INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

# Automatic Evaluation of Programming Projects

**Miguel Rocha Ferreira Aguiar Nogueira**

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

**Júri**

May 2011

# Acknowledgements

This thesis could not have been done without the help and support of several people who undoubtedly need a word of appreciation.

First and foremost I need to thank my parents for having given me all the possibilities to have followed my studies and for all the support given during not only the time used to work on this thesis or the degree but during all my life.

I would also like to thank my advisors for the opportunity given to work on this thesis and the good advices given to steer my work in the right path.

I cannot forget my family and friends for the giving me a solid foundation to endure good and bad times and to having kept supporting and cheering me in this academic road coming to an end now (at least for now).

Lisboa, November 19, 2011
Miguel Aguiar Nogueira

To my parents Alberto and
Manuela

# Resumo

Cadeiras de programação são uma componente estrutural de qualquer curso da área das Ciências da Computação. Para os alunos provarem que aprenderam os tópicos leccionados em cada cadeira, normalmente é-lhes exigido para resolver problemas programaticamente ou para desenvolver uma aplicação cumprindo alguns requisitos. Para avaliar os projectos implementados pelos alunos, o corpo docente das cadeiras ou os avalia manualmente ou usa um sistema de avaliação automática. Actualmente existem poucos sistemas de avaliação automática e os que existem não cumprem os requisitos considerados importantes, sendo na sua maior parte orientados a concursos de programação.

O objectivo desta tese é a criação de um sistema de submissão e avaliação de projectos para dar suporte a cadeiras de programação. O objectivo é que o sistema ajude o corpo docente e os alunos, automatizando grande parte da avaliação dos trabalhos e dando informação instantânea aos alunos de como é que as suas soluções resolvem o problema proposto.

Este documento apresenta a ideia por detrás dum sistema destes e os seus requisitos, o estado da arte dos sistemas utilizados actualmente e apresenta a arquitectura e implementação de um novo sistema que respeita tais requisitos.

# Abstract

Programming courses are the basis for any Computer Science degree. For students to prove they have learnt the topics taught by each course, they are generally asked to programmatically solve a problem or to build an application according to some requirements. To evaluate and grade the projects implemented by students the faculty of the courses either evaluates them by hand or resorts to an automatic evaluation system. Currently, there are few automatic evaluation systems and those there are do not comply to the requirements considered important, being most of them programming contest oriented systems.

The purpose of this thesis is the creation of a project submission and evaluation system to support programming courses. The goal is for the system to help both the faculty of the course and students, by automating most of the evaluation, managing the course and giving instant feedback to students on how well their solutions solve the problem given.

This document presents the rationale and requirements for such a system, the state of the art of the systems currently used and presents the architecture and implementation details of a new system that complies with the requirements presented.

# Palavras Chave Keywords

## *Palavras Chave*

Avaliação Automática
Disponibilidade Submissões
Feedback aos Alunos
Apoio Cadeiras Programação

## *Keywords*

Automatic Evaluation
Submission Availability
Feedback to Students
Support Programming Courses

# Table of contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Introduction

Courses in which new programming languages, paradigms, methodologies and algorithms are first taught to students often include a practical component so that students can exercise the acquired theoretical knowledge. Usually, this practical part often involves creating a program in order to solve a given problem. The programs made by each student (or group of students) have to be evaluated by the teachers of the course. These programs can be evaluated accordingly to two parameters: i) whether the program executes as required and ii) whether students have implemented the program applying the concepts and constructs taught in the theoretical classes. Depending on the course and on the teacher, the evaluation can be just based on the first parameter or on the combination of both of them. When the course has more than a hundred students, something that is often usual in the first years of a degree, evaluating all the project submissions is a big effort for the faculty in charge of said course.

For the first parameter described, it is possible to define automatic tests that verify the correct behavior of student's programs but this is not possible for the second parameter that still requires manual verification.

For the students of a course the feedback about this programs is crucial, specially if they can still correct their work. Giving the possibility of finding errors and correct them during the implementation of a program is a big advantage since students will learn more by correcting their own mistakes.

For some years, with the advent of Internet and the ability for students to submit projects remotely, teachers have used automatic submission systems to receive the projects. These systems verified which student was logged and at what time he or she uploaded the project to ensure they were submitted before the deadline. But there was no help at all to run the code through a series of tests to verify its correctness and automatically publish the results online. Later on, some teachers started using contest-intended systems like Mooshak (Leal and Silva 2001a) (Section 2.2) to automatically run a series of tests on the submitted programs. Since these systems are not intended to support a course, they have several limitations (some of which will be described on Chapter 2) and do not support all the desired functionality needed to help run a course.

The two main problems that make these systems are hard to create are to ensure the system can always receive student's programs (availability) and the ability to gracefully endure high load peaks.

## 1.2 Goal

The purpose of this thesis is to create a new system or adapt an existing system in order to assist students and teachers of programming courses by automating most of the tasks related to evaluating programming assignments.

For teachers, the evaluation system should help to automate the evaluation process offloading most of the evaluation work from them. This includes aspects like being easy to create automatic tests and automating the grade publishing. The system should also handle student management helping on all the grouping and schedule issues, managing project reception inside the deadline and also managing project grades.

The evaluation system should also help students by giving them instant feedback on the correctness of the submitted solution, since the system will be available 24/7, allowing students to work at their own pace. This feedback consists in running a series of tests and displaying the tests which the students' code passed and those which failed. This interactivity with the student enters the e-learning topic by trying to engage students in the course which can help improve learning and consequently grades and reduce the drop of rates of the courses, as described in the tests made in the University of Murcia(García-Mateos and Fernández-Alemán 2009), topic covered in the Mooshak section (Section 2.2). Moreover, this system also should allow the student to check if the version being submitted is the correct one, decreasing the probability of delivering an older version of the code.

The 24/7 issue mentioned earlier, which is in itself an availability and quality of service (QoS) requirement is specially relevant. Students are being evaluated for their their projects and the grades normally account for a big percentage of the final course grade (normally around 50%), so the system has to be up to receive students' submissions. In a peak of traffic like those that usually happen nearing deadlines the system may get under heavy load, but it is essential for it to continue to be able to receive the submissions, even if some limitation in the automatic evaluation part is enforced based on system load. To give an example of an extreme situation in which the system might have to encounter and handle gracefully, on the 2009/2010 Object-Oriented Programming course at Instituto Superior Técnico, a single group submitted their project 200 times in a few minutes/footnoteInformation provided verbally by the faculty of the course..

More concretely, the goal of this thesis is to develop a system able to help the faculty of programming courses to automatically grade at least a part of the student's programs. For this goal to be achieved there are some requirements the system will have to comply with to be more useful than the current system and the existing alternatives.

This new system should make course management easy on the project front. It should handle group definition and their maintenance, to allow teachers to visualize the submitted projects and define, in an easy way, the tests needed to evaluate a project and automatically grade them. Moreover, it should be easy for students to submit their solutions and get the corresponding result.

Project submissions for a course usually have a big growth in number as the deadline approaches. At the end of this deadline a very high number of submissions and therefore evaluations are requested to the system and the load of the system increases substantially reducing its effectiveness. The new system must have mechanisms to handle this load gracefully.

## 1.3 Requirements

This section describes in detail the requirements that an automatic evaluation system should meet. First, we present the functional requirements and then the non-functional requirements. The desired system should have a set of functionalities that span from authentication to the ability of automatically grading a project.

### 1.3.1 Functional Requirements:

The functional requirements of an automatic evaluation system are:

**Authentication:** The system should allow a student to authenticate himself univocally with the system since the system is used to evaluate part of his grade on a course.

**Grouping:** The system must support groups since there are courses in which some practical projects are to be done in groups of two or more students. The system must allow the creation of groups of students. The definition of groups should be easy since we do not want to overload teachers.
A possible support for using the grouping system used in Fénix (Section 2.7) by importing the groups and their composition. Teachers should be warned if there was any change in the Fénix groups and decide if they want to apply the changes on the system.
The system also must support the possibility of the groups being created by Project/assignment instead of being independent. This would allow more flexibility for creating ad-hoc groups for class assignments for example.

**Group evolution:** There are always some problems regarding grouping during the execution of a course and it is always necessary to change the constitution of some groups, delete existing groups or create new ones. This task should also have a low overhead from the point of view of the teachers.

**Individual vs group:** A course can have individual and/or group projects. The system should support both types of projects on a transparent way for teachers and students. When a student logs in the system, he has access to all group projects of his group and to his individual projects.

**Schedules:** It is desirable the system can be aware of the course's schedules and shifts including theoretical and practical classes and which teachers lecture each class. Each group should be related to a practical class shift.

**Submission of files:** The source code of the student's solution has to be able to be uploaded remotely to the system. If the project the student wants to submit is a group project, any member of the group can submit it. Each submission must be timestamped.
As there are always some last minute e-mail submissions 4 or 5 minutes after the deadline expires, the system should allow a teacher to submit a student or group project after the deadline has ended.

**Submission deadlines:** Each system has a deadline. The deadline can be static (a specific date and hour) or variable (until the start or the end of the shift the student/group is enrolled of a given week).

**Access to prior submissions** The system should be able to save all the submissions the students have made and their correspondent grade (if it has already been evaluated). The students should be able to access all of their submitted programs and the correspondent results of the automatic evaluation. The access to the submitted programs is important since sometimes students "lose" their local copy of the submitted programs.

**Grades:** The system must be aware of the concept of grades. The faculty of the course must be able to specify how are the projects graded (for example: 70% automatic evaluation and 30% the quality of the solution and of the code manually graded by a teacher).

**Specify automatic tests:** The system should support the specification of automatic tests for each project to assert the correctness of the students' code. Each test must be composed by one or more input files and one expected test output file. The system evaluates a test over a student project by submitting the input files related to each test to the compiled project and verifying if the project executed without errors and the expected result is equal to the obtained output. The teacher should also be allowed to specify which tests are to be run on submission, which are to be run after the deadline when the system grades the projects and which are to be run on both situations. Teachers might not want to allow the difference between the intended output and the actual output of the execution of a test to be shown to the student, only if the test passed or not, so the system must also allow to change this visualization setting.
Each test has an upper time limit and might have an upper limit in process memory size. Test specification must be simple and fast. Teachers should only need to spend a few minutes to specify the tests for a project.

**Run automatic tests:** The main purpose of the system is to run a series of tests for each project submitted by the students and grade them accordingly to the number of passed tests and their respective weight. The grades are given after the deadline has ended in a final evaluation of all the last submissions from students with the offline tests. When the tests are run before the deadline ends, only the submission tests are executed and no grade is applied, the system only needs to show which testes passed and which did not.
For each test the outcome of its execution may either be correct or there may be some kind of execution error. This execution errors may vary in seriousness and some kind of errors can still give a smaller grade like a presentation error in which the output is correct with the exception of a whitespace.
A security mechanism has to be provided for disallowing students to access (or transfer them to a remote location) the test input files in order to optimize their code for those tests or to have the correct answer hardcoded.
If the system cannot give an immediate evaluation result of a submission (because of having more submissions to be evaluated before it), the system should indicate an estimated time for the result being available and when it is available warn the student/group of it by e-mail.

**Language independent:** The system should be able to evaluate programs written in any language, provided they can be compiled and executed from the command line. For each project the teachers should be able to specify which programming languages would be allowed.

**Submission evaluation strategies:** The system must support various evaluation strategies to be applied to the submitted projects and be able to change them at runtime. These strategies could be for example: receive project compile and execute it against the tests; receive project and compile; receive project and put it in a FIFO queue to be compiled and tested; limit the number of submissions per student/group. etc. This would be useful to control the load of the system and to adapt the system to several types of assignments for the students.

**Rank student solutions:** In order to promote a healthy competition among the students, the system should rank their solutions according to several criteria: number of tests passed or the grade that the submission obtained, number of submissions (lowest number the better, provided it passes the tests), faster solution, lowest memory used, etc. By ranking the projects, we hope that students submit projects with an higher quality so that they have a good classification on the ranking.

**Manual grading:** Teachers must also be allowed to download the projects of the students that belong to their practical classes in order to be able to do the non-automatic evaluation of the code.

**Visualization:** Apart from the features previously introduced which already impose some user interface requirements, there are some other UI requirements that do not fit any of the requirements previously listed.
Teachers should be able to view and sort student's submissions in a hierarchy composed from top to bottom by:

**Top Level:** Shows all the projects and allows each one to be evaluated, to download the last submission of every group for that project.

**Project:** For each project selected in the upper level, this view should list all the class schedules and allow each one to be evaluated, download the last submission of every group for each class schedule. Optionally, the class schedule division sorting can be removed and all the groups for each project would be visible.

**Class:** For each class, the groups belonging to that class are to be shown with the same options: evaluate last submission and download it.

**Group:** Finally, in the lower level of the view hierarchy, all the submissions of each group are to be shown with the same actions as stated on the hierarchies above, but instead of being applied to the last submission, applied to that submission.

**Practical evaluation:** There are some courses in which a practical examination is done after the project is submitted to assert if the student knows the project code, structure and in the limit, how to program. This is particularly important in projects done in groups. Therefore, support for this kind of evaluation is needed. Students should be able to download their last submission of the project and submit it again with the changes needed to run the new tests within the deadline for the examination.

### 1.3.2 Non-functional Requirements

In order to support the functionalities described in Section 1.3.1, the following non-functional requirements are desirable:

**Accessibility:** The system should be accessible from a wide variety of locations, being desirable to have a web interface for both student and teacher interaction.

**Adaptability:** The system must be adaptable. It should be easy to integrate the system with other systems like Fénix for example. Fénix along with other functionalities knows the enrolled students on the courses. This way, it can be used to handle grouping for instance.

**Auditability:** The system, being used in official courses, must be auditable. For that end the system must log all user interactions with it like for example: date and time of the submission along with what files were submitted and by whom it was submitted.

**Availability:** The system must be able to accept project submissions even if under heavy load. This may lead to a variation of the Quality of Service taking into account the load of the system. If the system is too loaded the system can automatically change the evaluation strategy to a less demanding one in terms of CPU requirements in order to guarantee student's submissions are always received on time. In the limit, the system would only accept submissions and leave the evaluation for later.

**Extensibility:** The system must be easily extended for new functionalities. It should be easy to add a new evaluation strategy for when the system is under heavy load for example.

**Performance:** Under normal load the system should be able to run student's projects without any performance hit.

**Scalability:** It is desirable that the system can scale to support a situation where the number of submissions grows more than it was intended in the first place.

**Security:** The system must have secure authentication. The system cannot allow a student to see another student's source code.
In order to protect the execution of the system and the content of the tests a student's program must be executed in a sandbox. This way, if the submitted code has malicious instructions it should not be able to affect the normal behavior of the system or corrupt data.

## 1.4 Outline

The document is organized as follows. Chapter 2 presents the State of Art, describing some systems that have similar purposes of automatic evaluation, project submission and/or course management. Chapter 3 shows the architecture followed to create our evaluation system and Chapter 4 details how the system was implemented, the parts that compose it alongside a rationale on why some decisions were made. Chapter 5 details the methodology of how the system was evaluated. Finally, Chapter 6 presents the conclusions and the proposed future work.

# 2 State of the Art

## 2.1 Introduction

This chapter presents an overview of the main existing e-learning and automatic program evaluation systems. This division is made since their goals are intrinsically different. At the end of the section, all the systems are compared between each other.

Automatic Evaluation Systems are systems able to receive the source code of programs created as solution to a given problem, compile and execute them with a set of tests and assert the amount of tests passed by the user's solution. This kind of systems are used to support the realization of programming contests such as the ACM-ICPC[1] and its national and regional sub-contests, and in several programming courses throughout the world. Three contest systems and an academic evaluation system will be introduced in this section: Mooshak (Section 2.2), PC$^2$ (Section 2.3), UVa Judge (Section 2.4) and PO Eval (Section 2.5). A brief description of e-Learning Systems is also presented (Section 2.6) along with a fully fledge Academic Management System (Section 2.7). Finally we present the conclusions taken of the study of the State of the Art (Section 2.8).

## 2.2 Mooshak

Mooshak[2] is a web-based programming contest manager and automatic judge (Leal and Silva 2001a; Leal and Silva 2001b) developed by João Paulo Leal and Fernando Silva at the Faculdade de Ciências of the Universidade do Porto.

Having been conceived as a contest manager, Mooshak allows the registration of teams, each of which has a single password. Mooshak supports an HTTP interface. This way users can access the system using a browser.

When Mooshak is evaluating a student's solution, the automatic judging functionality is divided in two main phases: static analysis and dynamic analysis. The static analysis phase verifies the team is valid, which programming language is the solution coded on, what is the problem the team is trying to solve with this submission, checks if the program size does not exceed a predefined reasonable size (security feature) and sees if the solution compiles properly. If all static tests pass, the system enters in the dynamic analysis phase which has to deal with the execution of the compiled code running some test cases according to the problem the code is trying to solve. These tests consist in an input and an output file. The compiled program is executed and is passed the contents of the input file through the standard input (*stdin*) and its

[1] ACM's International Collegiate Programming Contest - http://cm.baylor.edu/welcome.icpc
[2] http://mooshak.dcc.fc.up.pt/

output (*stdout*) is compared with the test's intended output. The execution of the tests can have several outcomes:

- **Pass:** The output of the program is same the expected one.
- **Time-limit exceeded:** At least one test of the test suite took more time to execute than its predefined time limit.
- **Output too long:** The output of at least one of the tests is longer than intended.
- **Runtime error:** The execution of one of the tests exits abnormally.
- **Wrong Answer:** The answer of one of the tests is not the desired one.
- **Presentation error:** The answer is correct, only white spaces of at least one of the tests differ.

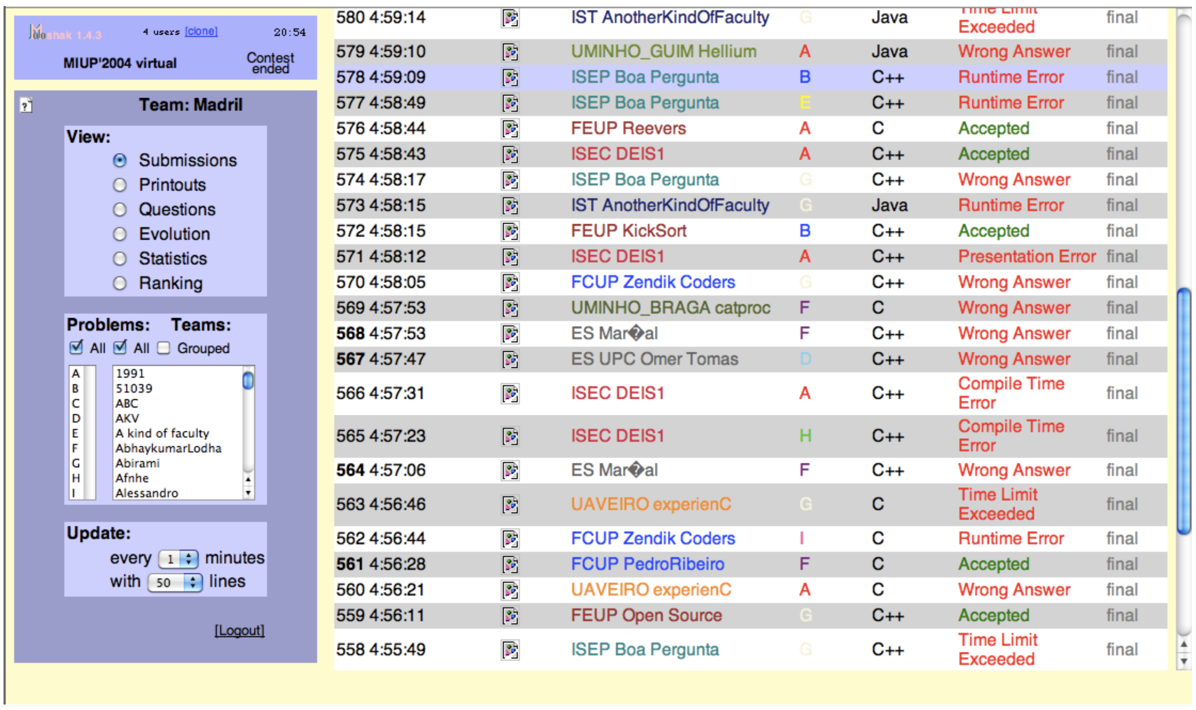
Compilation and execution of the submitted code are the most insecure parts of this process, therefore they are done in a secure environment with several restrictions and limits enforced.

Mooshak was developed using the Tcl scripting language, and is deployed in Apache with CGI. Instead of using a database system to store the persistent data of the system, the creators decided to use objects saved directly in the filesystem naming them Persistent Objects. For replication and scalability purposes rsync[3] is used since it allows fast and incremental file transfer and synchronization between various systems. It was created with a scalable architecture in mind where it can go from a single-site single-server layout up to a multi-site multi-server one able to securely host a big distributed international event.

From the requirements detailed on Section 1.3, we can see Mooshak does not comply to several of them. It doesn't support grouping in an easy way, i.e in order to support a group of two students, each of the students would have to log in individually for the individual assignments and then have a different login for the group. This has more implications than only having to remember the second username and password. For a teacher it would imply having to manually pair which students belong to each group to assign them the grades. Which leads us to another characteristic Mooshak lacks, the handling of Grades. For Mooshak a test is either a pass or a fail, in any of its types like detailed previously. There can not exist more important tests than others, have dependent tests nor grade them. This system can not handle schedules either, only final deadlines for the projects. Finally there are no variable evaluation strategies and the tests can not have multiple input and multiple output files.

This system is used in some university courses around the world such as Análise e Síntese de Algoritmos here at IST and in the Algorithms and Data Structures course (García-Mateos and Fernández-Alemán 2009) in the University of Murcia. In the latter they used the system with the main intent of reducing student dropout which was at 70% at the time. To try to reduce the dropout rate and in order to increase the students' self-learning initiative and interest in the course they changed the evaluation model from a monolithic approach with a big programming project and an end of course exam to a more flexible and interactive continuous evaluation approach with smaller programming assignments (individual and in groups) and

[3]rsync is an utility that provides incremental file transfer. http://rsync.samba.org/



MIUP'2004 virtual — Contest ended

Team: Madril

View:
- [✓ Submissions]
- Printouts
- Questions
- Evolution
- Statistics
- Ranking

Problems: [✓ All] Teams: [✓ All] Grouped

| | |
|---|---|
| A | 1991 |
| B | 51039 |
| C | ABC |
| D | AKV |
| E | A kind of faculty |
| F | AbhaykumarLodha |
| G | Abirami |
| H | Afnhe |
| I | Alessandro |

Update: every 1 minutes with 50 lines

[Logout]

| | | | | | | |
|---|---|---|---|---|---|---|
| 580 | 4:59:14 | IST AnotherKindOfFaculty | G | Java | Exceeded | final |
| 579 | 4:59:10 | UMINHO_GUIM Hellium | A | Java | Wrong Answer | final |
| 578 | 4:59:09 | ISEP Boa Pergunta | B | C++ | Runtime Error | final |
| 577 | 4:58:49 | ISEP Boa Pergunta | E | C++ | Runtime Error | final |
| 576 | 4:58:44 | FEUP Reevers | A | C | Accepted | final |
| 575 | 4:58:43 | ISEC DEIS1 | A | C++ | Accepted | final |
| 574 | 4:58:17 | ISEP Boa Pergunta | G | C++ | Wrong Answer | final |
| 573 | 4:58:15 | IST AnotherKindOfFaculty | G | Java | Runtime Error | final |
| 572 | 4:58:15 | FEUP KickSort | B | C++ | Accepted | final |
| 571 | 4:58:12 | ISEC DEIS1 | A | C++ | Presentation Error | final |
| 570 | 4:58:05 | FCUP Zendik Coders | G | C++ | Wrong Answer | final |
| 569 | 4:57:53 | UMINHO_BRAGA catproc | F | C | Wrong Answer | final |
| **568** | 4:57:53 | ES Mar�al | F | C++ | Wrong Answer | final |
| **567** | 4:57:47 | ES UPC Omer Tomas | D | C++ | Wrong Answer | final |
| 566 | 4:57:31 | ISEC DEIS1 | A | C++ | Compile Time Error | final |
| 565 | 4:57:23 | ISEC DEIS1 | H | C++ | Compile Time Error | final |
| **564** | 4:57:06 | ES Mar�al | F | C++ | Wrong Answer | final |
| 563 | 4:56:46 | UAVEIRO experienC | G | C | Time Limit Exceeded | final |
| 562 | 4:56:44 | FCUP Zendik Coders | I | C | Runtime Error | final |
| **561** | 4:56:28 | FCUP PedroRibeiro | F | C | Accepted | final |
| 560 | 4:56:21 | UAVEIRO experienC | A | C | Wrong Answer | final |
| 559 | 4:56:11 | FEUP Open Source | G | C++ | Accepted | final |
| 558 | 4:55:49 | ISEP Boa Pergunta | G | C++ | Time Limit Exceeded | final |

Figure 2.1: Mooshak being used in the MIUP 2004 event.

faster feedback. For that they resorted to Mooshak to allow students to submit their assignments and get instant feedback on whether their solution is correct or not allowing to make faster corrections. This system also allows the creation of a healthy competition between students, since their submission results (passed or error) are public. By comparing the results with the previous year where they used the monolithic approach described earlier, they got the dropout rate down from 70% to 45% and increased student satisfaction according to a survey made at the end of the course.

## 2.3 PC²

PC² is the acronym of the Programming Contest Control System[4] developed at the California State University in Sacramento. The system was created to support programming contests. The main difference to the other systems is that users access the system through a Java desktop application installed in all contestants' computers. PC² does not support an web interface which is a big disadvantage since it is needed to install a local application in all computers that need access to the system. The contestants receive their problems in the client application and then submit their solutions to the local application which will in turn send them to the judges' server application over the local network.

In its early versions the system lacked automated judging capabilities and required several people to run the contests since the judges had to manually compile, execute the submissions

[4]http://www.ecs.csus.edu/pc2/

against the tests, see the results and send optional text feedback back to the contestants. In newer versions it includes a default validator (Samir E. Ashoo 2008) (which can be changed to another one) that accepts execution output of the submitted code and asserts if it is the expected or not. PC$^2$ supports any programming language provided it can be compiled from a command line and it generates an executable file.

Being local area network and contest oriented it does not comply to many of the requirements for the intended system, only complying with the file submission and part of the automatic testing desired features.

## 2.4 UVa Judge and EduJudge

The University of Valladolid's Online Judge[5] is one of the oldest programming contest training sites for ICPC style contests and is also used for self study (Revilla, Manzoor, and Liu 2008). They also have an extensive Problem Set available[6]. Revilla goes through the history of the UVa Judge system which started as a series of Unix shell scripts and evolved from that to a more robust system able to run 24 hours a day without the presence of a system operator. They then had to drop shell scripts because they were not reliable enough to support a true judging system since they could not control the memory used by the contestants' programs nor was it made to generate status reporting events. On the first versions, contestants submitted their applications by e-mail. Some new iterations of the system are enumerated like the inclusion in 2002 of a SQL database and PHP tools to manage the interface and allowed HTTP updating of the files, turning the system into a client-server system with a web interface.

The system lacks several important requirements, such as grouping, proper grading, class schedules and evaluation strategies, only barely complying with the automatic evaluation related requirements.

EduJudge[7] is an European supported project which has as its main goal the improvement of UVA Online Judge into an effective educational environment able to use said judge in official courses from secondary and higher education on the area of mathematics and programming (Revilla, Manzoor, and Liu 2008). The objective is to integrate the system into Moodle and create a repository of problems. José Paulo Leal and Ricardo Queirós say (Leal and Queirós 2008) the EduJudge project forsees an integration with various LMSs[8], Evaluation Engines and Learning Object[9] Repositories running at the same time. The project is in its initial steps and there is not very much information available.

## 2.5 PO Eval

The current system[10] used at the Object-Oriented Programming course at Instituto Superior Técnico has been developed for some years by the course's faculty in Java and uses a MySQL

[5]http://uva.onlinejudge.org/

[6]http://acm.uva.es/problemset/ - Compilation of several contest-like problems for the contestants to train

[7]http://www.edujudge.eu/

[8]Learning Management Systems like Moodle, Sakai, LAMS, Dokeos, etc. Introduced in Section 2.6

[9]Object which contains a problem description and tests to prove the correctness of the solution

[10]The system's features were described in person by both my supervisors and with the help of Prof. David Matos, all lecturers of the course and with deep knowledge of the system and its quirks.

database. The system was created with the intent of automating project submission and evaluation.

The system handles schedules, students and groups sensibly in the way described in the requirements and it had hooks to Fénix (for authentication and for fetching group information about logged student) in a previous version, before Fénix suffered a major refactoring which broke many of these hooks. Projects can be either individual or in groups and are associated with a deadline. At the time of submission, some tests - named online tests - are run and the result is displayed to the students. Some offline tests are only run after the deadline and only on the last submission by means of a command-line launched script. Both types of tests are based in input and output, also being time limited. The system at the moment only supports Java programs, but it is adaptable to C through some code changes.

There are two kind of web interfaces with the users, one for students and one other for the faculty of the course. Students can see all the projects, their submissions for each project, and if the deadline has not still expired, they can submit a solution. On the other side, a teacher is related with a shift lectured by him and for evaluation purposes can download either a zip with the last submission of all groups for a given project or only for each of the shifts he or her lectures. For testing purposes a teacher can simulate he is a student and can submit projects (not limited by deadlines) and get them evaluated to check if the results are the expected.

This system had several performance and stability issues under load which lead to constant crashes and needing of relaunching/rebooting. One of the problems was related to the JVM quitting garbage collecting after a few submissions and starting using more memory than the available since old classes were not removed from memory, even if the project was already evaluated a long time ago. Some of the other problems the system has include not being possible to give a student their submitted source code from the web interface, crashing when the student's code threw some kind of exceptions and the difficulty of specifying tests.

## 2.6 e-Learning Systems

There are also some information systems, known as Learning Management Systems or e-Learning Systems, created with the main goal of supporting courses. Despite not having automatic evaluation of computer programs, these systems have some functionalities that fit the requirements. Some examples of these systems are LAMS[11] (Dalziel 2003) which is focused on learning workflow creation, Moodle[12], ATutor[13], Sakai[14] and Dokeos[15]. Dokeos is more targeted to corporations and governments and has a special version for medical training.

All these systems share a common set of features that include among others authentication, grouping, support for several roles such as teacher and student, collaboration tools (such as chat, forums, wikis and peer review of papers), evaluation tools (such as quizzes and tests), deadlines, submission of work related files and support grading. Being general purpose systems, none has automatic evaluation capabilities for student's software programs, only allowing the uploading of files on deadlines.

[11]http://www.lamsfoundation.org/
[12]http://moodle.org/
[13]http://www.atutor.ca/atutor/
[14]http://sakaiproject.org
[15]http://www.dokeos.com

The systems previously mentioned all follow the same basic web-based client-server architecture, either developed in Java or PHP and connected to a SQL-like RDBMS.

## 2.7 Fénix

FenixEDU[16] is a School Management Information System developed at Instituto Superior Técnico of the Universidade Técnica de Lisboa and used at some other academic institutions. It was first created as assignments for a course (taught by Prof. Rito da Silva) with the idea of having a course management system and grew to become a school management system supported by the university. It is also used as a research playground for MSc and PhD students and some innovations have been tested and applied on it.

The functionalities of the system that are interesting to this thesis are the course management ones. Fénix is designed to handle course enrollment, class schedules, class enrollment, grouping, examination dates, project deadlines, project submissions though it has no automatic evaluation, course announcements and final grades (passed then to the academic services).

Fénix is a Web-Based client-server system developed in Java that uses MySQL for storage purposes. It uses the Fénix Framework[17] that provides a Domain Model Language and a Software Transactional Memory system (JVSTM - Java Versioned Software Transactional Memory (Cachopo 2007; Cachopo and Rito-Silva 2006)).

## 2.8 Conclusions

A good way to better compare the capabilities of the automatic evaluation systems described in this section and to see to which requirements they answer is to place them side by side in a table. For simplicity sake, we consider that systems can comply (Yes), not comply (No), partially comply (+/-) with the requirements or their performance according to the requirement is unknown (?).

There are some capabilities of the systems that are not publicly available or hard to classify mainly in the non-functional requirements field. Being the PC$^2$ system composed of a group of desktop applications intended to run on a small network with few clients, it may not comply by design with many of the requirements. All the applications described on this section are at their core client-server applications. All except PC$^2$ offer a web interface.

Table 2.1 shows the comparison matrix. Analyzing the results, we can conclude there does not exist any system that satisfies all the requirements (laid down on Section 1.3), though some comply with several of them. As noted in their respective sections, Fénix complies with great part of the management requirements and Mooshak does a good job concearning the automated evaluation of tests.

[16]https://fenix-ashes.ist.utl.pt/
[17]Framework developed in IST and available at https://fenix-ashes.ist.utl.pt/trac/fenix-framework

| **Requirements** | **Mooshak** | **PC$^2$** | **UVa Judge** | **Fénix** | **PO Eval** |
|---|---|---|---|---|---|
| *Functional Requirements* | | | | | |
| **Authentication** | Yes | Yes | Yes | Yes | Yes |
| **Grouping** | No | No | No | Yes | Yes |
| **Schedules** | No | No | No | Yes | Yes |
| **File Submission** | Yes | Yes | Yes | +- | Yes |
| **Deadlines** | Yes | Yes | Yes | Yes | Yes |
| **Access Prior Submissions** | Yes | ? | ? | Yes | No |
| **Grades** | No | No | No | +- | +- |
| **Specify Automatic Tests** | +- | +- | +- | No | +- |
| **Run Automatic Tests** | Yes | Yes | Yes | No | Yes |
| **Language Independence** | Yes | Yes | Yes | No | +- |
| **Evaluation Strategies** | No | No | No | No | No |
| **Roles** | Yes | Yes | Yes | +- | Yes |
| **Visualization** | +- | +- | +- | No | +- |
| **Practical Examination Support** | No | No | No | No | No |
| *Non-Functional Requirements* | | | | | |
| **Accessibility** | Yes | No | Yes | Yes | Yes |
| **Adaptability** | ? | ? | ? | ? | +- |
| **Auditability** | Yes | Yes | Yes | Yes | Yes |
| **Availability** | +- | ? | ? | Yes | +- |
| **Extensibility** | +- | ? | ? | +- | +- |
| **Performance** | Yes | ? | ? | ? | No |
| **Scalability** | Yes | ? | ? | +- | No |
| **Security** | Yes | ? | ? | Yes | +- |

Table 2.1: Comparison Matrix

# 3 Architecture

Having raised the requirements for the system on Section 1.3 the first step to start developing the system was finding an adequate architecture in order to support it. It was clear early in the process the system would work in a Client-Server style on the Component-and-Connector Viewtype and display a Layered style in the Module Viewtype according to the definition of Viewtypes provided on (Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, and Stafford 2008). This derives from the need to have a system that supports several simultaneous users (either teachers or students) that can access the system remotely via a web browser.

## 3.1 System's Architecture

Figure 3.1 shows the architecture of an evaluation system and its access of external sources. In this section, we will discuss what each part does, why does it exist and map it into the layered architecture of the system.

The requirement of the system having to have high availability in order for the students to be able to submit their projects, made us choose to decentralize the two main functions of the system - submission and evaluation. This makes us be able to consider the the system as being composed of two theoretical subsystems, one for each function. This way even if the evaluation subsystem has some problem or is under load, the submission subsystem is unaffected, allowing students to submit their solutions without problems which from the point of view of both students and teachers is the crucial activity. If students are not able to submit their projects (or new versions) they will not be evaluated. If the files have been submitted in due time (before the project deadline ends) to the Submission Subsystem, they will eventually be evaluated.

The Submission Subsystem is responsible for storing the several versions of the projects submitted by students. Currently we have two types of submission systems: a Version Control System and the file system. In the former case we support the CVS version control system but the code was made in order easily support other types of version control systems (see Section 4.4.1 for details). In the second case, we use the Fénix system to store the submissions made by the students and then teachers have to download the last submission of each group into the file system and then the evaluation system automatically handles the evaluation of those submissions.

The main component of the evaluation subsystem is represented by the Evaluation Manager Component. This component is responsible for managing all the activities and data concerning the evaluation of projects and also for managing the results of those evaluations. Basically it implements the domain logic of our application and coordinates the access to the other components of the system.

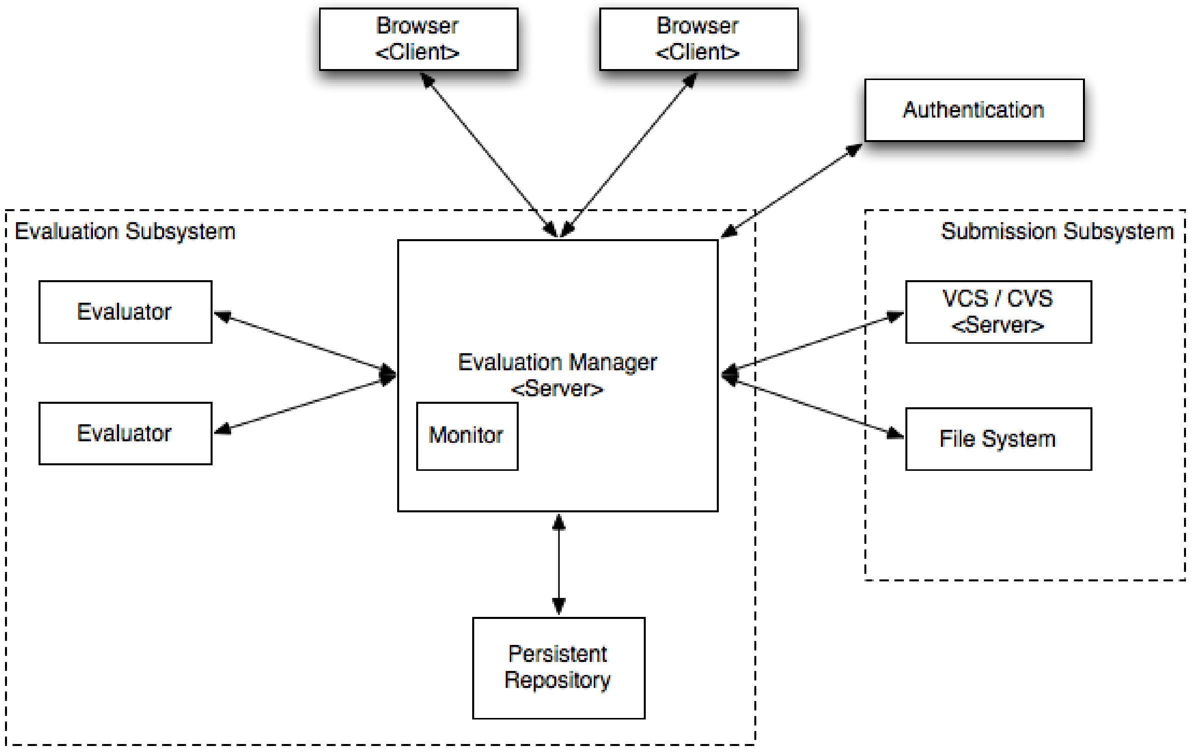


Figure 3.1: The System's Architecture

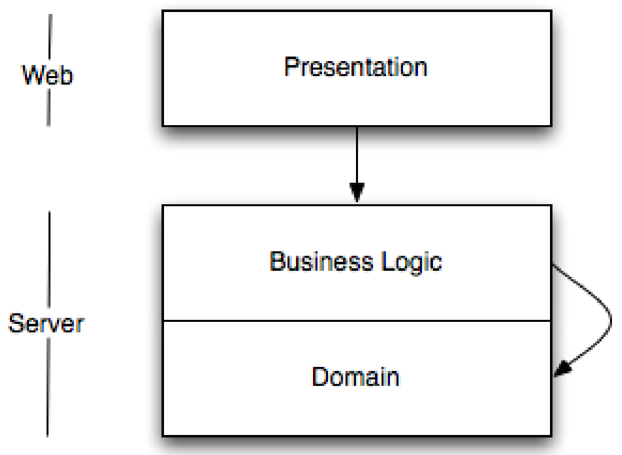


Figure 3.2: Layered Architecture used in the System

The authentication is handled by a module that contacts an external server in order to validate the user's credentials. This module can be easily extended to support various types of external authentication services or to use built-in authentication validation.

The Evaluation Manager component also has a module named Monitor that manages the projects to evaluate. It handles the queue of work (i.e. projects to evaluate) to do and dispatches it to the Evaluators. In order to be scalable and be able to support a high load of submitted projects, the projects are not directly evaluated by the Evaluation Manager Component. The evaluation of a project is made by one Evaluator component. The Evaluators can be local and/or remote and we can have several Evaluators subcomponents to allow the system to be able to evaluate more than one submission at the same time provided there is available computational power (more cores/processors or machines) distributing and controlling the load of the system. Their function is further explained on Section 4.2.2.1.

Finally, the Presentation Layer is created using web technologies and is remotely accessed by Web Browsers the users are using in order to communicate with the system.

## 3.2 Application's Layered Architecture

The Evaluation Manager subsystem presents a layered architecture having three layers: the presentation, the business logic and the domain.

Figure 3.2 represents a Layered Architecture where the Business Logic and Domain Layers are located in a server and the Presentation Layer is located wherever the client is accessing the server to get and send information through an Internet Connection. This type of architecture was chosen because we had the need to save persistent data (provided by the Domain and Persistency Layer, Section 3.2.1), provide ways to manipulate that same data (Business Logic and Services Layer, Section 3.2.2) and then present the results of said manipulations or the data itself to the end user of the system who is not on the same machine where the system is running

(Presentation Layer, Section 3.2.3). The layers visible on this system have their interactions well defined. The Business Logic and Services uses the entities from the Domain Layer and provide services for the Presentation Layer to use. The Presentation Layer cannot directly access the Domain Layer.

### 3.2.1 Domain and Persistency

This layer contains the domain entities of the system that model the problem. It also handles the instances of said entities (the actual system's data) and where and how they are saved persistently. This is the basis of the system, and the system is built upon this layer. No business logic is contained on this layer, only how the domain entities are related to each other and how to retrieve and access data.

The separation of this layer from the Business Logic layer allows for an easier change of how the system saves the data without affecting the layers above provided the interfaces are maintained.

### 3.2.2 Business Logic and Services

As mentioned before, the Business Logic and Services layer is where the system's data located in the layer below, the Domain and Persistency Layer, is created and manipulated following the rules dictated by the defined business logic. This is also the layer which the Presentation Layer has to accesses to get information and make the system do something. Every action made on the system is made through a Service contained on this layer. A service can contain the business logic itself, orchestrate other services and/or utilize functionalities located in other modules.

### 3.2.3 Presentation

Finally, the layer that presents the information to the users (in our case either teachers or students) and allows the users to interact with the system is the presentation layer. The presentation layer interacts with the layer below, the Business Logic Layer, through Services and Views. The services were already introduced in the previous section. The views are passed to this layer through the invocation of services and are a representation of the data of the system.

Having this layer decoupled from the rest of the system allows the existence of several presentation methods being one of them a web interface which allows for the accessibility requirement to be fulfilled since this way the system can be accessed from anywhere provided there is an Internet connection (or network provided the user is in the same network than the system). Theoretically, although given the ubiquity of web browsers in all sort of platforms the desire and need for this options is reduced, this design solution also allows for different presentation clients to be created such as command line applications, native mobile and desktop applications, etc.

# 4 Implementation

## 4.1 Introduction

In order to create a system that complies with the requirements raised in Section 1.3 our approach was to use a familiar programming language (for the stakeholders) and a straightforward architecture (presented briefly on Chapter 3) for it to be easily maintained and extended.

We chose to use Java as programming language since it is very widespread and has a big user base worldwide, which makes it easier to find people able to maintain and extend the system if needed.

The system separates two big functionalities: submission and evaluation. The submission is the act of a student or group of students to upload their project to the system and the evaluation functionality consists in evaluating said submission following some rules and passing given tests. In order of not having a single point of failure on the system, our approach was to decentralize and separate both functions, offloading the submission part to other systems usually running in other machines and to provide the means for the evaluation system to import submissions on demand.

The current version of the system supports two modes of submission: From a Version Control System (it currently supports only CVS, but it is easily adaptable to other VCS's like git[1]) or importing from a folder which allows teachers to use the universities' e-Learning or management platforms (like Moodle or Fénix) to handle the project submissions. Both approaches serve our availability requirements because the SLA[2] universities' systems provide are usually very good[3] and also allows for an offloading of responsibilities and work from the faculty of the course to specialized teams that run these systems.

This chapter details the parts composing the system, their functions and how they interact with each other, as well as the framework used. A brief outline is presented next:

We chose a layered architecture along with some extra modules as the architecture able to support the system we proposed to create. This topic is explained on Section 4.2. Having chosen the main architecture to use and the programming language, we decided to use a homebrewed framework created in the Software Engineering Group of INESC-ID and in the Fénix Project of the Informatics Centre from Instituto Superior Técnico. This framework is further detailed in Section 4.3. A rationale of the two modes of file submission currently supported

[1]Git is a Distributed Version Control System used by many famous projects like the Linux Kernel, Eclipse, Ruby on Rails. http://git-scm.com/

[2]Service Level Agreement

[3]This applies to the institution from where this thesis comes from since the hole institution runs on Fénix and the cluster of servers where the CVS server is running is also run on a fail-safe and redundant way by CIIST - Informatics Center of Instituto Superior Técnico. CIIST also provides hosting to other machines, but the QoS SLA is not that good, making the better solution to use Fénix or CVS running in their servers.

by the system and how the choosing of one of them varies the *modus operandi* of the system is presented on Section 4.4. Finally, an explanation of the Evaluation Process is presented on Section 4.5.

## 4.2 Layered Architecture

Using the layered architecture view presented on Chapter 3, we can now detail what each of the layers contains on this system.

In order to automate some of the functions of the system, a couple of modules had to be added detouring the architecture from a straight-forward and by the book layered architecture in which calls origin from the user. These modules are the modules responsible for managing the queue of submissions to evaluate and the worker threads that evaluate the projects. These modules are always running and can also be accessed through services. Every interaction with data in these modules is made through services in order to maintain data consistency. They are presented in Section 4.2.2.1.

### 4.2.1 Domain

As mentioned in the corresponding section of the Architecture chapter (3.2.1), the Domain layer represents the entities modeled by the system and whose instances (the data) are saved persistently. The method and where this data is saved is further explained in Section 4.3.

The domain entities of the system are represented in Figure 4.1. The root class of the domain which represents the system itself is the ASEPPApp class. From there on, the system models Courses[4]; Students, if they are enrolled in a given Course and if they are organized in Groups[5] on a given Course; Projects belong to Courses and contain the information of when submissions can start to be accepted and evaluated, if the project is a group project or an individual project, the submission deadline date and time and how are the submissions to be compiled, executed and tested. These Projects have associated a set of Tests that are used to evaluate them. The entity that represents the submissions for a given project of a given student or group is called Submission. Students' Submissions are first compiled, producing a Compilation Error if said compilation fails, and then evaluated according to the Tests producing one TestEvaluation instance for each Test ran which contains the result and stores the output of the execution of the Test if said test is failed. Their grade for the given Submission is the sum of the passed TestEvaluations. Courses might also contain Shifts for practical classes or laboratories to which Students belong and might be used to limit Submission deadlines instead of a single deadline for everyone[6] if the responsible Teacher chooses so. Finally a Login class exists which is the basis for a possible local authentication functionality not yet implemented.

[4]Although the first prototype uses only one course, there is already preliminary support for using the system in a multiple course scenario with few changes.

[5]Due to a framework bug which causes a MySQL error, this entity cannot be represented by the word Group, being instead named Grupo.

[6]Might be useful for homework evaluation.

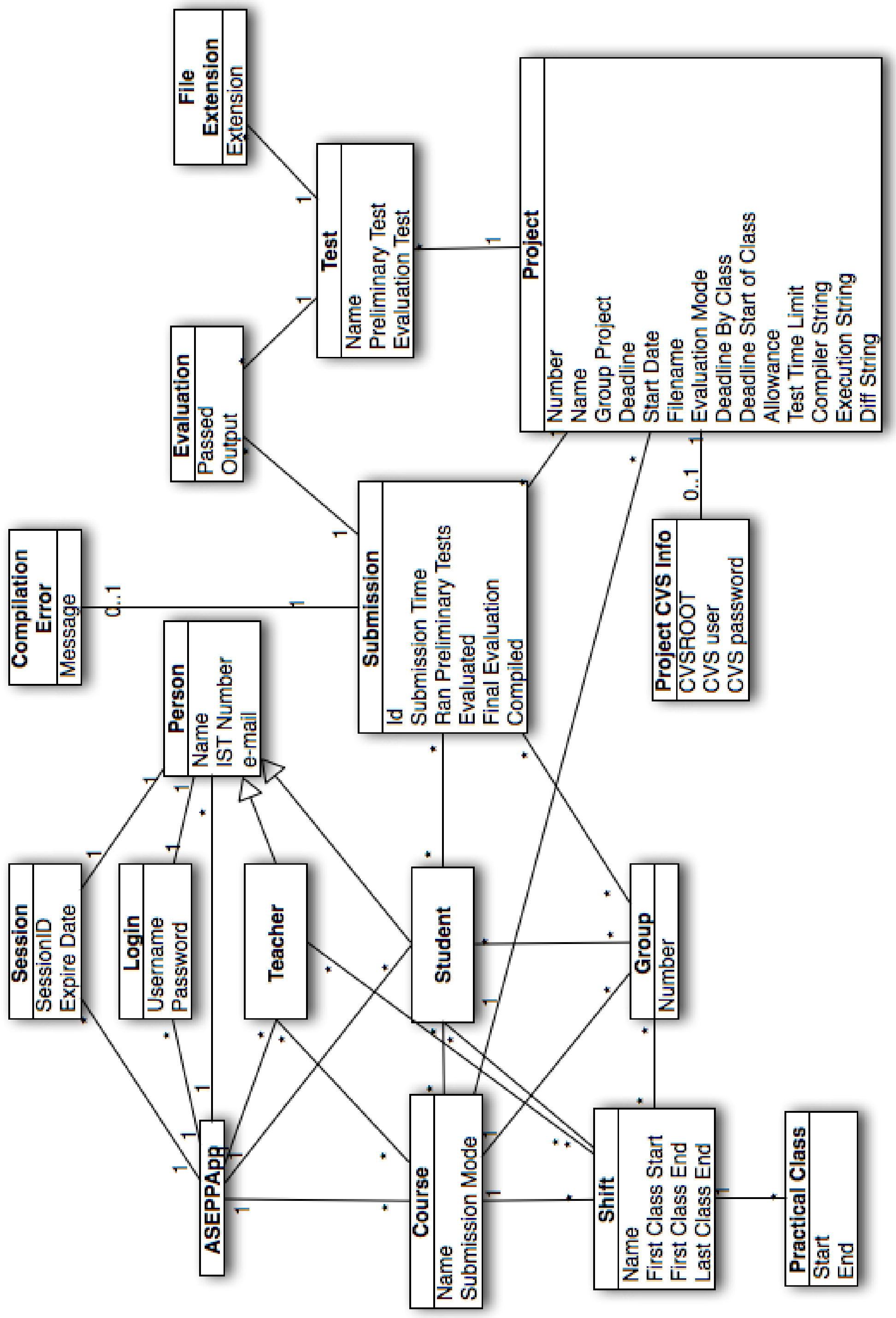


Figure 4.1: The System's Domain Model

### 4.2.2 Business Logic and Services

As mentioned on Section 3.2.2, the Services layer is where the system's data is created and manipulated, in other words, it is the layer that contains the Business Logic. Every action made on the system is made through a Service contained on this layer.

Figure 4.2 represents the services implemented in this layer to support the functionalities of the system. All the services are subclasses of the Service class. These services can consist in functionalities as trivial as adding a new Student to the system (**AddStudentService**) to more complex ones like importing a list of groups and students to the system and enroll them in the selected course in the case they don't exist or change their groups in case they already exist (**ImportGroupWithStudentsService**). In the case of the latter service, it uses the former service for each student found in the import order to register him in the system, enrolls him in the course and makes him part of the desired group. Some of the most relevant services are described bellow:

**AddProjectService:** This service creates a new Project in the system. A Filesystem folder is automatically created which will contain the tests and the associated libraries needed for the execution of the project. Depending on the mode the system is in, the VCS repository for the project is also inserted or a File Import folder is created[7]. This service receives as inputs the project name, deadline, start date, if it is a group project or not, the compilation, execution and evaluation strings, etc. It is further explained in Section 4.5

**AddSubmissionToQueueService:** As the name implies, this is the service used to add a submission to the evaluation queue. This is explained in more detail in Section 4.2.2.1

**AddTestsFromFolderService:** Service that adds the selected tests to the selected project marking them as Preliminary and/or Evaluation Tests. This service is used in conjunction with the **ShowTests ToAddFromFolderService** which lists the tests existing in the Project Folder. This service is explained in detail in Section 4.5.

**EvaluateSubmissionByIDService:** This service is a service that is called not from the presentation layer directly, but from a separate module called Evaluator included in the Business Logic, further detailed on Section 4.2.2.1 in order to evaluate a specific student submission. This service is invoked once per Submission either in File System or VCS mode. It is also invoked to determine the final grade for each group.

**ExecuteFinalProjectEvaluationService:** It is a service available in VCS mode that orders the system to make the final evaluation of a given project. This will fetch the last submission for each group or student from the VCS server, mark it as a final submission and add it to the evaluation queue. In File System mode this service is not used.

**GenerateDownloadableFileWithSubmissionsService:** For a given project, returns the final submission for every student/group in a zipped folder. This service is available only to teachers.

**GetNextSubmissionFromQueueService:** Service that returns the next submission in queue to be evaluated.

[7] These modes were introduced earlier and will be further explored in Section 4.4

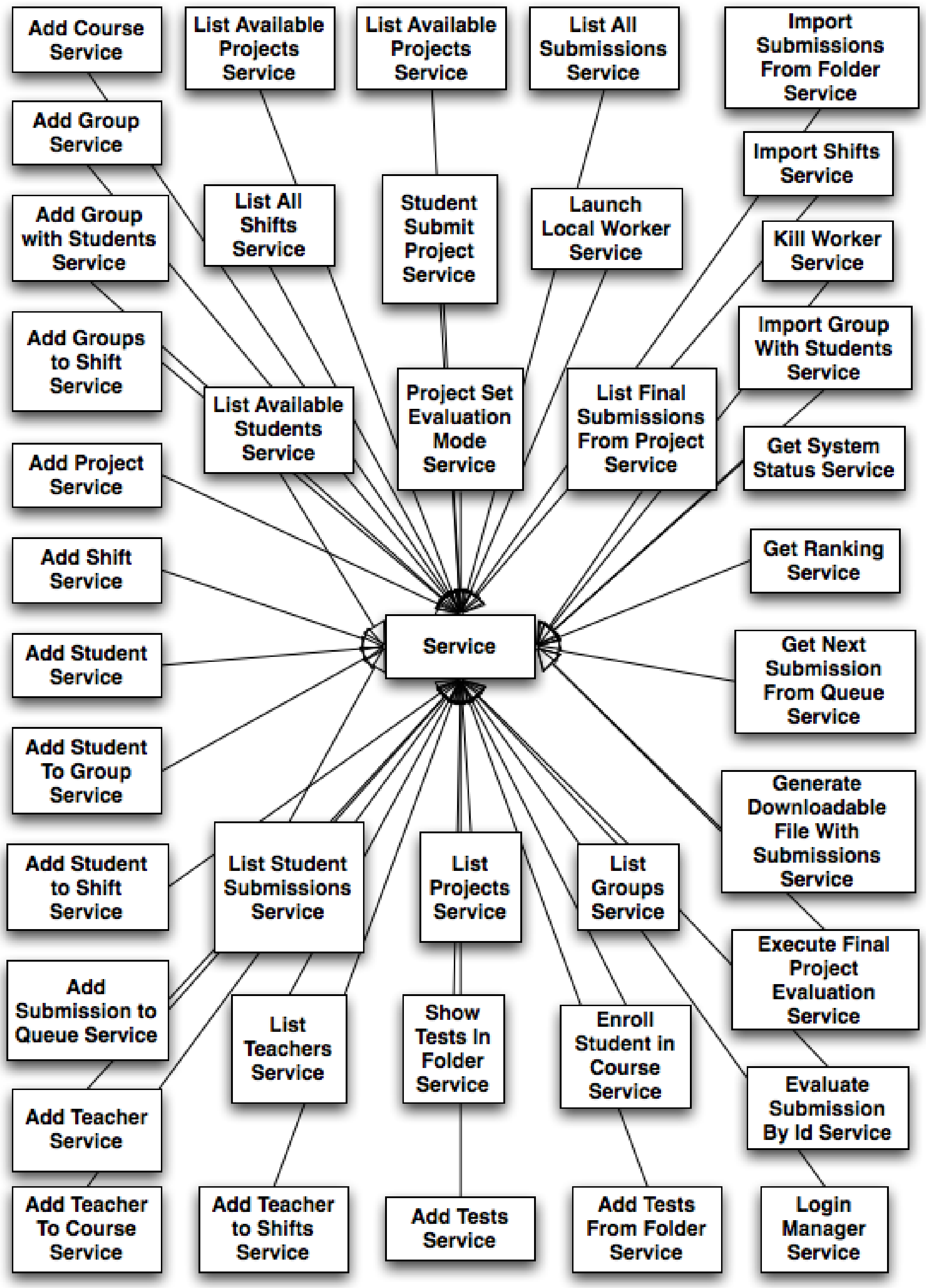

Add Course Service
List Available Projects Service
List Available Projects Service
List All Submissions Service
Import Submissions From Folder Service
Add Group Service
Import Shifts Service
List All Shifts Service
Student Submit Project Service
Launch Local Worker Service
Add Group with Students Service
Kill Worker Service
Import Group With Students Service
Add Groups to Shift Service
List Available Students Service
Project Set Evaluation Mode Service
List Final Submissions From Project Service
Get System Status Service
Add Project Service
Add Shift Service
Get Ranking Service
Add Student Service
Service
Get Next Submission From Queue Service
Add Student To Group Service
Generate Downloadable File With Submissions Service
Add Student to Shift Service
List Student Submissions Service
List Projects Service
List Groups Service
Execute Final Project Evaluation Service
Add Submission to Queue Service
List Teachers Service
Show Tests In Folder Service
Enroll Student in Course Service
Evaluate Submission By Id Service
Add Teacher Service
Add Teacher To Course Service
Add Teacher to Shifts Service
Add Tests Service
Add Tests From Folder Service
Login Manager Service

Figure 4.2: The System's Services

**GetRankingService:** Returns the current ranking of a given project. The ranking orders the best evaluated submission for each group. This ranking is always sorted by number of passed tests and the number of submissions evaluated or submission date (the last two criteria only make sense concerning the evaluation of submissions in VCS-mode with student requested evaluations since in File-mode all the submissions are created in the system at the same time).

**GetSystemStatusService:** This service returns the status of the system such as the number of active workers, what are they doing, the number of submissions still on queue, etc. This service is used to show the status of the system to the teachers.

**ImportSubmissionsFromFolderService:** Service available in the File Import mode which imports a set of folders from the File System creating a submission for each one of the group/students the folder name represents and queueing them for evaluation. With a special flag these evaluations are flagged as final.

**ListAllSubmissionsService:** Service accessible only to teachers that lists all submissions contained in a given course.

**ListStudentSubmissionService:** Service that lists all submissions from a given student[8].

**LoginManagerService:** Service that handles the authentication of the user in the system. In the current version it inquires a remote server to ascertain if the login credentials are the correct ones, but it is easily extensible to use a self-contained authentication solution (partly implemented already) or another third-party one.

**StudentSubmitProjectService:** This is the service invoked when a student (in VCS-mode) asks the system to evaluate a submission. The service fetches the submission from the VCS system and enqueues it for further evaluation.

#### 4.2.2.1 Monitor and Evaluators

Contained in the business logic of the system, there are a few modules (Monitor and Evaluators) that are always running, that play a big role on making the system work.

One module that is central for the system to properly work is the Monitor module. This module is a single entity whose function is to manage and monitor the worker threads, designated as Evaluators, and to dispatch queued work to them. For this reason, the module maintains a queue of pending submissions to be evaluated. The Monitor is accessed by services for several reasons. One of them is to add a submission to be evaluated, like service AddSubmissionToQueueService does. The other reason is to get a report on the status of the Monitor (made through the GetSystemStatusService service). Finally the Monitor also exposes services that are used to create and kill evaluators.

This entity also has a failsafe mechanism that tracks if an Evaluator is locked up somewhere and kills it, quarantining the submission the worker was currently working on and trying to evaluate it one more time (if it fails again the submission will not be evaluated again). This failsafe mechanism is implemented through a timer. This mechanism also ensures that there is at least one Evaluator running at any given time. On system startup, the domain is searched for

[8]Group submissions are also linked to each of the members of the group at the time of the submission.

non evaluated submissions and the queue is populated, making the need for queue persistency non-existent.

The Evaluators are threads (in a future iteration of the system they might be full programs on remote machines as well, see Section 6.4.1) that inquire the Monitor if there is new work (i.e. submissions to be evaluated) to be done and if there is, they ask the Monitor for it and proceed to evaluate the submission (done through the GetNextSubmissionFromQueueService service). The choice of using threads instead of processes for this first iteration was mainly because the way the projects are evaluated and to save computing resources on the server. For the former reason, the rationale is that the evaluation of a given submission consists in calling a process which compiles the student's program and for each test contained in the project, to run the compiled program with the test's input data, comparing the output later. Since the evaluator calls external processes, the risk of the thread getting corrupted and misbehaving is low. Easier to understand is the latter reason. If the Evaluators were to be processes, a new instance of the JVM would have to be created and maintained, causing the memory used for each evaluator to be orders of magnitude bigger than by using threads which is counter-productive. Moreover, the communication time between Evaluators and Monitors would be higher since they would be in distinct processes.

A representation of a simplified interaction between a student, the Evaluation System and the previously mentioned modules is shown on Figure 4.3.

### 4.2.3 Presentation

The system contains two different layouts depending if the logged user is a teacher or a student. The login screen is the actual landing page of the system.

The student interface allows the user to tell the system it wants it to evaluate a submission (available only if the system is in CVS/VCS-mode), to see his (or his group's) previous submissions and their results if they were already evaluated and to see the Ranking of results of all the students. The result screen, as it can be seen in Figure 4.4, shows if a given submission was already evaluated or not. If it was, it shows if the submitted program was able to compile without errors or if there was any error on the compilation, which is shown to the user. If the submission was compiled, then it shows the result of running the compiled program through the set of tests associated with the project. Again, if there was any error during the execution of the several tests, the student can access the error message. This error message shows the differences between the expected result and the obtained result.

The teachers' interface allows a bigger number of functionalities many of them related to the management of the course and setting up of the evaluation environment. Teachers can check the system's status (number of queued submissions, number of active evaluating threads, which submission is each of the evaluating thread working on, etc.), launch more working threads, kill a working thread, list and import students and groups, add another teacher, list and import shifts, list and add projects as well as their tests, import submissions if the system is in "File Import" mode, start the final evaluation of the final submissions of all students, list all the submissions and evaluations of the system in a per project basis as well as their rankings, etc. The teacher's point of view of the submissions the system contains can be seen in Figure 4.5.

In order to develop this layer, we used Google Web Toolkit[9]. This is a toolkit that allows a

[9]Version used: 2.0.4. Can be found at http://code.google.com/webtoolkit/

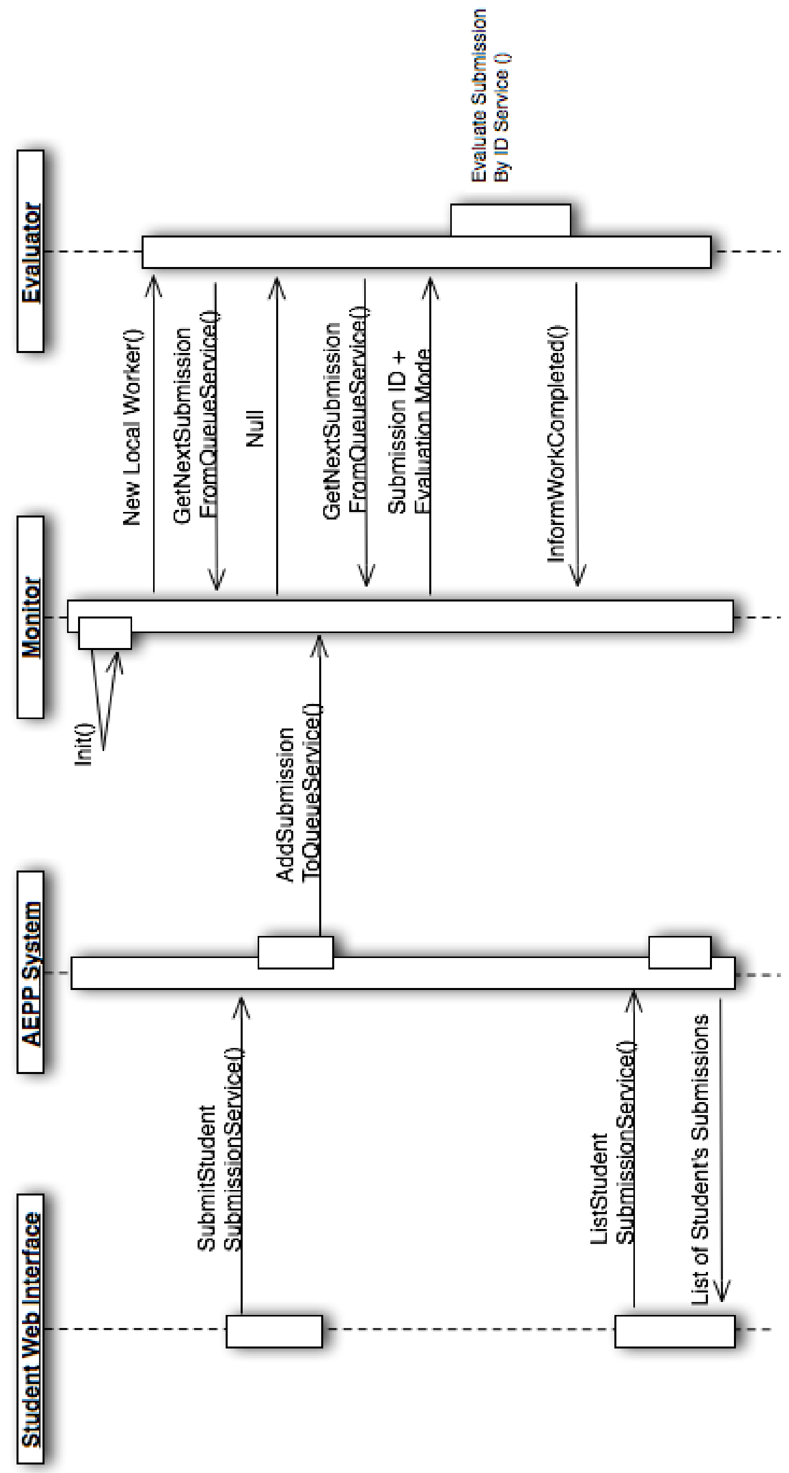
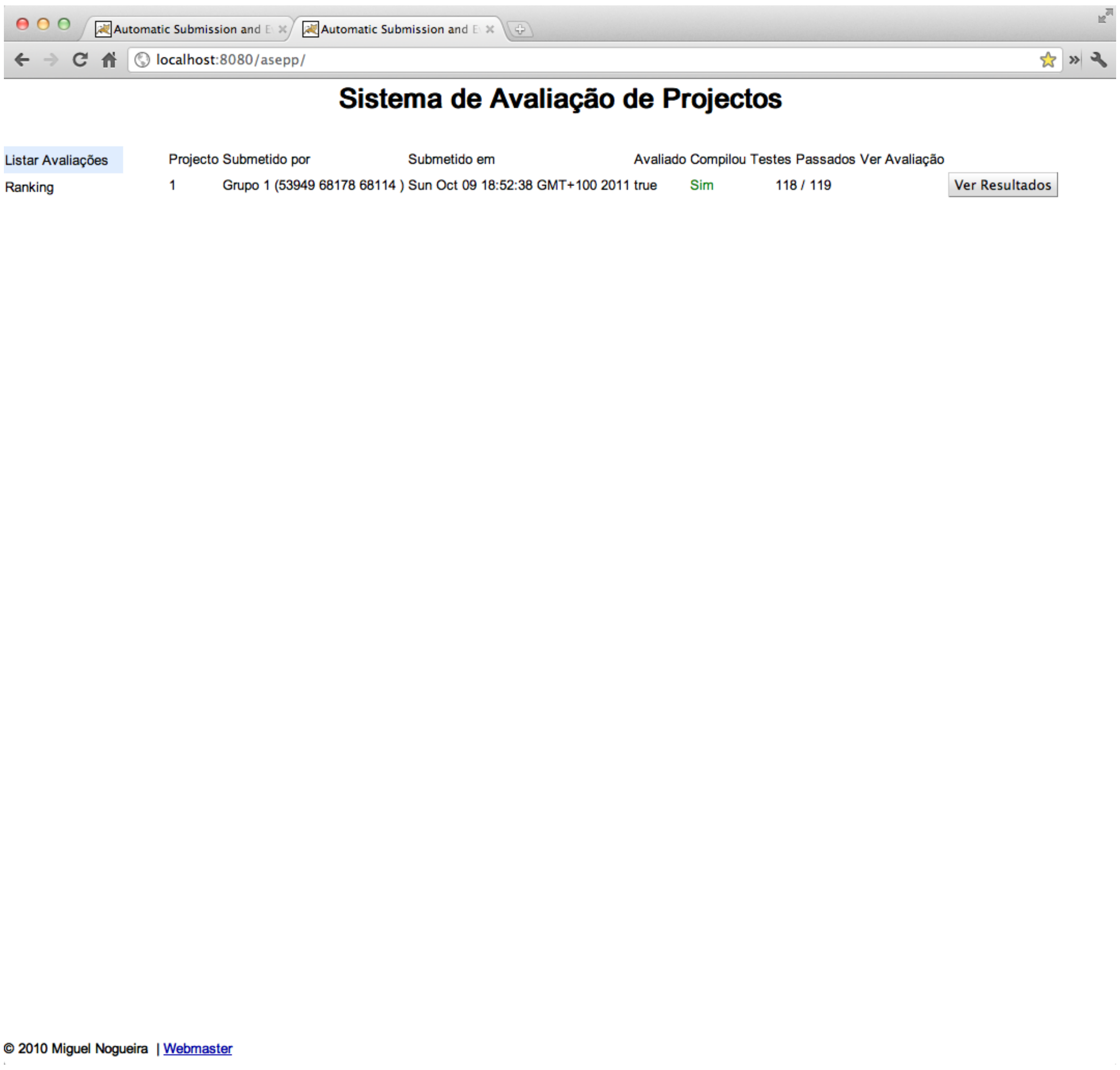

Student Web Interface
AEPP System
Monitor
Evaluator
Init()
SubmitStudent SubmissionService()
AddSubmission ToQueueService()
New Local Worker()
GetNextSubmission FromQueueService()
Null
GetNextSubmission FromQueueService()
Submission ID + Evaluation Mode
Evaluate Submission By ID Service ()
InformWorkCompleted()
ListStudent SubmissionService()
List of Student's Submissions

Figure 4.3: Simplified Sequence Diagram of the Monitor and Evaluator.

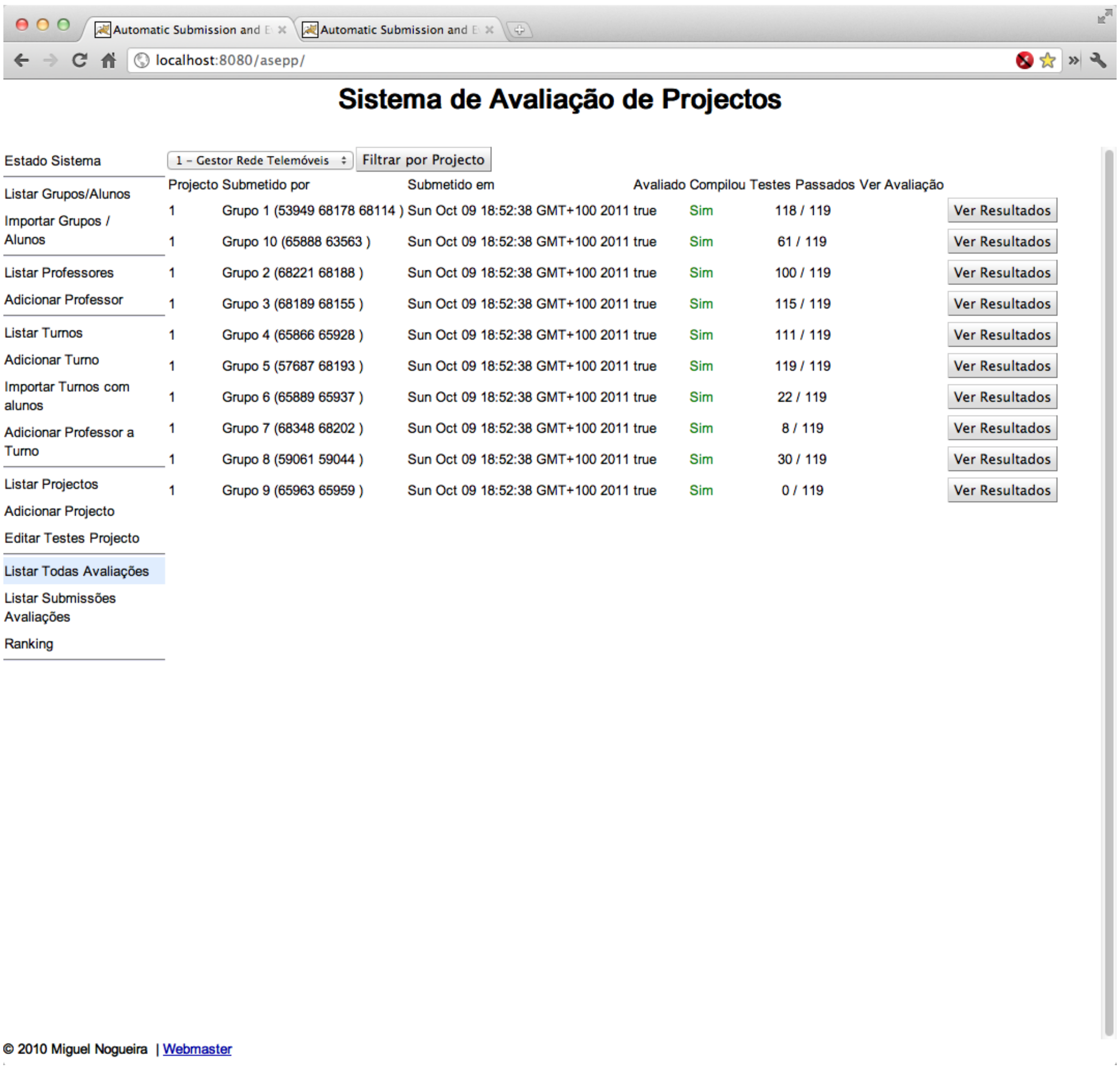


**Sistema de Avaliação de Projectos**

| Projecto | Submetido por | Submetido em | Avaliado | Compilou | Testes Passados | Ver Avaliação |
|---|---|---|---|---|---|---|
| 1 | Grupo 1 (53949 68178 68114 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 118 / 119 | Ver Resultados |

© 2010 Miguel Nogueira | Webmaster

Figure 4.4: Student's view of its evaluated submissions. One in this case.



Sistema de Avaliação de Projectos

Estado Sistema
Listar Grupos/Alunos
Importar Grupos / Alunos
Listar Professores
Adicionar Professor
Listar Turnos
Adicionar Turno
Importar Turnos com alunos
Adicionar Professor a Turno
Listar Projectos
Adicionar Projecto
Editar Testes Projecto
Listar Todas Avaliações
Listar Submissões Avaliações
Ranking

1 - Gestor Rede Telemóveis | Filtrar por Projecto

| Projecto | Submetido por | Submetido em | Avaliado | Compilou | Testes Passados | Ver Avaliação |
|---|---|---|---|---|---|---|
| 1 | Grupo 1 (53949 68178 68114 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 118 / 119 | Ver Resultados |
| 1 | Grupo 10 (65888 63563 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 61 / 119 | Ver Resultados |
| 1 | Grupo 2 (68221 68188 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 100 / 119 | Ver Resultados |
| 1 | Grupo 3 (68189 68155 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 115 / 119 | Ver Resultados |
| 1 | Grupo 4 (65866 65928 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 111 / 119 | Ver Resultados |
| 1 | Grupo 5 (57687 68193 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 119 / 119 | Ver Resultados |
| 1 | Grupo 6 (65889 65937 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 22 / 119 | Ver Resultados |
| 1 | Grupo 7 (68348 68202 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 8 / 119 | Ver Resultados |
| 1 | Grupo 8 (59061 59044 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 30 / 119 | Ver Resultados |
| 1 | Grupo 9 (65963 65959 ) | Sun Oct 09 18:52:38 GMT+100 2011 | true | Sim | 0 / 119 | Ver Resultados |

© 2010 Miguel Nogueira | Webmaster

Figure 4.5: Teacher's view of all submissions made on the system.

programmer to create web application front-ends by writing Java code. This Java code is then transformed into Javascript that generates the webpage itself. This toolkit was chosen because it is easy to use, the learning curve is not steep and it is Java as well, making it easier to maintain and extend to a larger user-base that might not know HTML, CSS and/or Javascript.

## 4.3 Fénix Framework and JVSTM

Having hinted on the framework used on the previous sections, this section presents the framework used and the rationale on why it was chosen.

The Fénix Framework and one of its biggest components, the Java Versioned Software Transaction Memory (JVSTM), were briefly mentioned on Section 2.7 and on the introduction of this chapter. The JVSTM was developed on the Software Engineering Group of INESC-ID (Instituto Superior Técnico), starting as a doctoral thesis from Prof. João Cachopo (Cachopo 2007) and evolving into the the basis of the framework in which the Fénix academic system is built upon.

In its essence the JVSTM is a Software Transaction Memory system which maintains the domain and its instances in memory and handles the way the data persists in the database without the intervention of the user(Cachopo and Rito-Silva 2006). This makes programming with this API easier, since the programmer works with objects as it would in any Java program, not having to worry to interact directly with the database with JDBC or other database connector. The programming style is close to programming with Hibernate with a few differences. To create the domain, there is a Domain Modeling Language in which the programmer describes de domain classes and its relations. The domain description in said language is then processed generating the Java classes used by the programmer.

The most important capability of this framework is that it supports almost for free from the programmer's point of view, transactions. This is important because a web application, like our proposed system, is a concurrent system, since there can be several users accessing the system at the same time and there can be multiple Evaluators running. Therefore we had to handle concurrency in such a way that the shared domain data is kept in a consistent state. Using the framework, every service call is transactional, which gives concurrency handling out of the box and consistent data all the time.

## 4.4 Submission and evaluation variants

The system supports two types of submission modes that lead to two different evaluation paradigms:

**Version Control System:** Submissions are imported from a Version Control System into the system. This method in detailed in Section 4.4.1;

**File Import:** Submissions are imported from the filesystem. This method is explained in Section 4.4.2.

One of the reasons for the existence of different submission methods is for the system to be able to adapt to different teaching philosophies the faculty might want to enforce in the course. Some teachers are in favor of instantaneous (or as soon as it is possible to evaluate the submitted solution) feedback. Others might prefer to have tests to be run all at the same time and once a day[10], others provide students with a small set of tests for them to check if their solution is on the right way, etc.

Another reason for the duality of submission criteria is some teachers do not want to overload students with having to learn how to interact with a Version Control System on top of having to learn a programming language and solve a problem although one may argue it is a needed skill for every Software Engineer and they will have to learn for some other courses which enforce collaboration of big groups.

The choice between the import methods is made at course level on system configuration, and there is never a mix of import methods on the same course which might cause some confusion on students.

### 4.4.1 Version Control System / Source Code Management integration

Having chosen the CVS or Version Control System (VCS) integration route for a course submissions can be queued to evaluate either by students or by teachers. For a student[11] to have its submission evaluated in this mode (prior to the final evaluation that is), he should first commit his new work to the VCS supported by the course, once that is made, the student should tell the system through its webpage he wants to evaluate the last submission contained on the VCS. When this is done, the student's latest submission is imported to the system and put in the evaluation queue, to be evaluated in order. The student can then check the result of the evaluation and how his solution to the project compares to the evaluated solutions from other students in a form of a Ranking.

On this mode, teachers don't have to do anything in order for submissions to be evaluated on an ongoing project. After the project deadline ends they can instruct the system to evaluate the last submission of all students to get the final grade of the project. Then the system accesses the VCS associated with the project to evaluate and checks out the latest submission for each student/group. For each submission, the system creates an internal submission and queues it for evaluation. The evaluation is done by applying the evaluation tests and is marked as final evaluation.

As mentioned earlier, the Version Control System currently supported by the system is CVS, but the extension to other VCSes like SVN[12] or git is easy to develop and integrate in the system. In order to add a new VCS system to the supported list of import methods, the user only needs to extend the abstract class SubmissionFileHandler with the new VCS system importer based on the CVSSubmissionFileHandler class and when the new system is configured invoke it instead. This importer class and its main method (getNewSubmissionFiles) receive as arguments the student/group, submission and project and course informations like the repository to use. Based in this information this class builds the identification of the project to obtain and checks it out from the VCS system associated with this importer class. The obtained submission is stored in the expected folder to be later compiled and evaluated by another compo-

[10]The current system only allows this to be made manually, but Section 6.4.4.2 details a scheduled approach that can be easily implemented on a later version of the system.

[11]The procedure is the same if the project is a group project.

[12]Subversion - http://subversion.apache.org/

nent of the system, the Evaluator. Section 6.4.2 explores the possible future work to be done on this area.

### 4.4.2 File Import

The other submission mode supported by the system introduces a variation on the way the users - either Students or Teachers - interact with the system in order to evaluate submissions and to check the evaluation results caused by the change in the evaluation paradigm.

Starting with the teacher's point of view, the way the teacher orders the system to evaluate the students' code is by getting the files students submitted on the file submission platform of choice and then copies them to the import folder of the intended project. When the files are in said folder, the teacher tells the system through its web interface those files are there and to start importing them. Each group/student with a project to evaluate is assigned to a subfolder with its number in the import folder. Then, for each student/group (again depending on the project) with an entry in the import folder a new submission is created and put in the "to evaluate" queue. From that moment on the process is the same as on the other mode: Evaluators fetch the queued submissions, evaluates them and the results are automatically published and accessible on the website as each submission is evaluated.

On the File Import mode, Students do not have the instant feedback they get on the VCS mode since they depend on the teacher to order the system to evaluate their submissions. They upload their solution to the chosen upload system and only when the teacher chooses to evaluate the projects they can access the website to check the result of their last submission.

## 4.5 Project Evaluation Process

As mentioned in the previous sections, the system evaluates students' project submissions. The system must be able to evaluate said programs. Therefore, it needs to know how to compile a submission, run it and compare the program's result to the expected result. In order to have the system as generic as possible (i.e. independent of the programming language applied to develop the submitted program), we decided to allow teachers to define how each project is evaluated, only enforcing a few rules.

The first step consists in defining the Project. For that, the teacher has to create a project in its web interface. On the Add Project interface, teachers have to detail the following information:

**Name:** The teachers should give a name to the project;

**Group Project:** If the project is an individual project or a group project;

**Start Date:** The date and time when submissions should start to be accepted and/or evaluated;

**Deadline:** The date and time until submissions concerning the project are accepted by the system;

**Main file:** The name of the file where the "Main" method is (for languages that use this) or the name of the file that starts the program;

**Deadline in class:** This is an option where the deadline changes for each student/group. If this option is active, the deadline is not on the date of the deadline, but on the last practical class before the deadline. If the option is active, there is also given the option of this new deadline being at the start or end of said class. This options were created to support weekly homework for example;

**Allowance:** Time in minutes students can still submit projects after the deadline.

**Compilation String** String used in order to compile the project;

**Execution String** String used in order to execute the project;

**Comparison String** String used to compare the test's output with the expected output;

**Maximum execution time:** Maximum execution time allowed for a test. If a test takes more than this maximum time, it should be aborted and the result of the evaluation if this test is considered unsuccessful.

**Project's Repository ROOT (optional depending on system mode):** CVS Root for the project, if the system is in CVS-Mode;

After defining the project (or rather before the system is to be used to evaluate projects), the teacher should create the tests that evaluate the project. The system also contains a feature that uses another set of tests to perform a preliminary evaluation. This set of tests to be executed for each submission is important for students since it warns students if the project does not comply with some tests and students can still correct their solutions before the deadline. The experience of teachers of the Object-Oriented Programming course in IST is that if those tests are made available to students so that they can execute the tests themselves, a large amount of students still does not use the tests to correct their solution.. Some teachers do not like to give access to the full evaluation test suite before the final evaluation is done. With this feature teachers can use either a subset of the evaluation tests or a new set of tests in order to do the preliminary evaluation.

These tests have to obey to a small set of rules in order for the system to be able to recognize them and be able to work with them. The input and output files that concern each test must have the same name and different extensions (only one dot per file). The extension .output is reserved for the output of the test execution which is available for the system to use while comparing to the expected result.

For example, we can have two tests for an imaginary project named *test1* and *test2*. Each of this tests has one input file named *testX.in* and one expected result named *testX.out*. The second test, *test2*, also has an optional file called *test2.import*. After the teacher creates these files, he should place them in the project folder. Having the files placed in the project folder, the teacher has to inform the system the new tests are there. The system then parses the files on the folder extracting the name and the associated extensions for each test and presents the list of tests it found to the user for the user to select for each listed test if it is a Preliminary Test, an Evaluation Test or both. Once the teacher submits this information the system finally is ready to evaluate student's solutions.

Student's projects are then submitted to the system by either one of the methods referenced in Section 4.4 and added to the queue. Then the Evaluators ask for the work as stated on Section 4.2.2.1 and start evaluating it.

For each submission to evaluate the system has to compile the code first. To compile the code, the system parses the Compilation String the teacher specified on the project definition in order to know how to compile the code. This method was chosen to give teachers some freedom in the way they choose programing languages for the course, since this way we can support any language that compiles through the command line. As a consequence, it also supports building tools like Make and Ant. At the moment in the first iteration teachers can use one variable on this string named *%ALLJAVA%* which represents all the Java files in the submission folder and on its subfolders. Using this variable, teachers can create their own compilation string. An example is:

*javac -encoding UTF-8 -cp libs/:. %ALLJAVA%*

When the compilation fails, the evaluation is aborted and the submission is marked as not able to compile and the error the compiler outputted is stored for later display to either the student or the teacher if they request it.

Having compiled the student's solution, the system now has to execute each test associated with the project. This part of the process involves executing the compiled submission code with the files of each test in order to produce the expected result. For this, the Execution String is processed. This string supports the variable %MAINFILE% which represents the file containing the "main" function (e.g. *%MAINFILE%.java*), and also supports the variable %TEST% representing the current test name. This can be specified to utilize the files associated with the test as inputs. Every execution has its output automatically saved to a file named *temp%TEST%* which can be used later. Following our example, if the teacher defined the project (enforcing it in its rules) as having its input file provided in the -Din flag, the import to the -DImport and to output the result to -Dout, the Execution String would be:

*java -cp libs/:. -Dfile.encoding=UTF-8 -DImport=%TEST%.import -Din=%TEST%.in -Dout=%TEST%.out %MAINFILE%*

One other feature this execution phase has is to selectively exclude parts of the execution command. In the current example, on *test1* the system would not include the *-Dimport=%TEST%.import* part of the string because *test1* doesn't contain the file *test1.import*. This feature has its limitations in the current form in which the only part that is not included in the final execution command is the substring (separating the given string by the space character) that contains the description of the file that does not exist.

Finally in order to evaluate if the result the execution of the student's program is the expected one, the Comparison String is used. This step compares the output of the student's program executed with the test's input with the test's expected output. For this the system also uses a string defined in the project named Comparison String to make the comparison. This string can use the variables introduced in the Execution String, and the *temp%TEST%* also presented before. So, in our example, a possible Comparison String could be:

*diff -iwc %TEST%.out temp%TEST%*

This approach of letting teachers define the strings for compilation, execution and comparison allows a degree of freedom in the choice of programming language used since the system can use any language whose compiler and execution can be called from the command line. Further developments in this area are possible and some of them are introduced in Section 6.4.5.

After the submission has been evaluated the results are automatically made available for the users, teachers or the student (or students belonging to the group) that submitted the project. To access these results, the users must use the web interface of our application.

## 4.6 Conclusions

The system presented in Chapters 3 and 4 complies with the majority of the requirements analyzed in Section 1.3.

One of the main requirements was the availability. The system needs to be able to accept student submissions. Offloading that responsibility and load to proven reliable systems helps fulfilling this requirement.

Another of the requirements tackled is scalability. The proposed system allows for vertical scalability (adding more local workers to take advantage of multiple cores of the new computers) and has the foundations in place for horizontal scalability (distributing work by remote workers).

Although there were some requirements that we were not able to fully fulfill on this iteration of the system to full extent such as the ability to support multiple evaluation strategies. However, for this case remark that the system does have various evaluation strategies since it behaves differently when in VCS mode and on File Import mode, but there is still not a mechanism in place to make these evaluation strategies pluggable and interchangeable. Although, the main intent of these requirement was to control system load and this load can be controlled by the number of evaluators running. If only one evaluator is running the queue might grow bigger, but the execution load will not increase since the system only evaluates submissions through the Evaluators and they only evaluate one submission at a time. This didn't happen with the POEval system since the submissions were evaluated as soon as they were submitted to the system. If there were to many submissions being made during a small interval of time (which could happen near the deadline of a project) the system was not able to handle such a load.

# 5 Evaluation

## 5.1 Methodology

To evaluate the effectiveness of this system testing must be done emulating real load conditions in order to assert if the requirements are being complied with. As noted before, the system has to be able to receive submissions even if under heavy load, and when heavy load usually happens is on the deadline day. In order to verify if the performance requirements are met we have to submit projects to the system with a high ratio of submission and having several simultaneous users. Performance testing could be made using the system in a production environment (supporting a running course) solo or alongside the previous system, or using submission data from previous and current years and simulating the requests. In either options, tests have to be specified for each submission artifact (project or assignment). In the former case, running the system alongside the old one, a proxy would have to be set up in order to receive student submissions, timestamp them and send to both new and old systems. This solution would introduce a level of indirection that would need to be evaluated by the faculty of the course in which the test would be conducted. In the latter option, former years submissions would be fed to the system following their timestamps in order to simulate the actual load that the previous system was under and verify how the new system behaves in the same situation.

We decided to choose the second case since this one gives us more freedom to specify the load conditions. This way we are going to evaluate the system running a set of final student submissions from last semester's Object-Oriented Programming course in the Taguspark Campus of Instituto Superior Técnico using the created system and then comparing it qualitatively with the method used to evaluate these submissions on production. For all the tests run the dataset consisted of 55 different project submissions and the time limit for each test to execute was 5 seconds[1].

The system used to evaluate students' projects last semester can be seen as a manual version of our automatic system:

- Students submitted their work to the university's academic system Fénix as tar.gz files,
- Teacher downloaded all submissions to his computer,
- Ran a series of scripts to evaluate students' submissions which for each submission created a file containing the result of the evaluation and created a consolidated report for all submissions,
- and then had to manually publish it in the course's webpage.

[1]After the 5 seconds the process would be killed and the test evaluated as a failure.

What we will focus on this evaluation on is on achieving the same results to prove the correctness of the evaluation method and try to achieve them in a relatively close amount of time as the previous method.

The tests were run in a Early-2011 Macbook Pro containing a Intel Core i7 2.2Ghz with 8Gb of RAM and a 7200rpm HDD running Mac OS X Lion.

Having a multicore computer available, we will also test the performance of the system using various Evaluators which will test scalability and performance gains. The tests will be run with various numbers of Evaluators running to assert the scalability of the system. Note that the processor on the test machine is a quad-core with Hyper-threading technology allowing for each core to run two simultaneous threads, which means the processor can run eight threads at the same time.

For the first two experiments - Section 5.2 and Section 5.3 - we are evaluating correctness and pure performance, so the system was used in File Import mode. This is the mode where teachers fetch student's programs from some other submission method and feed them to the system in one bulk operation. This approach was chosen because the two experiments are comparing the new system's performance to the old system, which uses the same approach.

## 5.2 Experiment One: Checking the correctness of the system

The first experiment consisted in ascertaining the correctness of the system. This was accomplished by evaluating last year's final submissions of the Object-Oriented Programming course's project on both the new system proposed in this work and the system used to evaluate and grade students during the course. The new system would be correct if the results matched the old one.

After submitting the 55 submissions to the system we verified that the results obtained were the same for both systems, the "old version" and our system.

Another metric measured was the time both solutions took to process and evaluate all the submissions. Table 5.1 contains the time both solutions took to complete. We were expecting a performance hit comparing to the previous version, but 20% was is a bit more than we expected. A performance hit was expected because the system is running in a web-application container (Apache Tomcat), on top of a framework that saves data in memory and posteriorly in a database (Fénix Framework), the mechanisms and services needed to answer the requests to the system's users and has a series of mechanisms like the Monitor - that ensures the Evaluators (only one in this Experiment) are running and not faulty - which run in separate threads than the Evaluator.

Part of this performance hit can also have a relation with the processor architecture of the hardware in which the tests were run, since these processors can overclock one core automatically if the others do not have load. The old system only executes one action at a time and does not involve web interactions. At the time of this evaluation data points were gathered I did not have access to another machine that would not auto-overclock when running single threaded programs.

We also verified how the new system handles some special cases concerning the type of errors that we can have while evaluating the system. The special cases were:

| System | Start Time | End Time | Time Delta | Percentage Delta | Evaluators |
|---|---|---|---|---|---|
| Old System | 22:57 | 23:25 | 28m | 100% | 1 |
| New System | 20:40 | 21:14 | 34m | 121.4% | 1 |

Table 5.1: Experiment One time comparison

| System | Start Time | End Time | Time Delta in minutes | Percentage | Evaluators |
|---|---|---|---|---|---|
| Old System | 22:57 | 23:25 | 28m | 100% | 1 |
| New System | 20:40 | 21:14 | 34m | 121.4% | 1 |
| New System | 21:30 | 21:52 | 22m | 78.6% | 2 |
| New System | 22:00 | 22:17 | 17m | 60.7% | 3 |
| New System | 22:23 | 22:37 | 14m | 50% | 4 |
| New System | 22:41 | 22:54 | 13m | 46.4% | 5 |
| New System | 21:14 | 21:26 | 12m | 42.8% | 6 |
| New System | 21:34 | 21:46 | 12m | 42.8% | 7 |
| New System | 21:52 | 22:04 | 12m | 42.8% | 8 |
| New System | 22:12 | 22:23 | 11m | 39.3% | 9 |
| New System | 22:29 | 22:41 | 12m | 42.8% | 10 |
| New System | 22:52 | 23:03 | 11m | 39.3% | 11 |
| New System | 19:47 | 19:59 | 12m | 42.8% | 16 |
| New System | 22:54 | 23:05 | 11m | 39.3% | 32 |

Table 5.2: Experiment Two time comparison

- Submission with a very slow implementation of the project. In this case the tests will fail due to the fact the tests reached the timeout limit.
- Submissions that fail all test cases associated with the project.

On the 55 submissions there are some of them that don't compile, some that are slow implementations and fail some tests because they are too slow and some that fail all test cases associated with the project. Our system handles well these types of special submissions.

## 5.3 Experiment Two: Performance of the system

The second experiment used to test the system and compare it to the previous version is related to how the system behaves under load and if it benefits of the growing number of cores in new machines. For this, we tested the new system running various numbers of Evaluators.

Table 5.2 contains the results of the runs made. The result of running the old system served as a reference time (100%) to which all of the other results were compared to. As can be noted, if two evaluators are run at the same time the results are already faster than the old system and the performance gains keep increasing in the tested hardware up until six different Evaluators are running. After six Evaluators, the performance of the system remains almost constant.

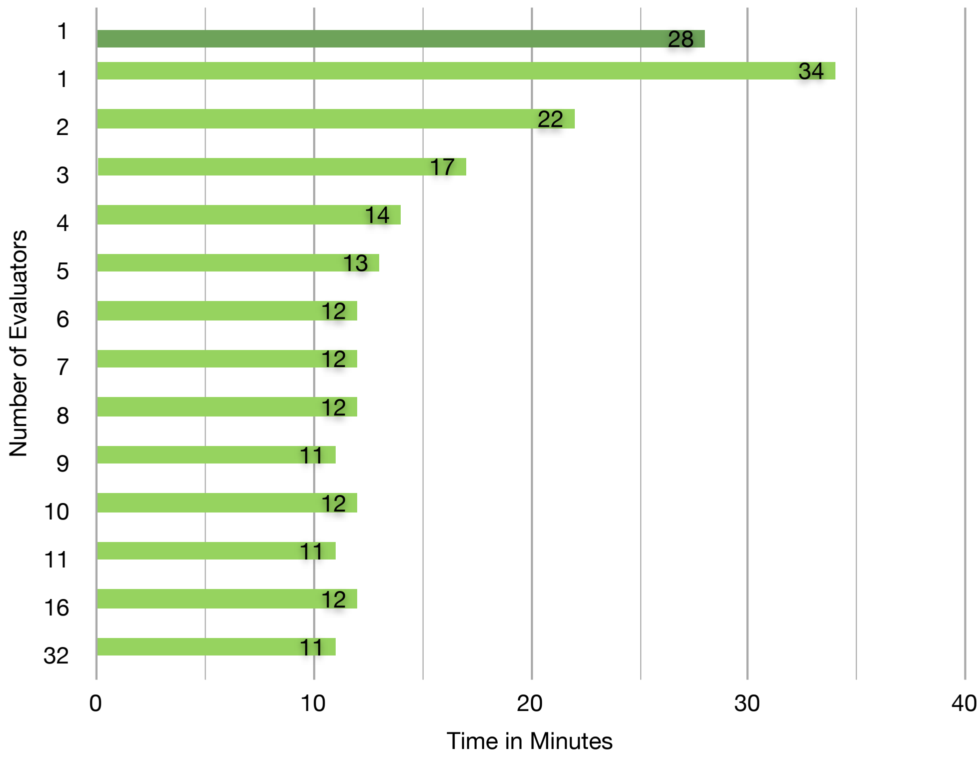


Figure 5.1: Experiment Two time comparison chart

The performance gains and its leveling can be better seen in Figure 5.1.

Another aspect tested in this experiment was how the system behaved retrieving answers through the web interface under high load. During the execution of the tests the web interface was used to access the results of the submissions (already evaluated or being evaluated) several times on all runs[2]. Slowdowns only became noticeable when the system was under very high loads like on the run with 32 Evaluators where the system's load average surpassed 50. And even then the system would retrieve the information in 1 or 2 seconds. On tests up to 10 Evaluators no performance hit was noticeable on the web interface.

## 5.4 Experiment Three: Multiple Clients on CVS mode

Since the system supports CVS, another feature needed testing is the retrieval of submissions under load. For this experiment, the system was setup in CVS mode with the CVSROOT pointing to a server in the Rede de Novas Licenciaturas in Instituto Superior Técnico. A program was developed that would emulate four students submitting projects in intervals of 0.5 sec-

[2]Even though this information was probably in memory due to the Fénix Framework, the system still had to use some cpu resources to retrieve it.

| System | Number of Submissions | Time Delta | Evaluators |
|---|---|---|---|
| New System | 10 | 2m | 4 |
| New System | 20 | 5m | 4 |
| New System | 30 | 7m | 4 |
| New System | 40 | 9m | 4 |
| New System | 50 | 11m | 4 |
| New System | 60 | 13m | 4 |
| New System | 70 | 16m | 4 |
| New System | 100 | 22m | 4 |

Table 5.3: Experiment Three time table

onds. Each student would submit 25% of the submissions.

The goal of the experiment was to check if all submissions were properly acquired and to time how long the system would take to empty the queue.

Table 5.3 presents the results of this experiment. As expected, all submissions are properly retrieved and evaluated and the time the Evaluators take to empty the queue grows linearly with its size as can be seen if Figure 5.2. Note that once the system gets enough programs to feed all the active Evaluators, the behavior in terms of load is comparable as the one in the previous experiment, even if there are still other threads fetching the submissions from the Submission System.

This experiment was also conducted using the browser instead of an automated program. Four students were logged in and submitting projects within 2-3 seconds of each-other[3] consistently and the behavior was the same as in the automated experiment albeit with less submission load.

[3]One each student and then looping back to the first student.

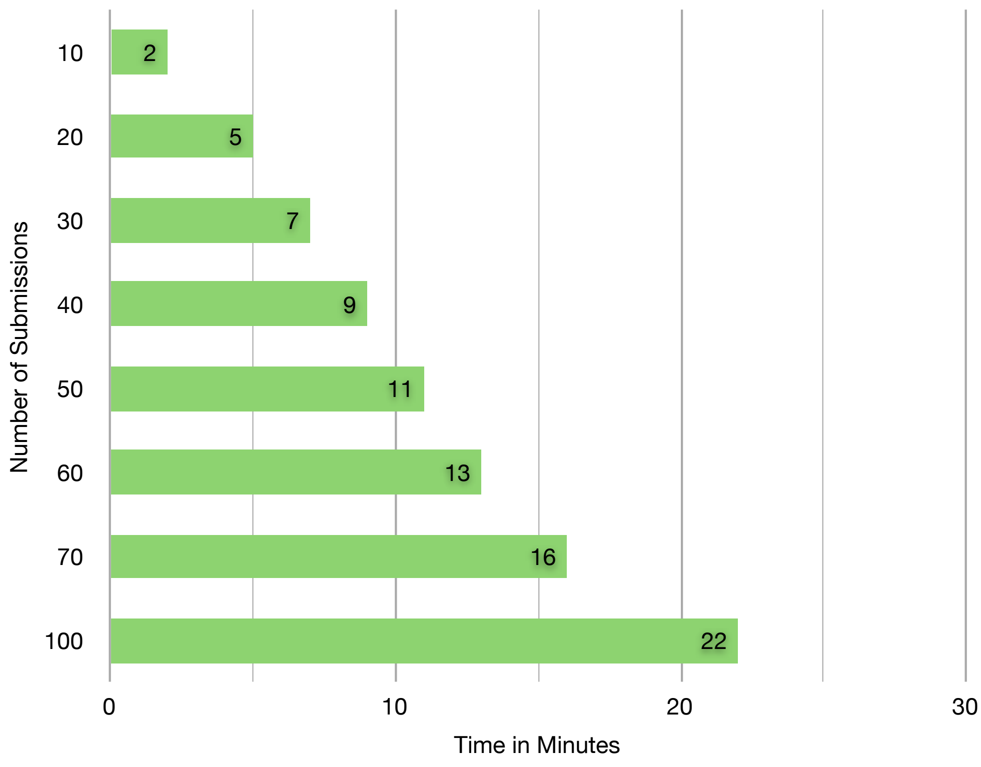


Figure 5.2: Experiment Three time comparison chart

# 6 Conclusions

## 6.1 Contributions

The use of automatic evaluation systems in education proved to be useful easing the workload on the teachers and improving the amount, quality and speed of the feedback to students. Though, most of these systems do not comply with all requirements deemed important for using in programming courses.

In this work we have identified a set of requirements(Section 1.3) that should be met by an automatic evaluation system to be used in the context of a programming course. Some of the requirements are the following:

- File submission
- Specification of automatic tests
- Execution of automatic tests
- Language independence
- Accessibility
- Performance
- Scalability

We have built a system that meets these requirements, namely that handles load situations, is fast and is easy to use. Moreover, the experiments presented in Chapter 5 show that this system is efficient and is able to support a high load, being able to handle a high number of simultaneous users and a high rate of project submissions.

## 6.2 Limitations

This work has its limitations in evaluating programming assignments since it is based in input/output tests. Other kinds of tests involving code introspection and code quality (an example would be whether the program under test implemented a given Design Pattern or not) can not be addressed by the proposed system.

Although, the system could be extended or adapted to run tests based on automated testing frameworks such as JUnit[1] or TestNG[2] for example if students were forced to implement and respect a given a interface.

[1]JUnit - A test framework for Java projects. http://www.junit.org/
[2]TestNG - Another test framework for Java projects. http://testng.org/doc/index.html

Out of the scope of this thesis was for the created system to have all of the functionalities needed to manage a course including course announcements, e-learning capabilities such as content spreading functionalities such as the course's bibliography and support materials, tests, exams, final grades, etc. The proposed system could in the future provide an API for it to be able to cooperate with other systems such as Fénix (Section 2.7).

## 6.3 Alternative Applications or Uses

The scope in which this system could be used would not only be reduced to programming courses but, like some of the systems presented on Chapter 2 such as Mooshak, could also be used to evaluate programming competitions.

## 6.4 Future Work

### 6.4.1 Remote Evaluators

One of the possible updates the created system can have is to be able to support external or stand alone evaluators. This would allow the main server to contain only the Evaluation Manager and only one or two Evaluators and offload the rest of the hard work to simple clients located in other machines in cases the workload is very big and the system is running several courses at the same time. The system is currently configured to run in a single course mode, but the infrastructure is in place for the system to support several courses with a few changes in the system's code mainly in the Presentation Layer.

On the Remote Evaluator front, the Monitor is already ready to handle them, only needing to expose the used services as Web Services for the Remote Evaluators to be able to access the system. There is also the need to upload the project's descriptions and tests for each of the Remote Evaluators on startup or when they are modified. For each submission to be evaluated it also needs to be fetched, what would be easy in VCS mode having the Evaluator downloading it directly from the source. This functionality would not be that useful on File Import mode since the Evaluation Manager would have to send the submissions to the Remote Evaluators.

### 6.4.2 Support for more import solutions

As discussed previously on Section 4.4 the system supports two methods to import the student's submissions: from a Version Control System or in File Import mode.

On the former case, the created system only supports CVS as version control system, but the extension of the import system to include other version control systems is trivial as it has already been detailed in Section 4.4.1. New handlers for Subversion, git, Mercurial[3], Darcs[4] or any other source control management/versioning system can be easily created and added to the system.

[3]Mercurial - a free distributed source control management tool. http://mercurial.selenic.com/
[4]Darcs - Another free, open source source code management system. http://darcs.net/

On the latter case and given the case the faculty of the course in which the system would be used do not want to support a VCS system, a new functionality of file upload could be easily implemented allowing for students to upload their files to the system.

### 6.4.3 Fénix Integration

As mentioned on Section 6.2, one of the possible followups for this work, considering the university on which both projects were born, is some kind of integration between the Fénix System and the proposed evaluation system could be achieved.

There are several kinds of integration possible on this matter, from simple API or Web Service calls to full code integration of the created system into the Fénix Academic Management System to support all the programming courses lectured on the universities using it.

### 6.4.4 Other improvements

This section presents some minor functionalities the system and its users would also benefit from.

#### 6.4.4.1 Sandbox

The system already encapsulates the execution of the tests in their own processes and revokes it of some privileges. The next step would be to completely sandbox the execution of the tests making it impossible for some malicious[5] or buggy code to access to parts of the system it should not.

This would also allow the Remote Evaluator (Section 6.4.1) idea to flourish allowing personal computers of the faculty of the courses, per example, to evaluate some of their students' submissions without any problem.

#### 6.4.4.2 Scheduled evaluations

For the purpose of some pedagogic exercise or for automation purposes, teachers might want to be able to schedule the evaluation of a project (or programming assignment) to a certain date and time. This could be used to automatically evaluate the project submissions with the final evaluation tests per example one hour after the deadline allowing the students to have faster feedback and the teachers not to have to worry to manually start the final evaluation. This feature would only be useful in VCS-mode.

#### 6.4.4.3 Localization / Internationalization

Mainly a cosmetic feature that would be useful to add to the system is the ability for the presentation layer to be displayed in different languages, which would allow the system to be used in schools and universities of a broader range of countries.

[5]Students can be mean these days :D

The framework in which the web front-end was built upon, Google Web Toolkit, provides a set of tools which allow an easy internationalization implementation.

#### 6.4.4.4 Statistics

The amount of data collected during a course or set of courses could be statistically analyzed and used to extract patterns (data mining) which could help both students and the faculty of the courses valuable information.

For students this could provide some indicators on the evolution of their work against their colleagues in a more precise way than the Ranking feature per example.

To the faculty of the course, this data could provide valuable information in how the current promotion of students is complying to the expected results, if the current project difficulty is adequate, if new teaching methods or techniques are making any sort of effect, among others.

#### 6.4.4.5 Evaluation Policies

Another interesting mechanism the system could have in the future are evaluation/execution policies. These policies may range from limiting the number of submissions per project a student/group has, to daily limits of evaluations, etc.

Some of the ideas behind having these policies are to enforce some control over the resources of the system, for students to be forced to think if they are really sure if they want to lose one of their available options to evaluate their solution, etc.

### 6.4.5 Improvement of compile/execution/comparison string parsing

The new system already allows for a variety of options regarding the compilation and execution of the students' programs and the comparison of their output with the test's expected output. However, as in everything, we find there is some room to improve these features.

One example that could be improved is the execution of programs using UNIX pipes and the selective exclusion feature presented in Section 4.5. This would not work if the execution string contained *¡ test.import* and the file *test.import* did not exist because, in the current implementation, it would parse them as different arguments making the pipe appear in the string with no file to pipe to the executing program.

A possible solution is the definition of a simple syntax to allow a bigger range of possibilities while creating the compile, execution and comparison strings.

# Bibliography

Cachopo, J. (2007, September). *Development of Rich Domain Models with Atomic Actions*. Ph. D. thesis.

Cachopo, J. and A. Rito-Silva (2006). Versioned boxes as the basis for memory transactions. *Science of Computer Programming 63*(2), 172 – 185. Special issue on synchronization and concurrency in object-oriented languages.

Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford (2008). *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Addison Wesley.

Dalziel, J. (2003). Implementing learning design - the learning activity management system (lams).

García-Mateos, G. and J. L. Fernández-Alemán (2009). A course on algorithms and data structures using on-line judging. ITiCSE'09, July 6–9, 2009, Paris, France.

Leal, J. P. and R. Queirós (2008). Design of a repository of programming problems.

Leal, J. P. and F. Silva (2001a). Managing programming contests with mooshak. *Eunis 2002 - The 8th International Conference of European University Information Systems, 2002, July, Porto, Portugal*.

Leal, J. P. and F. Silva (2001b). Mooshak: a web-based multi-site programming contest system. *Software—Practice Experience, Volume 33 , Issue 6 (May 2003), Pages: 567 - 581, 2003, ISSN:0038-0644*.

Revilla, M. A., S. Manzoor, and R. Liu (2008). Competitive learning in informatics: The uva online judge experience. Olympiads in Informatics, 2008, Vol. 2, 131–148 © 2008 Institute of Mathematics and Informatics, Vilnius.

Samir E. Ashoo, Troy Boudreau, D. A. L. (2008). *PC$^2$ Documentation - Version 9.1 What's New Document*. California State University in Sacramento. http://www.ecs.csus.edu/pc2/doc/v9/WhatsNewInV9.1.pdf.