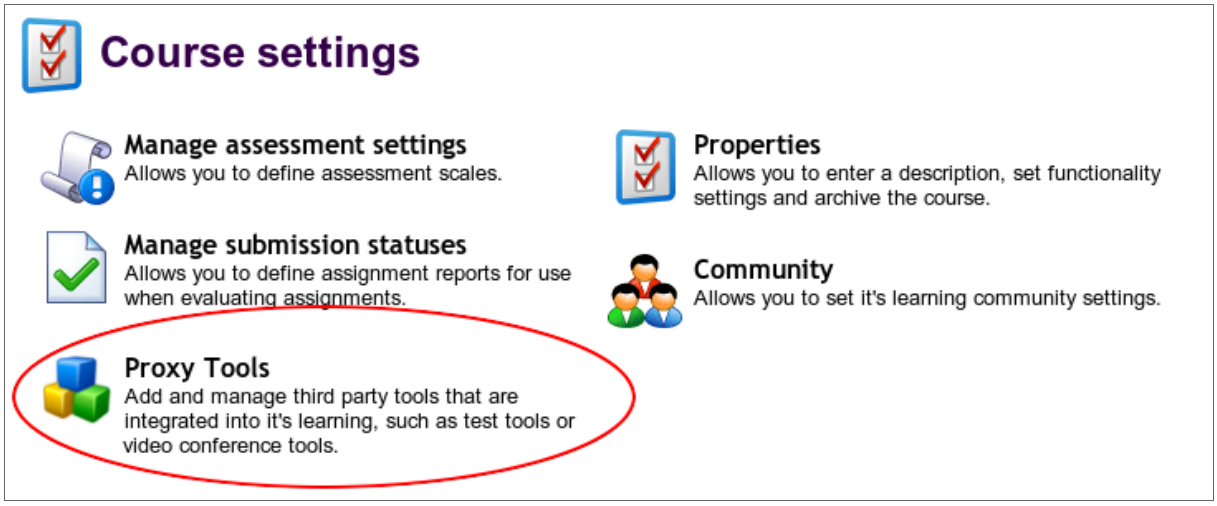

Illustration 10: Adding a proxy tool to it's learning (screenshot of the website).

The vendor of it's learning does not provide publicly available documentation for the proxy tool implementation support, other than referring to the IMS Tool Interoperability guidelines [10] and a smaller section about proxy tool *usage* in the internal help system.

5.2.1 Integration Opportunities

We need to look at the current situation to be able to discuss how a potential integration can benefit students in the programming course.

During spring 2008, this is the situation in the course TDT4100, "Object Oriented Programming":

- it's learning is used for course information, and exercises are made available for download on it's learning. Students logs on to the it's learning website through a university web portal "Innsida", a single-sign-on-solution for many university services.
- JExercise is used by students to help them with implementation details and test their code.
- Teaching assistants uses an in-house web-based system to register exercise results for individual students. Teaching assistants evaluate and grade exercises (live, with students) in computer labs, and register the results in this system.

- Students are encouraged to also submit the exercises to Jexserver for electronic grading with the newly built submission feature in JExercise. Use of this system requires students to configure JExercise correctly with login information to their user account on Jexserver. Electronic submission was not made mandatory.

A problem with this setup, with electronic submissions, is that there are unnecessary many systems involved:

1. A student must use it's learning to access exercises
2. A student must use JExercise to work with the exercise
3. A student must have an account on Jexserver, and must configure JExercise to submit completed exercises to this account.

Eliminating it's learning is not really alternative; course information of all sorts are managed here, and the students already has access to it's learning by default, through Innsida (the university web portal). It does not make sense to eliminate JExercise from the equation either, since this tool helps the student during programming.

Obviously, Jexserver is also needed, since neither it's learning nor JExercise can provide a server for receiving and grading exercises on-line. But a successful integration of Jexserver and it's learning could reduce the perceived complexity of the system.

1. Jexserver already has an internal user database, where submissions are associated with users in this database. But integration with it's learning could remove the need to have a separate user account on Jexserver, if the user databases could be linked somehow.
2. If a seamless integration with it's learning is successful, it could be possible for students to use Jexserver without knowing it is a separate system, including without needing to know the web address (or user account details) for Jexserver.

5.2.2 User Authentication

One of the most appealing aspects of tool integration is to be able to let the LMS handle the users authentication. While it is not particularly difficult to create a user account system and web based login system, it is an inconvenience to users to remember an additional set of username+password, and it also creates more support issues for course staff.

By deferring authentication to it's learning, a log-in interface is not needed, and the user can immediately access tool functionality after launching the tool from it's learning.

5.2.3 Jexserver and Tool Support

We now have to consider how we can use the IMS Tools Interoperability Guidelines to provide the best integration possible between Jexserver and it's learning.

We can identify three goals that should improve both students' and course staffs' experiences with the system, in order of priority:

1. Not having to register an account and memorize another set of username, password and server address will be helpful to students, but course staff will also benefit from this since support issues related to accounts (lost passwords, ...) will not arise.

2. Embed the grading system (Jexserver) in the interface of it's learning to reduce the perceived complexity of the system. This is most useful when combined with (1).

3. Report the students' scores back to the LMS so that the regular LMS tools can be used to aggregate student results and determine passed/not passed status or grade.

5.2.4 Tool User Interface Integration

The IMS Tools Interoperability Guidelines does not specify any method to integrate intimately with the user interface of the hosting LMS. This is identified in the guidelines as an area of improvement for future development, and the authors recommend that in the future, the LMS will render the tool's user interface based on an abstract user interface description from the tool [10].

In the current form of the guideline specification, the tool can only provide an URL to a web page, which the LMS must then present to the user, but the document gives some leeway in how to present this URL. The document suggests using *HTML iframes*, regular *HTML frames*, *HTTP redirect* or a new browser window [10]. Referring to the administration system of it's learning for configuring tools, we can choose to use either regular HTML frames or display the tool URL in a pop-up window (a form of new browser window).

Pop-up windows are an increasingly bad way of presenting web content. In Q1, 2007, 75% of North-American computer users were found to have pop-up blocking software installed [17].

While some pop-up blockers will allow pop-ups initiated by mouse clicks on links, it is difficult to create a universally workable interface across all web browsers and pop-op blocking solutions.

This leaves us with the embedded HTML frame solution. HTML frames allows us to define a rectangular area in a website interface where content is included from another website [11]. This rectangular area can also be scrolled independently of the original web site interface. The interface if it's learning is made up of HTML frames assembled into a HTML frame set. This can be verified by checking the HTML source code of the website in a browser. The interface of it's learning is mainly composed of a slim top frame, extending the entire horizontal with of the page, containing the main menu of the site. Along the left of the web site is a hierarchical navigation list containing links to content within the currently selected context, for instance a course. The rest of the available space is devoted to the actual site content (the "content frame"). Selecting links in the navigation tree or the main menu changes the content in this frame.

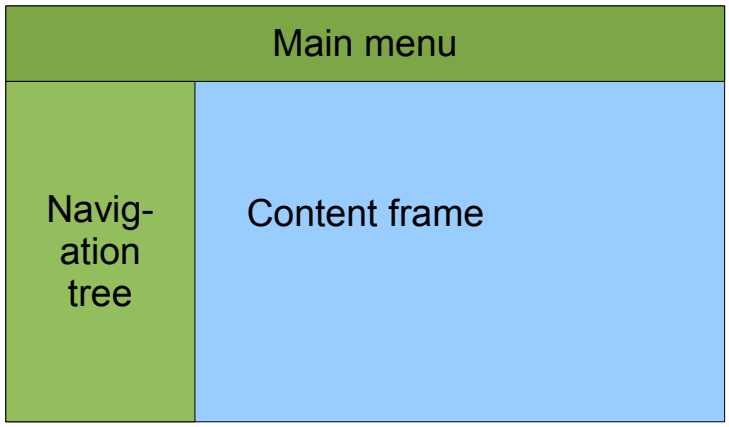

Figure 4: Interface layout of it's learning

Since the interface of it's learning is based on displaying content in the content frame when the user chooses a navigational element, choosing to use *HTML frame* for displaying the tool interface will display the tool interface in the same way as native content, thereby making it possible to create the illusion that the tool is an integral part of it's learning.

While the Tool Interoperability Guidelines does not allow for the LMS to render tool user interface elements, to provide for a seamless interface integration, it is possible to optimize a tool for a specific LMS. This is not in the spirit of *interoperability*, but the main focus of this thesis of to see how we can integrate with it's learning specifically. it's learning uses

Cascading Style Sheets (CSS) to specify the look of the the site contents. CSS is a technology for adding style to web documents, with style information decoupled from the actual content [19]. By referencing the style sheet used by it's learning, we can apply the same style to Jexserver content, provided that Jexserver uses similar user interface elements (styles may not apply otherwise).

5.2.5 New it's learning-specific Web Interface

A necessity for the integration between it's learning and Jexserver, a new interface is created specifically for use when Jexserver is accessed from it's learning. The reasons for creating a new web interface is:

- The existing user interface used a log-in form to allow access to the system. This is no longer relevant since the user logs in via it's learning[14].
- With the existing user interface, the student chooses from among a list of courses[15] and then from a list of exercises for the chosen course. This is not necessary with the new integration with it's learning. The tool deployment descriptor created for the Jexserver tool has settings for course and exercise. This means that when the tool is launched by a student, the exercise and course information is sent along the launch request and information relevant to the exercise can be displayed immediately.
- Another set of functions need to be available in the user interface, as we will see later in this chapter (download of student-specific exercise zip files and reporting scores to the LMS / it's learning).
- For a better user experience, the user interface should blend in with it's learning.
- It is a good idea to keep the old user interface intact. This way, Jexserver can be used standalone as before, without requiring an LMS interaction.

The new web user interface for Jexserver, as seen by a student, is shown in illustration 11. The visual style closely matches that of it's learning, by referencing the same Cascading Style Sheet (CSS) and using corresponding HTML document elements. The content outlined with a red stippled line is a HTML frame displaying a web page from Jexserver. The rest of the

14 For NTNU students, the log in is performed by *innsida*, an intranet solution used by NTNU, but this detail is not relevant here.

15 Based on which courses the students is allowed access to

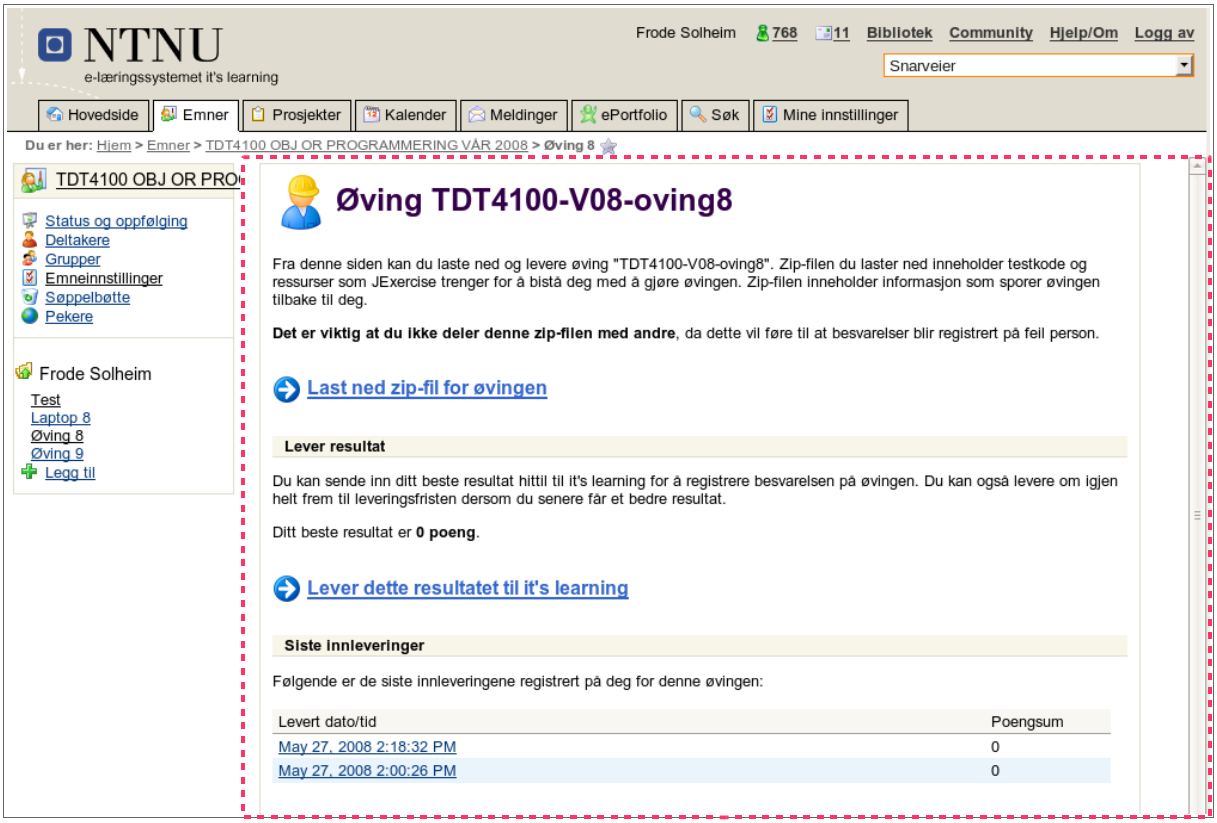

Illustration 11: The new main interface of Jexserver as seen when accessed from it's learning. The area outlined with a red stippled line is content from Jexserver.

interface is it's learning's own interface.

When the user clicks on the ”Download exercise zip file” link on the page, a zip file is dynamically created based on the original distributable zip file for the exercise, but information is added to the download to be able to automatically identify the student when he/she submits the exercise back to Jexserver via JExercise.

At the bottom of the page is an upload for for students that wishes to upload a zip-file directly via the web interface. This can be used for students that do not wish to, or can not, use JExercise.

5.2.6 Implications for JExercise

Since information that can be used to track the exercise back to the student downloading it is embedded in the downloadable zip file, it is no longer necessary for the student to enter the username and password to their Jexserver account in JExercise's preference dialog. In fact, the student does not even know this information, since the username is now just a matter of

internal details (see the implementation chapter).

The downloaded exercise can be imported into Eclipse via the zip archive import mechanism and the submission uploaded back to Jexserver without any configuration at all.

Similarly, the embedded information also alleviates the need for the course staff to add a submit URL to the .jex file when creating the exercise. The correct information is automatically added to the zip file when it is downloaded by the student.

These improvements reduces extra work for for both students and course staff, and human errors such as typing in a wrong submit URL in the .jex file or entering wrong user account information is practically eliminated.

5.2.7 Reporting Results Back to it's learning

When the student is satisfied with the achieved score, he/she can report this result back to it's learning. The “Report this result to it's learning” link (illustration 11) is used to perform this action. Information about highest achieved score and this link is only displayed when the student has uploaded at least one submission.

When the link is clicked, the IMS Outcome Service implemented by it's learning is invoked to return the result back to it's learning.

The IMS Tool Guidelines does not specify a way to return the student automatically to it's learning. The navigational controls are still present in the interface of course, so the student can always click on a content link manually.

But it makes sense to return the student to it's learning when his/her interaction with Jexserver is done. Since the guidelines does not specify how this should work, a trick specific to it's learning is used to return the student to the page from which he/she launched the Jexserver tool. An added benefit of this is the the student clearly sees that the score was in fact reported correctly back to it's learning. Illustration 12 is a screenshot of the it's learning interface after having reported the score back to it's learning, and subsequently redirected the web browser to the it's learning launch page for the exercise.

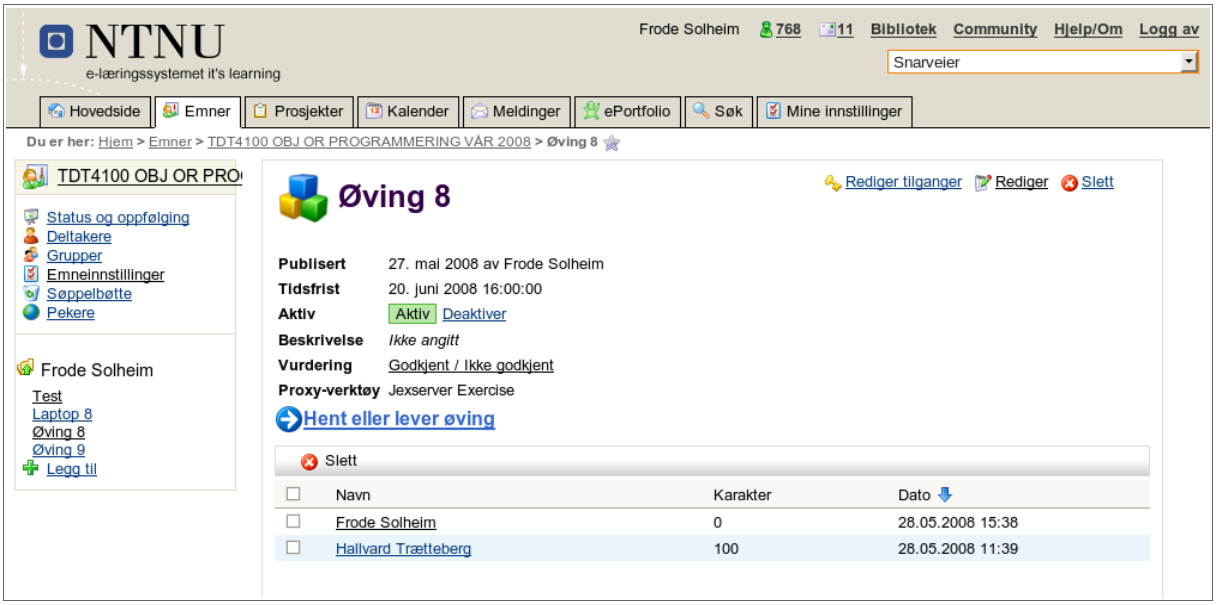

Illustration 12: Web interface of it's learning; the launch page for the Jexserver tool with a table of results for each individual students.

5.3 The Teacher Role

This project has not focused on the teacher's role in the integration with it's learning, but the teacher can use existing functions in Jexserver while integrating the system with it's learning:

- Jexserver has an existing web interface were teachers can define new exercises [2].
- The teacher interface was extended with a function to upload distributable zip files.
- Finally, the teacher must create new instances of the tool for each exercise by using the functions provided by it's learning.

A follow-up to this project could look at teacher integration opportunities with Jexserver and it's learning.

6 Implementation

This chapter discusses implementation and technical details concerning modifications to JExercise, Jexserver, and the newly developed integration with it's learning.

6.1 JExercise

The following sub chapters discusses implementation specific details for the new features in JExercise.

6.1.1 Automatic Zip-creation and Submission

JExercise was already integrated with a submission plug-in from another project, the Web-CAT project [4]. Initially, the plan was to use this open source plug-in to provide the submission feature for JExercise, provided that a compatible server side component in Jexserver would be capable of receiving the submission from the Web-CAT submission plug-in.

Creating the service side component for Jexserver would be fairly trivial, since Web-CAT used the multipart/form-data standard to upload the file [5][6]. This means that the request as received by the server is identical to a request from a web browser sending data from a standard HTML form (including file uploads) [11], with the implication that standard server side technologies can be used to process the request.

JExercise has an embedded HTML display component, also capable of loading web pages, used to display the exercise text so all requirements are readily available to the student as he/she makes progress in the exercise (see illustration 1 on page5). It was desirable to display the grading results in this display window after the submission had been uploaded and graded.

The problem with the Web-CAT plug-in was that there were no way to access the response

sent by the server from code using the plug-in. The Web-CAT submission plug-in did support displaying a simple text message respons from the server in a pop-up, this was not sufficient because of two factors:

- A short text message would not be sufficient to communicate all potential problems with the exercise, when a fine-grained response with details on every failed test is desirable.
- The grading result would not necessarily be available immediately. In scenarios with a high server load, and many submission queued for grading, a grading result could in theory take several minutes to generate. In the meantime, the connection from Web-CAT to the server would have timed out, preventing any useful feedback to be returned.

With this is mind, there were two options to consider:

1. Altering the functionality of the Web-CAT plug-in to suit the requirements of JExercise.
2. Writing new functionality from scratch into the JExercise code base.

Ultimately, option 2 where chosen, writing the necessary code into the JExercise plug-in. Reasons for this choice include:

- The effort to write the functionality was determined to be quite small. The Java standard library includes classes to deal with the Zip file format, and contains also HTTP classes for network communication with HTTP servers. Eclipse contains a rich API for accessing project resources (student's Java files), and creating a valid multipart/form-data request to be dispatched with the HTTP classes is fairly straight-forward by following the requirements of the relevant RFCs [5] and [6].
- It would be easier to maintain and adapt the submission system to changing requirements in the future, since the code would be tailored to JExercises requirement. However, if the rewrite effort had been considerable, this point would not be reason enough alone.

The code for bundling the students' Java files into Zip files were created and provided by the JExercise author. At the same time, he also implemented most of the code to generate a multipart/form-data request. Left was finishing the multipart/form-data code, finding a

reasonable way to dispatch the zip file to Jexserver from within Eclipse, and display the grading results when they are ready, keeping in mind the aforementioned problem with potential long grading time.

6.1.2 Eclipse and the Eclipse Jobs API

To be able to add elements to the user interface (UI) of Eclipse and JExercise, you must program to the API of The Standard Widget Toolkit[16] (SWT). SWT is an open source toolkit for Java for constructing user interfaces, created as part of the Eclipse project. SWT is a API built on top of native user interface toolkits, which means that there are separate versions for Windows, Linux and Mac OS X, each version interfacing with a different underlying user interface toolkit. The benefit of this approach is that the resulting application user interface will look like any other application developed for the platform.

Using SWT, UI components can be added to JExercise to let the student invoke the zip and submission procedure. While this can be implemented using traditional techniques; adding a button to the interface and do all the work in a event handler for the a button click event, this is a not a good solution for usability reasons. There is no telling exactly how long the entire operation could take, especially if we want to wait for grading results, and the the user will be blocked from perform any other work in Eclipse while it is sending the submission and waiting for the response.

A way to solve this usability problem is to use a new thread of execution to perform the zipping of the Java resources, submitting the resulting zip file and waiting for the grading results. If properly implemented, the operations would not block the user interface and would let the user continue working simultaneously. Using threading introduces a new set of problems related to resource access contention: several threads simultaneously accessing a data structure can either result in invalid program states (when threads are not properly synchronized to avoid this) or deadlock situations, where two (or more) threads each hold one set of locks while waiting for locks held by the other thread.

The solution is of course to synchronize access to resources shared between threads, while make sure deadlock situations can not occur. In this case, the shared resources are the Java resources and the project state in Eclipse. Eclipse automatically locks these resources when accessed by any thread, so the program should never reach an invalid state, but deadlocked situations are still possible.

16 http://www.eclipse.org/swt/

Eclipse has solved this problem by providing a new Jobs API in Eclipse 3.0 and later versions. The Eclipse Jobs API is specifically designed [7] to implement:

- Resource locking to avoid concurrent modification of the same resources
- Support for perform work in the background (not blocking the user interface)
- Support for determining which jobs can be run simultaneously (based on what locks the jobs needs).
- A special lock interface with support for deadlock detection and recovery.

Eclipse Jobs are implemented in Eclipse by executing the jobs in separate threads, though the user of the API does not necessarily notice this.

6.1.3 Implementing the Submission Feature

In the event handler for the submit button (illustration 6 on page 26), code is added to create a new Eclipse Job. The job invokes the submission zip procedure and then invokes the procedure to upload the submission to Jexserver. But how does the JExercise plug-in know where to upload the submission to? One solution would be to program the address directly into the program code ("hard coding"), but this make it more difficult to change later (require software updates) and it would not be possible to have different destinations for different exercises.

Instead, the submit location URL was added to the .jex file specification. A .jex file always accompanies an exercise and contains details about code requirements, tests and scores [1]. In order to be be able to deploy the functionality quickly, the URL was placed in the unused comment field of the .jex files. This meant that the visual editor for creating .jex files already could be used to insert these URLs.

A way to identify students was also needed, since Jexserver associates submissions with user accounts on the server. If the exercises distributed to each student were the same file, this identification information could not be added to the comment in the .jex file. The solution to this problem was to have the student type in the username and password in a preferences dialog that already existed in JExercise. The username and password would then be added as URL parameters to the original URL read from the .jex file.

The submission job assembles the complete URL after having created the zip file, and uploads the file to the server using a HTTP POST method with multipart/form-data-encoding, a common way of posting data to web servers [5].

The job could wait until the grading processing on the server were complete before showing the results to the user. On way of achieving this is to repeatedly connect[17] to the server to check the grading status, until the results are ready[18]. A more flexible approach were implemented where this logic can more easily be altered on the server, instead of requiring updates to the JExercise plug-in.

6.1.4 Results View

As soon as the submission is delivered to the server, the server returns a HTTP URL in the response to the upload. This URL points to a valid and dynamically generated page on the server. The HTML display component of the JExercise view is then instructed to display this URL. If the results are ready when JExercise loads the page, the embedded web page will show the results immediately. If not, a simple page will be shown explaining that the results are not ready yet (see illustration 7 on page 27). This page uses embedded JavaScript to reload the page after a set amount a time (a few seconds). Currently, this is a fixed set of seconds, but server-side logic could be implemented to estimate the number of seconds left based on the server load and the number of submission queued. This process will be repeated until the results are ready to be displayed (illustration 8).

Since the embedded web browser / HTML view of Eclipse can use technologies such as Cascading Style Sheets (CSS) and JavaScript[19], these are used to generate a tree view similar to the exercise requirement tree view already in JExercise. Style sheet instructions are used to show and hide nodes, which is implemented by nesting HTML *div* elements. JavaScript is then used to display and/or hide elements when the (+)/(-) icons are used to expand or collapse the tree. The logical tree structure is generated based on the hierarchial structure of exercise requirements in the .jex file.

Also note that while we are currently discussing JExercise, the result view is actually

17 With some delay between connections.
18 This is necessary because web clients, servers and proxies might have all have limits to how long a request can wait for a response from the server.
19 On Windows platforms, Eclipse uses embedded Microsoft Internet Explorer controls, and Mozilla browsers are embedded on other platforms. This provides support for many modern technologies within the browser component in Eclipse.

generated by Jexserver, but the component does logically belong to JExercise, at least seen from the user interface perspective.

6.1.5 Auto-submit Functionality

One the research questions involves collecting data for usage analysis. Due to the effectiveness of the implemented submit functionality, a very powerful data collection tool could be implemented with fairly little effort.

The existing submit function invoked by the user pressing the “Submit >>>” button in the user interface was hooked up to the calculate score button as well. The calculate score button is used by the student to run the associated tests locally on the computer and get a locally computed score for the exercise. This function is used frequently by students while working on the exercise[20].

After the change to this button event handler, when the student runs the tests to check the score, the submission function is simultaneously started in the background. The Eclipse Jobs API provides a way to specify that a job should run completely in the background, and this is exploited here to re-use the former developed submission job without interfering with the student's interaction with JExercise. The grading results of these automatic submissions are never displayed to the student.

The consequence of this change is that a large number of submissions will be sent to Jexserver (and graded by Jexserver) per student, spread over time. To avoid negative technical consequences (of performance), JExercise also tells Jexserver whether a submission is auto-submitted, and Jexserver is modified to always prioritize manually uploaded submissions where the student may actively wait for grading results.

See the results and evaluation chapter for a more in-depth discussion of this feature.

6.2 Jexserver

The following sub chapters details changes made to Jexserver as part of this master's thesis.

20 See the student survey chapter

6.2.1 Database Replacement

From the Jexserver project report [2]: *"The database was sufficient for this project, but SQLite is not designed to be very efficient with concurrent processes accessing the database (especially when writing to the database, due to very coarse locking mechanisms), and testing suggests that moving to a database such as PostgreSQL or MySQL could increase performance when multiple users are accessing the system."*

One of the first tasks that were completed when preparing to use Jexserver in TDT4100 was to replace the database with MySQL[21]. MySQL is an open-source database software available for several operating systems, including Linux and Microsoft Windows, for no charge. The most pressing reason for replacing the database was because to work around a specific locking problem with the SQLite database; the database had to be opened/closed repeatedly to avoid hanging processes[22], and this incurred a noticable performance hit.

The database code in the original Jexserver project was localized to a few core classes mapped against database structures, so rewriting to use MySQL instead of SQLite was not a particularly large effort. Jexserver used Java's Database Connectivity API (JDBC) from the start, which meant that switching database could be done without changing the Java code, as the API is the same whichever back-end is used. Unfortunately, although both databases are based on SQL, actual SQL code is seldom portable because often vendor-specific functions must be used to perform some tasks. Still, the necessary changes to the SQL were quite small.

6.2.2 Receiving Submission from JExercise

To accommodate the new automatic submit feature in JExercise, a new Java Servlet were designed to receive the submissions. To facilitate code re-use, the communication protocol between JExercise and Jexserver were based on HTTP and HTTP multipart/form-data posts [5] [6]. In practical terms, this means that the submit HTTP request from JExercise were designed to be compatible with a file upload to a HTML form as used in the original Jexserver web interface [2]. Thus, the code to receive the zip file would mostly be the same as in the original Jexserver. The extensions needed were to allow the client to send username and password with the request instead of using a session ID cookie as with the existing web interface. Also, the server checks a parameter to determine if the submissions was manually or automatically uploaded from JExercise.

21 http://www.mysql.com/about/
22 SQlite was never designed specifically to be accessed by multiple processes.

Some changes to the Jexserver core was also needed:

- The database must be able to store whether the submission was sent automatically or not, because we want to able to separate these submission. The table schema for the submission table was extended with a *autosubmitted* field.
- A *priority* field was added to submissions in the database. Auto-submitted submissions are given a lower priority than manual submissions. While the *autosubmitted* field could be used for this purpose, adding a separate *priority* field allows for easy addition of more priority levels in the future.
- The dispatcher was updated to respected the new *priority* field. Submissions with higher priority are always sent to Evalclients before lower priority levels are considered.

To accommodate easy retrieval of data useful for usage statistical analysis, a new table was added to the database schema from [2]. This table includes all individual results for all point-awarding requirements in a submission. This allows for queries based on individual requirement results. An example of such query is "retrieve the number of students failing test 9" for a specific exercise. Having this data readily accessible in the database also allows for real-time visualization of submission results for an exercise or an exam. The alternative would be to examine the XML reports[23] generated by Evalclients, which would require more processing time and more complex code to retrieve results.

6.2.3 Installation on a Dedicated Server

As part of moving Jexserver from a prototype state to a production-ready system, it was installed on a dedicated Linux server procured from the IT department at the institute of informatics at NTNU. A dedicated server setup for Jexserver was needed since the server would be actively used by students in TDT4100, and performance tests run against the system should be isolated from other systems. Co-hosting the system on a shared server would make performance testing and monitoring both difficult and error-prone.

After the completion of this project, the software will most likely be installed on a Linux or Solaris server run by the IT department. Using Linux for the dedicated server was thus a natural choice because of the price (free), and similarity between Linux and Solaris; the latter

23 For more complex analysis, accessing the XML-based grading reports is still useful since these contain more data on failed tests such as stack traces and error messages.

is a Unix operating system while the former is a Unix-like[24] operating system. Since the server software is developed on Microsoft Windows system, a side effect of this choice is that the software will automatically receive active testing on both Windows and Linux platforms, making sure the software is platform agnostic.

In the author's experience, testing a software on multiple platforms is a good strategy even if one targets just one platforms, because differences in operating systems can reveal subtle bugs in the software that could otherwise go undetected during testing and appear infrequently in a production setting making the problem difficult to track down.

6.2.4 Performance Problems

After running some tests on the Linux server, it became apparent that it suffered from performance problems while grading exercises. The performance problem did not occur when running the software on Windows, the development platform.

Investigating into the issue revealed that the performance problem was not due to system resources, not the actual grading process. Benchmarking showed the the actual grading performed as expected, utilizing the system CPU fully. The performance problem was related to a delay between each graded submission.

Ultimately, the problem was found to be due to a subtle problem in the communication between the Jexserver Dispatcher and the Evalclients. When Evalclients reported results back to Jexserver, they did not flush the sockets after writing all the data. Because the operating system buffers the socket data to improve performance, the OS waited for more data to be written. As there were no more data, the process just waited until a timeout caused the OS to send the data in the buffer.

This a good example of the problem with platform-independent software development. Even though the API is platform agnostic, as is the case with Java itself and most Java libraries, the semantics of function calls may differ slightly. This is especially the case with functions using operating system resources such as sockets, file access and process management.

In this case, the fault was with the Evalclient implementation for not explicitly flushing the socket buffers. Flushing the buffers after Evalclient had finished sending data resolved the performance problem completely.

24 While mostly compatible with Unix and POSIX standards, Linux has never been officially certified against these standards.

6.2.5 Problem With Use of AWT in Exercises

While all exercises created by course staff for TDT4100 eventually graded successfully in Jexserver, there were some specific issues regarding AWT that required changes to the Jexserver implementation. The underlying reason for the problems is that the tests are run in a sandbox Java environment to avoid security problems [2].

A few exercises were designed around the use of some AWT classes, but including the *java.awt.** classes triggered security exceptions in Java. The actual Java grading process runs under a Java security policy restricting what the code can do. To allow the AWT classes to be used in tests and student code, the following permissions had to be added to the policy file:

```
permission java.lang.RuntimePermission "shutdownHooks"
permission java.lang.RuntimePermission "modifyThread"
permission java.util.PropertyPermission "user.dir", "read";
permission java.util.PropertyPermission "user.home", "read";
permission java.io.FilePermission "{home}", "read";
```

These permissions were identified by running the exercise grading process, noticing the exception message when Jexserver refused to grade a requirement due to a security exception, add a permission allowing this operations, and restart. This cycle was repeated until all the tests graded perfectly.

Additionally, the documentation for these permissions were consulted to make sure the permissions did not pose any potential security risks. Allowing read access to the user's home directory for instance is not a problem when the Evalclient is run under a dedicated unprivileged user account on the operating system.

6.2.6 Accessing File Resources from Tests

A few exercises required that the JUnit tests was able to read files from the file system; files that are distributed with the exercise as part of the distributable zip file. Example of such files are text files with data used by tests and student code.

The files were accessed using relative paths (paths relative to the project root). This did not work very well when the exercises were graded in Jexserver, since the current working directory was not set to the location of the unzipped exercise.

And although Java can access the path to the current working directory through system

properties, it has not API to change this during runtime. A solution to this problem was to create a small utility class containing a function that could be included in exercises that needed to access local files bundled with the exercise:

```
public String getExerciseFilePath(String relativePath) {
    String basePath = System.getProperty("jex.exercisepath");
    if (basePath == null) {
        return relativePath;
    }
    return new File(basePath, relativePath).getPath();
}
```

Jexserver was modified to save the location of the exercise into a Java system property *jex.exercisepath.* The *getExerciseFilePath* method from the utility class could then be used to resolve a relative path. If the tests were run from within JExercise, the *jex.exercisepath* system property would not be set, and existing behavior would be kept by just returning the same relative path. When run on Jexserver, an absolute path would be returned by resolving the relative path against the base exercise path.

This was the only *exercise modification* needed to grade any exercise created for TDT4100.

6.2.7 Students Delivering non-zip Files

A lesser problem was that a couple of submissions were not proper zip files. The existing Jexserver implementation did not handle this very well and caused the submission to fail the grading process. The Jexserver core was updated to handle problems occurring before start of grading much better. A general field for high-level error messages was added to the grading XML report. The web interface was also updated to check for this error message, so subsequent invalid submissions will get a proper error message displayed on the grading result page.

This was of course only a problem with manually uploaded files, since the new submit functionality in JExercise creates proper zip files automatically.

6.3 it's learning Integration

The following sub chapters discusses technical solutions and implementation details related to the integration work with it's learning.

6.3.1 Tool Support in it's learning

As documentation of tool support in it's learning is not publicly available, an investigation of existing tool support was necessary. A natural cause of action was to use the full-featured demonstration tool provided by The IMS Global Learning Consortium, the Proxy tool test harness, and try to integrate this tool with it's learning.

The proxy tool test harness is distributed as a Java web application, requiring a Java Servlet container to host it. This test harness provides a demo tool with can be used to test Learning Management Systems. In order to use this tool with it's learning, the tool must be published on a web server accessible from both it's learning and the user's computer (since the user will communicate with the tool once the session has been negotiated between the tool and it's learning).

Apache Tomcat[25] was chosen to host the demo tool, since it can function both as a web server and a Servlet container, reducing the need for other dependencies. Apache Tomcat is not the reference implementation of the Java Servlet specification [13], but is a well established application available free of charge. Another reason to choose this platform is that it was the only Servlet platform mentioned by name in the documentation included with the test harness.

Before trying to connect the tool to it's learning, the tool test harness (also provided by IMS Global Learning Consortium, but not to be confused with the proxy tool test harness) was also installed on the Tomcat server. To validate the setup of the tool, it was loaded into the tool test harness. After some configuration, the systems was successfully integrated; the tool test harness correctly launched the tool and redirected it's user interface. Upon tool completion, a score was reported back to the tool test harness.

With a working tool published on a web server, the integration capabilities of it's learning could now be probed.

6.3.2 Deployment Descriptor Problems

The tool deployment descriptor that was used with the tool test harness was uploaded to it's learning via the available "add proxy tool" web interface. Unfortunately, this did not work as expected and resulted in the following error message:

"Error! The uploaded package is not valid or cannot be read: There is an error in the

25 http://tomcat.apache.org/

XML document..”

This was unexpected since the uploaded deployment descriptor was the only available example apart from the information in the guidelines document [10],

This is the original deployment descriptor from the IMS proxy tool test harness package, ConceptTutorDeploymentDescriptor.xml[26]:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProxyToolDeployment xmlns="http://www.imsglobal.org/xsd/imsti_ptdd_v1p0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <DeploymentProfile>
        <CoreSettings>
            <ProxyTool>
                <ProxyToolName>ConceptTutor</ProxyToolName>
                <ProxyToolVersion>
                    <major>1</major>
                    <minor>0</minor>
                </ProxyToolVersion>
                <IMSTIVersion>
                    <major>1</major>
                    <minor>0</minor>
                </IMSTIVersion>
                <ProxyToolDescription>ConceptTutor provides just in time, just
enough information for basic comprehension of a concept.</ProxyToolDescription>
            </ProxyTool>
            <TIR>
                <LaunchService>
                    <ServiceName>ConcepTutor launch service</ServiceName>
                    <ServiceLocation>http://localhost:8080/ti-
tool/services/TIRLaunchServiceSyncSoapPort</ServiceLocation>
                    <ServiceProfileName>MinimalOutcomeProfile</ServiceProfileName>
                </LaunchService>
            </TIR>
        </CoreSettings>
        <SupportedLaunchRoles>
            <Role>Student</Role>
            <Role>Instructor</Role>
        </SupportedLaunchRoles>
        <ContextualProfile Required="true"
ContextualProfileType="SimpleUserProfile"/>
        <ContextualProfile Required="true"
ContextualProfileType="AcclipContextProfile"/>
        <ContextualProfile Required="true"
ContextualProfileType="DeliveryContextProfile"/>
        <SecurityProfile Required="true" Preference="0"
SecurityProfileType="SharedSecretProfile"
xmlns:dd="http://www.imsglobal.org/xsd/imsti_ptdd_v1p0"
xsi:type="dd:SharedSecretSecurityProfile.Type">
            <SharedSecret>D/utj51IkvfmY05yD/W6pzCCLFNqITliVbs5+X9rtqxwW4N9OXXA2IrVl
baJ/iYhwBvqRBQhtSEZ81dd0LWdlg==</SharedSecret>
            <DigestedMessageElement>syncRequestHeaderInfoDType/messageIdentifier</D
igestedMessageElement>
        </SecurityProfile>
        <OutcomeProfile Required="true" Preference="0"
OutcomeProfileType="MinimalOutcomeProfile"/>
        <OutcomeProfile Required="true" Preference="0"
OutcomeProfileType="SimpleOutcomeProfile"/>
```

26 Before deployment, the service location was altered to point at the correct web URL.

```
        <ToolSettings>
            <SettingsDefinition>
                <documentation><p xmlns="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
                    A ConceptTutor requires that the launch a tool settings
instance that
                    satisfies the schema listed in-line below.  In particular, a
launch will succeed only
                    if the resourceIdentifier resolves to a resource known to the
ConceptTutor instance.
                    The instance contains a sample resourceIdentifier that would be
overridden in any
                    particular proxy tool instance.</p>
                </documentation>
                <schema>
                     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        xmlns="http://engage.doit.wisc.edu/schemas/ConceptTutor_v1p
0"
                        targetNamespace="http://engage.doit.wisc.edu/schemas/Concep
tTutor_v1p0">

                        <xsd:complexType name="resourceIdentifierType">
                            <xsd:sequence>
                                <xsd:element name="id">
                                    <xsd:simpleType>
                                        <xsd:restriction base="xsd:string">
                                            <xsd:minLength value="1"/>
                                            <xsd:maxLength value="256"/>
                                        </xsd:restriction>
                                    </xsd:simpleType>
                                </xsd:element>
                                <xsd:element name="instance">
                                    <xsd:annotation>
                                        <xsd:documentation>Setting instance to base
will retrieve the resource from the Fedora repository.
                                            Setting instance to test will by-pass
the Fedora repository and deliver a test page.
                                        </xsd:documentation>
                                    </xsd:annotation>
                                    <xsd:simpleType>
                                        <xsd:restriction base="xsd:string">
                                            <xsd:enumeration value="base"/>
                                            <xsd:enumeration value="test"/>
                                        </xsd:restriction>
                                    </xsd:simpleType>
                                </xsd:element>
                            </xsd:sequence>
                        </xsd:complexType>

                         <xsd:element name="resourceIdentifier"
type="resourceIdentifierType"/>
                     </xsd:schema>
                </schema>
                <instance>
                    <resourceIdentifier
xmlns="http://engage.doit.wisc.edu/schemas/ConceptTutor_v1p0"><id>active/tutor.html
</id><instance>base</instance></resourceIdentifier>
                </instance>
            </SettingsDefinition>
        </ToolSettings>
    </DeploymentProfile>
</ProxyToolDeployment>
```

Considerable time was used checking with the tool guidelines, and trying to strip elements from the deployment descriptor to probe for features left out by it's learning. All tests resulting in failure.

The company behind it's learning was then contacted and asked for documentation specific to their implementation. While such documentation did not exist, they were kind enough to send a deployment descriptor XML file known to work with it's learning. The received descriptor was by the looks of it identical with the reference descriptor, apart from another textual description, but this descriptor was accepted by it's learning.

Output from the UNIX program diff shows us the differences between the two descriptors.

```
<                 <ProxyToolDescription>ConceptTutor provides just in time, just
enough information for basic comprehension of a concept.</ProxyToolDescription>
---
>                 <ProxyToolDescription>Simple example of a proxytool that delivers
a test for a student, and edit-option for teacher.</ProxyToolDescription>
21c21
<                     <ServiceLocation>http://localhost:8080/ti-
tool/services/TIRLaunchServiceSyncSoapPort</ServiceLocation>
---
>
<ServiceLocation>http://localhost/tests/WebServiceProxy/WebServiceProxy.asmx</Servi
ceLocation>
51c51,53
<
targetNamespace="http://engage.doit.wisc.edu/schemas/ConceptTutor_v1p0">
---
>
targetNamespace="http://engage.doit.wisc.edu/schemas/ConceptTutor_v1p0"
>
          elementFormDefault="qualified"
>                                                  >
```

The only technical difference seems to be the addition of elementFormDefault="qualified" to the <xsd:schema> element in the descriptor accepted by it's learning. By adding this XML attribute to the reference descriptor, it was also accepted by it's learning.

The attribute *elementFormDefault* is specified by the XML Schema standard [14]. Its significance is whether instance documents are allowed to contain unqualified elements, or all elements must be qualified. XML Schema is used by deployment descriptors to specify validation rules and requirements for tool settings. The fact that this attribute should change whether it's learning accepts the descriptor or not is peculiar, and can best be described as an implementation-specific requirement. The IMS guidelines does not reference this attribute.

That being said, the IMS Tools Interoperability Guidelines does seem weak as a specification

or standard, and the use of the word "guidelines" in the document name does seem fitting. Although the guideline document is rich in diagrams, use cases, and specifies allowable elements in data structures (via an XML Schema), it does lack concrete examples, and the semantics of most data fields are left unspecified. That implementors must resort to some guesswork to implement the guidelines goes a long way to explain the encountered problem.

6.3.3 Conformance to Tool Guidelines

With it's learning's acceptance of the deployment descriptor, it is now possible to check how it's learning integrates tool, and what information is sent with a launch request to the tool. In the absence of documentation this knowledge must be retrieved through experimentation.

The tool support in it's learning is based on defining the proxy tool in the interface, and then uploading the deployment descriptor to this tool definition. After the tool definition has been saved, it is then possible to add *instances* of this tool definition to content folders in courses, on the same level as adding documents, sub-folders, questionnaires or other learning content.

When adding a tool *instance* to a content folder, you must also configure the settings for this tool instance, as defined in the deployment descriptor. This makes it possible for a learning tool to have just one deployment descriptor, and a corresponding tool definition in it's learning, but still be usable in different contexts. For instance, the tool can specify that exercise number is a required option, thus forcing tool instances in it's learning to choose an exercise number. In this way, the same tool definition can be used with all exercises in the course.

By experimentation, it's learning was found to have another requirement that is not described in the IMS Tools Interoperability Guidelines. Even after creating the tool definition and adding an instance to a folder, with valid settings, there was no way in the user interface for the student to launch the tool; it's learning merely displayed a page with information about the tool. it's learning has introduced a concept called "operations" with its tool support. After creating the tool definition, you must then add one or more operations to this tool definition. These operations has titles and some associated settings such as launch roles. By adding an operation, you are effectively adding a launch link (with the same title) to all tool instances. This allows tool instances to have more than one launch operation.

With tool launch mechanisms in place in it's learning, the demo tool is launched. The descriptor is set to a fake HTTP server, redirecting the traffic to the real Tomcat server

hosting the tool. This allows monitoring of the traffic between the LMS and the tool. Since XML-based protocols are being used, the communication is quite easy to read directly. Analysis of the traffic shows that the following information is sent by it's learning to the tool:

- Required fields such as IMS Tools Interoperability version.
- A launch profile with a unique launch identifier
- A context profile, giving the name of the content element containing the tool instance in it's learning.
- A user profile, including a numerical user ID, first and last names, username, course code, and locale settings.
- The tool settings specified when adding the tool instance to a content folder.

When the tool returns a URL based on the launch request, it's learning loads this URL in the main content frame[27] of the web interface.

6.3.4 User Authentication

The tool integration lets us use the user account system already present in the LMS. The following is an XML fragment sent as part of the SOAP call when launching the Jexserver tool from it's learning. The fragment is from an authentic call, extracted by monitoring web traffic, but actual user id, user name and name has been altered for reasons of privacy.

```
<UserProfile xsi:type="SimpleUserProfile.Type">
<IMSSourcedid>
<SourceName>VLE User</SourceName>
<SourceID>9173</SourceID>
</IMSSourcedid>
<Userid>9173</Userid>
<FirstName>John</FirstName>
<LastName>Doe</LastName>
<Ancillary>SyncSource:NO-FS,Username:johndoe,Language:en-
GB,FormatCulture:nb-NO,TimeZoneOffset:120,TimeFormat:24,CourseCode:TDT4100
</Ancillary>
</UserProfile>
```

As can be seen from the listing, it's learning tells us, among other things, the username of the user invoking the tool and a unique number identifying the student in it's learning. This

27 The web user interface of it's learning is based on HTML frames.

information can be used by Jexserver to implicitly authenticate a student when he/she accesses the tool. But this implicit authentication requires that we also trust the information that is sent along with the launch request.

Since the tool interface consists of a web interface running on a server hosting the tool, and the usage of the tool may involve more than one web pages, the tool must keep information about several students accessing the tool apart. One way of doing this is to use sessions, a common technique in website design, where we store user details in a session that is preserved while the user is accessing the site. This is also how Jexserver originally works [2].

Jexserver, which stores all submissions by students on the server [2], must still have a way to associate submissions with specific students. The original student database described and developed in [2] is therefore still needed. but the information from it's learning can be used to create new student entries on demand, and "log in" students as the access the tool from it's learning.

6.3.5 Reporting Scores

Via the Outcome Service defined by Tool Interoperability Guidelines [10], which is implemented by it's learning, Jexserver is able to report back scores to it's learning when a students submission is graded. The Outcome Service supports a simple score reporting where a single number or text string is returned to the LMS. Jexserver grades submissions by assigning a numeric score, so the simple reporting mechanism is sufficient to support Jexserver.

6.3.6 Tomcat Web Server and Servlet Container

By the same reasoning as in section 6.3.1 (page 49), and since Jexserver already runs on Tomcat [2], the Apache Tomcat server was chosen to host the Jexserver tool. This also made it possible to re-use components developed earlier for Jexserver.

6.3.7 Apache Axis and SOAP Web Services

While Apache Tomcat is a full-blown Java Servlet container and can also function as a general-purpose web-server, it does not natively contain support for SOAP web services.

Apache Axis is SOAP implementation written in Java using Java Servlet technology. This

makes it possible to run Axis from within Tomcat. Axis comes as a pre-configured web-app which can be deployed directly into compatible web-app containers such as Tomcat. You are then able to, via configuration files, add web services to the Axis web-app which would then collaborate with another web-app (Jexserver) to do the actual work. There is another way to deploy Axis in Tomcat. Since Apache Axis is servlet-based, it can also be integrated into any other web-app by copying the necessary files from the Axis web-app and defining the servlet mapping in the configuration file for the web-app. From the Axis installation guide [18]: "If you are experienced in web application development, and especially if you wish to add web services to an existing or complex webapp, you can take an alternate approach to running Axis. Instead of adding your classes to the Axis webapp, you can add Axis to your application." This was the method chosen for Jexserver since it then avoids complexities caused by either inter-web-app communication, or by trying to embed Jexserver into the Axis framework.

6.3.8 Tool Web Services

To be able to fulfill a two-way integration with it's learning, two components are needed:

- A server implementation of the IMS Tool Launch Service. it's learning acts as the client here, invoking the service when a user starts the tool.
- A client implementation of the IMS Tool Outcome Service. Jexserver is the client and needs to contact the server implementation running in it's learning.

Axis comes bundled with a tool, *WSDL2Java*, which simplifies writing SOAP services and clients. By using the two WSDL[28] files provided by IMS Global Learning Consortium, WSDL2Java can automatically create Java classes to perform the necessary serializing and de-serializing from the XML format of SOAP, to Java objects.

WSDL2Java was used to created both a client library and a server stub implementation. The client library is automatically ready to be used, but the server sub implementation is best described as a "skeleton". All required classes are created, along with all methods defined in the service, but the method bodies are empty. This is where the actual logic of the service must be written. To be able to re-create the server stub from updated WSDL-files if necessary, without overwriting the service logic, the implementation was not written directly into the body of the service methods. Instead, another separate class was created, containing similar

28 WSDL is an XML standard for describing web services

methods with the same method signature. The service logic were implemented in these methods, with just a forwarding call placed into the server stub methods. This creates a clear separation between automatically and manually created code and should ease maintenance of the code.

6.3.9 Customized Exercise Zip Download

With the current situation in the course TDT4100, exercise zip files are distributed on it's learning; all students access and download the same zip file. Reviewing the earlier section about JExercise improvements, the JExercise client was extended with functionality to submit the student's work directly to Jexserver. This function requires that the student has configured their Jexserver username and password in the JExercise preferences dialog.

The problem with the new JExercise--it's learning integration is that the Jexserver username and password is no longer used. While there is an implicitly created user in the Jexserver user database, this user has no password and can only log in indirectly via it's learning. Also, the username is not known to the student because it is based on a numeric student id from it's learning.

One way to solve this problem is to create a random password for the student when they access Jexserver from it's learning for the first time, and display on the web page the username and password they need to enter into JExercise. A similar solution is to let the students associate a second set of username/password with the Jexserver account.

The chosen solution does not only solve the problem, but should also improve the user experience. By creating a personalized zip file for each student, the submit URL can contain all the information needed to identify both the student, the course and the exercise. The most basic way of achieving this would be to embed the user name, course id and exercise id directly into the submit URL (as URL parameters). While the internal student id may not be sensitive personal information, there is no need to disclose this information in the submit URL and send it over the Internet. A better way is to generate a unique identifier in Jexserver each time a student downloads an exercise zip file and associate this identifier with the student and the exercise on the server side[29].

This implies that the exercises cannot be hosted on it's learning any longer. Jexserver needs to manage the distribution of zip files to students. The administrative interface of Jexserver [2]

29 The unique identifiers are stored in the Jexserver database.

has been extended to allow course staff to upload the exercise zip file for distribution. For students, the exercise page shown in illustration 11 has a "download zip file for the exercise". When the student clicks this link, a new unique identifier is added to the database as previously suggested, and a submission URL is created containing the upload destination and the identifier as a unique parameter. Here is an example URL[30], the unique identifier is included as the *exerciseref* parameter:

```
http://jexserver.idi.ntnu.no:80/jexweb/submit_api.do?
exerciseref=37619830-54c1-467d-b9a8-eb303f74c8a9eref
```

A copy of the original zip file is created on the server, and this URL is added to a special file within this zip archive, this file is based on the name and location of the .jex file within the archive. If the .jex file is called jex/exercise8.jex, the submit URL is added to the archive under the file name jex/exercise8.jex.submit[31]. The modified zip file is then sent to the user.

Since the zip file contains a submit URL tailored to the student, it should not be shared between students, or exercise results will be credited to the wrong student. A warning on the download page notifies the student about this.

6.3.10 Enhancing JExercise Submission Function

The submit function developed for JExercise was extended with additional functionality for the integration with it's learning. It now first tries to find the submit URL by looking for a submit file corresponding to the active .jex file. JExercise has an API function to retrieve the active .jex file[32]. The resource path to the submit URL file is generated by taking the path to the .jex file and appending ".submit". The Eclipse resource API is then used to read the submit URL from the file if it exists. If the submit URL could not be read, the old behavior is then tried. The JExercise plug-in can be used with either methods, which makes Jexserver upload possible still without using the it's learning integration.

6.3.11 Receiving Submissions from JExercise

The Java Servlet responsible for receiving and handling submissions from JExercise was extended to also look for the new *exerciseref* URL parameter. When this is present, the Servlet looks up the student and exercise from the database, given a valid *exerciseref*.

30 The URL is shown on two lines here, but is in reality one continuous line without any newlines
31 A regular text file containing the submit URL in a single line.
32 There could in theory be more than one .jex file in the same Eclipse project.

Additionally, instead of returning the normal result page URL to the JExercise client, a URL pointing to a custom created results page for it's learning integration is returned. This allows tailoring the information displayed to include details specific usage together with it's learning.

6.3.12 Reporting Results Back to it's learning

When the student is satisfied with the achieved score, he/she can report this result back to it's learning.

While the IMS Tool Guidelines does not specify a way to return the student automatically to it's learning, a trick specific to the communication with it's learning makes it possible to return the student to the initial web page.

Here is an XML fragment from a launch request from it's learning:

```
<DeliveryContextProfile>
<IMSSourcedid>
<SourceName>Learning Element
</SourceName>
<SourceID>1056543</SourceID>
</IMSSourcedid><Name>Exercise 8</Name>
<ContextDescription />
<Ancillary />
</DeliveryContextProfile>
```

The interesting part here is the source id. Note that the IMS Tools Interoperability Guidelines does not specify exactly what these elements represents. From the specification: *"DeliveryContextProfile - this optional element provides information about the delivery context within which the Proxy Tool launch was initiated, e.g., a course or a section. Thus, this includes an IMS SourceId as well as a name, description, and an arbitrary number of ancillary (string) elements."* [10].

The Source ID is clearly related to the launch element. By further investigation, it was found that this source ID is reflected in the URL for the page launching the tool:

```
https://www.itslearning.com/ProxyTools/ProxyElementViewer.aspx?
ProxyElementID=1056543
```

The number 1056432 in the URL is identical to the SourceID from the launch request. This

means that we can reconstruct the URL to page preceding page by appending the string “https://www.itslearning.com/ProxyTools/ProxyElementViewer.aspx?ProxyElementID=” with the SourceID. The reconstructed URL is then sent back to the web browser with instructions to load this page. This is unspecified behavior and this specific implementation will not work with other Learning Management Systems.

7 Survey and Performance Testing

At the end of the semester, a student survey was performed amongst student in the programming course TDT41000. The survey included a questionnaire and required students to submit their answer to a specific earlier issued course exercise.

7.1 Motivation

The motivating factors for issuing this survey was:

- Too few students used the system to be able to say anything about performance.
- Likewise, too little data was available to conclude anything about JExercise and Jexserver usage.
- A test data set was needed for running an exam simulation, proving adequate Jexserver performance.

Possible reasons why few students submitted to Jexserver include:

- Jexserver grading was not used immediately at the start of the course. It was thought best to complete the submission feature in JExercise first. Because of the time required to implement and test this feature, the first few exercises were completed by students without knowing about Jexserver.
- After completion of the submission function, the JExercise plug-in needed to be updated in order to be able to submit automatically to Jexserver. At this time, all students had already installed the original version and were not necessarily motivated to go through the hassle of updating.
- It was never mandatory to submit exercises to Jexserver for grading. The actual

grading were performed for teaching assistants as before, but students were encouraged to also deliver submissions to Jexserver.

- Some students never knew that it was possible to deliver exercises to Jexserver. This was uncovered by the student survey (see discussion in section 8.6.1on page 72).

7.2 Student Survey

Partly because of few submissions to Jexserver during the semester, a survey was conducted among students taking the class TDT4100 towards the end of the semester. Participating students had too fill out a questionnaire (see appendix C) and also submit their answer to exercise 8 via upload to Jexserver. A special upload form were created to allow students without existing users accounts on Jexserver to quickly upload their submission.

To encourage participation in this survey, a gift card worth NOK 2500 was promised as award to one randomly chosen participant after the completion of the survey.

A discussion of the survey results is provided in section 8.6.

7.3 Performance Testing

Research question RQ4 concerning system performance during an exam led to the performance requirement in 4.2.2.1 (page 22).

To be able to test the performance requirement and answer the research question, a performance test must be employed. As a substitute for evaluating the performance on a real exam, which was not an option at this time, a simulated exam is run by computers.

7.3.1 Test Data Set

The test data set consists of 50 students submissions collected as part of the student survey. The survey got 56 responses in total, but of these, only 50 respondents managed to upload the required submission. The submissions were their answers to the earlier exercise 8, randomly chosen to be used as the 'exam' for the test. Using exercises instead of actual exams is an acceptable substitute, since the complexity, in terms of code size and there grading cost, of an

exercise would normally not be less than for much shorter 4-hour exam.

Since the data set consists of only 50 submissions, and our goal is to run a performance simulation with 250 "students", we let each submission be submitted by 5 simulated students, thus reaching a total of 250 submissions[33].

7.3.2 Testing Strategy

While one could simulate an entire 4-hour exam, the relevant part of the exam is when the majority of the students submit their answers. At NTNU exams, students are often notified when there is 15 minutes left before the submission deadline. Most students also use the time available on an exam. This means that the majority of the submissions will occur during a quite limited time frame, and this is the basis for the requirement in 4.2.2.1, where the ability to receive and grade 250 submissions during 15 minutes is considered to be required performance.

The actual probability distribution for when students will deliver a submission during the last part of the exam is not known. The most straight-forward way to simulate this would be to use a uniform distribution, where statistically, all submissions would be spread out during the course of the 15 minutes. This would also be the most beneficial to the grading system as the load would be evenly spread out over time. It is instead quite possible that the number of submissions will increase as the deadline approaches, while possibly decreasing sharply right before the deadline. The effect is this would be that a larger portion of the submissions would be clumped together in a time span shorter than 15 minutes.

Since the student submission pattern is not known, and to avoid running the simulation with a overly optimistic scenario, we tighten the orignal requirement to:

- Jexserver must be able to receive 250 submissions within 7.5 minutes and should also give the student the resulting score within reasonable time. We defined reasonable time to be within 30 seconds.

The motivation here is to create a "realistic worst-case" scenario, and if Jexserver can handle this sufficiently well, the requirement in 4.2.2.1 will also be fulfilled.

33 That these numbers add up exactly is purely coincidental. The target number of 250 students was based on the number of students taking the course, and was determined before the number of submissions was known.

7.3.3 Test Setup and Execution

The test is performed against the production server described in the implementation chapter. A small program was created to represent an individual student, and the instances of the program is assigned one the 50 submissions each. A controlling program spawns 250 instances of this program at the start of the test, representing the 250 simulated students. The programs were designed to have minimal system overhead to avoid system resources being a liming factor in the test. This was verified during the course of the test, where processor utilization averaged 3% on the computer running the student simulations.

The behavior of the student simulation program is to sleep for a random amount of time between 0 and 7.5 minutes before uploading the submission to Jexserver. The time spent uploading the exercise and retrieving the acknowledgment of the upload is logged. The end result of running 250 instances of this program is getting a near-uniform distribution of submission over the 7.5 minute interval.

The student submission program uploads the submission to the Jexserver interface, simulating exactly the upload mechanism used by JExercise. This means that all processing overhead from the web server, database and related components is included in the performance test.

A custom monitoring system is created to log the following variables on the server:

1. Number of ungraded submission in queue for grading.
2. Average grading time of the last 10 graded exercises.
3. Average total waiting and grading time of the last 10 graded exercises.

The server-side logging system is based on rrdtool[34], an open source data logging and graphing system for time series data. The monitoring system collects the variables every 10 seconds and logs them to rrdtool databases.

Jexserver runs on a computer procured from the IT department at IDI, NTNU for the duration of the master's thesis. The main specifications for this computer are:

- Intel Pentium 4 CPU 2.40GHz.
- 512 MB RAM

34 http://oss.oetiker.ch/rrdtool/

The system does not run any other services but Jexserver. The active components of Jexserver are:

- Apache Tomcat Server running the web interface.
- MySQL server hosting the database.
- Jexserver Dispatcher, the controller for grading submissions.
- Jexserver Evalclient, a grading client connected to the Dispatcher.

After running a complete simulation with the aforementioned setup, another simulation is run with more processing power for grading exercises. Jexserver was from the start designed to be able utilize distributed processes to grade submissions, to provide for scalability when extra performance is needed [2]. A second Evalclient is run on a separate computer, a Intel Core 2 Due CPU clocked at 2.66 GHz, connected to the Dispatcher over Internet through a 20Mbps/1Mbps ASDL link. The submissions are quite small, averaging under 10 KiB, so the connection speed and transfer time of submissions between the Evalclient and Jexserver over the Internet is insignificant. The transfer time of a 10 KiB submission over a 20 Mbps link is as little as 4 milliseconds.

The results and evaluation chapter contains the results of the performance test, and how adding a second Evalclient affected performance.

8 Results and Evaluation

This chapter presents and discusses the results of this project. It also contains a discussion of the results from the student survey and details of the the performance test.

8.1 Completeness of Functionality

All functional requirements identified in chapter 4 were successfully implemented, as demonstrated in chapters 5 and 6. The non-function requirements were also successfully met, as discussed in chapter 5 and this chapter.

8.2 JExercise Improvements

JExercise was successfully extended with an integrated function to automatically assemble and deliver submission to Jexserver when the student chooses to submit an exercise. When used together with Jexserver and it's learning, this functionality is available without any configuration required by the student.

JExercise also received a function to automatically assemble and send submissions of work-in-progress while the student is working with the exercise. These submissions can be used to monitor overall student progress on exercises and discover problematic exercise parts, as well as analyzing students working patterns.

8.3 Jexserver

The Jexserver prototype implementation developed in autumn 2007 was further developed into a production-ready grading system. It was set up on a production server and successfully

tested during the spring semester 2008.

8.3.1 Configure a Dedicated Jexserver System

A dedicated Linux server was configured to host Jexserver. Since Jexserver was designed to be platform agnostic from the start [2], the effort to get in running on Linux[35] was quite small.

As discussed in the implementation chapter, Jexserver also received some improvements

8.3.2 Test-driven Jexserver Improvements

While using Jexserver to grade exercises during the 2008 spring semester of TDT4100, the following problems were found with Jexserver:

- Some exercises used classes from the the Java AWT library, but AWT initialization triggered some security exceptions. Some security permissions were given to the grading process to allow these exercises to be graded, after evaluating and concluding that they did not pose a security problem.
- A couple of exercises files bundled with the exercise accessed using relative paths needed a small patch to successfully grade in Jexserver.

Generally, all exercises were graded successfully in Jexserver. But there is still a possibility that some future exercises will want to use functionality currently forbidden by the grading process. One way to solve this problem is to add security permissions to the .jex file format. These permissions would be added to the Java security policy file under which the grading process is run. The .jex file is always loaded from the original exercise zip file contained in Jexserver and not from the student's submission, so students would not be able to exploit this.

8.4 Integration with it's learning

The following sub chapter discusses the results of the integration with it's learning.

35 The software was developed on the Microsoft Windows platform using open source technology.

8.4.1 Applicability of Tool Support

As discussed in the previous chapters; while the Tool Interoperability Guidelines defines a quite limited set of integration options, it is actually a perfect match for Jexserver. Jexserver is now able to:

- Let it's learning handle user authentication.
- Integrate its interface seamlessly into it's learning.
- Report submissions scores back to it's learning.

8.4.2 Results of Integration Work

While the Tool Interoperability Guidelines does not specify any way to directly integrate the tool interface with the user interface of the Learning Management System, the project still succeeded in creating a user interface and interaction that blurs the limits between it's learning and Jexserver.

Because of the absence of user interface integration possibilities in the guidelines document, the tool was implemented to specifically blend in with it's learning. This means that the tool is somewhat tied against it's learning, and that the Jexserver tool must be modified slightly to work with any other LMS.

The IMS Global Learning Consortium has announced its intention to create an abstract user interface description that can be used by tools to describe its user interface [15]. The LMS will then be able to render this user interface using native interface components, making the user interface itself truly portable against all Learning Management Systems supporting the standard. This work could possibly be included in a future 2.0 version of the specification.

8.4.3 Using the Reported Outcomes

As can be seen in illustration 11 on page 35, the outcome of the grading process is successfully reported back to it's learning. The intention was to report the results to it's learning and use the report functions present in it's learning to aggregate the results for all students and exercises. The aggregated results would then be used to decide whether students would be allowed to take the final exam or not.

it's learning contains two functions to aggregate assignment results, and is primarily used by the built in assessment tool. On of these functions creates a table of results with students in rows and assignments in columns. The other function is a virtual grading card, where assignment grades for individual assignments are displayed and can be aggregated into a final grade based on weighting the individual results.

Since it's learning implemented the IMS Tool Interoperability Guidelines' Outcome Service, it was assumed that the tool results would be treated like the results on the internal assessment tools. Unfortunately, this proved to be wrong. scores were reported back to it's learning, but the results for tool instances could not be added to the tabular results report. it's learning did however allow the tool instance results to be added to the virtual grading card. But for some reason, the results were always displayed as “no result” on the grading card report no matter the result reported back from the tool.

Because of this seemingly weird behavior, the company behind it's learning was contacted to clarify the situation, and they confirmed that tool instance results could be added to the grading card. After trying several more combinations of tool configurations and different grading results from Jexserver, with no changes to the grading card, the it's learning vendor was contacted again with the negative test report. After the contact person checked some details, the previous statement about tool result support was retracted, and he could now confirm that it's learning did not support aggregating the results.

Why it was possible to add tool instances to the virtual grading card was not explained, but this situation leaves two alternatives:

- Work with the vendor it's learning and try to get this issue fixed in it's learning. This is the preferred way to handle this situation, because results from tool instances could then be aggregated with other assessment tools in it's learning along with additional points assigned by teaching assistants.
- A fall-back solution is to implement a reporting function in Jexserver itself. This is quite easy to implement, since Jexserver has all the necessary information, but ideally this function would be in it's learning instead.

8.5 Using Submission Data for Analysis

The auto-submit data along with regular submissions can be used to perform a variety of analysis. While the scope of this master's thesis did not include performing such analysis, a couple of usage scenarios is presented to give examples to how this data can be used.

8.5.1 Analyzing Student Working Patterns

Since several submissions are sent to Jexserver during the student's work on an exercise, working patterns can be analyzed. Figure 5 contains a graph created based on submission data for a single student working on exercise 9.

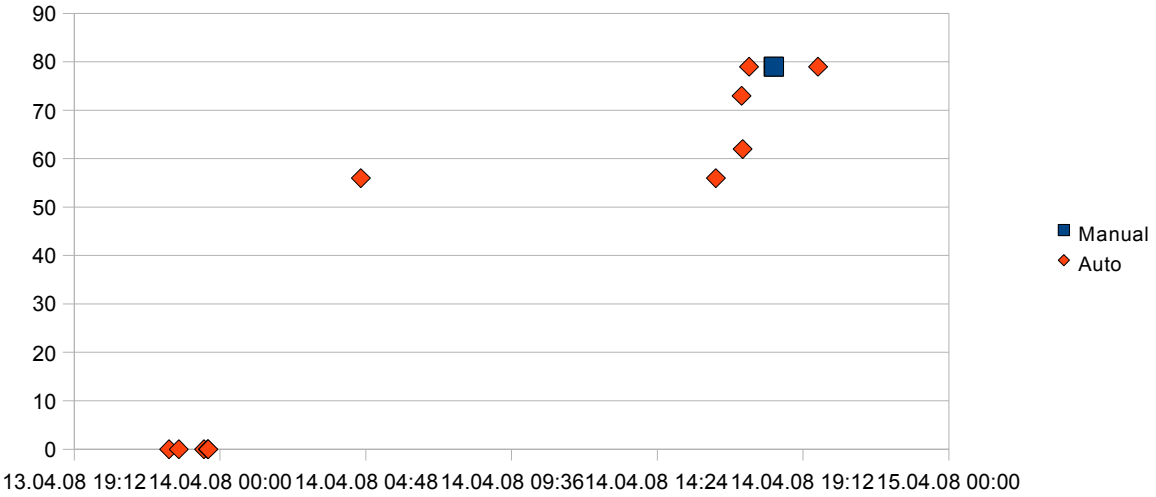

Figure 5: Exercise scores over time for a student working with exercise 9. Red points represents auto-submissions and manual represents explicit submission to server by the student.

We can see both the approximate time spans when the (anonymous) student worked on the exercise, and how fast he/she progressed score-wise.

8.5.2 Identifying Troublesome Exercise Requirements

The graph in figure 6 is generated dynamically based on information in the database. Green rectangles represents passed requirements and red rectangles are failed requirements. The rows contain the results of the last submission for each students, with one column per point-giving requirement.

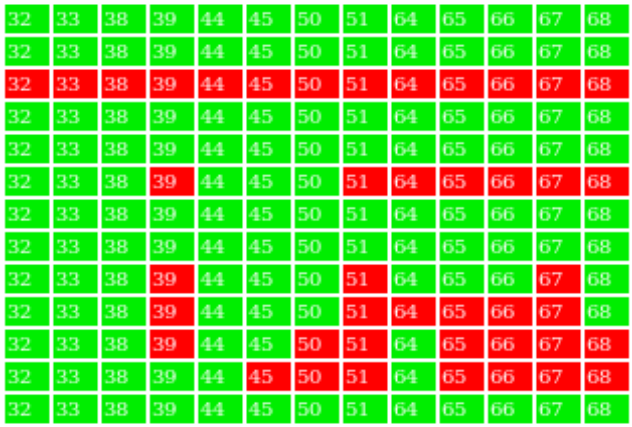

Figure 6: Representation of success and failure for individual requirements for all 13 students submitting exercise 9. The number are referring to specific requirements in the .jex file.

This graph can be used to monitor overall student progress on the exercise in real-time, by refreshing the web page generating this graph. The graph is dynamically generated by the teacher administrative interface available in Jexserver, but as noted in chapter 5, the teacher's interface is not integrated with it's learning.

It can also be used to identify particularly troublesome requirements - requirements where more than an expected number of students failed to implement the requirement properly. This information in turn can be used to evaluate the requirement itself, or whether the exercise description was not sufficiently clear. The results of such analysis could be used to improve future exercises and learn more about problematic topics. One could possibly also identify parts of the curriculum that should receive additional attention in classes.

8.5.3 Other Possibilities for Data Extraction

The following relevant information is collected for each submission:

- The student ID.
- The total score.
- The date and time of submission
- Individual score for each point-giving requirement.
- Whether the submission was sent automatically or manually.

This data can be exported in tabular form from the SQL database, and should provide ample opportunity for a wide variety of analysis, especially considering that one can also access a complete machine-readable grading report in XML format, detailing all code errors, and ultimately also access the complete source code for a submission.

Since automatic submissions are stored on the server, one could also perform some code analysis to see how the code for an exercise evolves over time.

8.5.4 Auto-Submit and Potential Privacy Concerns

Collecting in-progress work by students may be seen as an invasion of privacy, and before deploying this function in a course, this topic should be reviewed. See 8.6.3 for a discussion of this matter.

8.6 Student Survey

The student survey was conducted at the end of the semester, and only students in TDT4100 were eligible to participate. The survey received 56 responses. The total registered number of students in the course TDT4100 according to it's learning is 279. However, not all of these students are actually attending the course, and some students are just registered to take the exam and does not follow the exercise setup.

Course staff estimated the actual number of students to be about 240. This means that roughly one in four students responded to survey, which is a quite good result considering that the survey was performed during a period of exams for most students.

8.6.1 Why Students Did not Submit to Jexserver

The survey gives quite clear answers to why students did not submit to Jexserver. Students were asked to give reasons why they did not use this functionality (multiple answers were allowed):

Reason	Responses	Percent[36]
It was not mandatory	38	80.9%
There were no benefits in doing so	21	44.7%
It seemed cumbersome to configure	6	12.8%
It did not work for me	5	10.6%
Other reasons[37]	36	76.6%

Over 80% of the respondents answered that one reason was that it was not mandatory, and 44.7% also added that they saw no benefit of doing so. It was mandatory to deliver the exercise in person, in a computer lab at NTNU, to a student assistant.

Students were also allowed to write their own comment on why they did not use this feature. Answers revealed some additional reasons, including that several students did not know about this function at all, due to not attending all classes, and some also though the system was non-operational. See appendix D for a list of all responses.

Reasons why the submission feature did not work for 5 respondents are not known, but reasons might include:

- They may have had problems upgrading their plug-in to the required new version, or they may have had problems figuring out how to configure JExercise or how get an update. Comments given to the questionnaire supports this for some cases (see appendix D).
- Students may have failed to enter user account details properly.

If mandatory submissions to Jexserver is used in the next semester for this course, the situation should be more favorable:

- Students would get the JExercise plug-in with the submission feature from the start of the semester, removing the obstacle of upgrading.
- There will not be a need for students to manage a user account on Jexserver, since this will be handled transparently by the integration with it's learning.
- Students will not have to configure anything in the JExercise installation to be able to

36 Percentages based on number of respondents answering no to question 5.
37 See answers to question 7 (open question).

submit exercises, as all required information are bundled with the exercise they download from it's learning / Jexserver.

8.6.2 Effectiveness of Auto-Submit Function

While having discussed the potential of the auto-submit function to collect data for analysis, the effectiveness of this strategy on collecting fine-grained data is depended on how often students use the calculate score button in JExercise, since this is the mechanisms that triggers the auto-submit function.

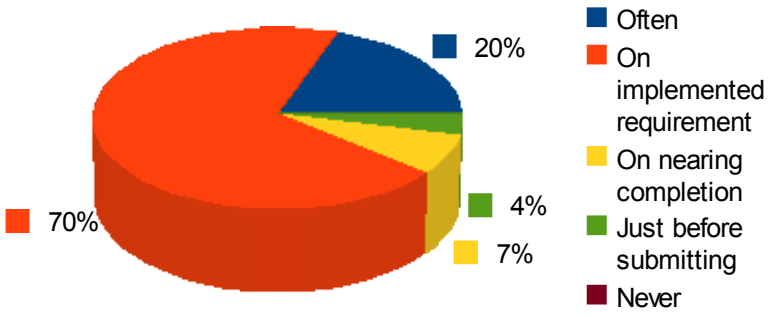

Figure 7: How often students use the button for calculating score in JExercise

Therefore, a question was included in the questionnaire to discover the habits of using this button while working on the exercise.

The results are very favorable to the auto-submit function. Almost 90% of the respondents used the function whenever they had implemented a requirements (or thought they had), or when they wanted to check if they had fixed a problem, or more often.

This suggests that the auto-submit function should be able to collect many submissions of work-in-progress during the students work on an exercise.

8.6.3 Acceptability of Auto-Submit Function

Another central question is whether the students will allow the auto-submit function to collect this amount of data. It is possible that some students may feel intimidated by this function, or feel that their work is being to closely monitored.

The questionnaire therefore posed the following question to the students:

> *"Which data would you accept being collected whilst working with the exercise, where the goal is to improve exercises (provided that data is treated anonymously)?"*

It was stressed in the question that the motivation was to collect data for analysis that data would be treated anonymously, implying that the automatically sent submissions would not impact their score or grade in any way.

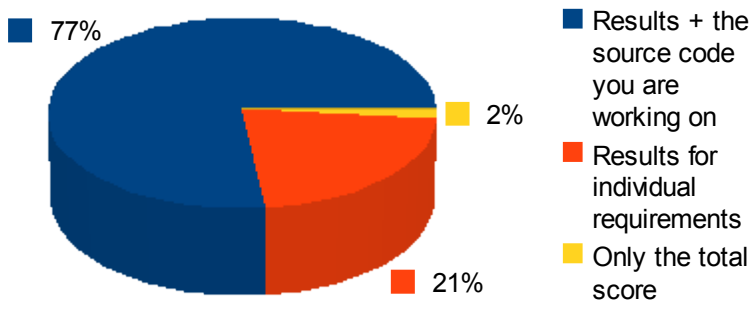

Figure 8: Which data students would accept being collected whilst working with the exercise

More than 3 in 4 students participating in the survey found it acceptable to collect the source code they are actively working on. This is a very encouraging result. But on the flip side, 1 in 5 students drew the line at reporting achieved results only.

In the end, the course staff must choose whether to use this function, but collecting in-progress work by students may be seen as kind of a privacy invasion, so the students should be notified beforehand if this function is actively used. On the positive side, most students

would likely ultimately find this acceptable according to the survey results.

It may be a good idea to provide a way for students to opt out of this feature, possibly providing an off-switch in the JExercise client. This way, one would still be able collect a large set of data while also respecting the privacy of the students who feels uncomfortable with this.

8.7 Performance Testing

The following sub chapter presents and discusses the results of the performance executed as described in section 7.3.3.

8.7.1 Primary Test

The primary test was run with the setup described in section 7.3.3. 250 simulated students submitted exercises during a 7.5 minute interval. The most critical benchmark of Jexserver is the ability to receive the the submission from the students. The time between the upload was started and an acknowledgment was received from the server was logged for each simulated student.

All submission were accepted by the server without errors. The mean time for submitting and receiving acknowledgment, as seen by simulated students, was 232 ms with a standard deviation of 208 ms. One must conclude that the server is more than capable of receiving and storing the submission from the students in real-time.

The above results did not include waiting for the grading results. The requirements stated that the server *should* be able to give the students a grading result as well, in real time.

To discuss how well Jexserver fares in regard to grading the submissions, we look at some graphs generated by the monitoring and logging system developed for the test. Figure 9 displays the amount of submissions waiting for grading on the server as the exam submissions progresses.

As one can see from the graph, the system is not capable of grading the submissions quickly enough, and a queue of submissions are slowly building up on the server. Too see how this affects the total grading time, the time passed between the submission were registered and the

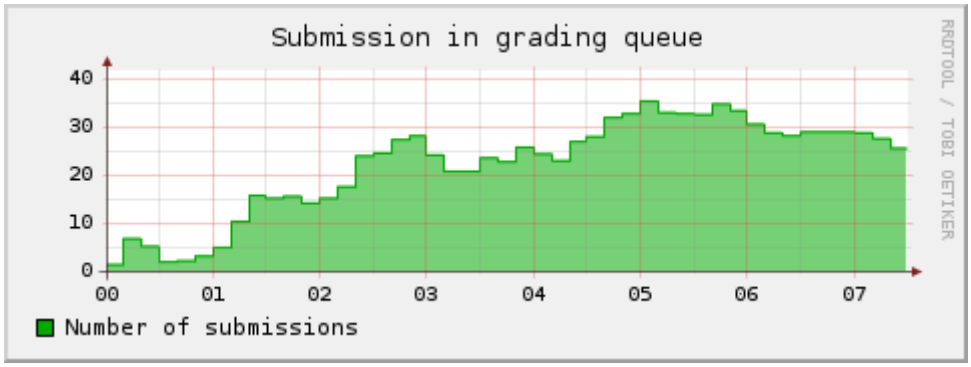

Figure 9: Number of submission in queue over time

grading results were ready, we can look to figure 10.

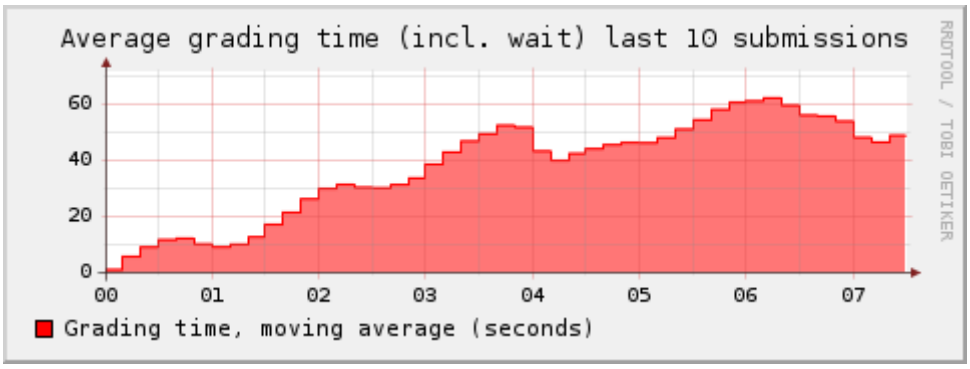

Figure 10: Increases in grading time over time

We can see a clear increase in grading time as the performance test progresses. This result was expected on the background of the graph in figure 9. The grading time for submissions surpasses 60 seconds 6 minutes into the experiment.

For reference, the change in actual grading time, over time, is illustrated in figure 11. Not surprisingly, the actual grading time is close to constant, just below 2 seconds. Since Jexserver grades only one submission at the time (in this setup), and remaining submissions wait in a queue, there is no reason why the grading time should be affected significantly over time.

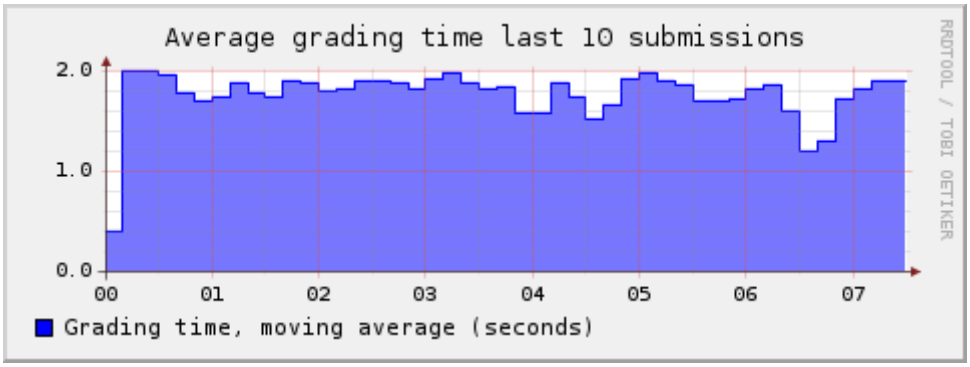

Figure 11: Average values of actual grading time

In summary, Jexserver did manage to receive and store all submissions, while giving a good response time of about 0.2 seconds. The submissions did however come in at a greater rate than submissions were graded, so the server gradually built up a queue, worsening the total waiting time for grading results as time progressed.

8.7.2 Secondary Test

In the secondary test, we took advantage of the distributed grading system built into Jexserver and added another grading process, running on another computer (see section 7.3.3). The rest of the test setup was equal to the first test.

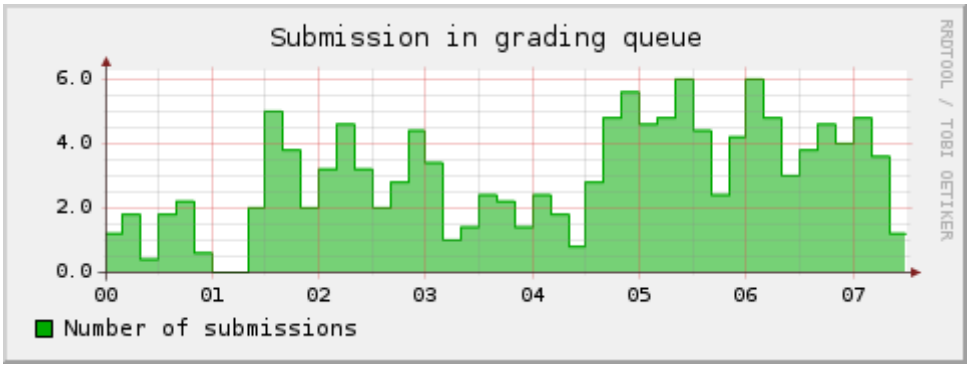

Figure 12: Number of submission in queue as the test progresses

The results of this test are very encouraging. As figure 12 illustrates, the gradual buildup does not occur in this test as it did in the first test (figure 9). The maximum number of submission queued at any (logged) time was 6 submissions, compared to 35 in the first test. This improvement is mirrored in figure 13.

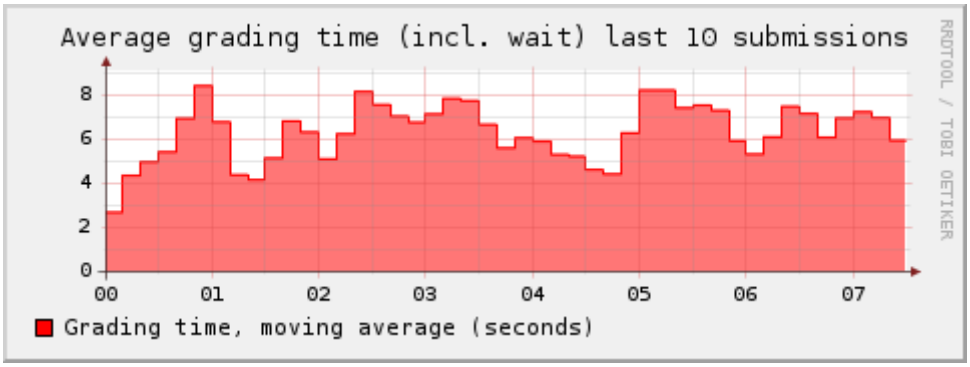

Figure 13: Grading time for submissions as the test progresses

In this test, the waiting time for results for a submission to be ready varies roughly from 6 to 8 seconds, without any sign of this value increasing. This means that the two Evalclients grading the submissions are capable of handling the volume of submissions. Using more than two Evalclients would also be an option if even better results were required.

8.7.3 Summary of Performance Results

The server did not have any problems receiving and storing all submissions delivered in the 7.5 minute time interval. With two processing cores grading submissions, all submissions were graded quickly, in the range of 6-8 seconds. This more than qualifies as *within reasonable time* as we determined to be 30 seconds in section 7.3.2.

The original requirement called for being able to receive and grade 250 student submission within a time frame of 15 minutes. The system managed to perform well when the time span was reduced to 7.5 minutes. The primary reason to halve the time span was to create a reasonable worst-case scenario, since there is no guarantee that students would submit their exercises throughout the full 15 minutes. It is likely that the number of submissions would increase somewhat as the deadline approaches, while possibly decreasing sharply right before the deadline. The effect of this would be that a significant portion of the submissions clumped together in a time span shorter than 15 minutes. But at the same time, some students would probably also deliver their answers before the exam reaches its final 15 minutes. Shrinking the time span to 7.5 minutes while still receiving and grading 250 submissions should make up for uncertainty inherent in the probability distribution of the timing of students' submissions.

For an exam, if “real-time” feedback of grading results to students is desirable, using at least two processing cores is recommended. The computers used during the test was commodity

computers with average hardware specifications. A good choice for a production server would be a computer with a modern multi-core processor. Alternatively, a distributed system with more than one computer would also work well.

9 Conclusion and Further Work

The integration work with it's learning resulted in a very tight integration, blurring the lines between it's learning and Jexserver as seen by the student. Although the IMS Tool Interoperability Guidelines does not address user interface integration, the tool blends naturally into it's learning. Using Jexserver and it's learning together also alleviates the need to use a dedicated system to track a student's progress through exercises as it is done today.

Having the exercises available for both download and submission in one place, easily accessible from the it's learning course interface should be able to provide a great improvement in usability for students. The new zero-configuration feature of JExercise when used together with it's learning and Jexserver should eliminate common student errors in the submission process and also reduce the amount of support needed by teaching assistants and course staff.

Jexserver was successful in grading submissions for all exercises that where developed for the course TDT4100. The exercises were developed completely independent of Jexserver and only a minor change to a couple of exercises were needed. Jexserver itself needed to relax a couple of security restrictions on the grading process to be able the grade some of the exercises.

The performance of Jexserver was shown to be more than sufficient to handle submissions from 250 students in the final 15 minutes of an exam. The scalability of Jexserver's grading system made it possible to provide, for all submitting students, grading results in less than 10 seconds. The performance of Jexserver is good enough to perform a digital exam based on JExercise and Jexserver.

As final conclusion, with the work and testing done throughout the semester, Jexserver has finally transitioned from a prototype implementation to a production-quality system ready to be used to manage and grade exercises for “TDT4100 Object Oriented Programming” and other programming courses.

9.1 Further Work

Here are some thoughts for future work on Jexserver and follow-up to this thesis:

- Either work with it's learning to improve their handling of results from learning tools, or implement a web-based reporting in Jexserver with possibility for teaching assistants to add points to students' scores.
- Include possibility to add security exception to the .jex file, in cases where exercise code needs special access, for instance be able to contact an Internet host.
- Improve the teacher management interface with more statistics.
- Integrate the teacher management interface into it's learning.
- Use Jexserver actively in a programming course with mandated submitting to Jexserver for grading, and evaluate the system in full-scale use.
- Allow JExercise to look for updates to the exercise itself. It has happened on occasion the exercises contains errors which needs to be distributed to students. Automating this process would further improve usability of the system.
- Evaluate more closely the usefulness of the auto-submit submissions and use try to use data to analyze and find problems with exercises.

The complete source code for the JexServer implementation and JExercise is available on NTNU's open source portal at http://www.opensource.idi.ntnu.no/projects/jexercise/.

The IMS Global Learning Consortium, Learning Tools Interoperability v2.0 Working Group is currently working on version 2 of the interoperability guidelines. Since the integration opportunities in version 1 was quite limited, a future project could also involve looking at how the next version could improve integration, and ultimately, the user experience. The charter is available for download from the IMS Global Learning Consortium website [15]. A Google Summer of Code 2008 project has been launched to provide reference implementations of Interoperability Guidelines 2.0 [16].

Appendix A: References

[1] Trætteberg, H. and Aalberg, A. (2007) Authoring specification- and test-based Java exercises with JExercise. Presented at the NIK 2007 conference (http://www.nik.no).

[2] Solheim, F. (2007) Automated Grading of JExercise Programming Exercises and Exams. Master's thesis background project report, available from NTNU.

[3] "t´s learning: Oversikt over noen av funksjonene i basisversjonen", accessed May 29, 2008: http://www.itsolutions.no/imaker.exe?id=2526

[4] “WhatIsWebCat” http://web-cat.cs.vt.edu/WCWiki/WhatIsWebCat, accessed Dec 5, 2007.

[5] Masinter L. (1998) RFC 2388 Returning Values from Forms: multipart/form-data.

[6] Freed N., Borenstein N (1996) RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.

[7] Valenta M. (2004) On the Job: The Eclipse Jobs API, accessed Jan 30, 2008: http://www.eclipse.org/articles/Article-Concurrency/jobs-api.html

[8] Apache Tomcat home page, accessed Jun 5, 2008: http://tomcat.apache.org/index.html

[9] SOAP Version 1.2 Part 0: Primer (Second Edition), accessed Jun 5, 2008: http://www.w3.org/TR/2007/REC-soap12-part0-20070427

[10] IMS Global Learning Consortium, Inc. (2006) IMS Tools Interoperability Guidelines. Available for download from http://www.imsglobal.org.

[11] Raggett D., Le Hors A., Jacobs I., W3C (1999) HTML 4.01 Specification, W3C Recommendation 24 December 1999.

[12] "Proxy-verktøy", accessed May 25, 2008:

http://www.itsolutions.no/imaker.exe?id=6833

[13] JSR-000154 Java(TM) Servlet 2.4 Specification, accessed May 25, 2008: http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html

[14] XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 October 2004, accessed Mac 25, 2008: http://www.w3.org/TR/xmlschema-0/

[15] "Learning Tools Interoperability v2.0 Working Group", accessed May 27, 2008: http://www.imsglobal.org/toolsinteroperability2.cfm

[16] "Functionality Mashup Using IMS Learning Tools Interoperability Google 2008 Summer of Code ", accessed May 28, 2008: http://www.sakaiproject.org/soc2008/

[17] Schadler T., Zavradinos G [2007] Suvery Highlists: North American Technographics Consumer Techology Online Survey, Q1 2007.

[18] Axis installation instructions, accessed May 10, 2008: http://ws.apache.org/axis/java/install.html

[19] Cascading Style Sheets home page, accessed May 28, 2008: http://www.w3.org/Style/CSS/

Appendix B: Glossary

CSS	Cascading Style Sheets. A standard implemented in web browsers that allows style rules to be applied to web documents, separating style from content.
Dispatcher	A server process handing out evaluation jobs to Evalclients.
Eclipse	An extensible integrated development environment. Available from http://www.eclipse.org.
ECJ	Eclipse Compiler for Java, also known as Eclipse JDT Compiler. Available from http://www.eclipse.org.
Evalclient	A long-running process connected to, and receiving jobs from a Dispatcher. Embeds Evaluator.
Evaluator	The controller for running Jextester. Is a part of Evalclient. Also responsible for compiling the students code before launching Jextester.
HTML	HyperText Markup Language. A standard for describing document structure and representation, used primarily by web browsers to display web site content.
HTTP	HyperText Transfer Protocol
IDI	Department of Computer and Information Science (Norwegian abbreviation). See http://www.idi.ntnu.no/.
IMS	Short version of IMS Global Learning Consortium, Inc. http://www.imsglobal.org/
IT1104	See TDT4100.
Jex	Short for JExercise, or the file ending for JExercise specification files.
JExercise	An open-source project at NTNU. See section 2.1 on page 4 for description.
JExercise plug-in	A plug-in for Eclipse. See section 2.1 on page 4.
Jexfile	A JExercise exercise specification file (in XML format).
Jexserver	The name of the prototype system developed in an earlier project and extended in this master's thesis.

Jextester	The program that runs the tests associated with a student's submission.
LMS	Learning Management System. it's learning is an example of an LMS.
MySQL	An cross-platform open-source SQL database, http://www.mysql.org/.
NTNU	Norwegian University of Science and Technology. Website: http://www.ntnu.no/.
SQLite	A lightview database, http://www.sqlite.org.
TDT4100	Course "Object Oriented Programming" at NTNU.
TDT4520	Program and Information Systems, Specialization Project.
Tool	A learning tool is defined by the IMS Tool Interoperability Guidelines.
WSDL	Web Services Definition Language, a XML-based standard to define web services.
XML	Extensible Markup Language. A popular markup language used to describe hierarchical and/or structured data.

Appendix C: Questionnaire

A survey was conducted with a questionnaire in Norwegian. The questions has been translated to English for inclusion in this document.

1. Did you use JExercise to work with Exercises in the course TDT4100?

- Yes
- No

2. If no, what was the reason(s) for this? (multiple choice)

- It was cumbersome to install / configure
- It was not necessary to complete the exercises
- It did not work on my computer
- I did not like using the system
- Another reason

3. How often did you use the button for calculating score in JExercise? (multiple choice)

- Often
- Whenever I implemented a requirement or fixed an error
- When I though I was nearing completion of the exercise
- Only to check if I was ready to submit the answer
- Never

4. Which data would you accept being collected whilst working with the exercise, where

the goal is to improve exercises (provided that data is treated anonymously)? (multiple choice)

- Results + the source code you are working on
- Results for individual requirements
- Only the total score

5. Did you use the submit button in JExercise to deliver exercises?

- Yes
- No

6. If not, what was the reason(s) for this? (multiple choice)

- It was not mandatory
- There were no benefits in doing so
- It seemed cumbersome to configure
- It did not work for me

7. If there was another reason for not delivering exercises via the submit button in JExercise, you can write it here (voluntary).

- (open question)

8. How often did the teaching assistants give you extra points beyond JExercises score?

- Never
- Some times
- Often

9. How would you feel about the JExercise-server having more and/or stricter tests than the ones you can run in Eclipse?

- (open question)

Appendix D: Survey Results

This chapter contains all responses to the questionnaire listed in appendix C.

Responses to Checkable Questions

Answers to yes/no questions and multiple choice questions are presented here. Responses to the open questions are listed in the next sub chapter.

1. Did you use JExercise to work with Exercises in the course TDT4100?

Answer	Responses	Percent
Yes	56	100%
No	0	0%

2. If no, what was the reason(s) for this? (multiple choice)

Answer	Responses	Percent
It was cumbersome to install / configure	0	0%
It was not necessary to complete the exercises	0	0%
It did not work on my computer	0	0%
I id not like using the system	0	0%
Another reason	1[38]	1.8%

38 An unexpected answer here, since all respondents answered yes to question 1.

3. How often did you use the button for calculating score in JExercise? (multiple choice)

Answer	Responses	Percent
Often	11	19.6%
Whenever I implemented a requirement or fixed an error	39	69.6%
When I though I was nearing completion of the exercise	4	7.1%
Only to check if I was ready to submit the answer	2	3.6%
Never	0	0.0%

4. Which data would you accept being collected whilst working with the exercise, where the goal is to improve exercises (provided that data is treated anonymously)? (multiple choice)

Answer	Responses	Percent
Results + the source code you are working on	43	76.8%
Results for individual requirements	12	21.4%
Only the total score	1	1.8%

5. Did you use the submit button in JExercise to deliver exercises?

Answer	Responses	Percent
Yes	9	16.1%
No	47	83.9%

6. If not, what was the reason(s) for this? (multiple choice)

Answer	Responses	Percent[39]
It was not mandatory	38	80.9%
There were no benefits in doing so	21	44.7%
It seemed cumbersome to configure	6	12.8%
It did not work for me	5	10.6%
Other reasons[40]	36	76.6%

39 Percentages based on number of respondents answering no to question 5.
40 See answers to question 7 (open question).

8. How often did the teaching assistants give you extra points beyond JExercises score?

Answer	Responses	Percent
Never	48	85.7%
Some times	7	12.5%
Often	1	1.8%

Responses to Open Questions

This sub-chapter contains all the responses for the open questions number 7 and 9.

Question 7

If there was another reason for not delivering exercises via the submit button in JExercise, you can write it here (voluntary).

The following are the answers given to this open question. They have not been altered or edited in any way except removing white space.

- Brukte datamaskiner på p15 til å gjøre øvingene, tror ikke det fungerte å gjøre det der.
- Visste ikke om det.
- JExercise måtte oppdateres for å kunne gjøre det, og ettersom alt likevel fungerte bra, gjorde jeg ikke det.
- Jeg hadde ingen knapp i versjonen jeg brukte, oppdaget ikke at den fantes før jeg var ferdig med øvingsopplegget.
- Tenkte vel egentlig aldri så mye over det...
- studass godkjente øvinger ved å se hvor mange poeng en fikk når en trykket på poengberegningsknappen - og noterte dette som godkjent/ikke godkjent
- I den versjonen av JExercise jeg har installert er der ingen submit-knapp. Og jeg har ikke tatt meg bryet med å reinstallere når alt fungerte bra slik det var.

- Personlig så visste jeg ikke at det var mulig å levere øvingen via "submit" knappen? Er dere sikre på at det var aktuelt? Det hadde ihvertfall spart meg for utrolig mye bry, ved å slippe måtte vise øvingene på sal...
- Fant aldri ut hvordan man skulle, tror jeg rotet litt rundt 2 ganger på nette uten finne noen ok veiledning.
- I gavekortinnleveringen så jeg for sent at det var en mulighet, I resten av øvingsopplegget var jeg ikke sikker på om det var helt ferdig implementert "på deres side".
- Submission Status: Awaiting grading... This page will refresh itself automatically until the results are in. JExercise hang seg opp her når jeg brukte submit knappen (ca. kl. 19:50, 19.05.08). Siden refreshet seg som det står, men ingenting skjedde. Ventet i snart 5 minutter uten resultat. Jeg likte ikke helt at passordet mitt er tilgjengelig i klartekst via. server linken though, og i tillegg sendt via http ikke https. Må vel være mulig at brukeren kan få spørsmål om å skrive inn slikt selv ved submit, evt med mulighet for en "Save details.." checkbox.Bare mine tanker :)
- Slet med å finne login
- Alle øvingene måtte uansett vises til studass, dessuten hadde ikke jeg noen submit-knapp før jeg oppdaterte for å bli med i konkurransen...
- Jeg oppdaterte aldri JExercise slik at jeg fikk en submit-knapp. Ble aldri opplyst om oppdateringen offisielt som jeg så, og jeg måtte likevel gå på sal for å levere, så var ikke noe vits i å gjøre det.
- øvingene ble godkjent av studass på veiledningstimene
- Fikk ikke med meg at det gikk an, var ikke på noen øvingsforelesinger av den enkle grunn at jeg fikk til øvingene på egen hånd...
- Mest fordi det ikke var nødvendig når vi allikevel måtte levere til studass. Hadde vi hatt mulighet til å bare levere på server hadde jeg nok brukt den.
- Systemet var ikke oppe å gikk for innlevering over nett, såvidt jeg vet.
- Visste ikke at dette var mulig.

- Jeg trodde at for å levere inn øvinger så måtte vi få det godkjent av en studass, har ikke fått med meg at det fantes en "submit-knapp" :)

Question 9

How would you feel about the JExercise-server having more and/or stricter tests than the ones you can run in Eclipse?

The following are the answers given to this open question. They have not been altered or edited in any way except removing white space.

- OK
- Det er helt greit. Det var av og til at det gikk an å lure testene
- Det kan nok være hensiktsmessig. Men skal dere ikke stille spørsmål om hvor vidt vi synes det å bruke JExercise var en god måte å jobbe med oppgaver på? Om vi fikk et godt faglig utbytte av det, og om vi synes opplegget var godt pedagogisk sett? Om oppgavene var bra nok formulert, eller om de var mangelfulle og lite spesifikke? Jeg vil iallefall si at det burde gis en bedre oversikt over hva som skal bli det ferdige resultatet av hver øving, hvordan de forskjellige klassene man lager skal samvirke, og at mange detaljer om hvordan oppgaver skulle løses burde vært bedre spesifisert. Litt for ofte måtte vi gjette hva JExercise var ute etter som resultat. Flere eksempler kan da også benyttes.
- Det høres ut som en god ide, hvis man i det minste har lov til å foreta et visst antall forandringer av øvingen etter innlevering.
- Det er ingen dum idé. F.eks. kan man da teste hvor effektiv kode studenten har skrevet også. Tenker da på å kjøre timeout-sjekk osv, slik det blir gjort programmeringskonkurranser til tider. IKke at dette nødvendigvis gir studenten noen mindre score, men kanskje gi en tilbake melding med "Du klarte testene så, så bra, og det er rom for forbedringer". Dette vil gjøre at studenten kanskje blir mer oppmerksom på hvordan han/hun skriver kode /løser problemer.
- Vet ikke.
- Vet ikke. Jeg stiller meg hverken positiv eller negativ til det.

- Dette var jeg ikke klar over, men testene mine har fått full score hver gang. Hvis testene på serveren er strengere enn testene på JExcercise synes jeg det burde understrekes ganske klart på forsiden og/eller ha en konfirmasjonsknapp hver gang man leverer øvingene over Submit-knappen. Også i og med at øvingene blir strengere gjennomgått på serveren ser jeg kanskje på som litt urettferdig. Hvis man har fått 100 poeng på JExcercise, kommer til studass og skal vise frem og man plutselig får 75 poeng, kan dette være et ganske greit "slap in the face".Om man får 3 ganger å submitte på, tilbakemelding på hva som kan være feil etter hver gang man submitter, synes jeg denne løsningen er fin. Opplegget med JExcercise generelt synes jeg kanskje er litt vel slapt, i og med at man kan spamme ned score-knappen, og også gå inn i testene og debugge, eller rett og slett finne ut hva oppgaven går ut på (synes ikke oppgavetekstene har vært spesielt gode, hvertfall ikke den første halvdelen av øvingsopplegget, etter det har det forbedret seg). Det sparer selvfølgelig masse tid for stud.ass. da øvingene kan være ganske komplekse, og det tar tid å sette seg inn i hver enkelt students kode (spesielt hvis studenten ikke har lært seg riktig indentering og/eller "korrekt" oppsett kan dette være et helvette). Alt i alt synes jeg øvingsopplegget og forelesningene har vært bra! :)
- likegyldig
- Så lenge det går klart frem, og de var bedre skrevet enn de vi fikk, ville jeg ikke hatt noen problemer med det.
- Hvis vi har en oppgave hvor vi forstår akkurat hva som kreves er det greit å få det testet. Vært et par ganger vi måtte implementere ting fordi testene krevde det uten at det sto i oppgavetekst...
- Helt greit, egentlig, fordi det var noen ganger at kode fungerte på gamle tester, men på nye tester hvor man rebrukte gammel kode, så fikk man feil fordi gammel kode egentlig ikke var helt riktig. Så det kan være greit å være enda mer sikker på at man faktisk har fått til alt.
- Hvis det skal bli strengere, bør en heller vurdere kildekoden enn å inføre flere tester..
- Ville vært fint, og evt at den tester på flere måter ettersom man kan ha riktig besvarelse i Eclipse, men mangle noen få ting som gjør forskjellen mellom kanskje 0 og 100p ettersom den ikke tester så grundig. Men hvis serveren tester mindre deler

grundigere, så er det kanskje lettere å se hva som er feil og hva som er riktig.

- Bra
- Likegyldig
- Jeg stiller meg negativt til det å ha JExercise tester på en servevr i det hele tatt(både fler og strengere, og bruk av det i det hele tatt). Det er så enkelt at jeg ikke vet hva som er kravet til koden jeg skriver, for øvingsoppgavene forutsetter en del fortolkninger og at vi skal ta antagelser. Men testene har ikke rom for disse antagelsene i det hele tatt. (For ikke snakk om at det av og til er feil i øvingsteksten, og uten selve test-koden vil jeg ikke finne den, og det vil bare være frustrerende og ikke ha feilsøkingsmuligheten). Når det har blitt sagt så liker jeg ikke JExercise heller. Det er lite resultatvennelig, og med det mener jeg at du ser ikke hva du koder når du bare koder for å tilfredstille testene. Det var i hvertfall slik jeg følte det, og at jeg ikke hadde muligheten til å bruke koden og utvide oppgaven litt for lekens skyld.
- Det vil gjøre øvingsopplegget til et enda mer "løs små algoritmer du ikke trenger tenke over hvordan er integrerte uten tanke for arkitektur whatever"-opplegg, så jeg stiller meg derfor negativ til det. Dette kombinert med noen "tenkeoppgaver", eller oppgaver der du må tenke ut litt arkitektur og sånn selv, derimot, hadde funket fint. Med jexie-knapp trenger en ikke tenke så mye egentlig!
- Forholdsvis nøytralt. Men opplegget slik det har vert har passet meg meget bra.
- Jeg liker måten det fungerer på i Algoritmer og Datastrukturer. Det å få en egen JExercise-server med flere og strengere tester høres positivt ut. Hvertfall i det lange løp.
- Flere tester er bra på en måte, og det er også strengere tester forutsatt at de da blir mer presise, men siden en oppgave ofte kan løses på flere måter må testene også være litt "løse" om man kan kalle det det...
- Blir bare dumt. Hele poenget med oppgavene er jo at vi skal jo vite hvilke inndata vi har med å gjøre. På samem måte som skal man lese inn fra bruker så vet man jo at dem kan finne på skrive hva som helst. På samme måte så er oppgave teksten alt for uklar i noen sammenhenger.. Uklar på hva slags feil vi faktisk kan få. Hadde teksten vært klinkende klar så hadde det ikke vært noe problem da kunne dere gjerne hatt en

million tester for min del. Flere av testene kunne også vært bedre skrevet. På ene oppgaven la jeg inn flere tester selv fordi den ikke testet alt jeg ville ha testet.

- Var mye artigere å jobbe med øvinger der JExercise testet mye og man fikk poeng for deloppgaver, f.eks: 5 poeng for en metode, det var motiverende å se testen bli grønn og man fikk poeng, selv for små ting :). Var værre når man f.eks kun fikk poeng når man hadde gjort alt (60 poeng, 30 poeng) Mulig å implementere i server testen slik at man kan få tips underveis?

- Det virker urimelig så lenge ikke det blir klart spesifisert hva som i så fall testes i oppgaveteksten.

- Siden JExercise først om fremst er et verktøy for å få bekreftet at ting er gjort riktig, ser det som mot sin hensikt å prøve å "lure" testen. Derfor høres dette (har ikke testet på server) strengt tatt bare bra ut.

- Jeg er positiv til å ha flere tester i JExercise for èn klasse, men helst bare èn som faktisk er gjeldende for poengtelling. Men kanskje tester som er strengere mht. krav om bedre kjøretid, kode konvensjoner o.l. Værtfall noe til å inspirere de ivrigste av oss til å yte det lille ekstra.

- dumt, for oppgavetekstene var ikke spesifikke nok...om de blir bedre, kunne det virket

- Dette vil berre vere positivt då dette vil tvinge oss til å jobbe betre på eiga hand, og sjølv teste om koden vår fungerer.

- Jeg synes det ville vært svært urimelig, siden det ville være vanskelig å vite hva kravene for å få poeng er. Flere ganger iløpet av øvingsopplegget har oppgaveteksten vært vanskelig å forstå, og vi har måtte lese gjennom test-koden for jexercise/JUnit-testene for å forstå hva det egentlig var vi skulle gjøre. Hvis serveren skal ha strengere krav må vi i så fall få bedre oppgavetekster, detaljert tilbakemelding på hvilke krav som ikke ble møtt, samt muligheten til å levere flere ganger.

- Jeg er ikke helt sikker på om jeg liker JExercise slik den er nå. Mange studenter (meg å) bruker den litt for mye for å sjekke om jeg har gjort en bitte liten ting rett eller galt. Jeg brukte selv JExercise ofte for å sjekke om jeg fikk til noe (men sp syns jeg også oppgavene var tvetydige, og det var bare ett svar, så ikke rart at man måtte bruke JExercise så mye. Men det er jo ikke JExercise sin feil). Hvis det ble stilt strengere

krav og flere tester, så burde oppgavene bli bedre formet. Ofte kunne det forståes at oppgaven kunne løses på mer enn en måte, men JExercise tillot bare en måte, så hvis du valgte feil metode, men likevel fikk alt til å kjøre slik det skulle, så ville du allikavel få feil. Jeg syns det godt kan bli litt strengere krav, slik at studenten virkelig jobber med programmeringen. MEN da må også oppgavene bli bedre formet.

- Når man først bruker JExercise i øvingsopplegget synes jeg at den burde ha tester som er gode nok til at man kan si at koden fungerer slik den skal (eller at dere i hvert fall skriver (med ord) hva som blir testet for, slik at man kan teste for andre ting selv). Hadde man i stedet oppfordret til at folk tester kodene sine selv (uten å gi tilgang til JExercise, men ved å tipse om for eksempel flittig bruk av syserr og debugging i eclipse) kunne koden godt vært testet strengt på en server.

- Det er ok, men man bør da få vite hva som feiler. Øvingene bør eventuelt spesifiseres mer nøyaktig, da det i år var rom for å tolke oppgaven annerledes enn de som utarbeidet LF, og dermed fikk man ikke godkjent, selv om man hadde riktig svar. Det kan være litt urettferdig, ettersom jeg flere ganger under arbeid med øvingsopplegget hadde vansker med å forstå hva JExercise egentlig ville ha som resultat, og måtte snoke litt rundt i testekoden for å finne ut. Hvis den ligger utenfor rekkevidde på en server kan det bli vanskeligere for meg å finne ut hva som skal til for å få poeng.

- Jeg ville syntes det var dumt. Noen ganger var man ikke sikker på hva det egentlig ble spurt om, og hvorfor det ble feil, så da måtte man sjekke testene for å se hvor den feilet.

- Det kan godt være strengere tester og slikt, men da må det kommer mer klart fram av oppgaveteksten hva som metodene skal gjøre. Det har til tider blitt gjetning om hva metodene skal gjøre, pga uklar oppgaveformulering. Det kan kanskje være greit å ha noen tester som kjører lolalt fra Eclipse slik at man kan få testet litt uten å måtte ha nettilgang, også heller ha en større og mer omfattende test på en server som bestemmer poeng.

- jeg synes det er bra, men ofte var feilmeldingene lite informative.

- det gjør ingen verdens ting.

- Helt greit så lenge den tellende poenggivingen blir riktig. Det viktigste er at man faktisk får riktig kompetanse, og da må man ta konsekvensen hvis man kun sikter mot

å få poeng i eclipse, og ikke ser på at øvingen må fungere som helhet. Men det krever bedre kravspesifikasjon, ettersom testene ofte er veldig spesifikke, mens det likevel kan være flere måter å løse oppgaven på. Dette skjedde flere ganger for min del.

- Høres helt greit ut.
- Det er gullverdt for all tilbake melding. Jeg synes at det er topp når jeg får beskjed om jeg er på rett spor.
- Det synes jeg ikke er så veldig bra
- Er greit nok med flere tester på serveren enn man kan kjøre i eclipse. Men det forutsetter at oppgaveteksten er veldig bra og at den spesifiserer godt alle kravene til hva programmet skal gjøre.
- Litt motvillig til det. Hvis du i utgangspunktet har en kode som oppfyller kravene som du kan se, mener jeg det blir feil å sette flere krav/strengere tester i etterkant av innleveringen. Dette vil ikke gjenspeile virkeligheten, der du kontinuerlig får vite hvilke krav ditt program skal ha.
- Bryr meg egentlig lite, er ferdig med faget.
- Hvis det skal bli strengere testing syns jeg dette heller skal integreres i øvingene. Ofte kan det være vel så vanskelig å faktisk forstå hva øvingsoppgaven ber om, enn å utføre kodingen. Da er det fint å ha testene som en sjekk på at man har forstått oppgaven riktig. Det har vært tilfeller hvor jeg først ved å studere testkoden, har blitt klar over punkter jeg har misforstått. Hvis det legges strengere tester på en JExercise-server, bør man derfor etter min mening ikke bli straffet hardt for eventuelle feil. Videre må også øvingene være bedre formulert en det de har vært.
- Flere tester kan sikkert være lurt da Eclipse-testen alene kunne ha feil i seg. Så hvis man fikk feil i Eclipse kunne man teste opp mot serveren og se om det funket.
- bra
- Helt greit så lenge man har mulighet til å sjekke resultatet lokalt på maskinen i tillegg til på serveren.
- Kunne tenke meg at serveren har færre tester som tester grovt på måte .. hele øvingen i

en test i stedet for deløvinger...

Appendix E: Code Structure

This chapter contains a short overview of the code structure for Jexserver.

Code Availability

The code itself is available from NTNU's open source portal under the JExercise project:

```
https://www.opensource.idi.ntnu.no/projects/jexercise/
```

The code for the entire Jexserver module is contained in the *jexserver* CVS module, while the enhancements to JExercise is commited to the CVS the modules *jexclient* and *jex.emf*. NTNU provides anonymous login to the CVS server.

Jexserver Structure

Directories

The *jexserver* module includes the following important directories:

bin	Output folder for Java classes (from src folder)
jexweb	Base folder for web application, contains JSP files for the web interface.
jexweb/WEB-INF/classes	Output folder for Java classes (from websrc folder)
src	Source code for Evalclient, Jexserver, and common core classes shared with the web application.
websrc	Source code for web-specific Java classes (Servlets)
wsdl	Contains WSDL files for IMS Tool Web Services as well

	as generated code and batch files for creating skeleton code.

Java Package Organization

This section gives an overview of the structure of the code by Java packages.

no.ntnu.stud.frodesol.jexserver.bin (src)
This package contains the executable programs that forms Jexserver; Dispatcher and Evalclient.

no.ntnu.stud.frodesol.jexserver.core (src)
These are core classes representing information stored in the database, such as *Exercise*, *Submission*, *Student* and classes representing other information such as *Report*, used to read and write XML-based grading reports.

no.ntnu.stud.frodesol.jexserver.dispatch (src)
Classes specific to the Dispatcher.

no.ntnu.stud.frodesol.jexserver.eval (src)
Classes used by Evalclient and the code used to run and evaluate exercises.

no.ntnu.stud.frodesol.jexserver.util (src)
Utiltity classes, including a helper program to create and update the MySQL database used by JExercise.

no.ntnu.stud.frodesol.jexweb (websrc)
This package contains the Java Servlets used to the Jexserver web interface, it's learning integration and submission interface.

no.ntnu.stud.frodesol.jexweb.imstool (websrc)
This package contains the implementation of the IMS Tool LaunchService, as well as a client wrapper class for easy communication with the LMS's OutcomeService.

org.imsglobal.www.services.ti.wsdl.sync.* (websrc)
These packages contain code automatically generated by Apache Axis's WSDL2Java tool, based on the WSDL files provided by the IMS Tools Interoperability.