Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

AUTOMATIC GRADING OF PROGRAMMING ASSIGNMENTS: PROPOSAL AND VALIDATION OF AN ARCHITECTURE

TRABAJO FIN DE MÁSTER

Julio César Caiza Ñacato

2013

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

AUTOMATIC GRADING OF PROGRAMMING ASSIGNMENTS: PROPOSAL AND VALIDATION OF AN ARCHITECTURE

Autor
Julio César Caiza Ñacato

Director
José María Del Álamo Ramiro

Departamento de Ingeniería de Sistemas Telemáticos

2013

A mi familia, de manera especial a mi hermana Pamela

Agradecimientos

Al Gobierno Ecuatoriano, que a través de la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación y su programa de becas Convocatoria Abierta 2011, me dio la oportunidad de poder realizar y culminar estos estudios.

A José María del Álamo por haber dirigido el presente trabajo. Sus ideas y recomendaciones me han servido de mucho durante la realización del mismo. De igual modo me ha ayudado a iniciar en el mundo de la investigación.

A mis amigas y amigos de siempre, aquellos que a pesar de la distancia y el tiempo, siempre están ahí, augurándome buenas cosas y esperando el tiempo de volver a vernos.

A mis nuevas amigas y amigos que he conocido gracias a vivir en esta Madrid cosmopolita. Porque me han brindado su sincera amistad y han hecho que toda esta experiencia valga aún más la pena. Seguro se añadirán a los amigos de siempre.

Finalmente, pero muy importante a mi familia, que son los que siempre me apoyan y me animan a seguir adelante.

Resumen

La calificación automática de tareas de programación es un tema importante dentro del campo de la innovación educativa que se enfoca en mejorar las habilidades de programación de los estudiantes y en optimizar el tiempo que el profesorado dedica a ello. La Universidad Politécnica de Madrid está interesada en este campo de investigación y dentro del proyecto "Sistema de Evaluación Automática de Prácticas de Programación" espera construir una herramienta para soportar dicha calificación automática. Así mismo, muchas instituciones académicas han reportado trabajos similares incluyendo detalles de la implementación y despliegue de las mismas; pero a pesar de tal cantidad de trabajos, aún quedan problemas por resolver. Uno de ellos y muy importante está relacionado con la diversidad de criterios para calificar las tareas de programación.

El presente trabajo tiene como objetivo el proponer y validar una arquitectura para soportar procesos de calificación automáticos de tareas de programación. La mencionada arquitectura provee de modularidad, extensibilidad y flexibilidad al proceso de calificación, que se traducen en la capacidad de soportar múltiples modos de calificación.

Para ello, en primer lugar se ha llevado a cabo una revisión sistemática de la literatura para proveer de un contexto al problema mencionado. Esta revisión contribuye con la identificación y construcción de una caracterización de criterios de calificación de tareas de programación, los cuales fueron reportados en trabajos similares. La descripción de las herramientas construidas en dichos trabajos ayuda a identificar una de ellas que pueda servir como base, para seguir con la implementación de nuevas características, y de este modo evitar el "reinventar la rueda". El plugin de Moodle, Virtual Programming Lab ha sido seleccionada como herramienta base. Además, la información recolectada en esta revisión ha servido para completar un conjunto amplio de requisitos para empezar con un proceso de desarrollo de software.

Se presenta entonces la definición, implementación y validación de la arquitectura siguiendo un modelo de desarrollo en cascada. Como primer paso, se realiza la definición de un nuevo artefacto de software nombrado como grading-submodule que permite evaluar código fuente considerando una métrica o un criterio de calificación determinado e independientemente del lenguaje de programación de dicho código fuente. A continuación se realiza: la identificación de un conjunto de requerimientos incluyendo aquellos funcionales y no funcionales, el análisis de la solución a desarrollar considerando la herramienta base, el diseño de la arquitectura y sus elementos, la implementación haciendo énfasis en consideraciones importantes acorde a las tecnologías utilizadas, y se termina con una validación a través de dos casos de estudio.

La arquitectura está basada en el uso de un orquestador que controla todo el proceso de calificación, teniendo en cuenta la información provista por un archivo de configuración. El proceso de calificación está definido por un conjunto de grading-submodules que pueden estar dispuestos de cualquier modo. Garantizando entonces la modularidad, extensibilidad y flexibilidad dentro del proceso de calificación

La validación se realiza en dos partes: la primera demuestra que la arquitectura puede ser llevada a la práctica, es decir puede ser implementada, para esto se han usado librerías Java importantes; la segunda parte de la validación se realiza a través de dos casos de estudio que se basan en tareas de programación reales dadas a los estudiantes en la Escuela Técnica Superior de Ingenieros de Telecomunicación en la Universidad Politécnica de Madrid.

Abstract

Automatic grading of programming assignments is an important topic in academic research. It aims at improving students' programming skills and optimizing the teaching staff time. Universidad Politécnica de Madrid is interested in this research field and is currently working on the project "Sistema de Evaluación Automática de Prácticas de Programación", which aims to build a tool to support this kind of grading. Several academic institutions have been interested in this research field as well. They have reported their works, which include the implementation and deployment of this kind of tools. But, in spite of the big quantity of work carried out in this field, there are still problems to be solved. One important gap is related to the diversity of criteria to grade programming assignments.

As a mean to solve the mentioned gap, this work aims to propose and validate an architecture to support the grading process of programming assignments. This architecture will provide modularity, extensibility, and flexibility features to that process. It implies the capability of supporting several different ways of grading assignments.

This work starts making a systematic literature review to get the context of the problem. This part of the work contributes to identify and characterize the grading criteria used in related works. Additionally, a description of already built tools is provided, which is helpful to choose a base tool and to continue working on it in order to avoid "reinventing the wheel". Virtual Programming Lab was selected as this base tool. Helpful information to complete a set of requirements, which allows starting a software development process, is provided as well.

Based on a waterfall development process, this work presents the design, implementation and validation of the mentioned architecture. This part of the work starts defining a new software artifact named grading-submodule, which allows evaluating source code considering a grading criterion or a grading metric independently of the programming language. After that, this work goes on with the identification of a set of functional and non functional requirements, the analysis of the solution considering VPL Moodle's plugin as base, the design of the architecture and the elements inside it, the implementation making important considerations to choose the most suitable technology, and a validation through two case studies.

The architecture is based on an orchestrator, which controls the whole grading process considering the information provided by a configuration file. The grading process can include a set of grading-submodules arranged in different ways. This features guarantee modularity, extensibility and flexibility in the grading process.

The validation is carried out in two steps: the first one is through the architecture's workability, which was carried out using powerful Java libraries; and the second one is through two case studies based on real programming assignments proposed to students at Escuela Técnica Superior de Ingenieros de Telecomunicación at Universidad Politécnica de Madrid.

Contents

Resumen i

Abstract iii

Contents v

List of figures ix

List of tables xi

Acronyms xii

1 Introduction 1

1.1 Justification 1

1.2 Goals 3

1.2.1 General 3

1.2.2 Specific 3

1.3 Structure of the document 4

2 State of the art 5

2.1 Automatic grading of programming assignments 5

2.1.1 Contrasting previous reviews 6

2.1.2 Analyzed key features 8

2.1.3 Tools 9

2.1.4 Current situation analysis 13

2.2 Grading criteria 16

2.2.1 Grading criteria characterization 16

2.2.2 Technologies to evaluate criteria 18

2.3 Technologies for orchestration 20

2.4 Virtual Programming Lab 22

2.4.1 Main features 22

2.4.2 Architecture 23

2.4.3 Technologies 24
2.4.4 The grading process 27
2.4.5 VPL's suitability 29
2.5 Learning management systems 29
2.5.1 Paid-for 30
2.5.2 Free and open source 30
2.5.3 Moodle 32
2.6 Chapter summary 33
3 Problem analysis 35
3.1 Actors 35
3.2 Defining the context 36
3.3 Requirements 38
3.3.1 Functional requirements 38
3.3.2 Non functional requirements 40
3.4 Solution approach 40
3.5 Existent tools' suitability 42
3.6 Defining the scope 46
3.7 The solution for the defined scope 47
3.7.1 Analysis of the VPL-Moodle subsystem 48
3.7.2 Analysis of VPL-Jail subsystem 48
3.8 Chapter summary 50
4 Design 51
4.1 VPL-Jail subsystem 51
4.1.1 Detailed architecture 51
4.1.2 Process perspective 54
4.1.3 Control perspective 56
4.1.4 Objects-oriented design 57
4.2 VPL-Moodle subsystem 66
4.2.1 Data Model 66
4.2.2 An abstract view of the grading process management module 69
4.2.3 User interfaces 71

4.3 Subsystems communication 73
4.3.1 Grading-submodules management 73
4.3.2 Students' submission 74
4.4 Chapter summary 74
5 Implementation and Validation 75
5.1 VPL-Jail subsystem implementation 75
5.1.1 Programming languages 75
5.1.2 Configuration file 76
5.1.3 Logging 78
5.1.4 Application's core 79
5.1.5 Application's submodules 81
5.1.6 VPL's integration 83
5.1.7 The lib directory 83
5.2 VPL-Moodle subsystem implementation 83
5.2.1 Programming languages 84
5.2.2 Configuration file 84
5.2.3 Lib file 85
5.2.4 User interfaces 85
5.2.5 VPL's integration 86
5.3 Subsystems communication 87
5.3.1 Code reusability 87
5.3.2 VPL-Moodle subsystem 87
5.3.3 VPL-Jail subsystem 88
5.4 Validation 88
5.4.1 Case study 1 88
5.4.2 Case study 2 99
5.4.3 Analysis 103
5.5 Chapter summary 104
6 Conclusions 106
6.1 Goals achievement 106
6.2 Main Contributions 108

6.3 Future Work 109

Bibliography 111

Annex I: Scientific article about the systematic literature review 114

Annex II: Description of important methods in the classes Orchestrator, CommandExecutor and GradingSumoduleProgram 125

Annex III: Detailed description of the case studies (in Spanish) 131

List of figures

Figure 1.VPL's architecture 23
Figure 2. XML–RPC request.....26
Figure 3. XML–RPC response. 26
Figure 4. Automatic grading process 28
Figure 5. High-level use case diagram 37
Figure 6. Solution approach 41
Figure 7. VPL-Moodle and VPL-Jail subsystems 47
Figure 8. Grading process inside the VPL–Jail subsystem 49
Figure 9. Default architecture for grading in VPL 52
Figure 10. Architecture considering many metrics to grade in VPL.....52
Figure 11. Proposed architecture for grading process 53
Figure 12. Elements and calls inside the grading process 55
Figure 13. Grading process inside the VPL-Jail subsystem 56
Figure 14. Controllers in the VPL-Jail subsystem.....57
Figure 15. Class diagram for orchestrator and grading-submodules 60
Figure 16. Extended class diagram with *GradingSubmoduleProgram* subclases 63
Figure 17. Package diagram of the orchestrator and grading-submodules levels.....64
Figure 18. Grading process interactions 65
Figure 19. Data directories inside the VPL-Moodle subsystem.....67
Figure 20. Semantic data model for the grading module 68
Figure 21. Architecture for the grading process management 70
Figure 22. View of grading-submodules management 71
Figure 23. View of grading-submodules creation / addition 72
Figure 24. General view of grading process 72
Figure 25. View for configuration of grading-submodules 72
Figure 26. Communication between subsystems in the grading-submodules management 73
Figure 27. Structure of the XML configuration file 76
Figure 28. XML configuration file for the logger in the grading module.....78
Figure 29. Piece of annotated code to map the XML configuration file 80
Figure 30. Structure for the *run()* method 82
Figure 31. Adding a new grading-submodule for testing Java programs.....91
Figure 32. Grading-submodules management user interface.....92
Figure 33. VPL activity selection.....93
Figure 34. VPL activity (programming assignment) creation 93
Figure 35. Settings' management for a VPL activity.....94
Figure 36. Execution options for a VPL activity.....94
Figure 37. Requested files for a VPL activity.....95
Figure 38. Grading process web form.....95
Figure 39. Definition of grading-submodules inside the grading process.....96

Figure 40. Definition of grading-submodules inside the grading process..........97
Figure 41. Configuration of grading-submodules' parameters inside the grading process..........97
Figure 42. Grading process already configured..........98
Figure 43. Student interface – assignment description..........98
Figure 44. Student interface – uploading the requested file..........99
Figure 45. Student interface – feedback provided..........99
Figure 46. Grading-submodules management considering the StyleGradingSubmodule..........101
Figure 47. Parameters for StyleGradingSubmodule..........102
Figure 48. Grading process considering a StyleGradingSubmodule..........102
Figure 49. Student interface – feedback considering code style..........103

List of tables

Table 1. Mature tools 14
Table 2. Recently developed tools 15
Table 3. Grading criteria characterization 17
Table 4. Tools for evaluating programs 19
Table 5. Functional requirements 39
Table 6. Non functional requirements 40
Table 7. Automatic grading tools suitability 43
Table 8. VPL's fulfillment of all requirements 44
Table 9. Requirements considered for this work 46

Acronyms

API	Application Programming Interface
bLTI	Basic Learning Tools Interoperability
CPU	Central Processing Unit
CRUD	Create – Read – Update - Delete
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
DoS	Denial of Service
DSL	Domain Specific Language
EHEA	European Higher Education Area
ETSIT	Escuela Técnica Superior de Ingenieros de Telecomunicación
FOSS	Free and Open Source Systems
FR	Functional Requirement
GNU	GNU's Not Unix
GPL	GNU General Public License
GUI	Graphical User Interface
HDL	Hardware Description Language
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technologies
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standarization
J2EE	Java Enterprise Edition
JAWS	Java Web Start
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for Restful Web Services
JSF	Java Server Faces
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol
LMS	Learning Management Systems
LRN	Learn Research Network
MOODLE	Modular Object Oriented Dynamic Learning Environment
NATO	North Atlantic Treaty Organization
NFR	Non Functional Requirement
Oauth	Open Authentication Protocol

OSGi	Open Services Gateway Initiative
PHP	HyPertext Preprocessor
POM	Project Object Model
RADIUS	Remote Authentication Dial In User Service
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
RSS	Rich Site Summary
SCORM	Sharable Content Object Reference Model
SEAPP	Sistema de Evaluación Automática de Prácticas de Programación
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCP	Transport Control Protocol
UC	Use Case
UDP	User datagram Protocol
UML	Unified Modeling Language
UNESCO	United Nations Educational, Scientific and Cultural Organization
UPM	Universidad Politécnica de Madrid
URI	Uniform Resource Identifier
UTF8	UCS Transformation Format -8bit
VHDL	VHSIC Hardware Description Language
VPL	Virtual Programming Lab
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

1.1 Justification

One of the priorities in the EHEA (European Higher Education Area) is encouraging the lifelong learning. This is a program that aims the inclusion of "people at all stages of their lives"[1] to contribute to develop education and training. ICT (Information and Communications Technologies) is considered as one of the four key points to support the program. They provide of help to make innovative practices, improve access to education and develop advanced management systems[2].

Additionally, the EHEA implementation brought a change in the typical teaching – learning process. It means, changing from an environment focused in just teaching to a new one, where there is a tutor guiding the student's learning (Méndez 2008) (Martínez 2011).

To do a good job as a tutor, it is necessary to trace the students' improvement. This is quite difficult considering the diversity and big quantity of students. ICT can be used to help teachers. The main advantages of this kind of tools include availability, distance suppression (maintaining student-teacher contact) (Méndez 2008), possibility to work with a lot of students, and so on.

There is a good set of ICTs to help in education. They include technologies that are oriented to a general scope and can be used in any education field, LMSs (Learning Management Systems) for instance. They are broadly used around the world. In Spain most universities use this kind of tools (García González et al. 2010). In the specific case of UPM (Universidad Politécnica de Madrid), Moodle[3] (Modular Object Oriented Dynamic Learning Environment) is used to get support on many tasks for different courses.

ICTs can be used more specifically as well. They can be oriented to do specific tasks in a given course. Programming courses can use this kind of technologies to improve the student's learning and to increase their skills. For example a tool of this kind could provide an automatic grading of the students' assignments. This tool would help students to receive their grades and good feedback quickly and it would be helpful for them to improve their programming abilities. For teaching staff, it would be useful to

[1] http://ec.europa.eu/education/lifelong-learning-programme/index_en.htm
[2] http://ec.europa.eu/education/lifelong-learning-programme/ict_en.htm
[3] http://moodle.org/

avoid the excessive and maybe repetitive workload. The saved time could be used in more focused task in the same programming learning process.

Considering aforementioned context, the UPM inside the program "Ayudas a la Innovación Educativa y a la Mejora de la Calidad de la Enseñanza" carries out the project SEAPP (Sistema de Evaluación Automática de Prácticas de Programación). This project aims to implement a tool for automatic grading of programming assignments to help students in their learning process and to support teachers in tutoring and tracing students' improvement[4].

Likewise to this case, other institutions reported similar requirements. Several researches have informed about the development of this kind of software tools and their correspondent implementations and deployments. The fundamental goals included providing a good feedback and optimizing the teaching staff time. Additionally, these projects informed about additional gaps. These were related to plagiarism detection, provision of a secure test environment, controlled resources' use, the diversity of ways for grading (Higgins et al. 2005), the definition of pedagogical models (Choy et al. 2008), and so on. These reports have been studied in some reviews, which can help to get a current perspective of this research field.

Douce et al. in (Douce et al. 2005) reported as the main improvements in the reviewed tools, the orientation of using web-based technologies for resources' access and the increment in support for more programming languages. They proposed as future work the grading of GUI (Graphical User Interface) programs, meta-testing (test of tests), LMS integration, means to protect the system against intentional or unintentional malicious code, and support for web programming.

A few years later Ihantola et al. in (Ihantola et al. 2010) and Romli et al. in (Romli et al. 2010) reported improvements in systems integration with LMSs, in security for the host system, and the use of static and dynamic analysis inside the grading process. Additionally, they reported the lack of a broad tool's adoption (due to every tool had been built considering specific requirements), and the lack of a common grading model. In the first case they made a set of suggestions which include working on open source projects. Some projects followed this suggestion and their acceptance have grown (Edwards et al. 2008) (Rodríguez-del-Pino et al. 2012). In (Romli et al. 2010) the building of a flexible and configurable system was proposed, which seemed a good path to reach broad adoption. In the second case, the lack of a common grading model is due to the fact that every institution and even every teacher has his own way to grade an assignment. In reviews carried out in 2010, the correctness is reported as the

[4] http://innovacioneducativa.upm.es/proyectosIE/informacion?anyo=2012-2013&id=954

main criterion considered to grade. At that time, there was not a common approach yet; maybe the first step to build a model could be to characterize grading criteria.

Most recent works have reported new improvements. Thus, RoboLIFT (Allevato et al. 2012) has the feature of grading GUI applications; web programming languages have been considered as well, for example VPL (Virtual Programming Lab) in (Rodríguez-del-Pino et al. 2012) reported grading of PHP (HyPertext Preprocessor) programs.

Nowadays, the problem of having a common model to grade persists. Then it is an open research path. To propose a model to grade any programming assignment that works for any teacher and for any institution would be very complicated or impossible. The reason is that different criteria will persist. The solution could start considering a more high level perspective, looking for a configurable process where different models could be supported, thus any grading metric or criterion could be selected as the academic staff needs.

1.2 Goals

1.2.1 General

To propose and validate a new architecture to support an automatic grading process, which will be extensible, flexible and modular to support many ways of assessments.

1.2.2 Specific

- To use the knowledge about **scientific research**, which was acquired in the master course, in a real problem.
- To make a **systematic review of related works** to get an actual context in automatic grading of programming assignments.
- To identify and use the most suitable features of **software engineering**, which can be applied in this work.
- To **gather a set of requirements** based on necessities of the students and the teaching staff inside the teaching-learning process of programming subjects.
- To **analyze the requirements and the context** to propose a suitable solution for the given problem.
- To apply principles of **software and services architecture to design a solution** for the given problem.
- To **validate** the archi**tecture proposed through the implementation of** a working prototype and with the use of it in real case-studies.
- To evaluate the results for establishing **conclusions and future works**.

- To **disseminate research** results through scientific publications in international forums.

1.3 Structure of the document

This work has been organized in five chapters after this introduction. The state of the art includes: a systematic literature review of tools for automatic grading of programming assignments; a characterization of criteria to grade programming assignments and technologies used to evaluate them are reviewed as well; a description of important features of orchestration technologies; a deep review of a chosen tool, which will be used as base to implement new features; and a sight in LMSs used nowadays. Next, the problem analysis aims to define the scope of this work and to propose a solution based on the requirements' analysis. After that, the design chapter explains how the solution will work and provides of useful software artifacts to help the implementation stage. The validation chapter makes the first level validation focused on implement the proposed architecture. It shows important considerations done while implementing the whole solution's modules. These are related with programming languages, useful libraries, integration issues, and so on. Additionally, a second validation is done through two case-studies based on real programming assignments. Finally, the last chapter shows the goals achievement, main contributions and future work.

2 State of the art

The previous chapter has established the context for this work. The main goal is to propose and validate a new architecture for automatic grading of programming assignments. It makes necessary to review which similar works have been reported.

The first appearance of a tool for automatic grading of programming assignments was reported in 1965 (Forsythe et al. 1965). It has been almost fifty years since that happened and nowadays there are a good number of tools. Establishing the actual situation of this research field will help to support that the given problem was correctly identified. Then, a systematic review is essential and it will be presented here.

At the same time of carrying out this review, the decision between building a new tool and taking one of the already built tools to validate the proposed architecture was made. Actually, the second choice was more suitable and therefore a deeper description about the selected tool is presented as well.

The systematic literature review shows an important element inside the grading process, it is the grading criterion. A grading criterion is always related to one grading metric at least. A characterization of grading criteria is presented to start with the solution of the problem. Additionally, almost every characterized criterion has the support of a tool to evaluate it, so a table of these tools is presented as well.

To manage the use of different grading criteria, their arrangement and their calls, a technology to orchestrate is necessary. Some technologies which could act as orchestrator are reviewed.

In spite of the proposed goals, it is necessary to consider that the goal of the SEAPP project is to provide a solution to UPM. One of the requirements given by this institution is the integration with its current LMS, Moodle. So, a quickly sight about existent LMSs is provided. Some important features about Moodle are highlighted as well.

2.1 Automatic grading of programming assignments

The common goal to build or to use this kind of tools has been to improve programming skills in the students, paying special attention to beginner students. The skills will be improved through solving many programming exercises. Students can go on the problems as quickly as they get good feedback. It would help them to understand their mistakes and therefore to improve their skills. Additionally students get a real benefit, which is to get a fair grade not dependent on personal considerations of the academic staff (Higgins et al. 2005).

Considering the quantity of students in a regular class of engineering and a big number of programming exercises, manual grading is not viable. The idea is not overwhelm the academic staff either, so another goal is to optimize the time of academic staff. The saved time could be used in more productive processes like planning and designing the lectures or just giving more personal attention in focused problems.

As the research in this field increased, new goals were proposed. Thus, in (Patil 2010), (Rodríguez-del-Pino et al. 2012), (Yusof et al. 2012) and (Queirós et al. 2012) an extra goal is getting the integration with a LMS to improve the performance of the programming assignments grading process. In (Amelung et al. 2008) was proposed the use of services to reach this goal. In (Spacco et al. 2006) one goal was to collect detailed information to research deeply the students' skill improvement process. More recently, Allevato and Edwars (Allevato et al. 2012) have as a goal to get the interest of students using the popularity of smartphones and mobile applications.

2.1.1 Contrasting previous reviews

Given the big quantity of already built tools since the first appearance, it is better to take advantage of previous literature reviews. When contrasting these works, it is possible to consider reported gaps, to identify which of them were solved, and which have persisted.

Douce et al. in (Douce et al. 2005) made a good and quick characterization of the evolution of this kind of tools until 2005. They reported three generations of tools. The first one refers to times when working on operating systems and programming languages was necessary, and the grading was only made considering a right or a wrong answer. The second generation refers to working with tools, which came with the operating system, to build new tools. C and Java languages were mostly used in development. The third generation is just around the time that this work was done. The main features in the reviewed tools were the orientation of using web-based technologies. An increment in support for more programming languages was reported as well.

Considering that Douce et al. (Douce et al. 2005) gave future paths for automatic grading of programming assignments, and Ihantola et al. (Ihantola et al. 2010) and Romli et al. in (Romli et al. 2010) made works covering tools developed until 2010, it is possible to contrast them to show the improvement in some issues. These issues can be classified as technical, pedagogical and for a system adoption.

Technical issues

Douce et al. indicated some research paths in (Douce et al. 2005), which included grading of GUI programs; meta-testing which refers to qualify applied tests; use and configuration of safe systems to test the programming assignments, the idea is to protect the host system of intentional or unintentional malicious code; integration of systems to avoid overwhelm the user, usually the idea would be integrate the tool with an LMS, it can be reached using web-services; and support for web programming grading because the use of web technologies in the normal life had increased, so universities started to teach web programming and grading this kind of assignments was necessary.

Ihantola et al. in (Ihantola et al. 2010) and Romli et al. in (Romli et al. 2010) reported improvements in systems integration with LMS and in security for the host system. Then, issues like grading of GUI programs, meta-testing, and support for web programming stayed waiting for more research.

Pedagogical issues

The reviewed works lack a common grading model. Every institution and even every teacher has his own way to establish a grade. So a reference model could be helpful. In reviews did in 2010, the correctness was reported as the main criterion to grade. Some works started to use static and dynamic analysis as well, but in general, every work proposes its own criteria set to grade. As a result, at that time, there was not a common approach yet; maybe the first step to build a model could have been a characterization of grading criteria.

About feedback, there were some implications: quickly feedback could trigger trial-error practices, how much useful is the automatic feedback, and which is the adequate quantity of feedback. Some works try to provide flexibility, through configuration of levels of feedback and allowing manual and automatic solutions (Edwards et al. 2008).

Some tools have considered the implementation of plagiarism detectors; this will be seen later in this chapter. Usually the plagiarism control module is inside an additional module but without affecting the grading process. Trying to consider a plagiarism inside the grading process could be reflected in a too much time required to grade an assignment. It is for sure that a module of this kind is necessary and a sanction in detected cases as well.

Systems' adoption

Regarding systems' adoption, both works (Ihantola et al. 2010) and (Romli et al. 2010) showed that a big number of tools had been built but they were not broadly used. It is because every tool had been built considering specific requirements. An important way to increase the adoption was to work on open source projects. Some projects had done this and its acceptance grew (Edwards et al. 2008) but a definitive broadly used tool has not been reached. In (Romli et al. 2010) was proposed the building of a flexible and parameterizable system and it seemed a good path to reach this goal.

2.1.2 Analyzed key features

It is necessary to define a set of key features to be able to evaluate the reported tools. The next key features have been defined considering they are important in implementation and deployment stages:

- Supported programming languages. It is a very important feature when making a quickly implementation is considered. It could define the use or not of a tool.
- Programming language used to implement the tool. This feature has great relevance when there is a set of policies regarding the software used in an institution. In the case of customization or maintenance, it would be a valuable feature to choose a tool.
- Logical architecture. It is an important feature when a modification of the tool is being considered. This architecture will show the modularity, extensibility and flexibility level. It could show how the different modules work and how the system could connect with other systems.
- Deployment architecture. It shows how the hardware over which the tool works is. It is helpful to know if a current environment will support the deployment of a tool. In the worst case it will indicate the resources needed and therefore will help to determine the cost of a possible deployment.
- Work mode. It indicates if the tool can work alone, or it work as a plugin when integration with another system (an LMS for instance) is required.
- Grading criteria. It includes a set of criteria on which the tool can establish a grade. It can include metrics associated to a given criterion. Even, how the grade calculation is done.
- Technologies used by the tool. It is helpful when deploying or building a new tool is considered. For a deployment case it is helpful to establish compatibility between the tool and a legacy system. It is useful for future maintenance as well. In a building case knowing which technology (standards, protocols, libraries, etc.) could be used to face a requirement is very helpful as well.

2.1.3 Tools

The previous mentioned reviews showed relevant information about tools already built and reported until 2010. It is necessary to make a new review of tools built in the last years. Additionally, it is worth considering some important tools that have had a continue actualization since their creation (CourseMarker, Marmoset, WebCAT, and VPL). All of them are considered in the next review.

CourseMarker

A tool developed in the Nottingham University to avoid the particular criteria of teaching staff. The main advantages are considered being scalability, maintainability, and security (Higgins et al. 2005). The supported programming languages for grading are Java and C++ and it has been built using Java. Its architecture shows 7 subsystems: login, it controls all the authentication process; submission, it receives the different submissions precisely; course, it stores information about the process; marking, it has in charge the grading process, and the storing of the submitted files and marks obtained; auditing, it has as responsibility to log all actions; and a subsystem to control the communication among the others.

As criteria to establish a grade it considers typography (indentations, comments, etc.), functionality through test cases, programming structures use, and verification in the design, and relations among the objects.

It works with technologies like Java RMI (Remote Method Invocation) for communication among the subsystems, regular expressions to verify results and DATsys (Higgins et al. 2002) to verify objects design.

Additional important features include: the capability to work with feedback levels, the orchestration among subsystems is defined by a configuration file, feedback and grades can be customized, there is plagiarism detector when grading, submissions number and CPU (Central Processing Unit) quantity are configurable, and finally there are security considerations which include: detection of malicious code and execution in a sandboxed environment.

Marmoset

It has been built in the University of Maryland. Its main goal is to collect information about development process to improve the student skills (Spacco et al. 2006). Its main advantages are making a complete snapshot about the student's progress, so the student development can be analyzed in detail; using different types of test cases (student, public, release, secret); and a personal support through comments' threads on the code.

Originally the paper reported grading of code written in programming languages as Java, C, Ruby and Caml Objective. Now, the official web page[5] informs that it works with all different programming languages. The architecture includes: a J2EE (Java Enterprise Edition) webserver, a SQL (Structured Query Language) database, and one or more build servers. These last are used in a safe and lonely environment to prevent effects of possible malicious code. The build servers' arrangement helps to provide scalability and security. The criteria to establish a grade include dynamic and static analysis. The dynamic analysis is done through test cases.

WebCAT

The main features are the extensibility because of its plugins-based architecture and a grading method based on how well students grade their own code (Edwards et al. 2008). The architecture design provides a set of important features: security, it is provided through means like authentication, erroneous or dangerous code detection; portability, because it has been built as a Java servlet; extensibility and flexibility, it is inherent to the architecture; and support for manual grading as well, it is because the academic staff can check students' submissions and enter comments, suggestions, and grade modifications. The official wiki[6] affirms that it is the only tool that integrates all these features.

The tool supports Java, C++, Scheme, Prolog, Standard ML, and Pascal, but it offers flexibility to support any programming language. The grade is based on code correctness (how many tests are passed), test completeness (which parts of the code are actually executed), and test validity (test accurate-consistent with the assignment). Additionally plugins can provide more metrics for grading (static analysis for instance). Additional features include: there are a lot of plugins for Eclipse and Visual Studio .NET IDEs, and it has been licensed as Affero GNU/GPL (GNU General Public License).

Grading tool by Magdeburg University

It has a really interesting goal, which is providing a tool which is not forced to work with a given LMS, but avoiding the use of two systems independently (Amelung et al. 2008). It can be reached using services. It shows a configurable focus. Then there are selectable components like the compiler, the language interpreter, the grading method, and the data set. The submissions' number, and time features are configurable as well.

The tool uses dynamic tests, compilers and interpreters to establish the grade. The supported programming languages include Haskell, Scheme, Erlang, Prolog, Python, and Java. The architecture is very interesting. It considers three servers: the front-end,

---

[5] http://marmoset.cs.umd.edu/

[6] http://wiki.web-cat.org/WCWiki/WhatIsWebCat

it will be an LMS system; the spooler server, it controls the request, the submissions queues and the back-end calls; and the back-end servers, which are the modules to evaluate a programming language. To communicate the servers, XML-RPC (Extensible Markup Language - Remote Procedure Calls) has been used.

JavaBrat

It is a tool reported in (Patil 2010), and built as a master thesis in San José State University. It gives support for two programming languages, Java and Scala. It uses Java to develop the grader software and PHP to build a plugin for Moodle. The design includes three important modules: a Moodle server with a plugin; a module which contains the graders depending on language and a repository of problems; and the last module is Javabrat which has a set of services to call graders and problems.

Although it can works as a Moodle plugin, this tool can work alone through a web interface developed as part of the project. This web interface was developed using JSF (Java Server Faces) 2.0. The services are implemented using JAX-RS (Java Api for Restful Web Services).

The work was centered in develop the web interface and the problems' repository. Then the grading process is not very complex and it is based on correctness, which is determined by test cases. It is a semi automatic tool because a revision of the report, generated when the grading process is done, is necessary.

AutoLEP

A tool developed in Harbin Institute of Technology and which is presented in (Wang et al. 2011). It has as main feature the combination between static and dynamic analysis to give a grade. The dynamic analysis refers to evaluating the correctness using test cases. The static analysis doesn't need to compile or execute the code. It is just about to make a syntactic and semantic analysis and it is reported as main difference with previous works.

The architecture includes: the client, a computer used by a student, it does the static analysis and can provide of a quickly feedback; a testing server which has to do the dynamic analysis; and a main server which has to control the information of the other components to establish a grade.

Petcha

A tool developed in University of Porto. Its main goal is the building of an automatic assistant to teach programming (Queirós et al. 2012). An important feature is the coordination among existing tools like IDEs (Integrated Development Environment), LMSs and even automatic graders. It supports the programming languages that IDEs do. The tested IDEs are Eclipse and Visual Studio.

Its architecture is defined as modules for every connected tool. Then, there is a module for the LMS, the IDE, the exercises repository, and for the grading engine. It relies on some technologies to guarantee interoperability: IMS Common Cartridge as format to build packages with resources and metadata, IMS Digital Repositories Interoperability and bLTI (Basic Learning Tools Interoperability). Additionally it used JAWS (Java Web Start) to build the client interface and it is working with MOOSHAK (Leal et al. 2003) as grading engine.

JAssess

It has been built by researchers in two universities in Malaysia, University of Technology and Tun Hussien Onn University (Yusof et al. 2012). Their goal is to have only one interface to access the grading process. JAssess is presented as an integrated tool with Moodle.

Their architecture shows the next modules: Moodle server, MySQL server, JAssess, and JAssesMoodle to communicate Moodle and JAsses.

About supported languages it only supports Java, and precisely it is the language used to build the tool. This tool uses technologies as Java File, Java Unzip, Java Runtime, Java Compiler and Java Reflection. About the criteria considered to grade, it is a weakness for the tool because it only depends on compilation. The evaluation process is not completely automatic.

RoboLIFT

The main approach is to get interest of students in programming using the popularity of mobile applications and smartphones (Allevato et al. 2012). The increasing market of android smartphones and applications makes increase the interest of students. This knowledge will be helpful when they will finish their studies as well. The tool supports grading of Android applications.

The tool is based on WebCat (Edwards et al. 2008), so the architecture is the same with an additional variation. The variation is the use of Robolectric[7], which is software to accelerate the grading process. The tool uses the development tools for Eclipse provided by Google.

Unit testing is considered to establish a grade. The tests are of two sorts, public and private tests. The students know the first one kind, and the second type is only used in the definitive submission.

[7] http://pivotal.github.com/robolectric/index.html

Virtual Programming Lab (VPL)

A tool built in Las Palmas University (Rodríguez-del-Pino et al. 2012). The goals of the project include to provide the students with many programming assignments, and to support the managing and grading process. The tool supports many programming languages including Ada, C, C++, C#, FORTRAN, Haskell, Java, Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL, and VHDL (VHSIC Hardware Description Language).

The architecture includes three modules: a plugin for Moodle, which allows the tool's configuration and making submissions; a browser-based code editor, which allows coding without the necessity of an installed compiler; and a jail server, which hosts the environment where the assignment will be evaluated. To develop this tool they have worked with PHP to build the Moodle plugin. To implement the jail server, C++ has been used. Every language has an associated Linux shell script for evaluation as well. The communication between Moodle and jail servers is done with XML RPC. The jail server provides their services through a Linux program called *Xinetd*. In addition the jail server implements a safe environment with the *Chroot* Linux program.

For grading it considers the correctness, done through test cases (in the default configuration). The test cases are specified in an own and easy syntax. The default scripts, which evaluate the programs, can be changed to improve the evaluation method.

Additionally, this tool has some interesting features that include: being built under GNU/GPL license, allowing automatic and semiautomatic grading processes, providing of a plagiarism control tool, and having configurable features for every assignment.

Moodle extension by Slovak University of Technology Bratislava

It is presented in (Jelemenská et al. 2012). Its main goal is managing and modeling digital systems using HDL (Hardware Description Language). The work reports managing features like assignments managing, and user type definitions. The only language supported is VHDL.

The tool evaluates a submission based on: compilation and syntactic analysis, functionality doing comparisons with a model, and then through a stage to detect plagiarism.

2.1.4 Current situation analysis

The key features of each tool reviewed in the last section can be used to identify the real improvements since the last literature reviews (Ihantola et al. 2010)(Romli et al. 2010) were carried out.

To analyze the improvements in a temporal perspective, two tables with tool's features are shown. Table 1 joins tools built a few years ago, previous to the work presented by Ihantola (Ihantola et al. 2010) which have been updated continuously. Precisely by their maturity, they count with really good features and in some cases with a broad use.

Table 2 shows more recent tools, which have not been broadly used but that present new features and propose new research lines.

Firstly it is necessary to consider the pending issues reported until 2010. They were mentioned in an earlier section and include: technical issues which include lack of a GUI grading tool, meta-testing, and support for web programming; pedagogical issues including lack of a model to grade, trial-error practices, adequate quantity of feedback, and plagiarism; and adoption issues.

Table 1. Mature tools

Tool's name	Main Features	Supported Languages	Work Mode	Grading criteria
CourseMarker	Scalability, maintainability. Security, configurability. Plagiarism detection. Work with levels of feedback.	Java, C++.	Standalone	Typography. Correctness. Structures use. Objects design. Objects relations.
Marmoset	Detailed information. Language independence. Security and scalability for evaluation module. Apache 2.0 license	Any language.	Standalone	Dynamic and static analysis.
WebCat	Extensibility and flexibility based on plugins. Access security. Portability. Semi and automatic process. GNU Affero license.	Java, C++, Scheme, Prolog, Standard ML, and Pascal. Flexibility for any language.	Standalone	Code correctness. Completeness. Test validity. Extensible by plugins.
Virtual Programming Lab	Moodle integration. Customizable grading mode. GNU GPL license. Plagiarism detection. Configurable activities. Jail environment.	Ada, C, C++, C#, Haskell, FORTRAN, Java,Octave, Pascal,PHP, Prolog, SQL, Ruby,Python, Scheme,Vhdl.	Moodle plugin.	Correctness based on test cases. Open for new methods.
Grading Tool (Magdeburg University)	Use of services. Configurable evaluation process.	Haskell, Scheme, Erlang, Prolog, Python, Java.	LMS extension.	Compilation. Execution. Dynamic tests.

Plagiarism has been seen as an important module inside an automatic grading tool. Then, some projects have already considered its implementation.

As it can be seen most of these issues have been solved. Thus, RoboLIFT has the feature of grading GUI applications because it uses LIFT, a library included in the WebCat project to grade GUIs. Web-oriented programming languages have been considered as well, VPL can grade PHP programs for instance. About meta-testing, WebCAT refers to validate tests. It can be done through determining how much of the code is being covered by executed tests.

The adoption of a tool depends on some features, which include: how long the tool has been tested, if the tool has been developed as open source, how flexible, scalable, and configurable the tool is.

Table 2. Recently developed tools

Tool's name	Main Features	Supported Languages	Work Mode	Grading criteria
JavaBrat	Use of services. LMS integration.	Java, Scala	Moodle plugin. Standalone.	Correctness.
AutoLEP	Static and dynamic analysis to grade.		Standalone	Static analysis. Dynamic analysis.
Petcha	Coordination among existing programming-support tools. Use of technology for interoperability.	Languages supported by Eclipse and Visual Studio.	Standalone	Based on test cases.
JAssess	Moodle integration.	Java	Moodle plugin.	Compilation
RoboLIFT	Grading mobile applications. GUI grading.	Java	Standalone	Unit testing (public and private).
Moodle ext. (Slovak University of Technology)	Oriented for digital systems. Plagiarism detection.	Vhdl.	Moodle plugin.	Compilation. Syntactic analysis. Functionality by comparison.

There is a big quantity of tools for programming assignments automatic grading. Then, does it make sense to continue building new ones? Usually the main reason to build a new tool is that the existing ones do not fulfill our requirements. If this is the case, to get the tool and extend it through a plugin may be a good idea.

Table 1 shows important information that supports the reuse of tools; this is based on existing tool features like extensibility, flexibility and configurability. All these set of features could guarantee the cover of many requirements for a given case; and for specific requirement it could be possible to build only an extension for the tool. It would reduce the implementation time.

The information in Table 2 shows that Java is the most common supported language by recent tools. Older tools have already supported this language and this cannot be a sufficient reason to build a new one. If new support for a given language is necessary, it can be done through adding a new submodule or plugin to extensible tools as well.

A remarkable fact is the use of LMSs in most universities. The ideal thing would be to seamlessly use the automatic tool within the LMS. Some recent tools are considering the integration with an LMS but they do not provide features like extensibility, flexibility, and maintainability as the older ones do. Maybe the next step to evolve with automatic tools is to add the LMS integration feature to the set of features of the mature tools. Probably it could be reached by a redesign of the tools to allow different gates (user interfaces) to access to the system. This goal could be reached through use of services. The gate could be a module in the LMS or a module developed in any technology to build user interfaces.

Finally, the lack of a common model to grade is still an important and persistent problem. Every institution and even every teacher has his own criterion to grade an assignment. Then, if defining a common model for grading is not possible, a solution could be designing and implementing a flexible architecture that supports different ways of grading programming assignments.

2.2 Grading criteria

As it was said in the former section, the lack of a common model to grade is still an important problem. Every institution and even every teacher has his own criterion to grade an assignment. Additionally, as Rodriguez at al. in (Rodríguez del Pino et al. 2007) say, it is necessary to recognize that some criteria cannot be measured. The creativity or the right sense of a comment cannot be determined by an automatic tool. Leaving out this kind of criteria, and considering the importance of defining a frame, a characterization of criteria is proposed as a first step to reach a possible grading model.

2.2.1 Grading criteria characterization

The importance of a characterization can be inferred by seeing Tables 1 and 2. There is diversity of criteria to grade. For example one tool just considers whether the code can be compiled while others include a criteria set (even considering extensibility of them through new plugins). Additionally, some tools refer to the same criterion by different names.

Looking at all grading criteria expressed in the previous tables, Table 3 shows a characterization for grading criteria. This characterization takes the diverse criteria and put them in a common and organized representation. This has been done considering applicable quality attributes given in (Sommerville 2005). These attributes are considered as external and cannot be measured directly. It is necessary to measure more internal attributes for software, they are the metrics. Quality attributes and metrics are related through a medium element, it is the criteria (Yelmo 2012). For example maintainability is a quality attribute, which can use some criteria to be determined, one of them is complexity; and this criterion can make use of some metrics, one of these metrics could be the number of flow control structures used in a program.

Table 3. Grading criteria characterization

Quality attributes (external)		Criteria
Execution		Compilation
		Execution
Functional Testing	Functionality	Correctness (system or method level)
Non Functional Testing	Specific requirements	Specific requirement for an exercise
	Maintainability	Design
		Style
		Complexity
	Efficiency	Use of physical resources
		Execution time
		Processes load
		Code weight

The quality attributes and criteria cannot be directly quantified but they can be evaluated through metrics quantification. The, there are some software tools that allow evaluating criteria through reports about metrics quantification. Thus:

- For compilation, a language compiler will allow knowing about the number of errors and warnings.
- For execution, a language interpreter will show some data including the success of a program, the number of warnings, and the number of thrown exceptions.
- For functionality, a program based on test cases (JUnit[8] in Java for instance) can be used. This will report the number of total and failed tests.
- For specific requirement, a particular program to see the use or not of a programming structure for instance will be necessary.
- For design, style and complexity, an external program will be needed (Checkstyle[9] for style in Java for instance).
- For the last four criteria, it could be useful shell script programs.

Some of the already reviewed tools offer the possibility of support any grading criterion through the building of plugins. Considering a complete grading process would be better. This grading process would have as features: a high level of

[8] http://www.junit.org/
[9] http://checkstyle.sourceforge.net/

configurability and flexibility to support any metric. The goal is not see just a metric or criterion; it is to consider the whole grading process.

2.2.2 Technologies to evaluate criteria

Considering that the goal is to propose an architecture to support a flexible grading process, and taking into account the previous criteria's characterization; here is presented a set of open source tools, which allow evaluating a criterion or some of them in source code files.

Initially, every criteria of the characterized set could be evaluated through our own programs or tools, which would be designed to reach a specific goal. But taking advantage of already built tools would be possible; these tools are shown in Table 4. This would avoid unnecessary implementations. The features about input parameters and output results could be used as helpful information to define wrappers to support them inside the proposed architecture (it will be seen in the design chapter).

Table 4 shows that most of these tools are focused on evaluating criteria for Java programs. This fact is because most of these tools have been developed based on JUnit testing framework. But, they add additional features as the possibility of getting data from a database, of evaluating GUIs, and returning evaluation reports for instance.

These tools evaluate criteria as correctness, use of physical resources, execution time, processes load, execution time, design, style and complexity. Correctness is the most supported criterion. It could be the reason which explains that most of the automatic grading tools seen in the systematic literature review carried out only use correctness to grade.

Some criteria have not been taken into account for the considered tools. These are compilation, execution, specific requirements, and code weight. It makes sense. In the first two cases it is because every language has its own associated compiler, debugger and program interpreter. These three elements will be used depending on the type of language: compiled or interpreted. For the third case, precisely its conception makes necessary a specific program. The program would evaluate the use of a given command or programming structure for instance. In the last case, it would be easy to implement it using a function or a shell command.

Table 4 provides useful information to define a wrapper when designing the solution. As first sight, it is notable that the input to every tool always includes a file or a parameter additionally to the program to be evaluated. For the output, all tools provide a result through the console or in a formatted file. It gives the idea that reading the standard output or to look for information in result files would be necessary.

Table 4. Tools for evaluating programs

Tool's name	Supported programming languages	Input	Output mean	Criteria
Junit	Java	Test cases file.	Console	Correctness
Feed4JUnit	Java	Test cases file. Test data for DB (Data Bases) or data sources (csv or excel files).	Console	Correctness
Cucumber	Ruby, Java, .Net, Flex, C#, Python.	Behaviour file, definitions file and steps definition file.	Console	Correctness
Luaunit	lua	Test cases file.	Console. XML file.	Correctness
Maverix	Java [GUI]	-	Console. Web report.	Correctness
Robotium	Android	Test cases file. Application to test.	-	Correctness
Harness	Java	XML file with test cases.	XML file.	Correctness
Jfunc	Java	Test cases file.	Console	Correctness
google Test	C++	Test cases file.	XML file.	Correctness
Phpunit	Php	Test cases file.	Console. XML reports.	Correctness
TestNG	Java	Test cases file. A build.xml file [for ant]	Console. Web report.	Correctness
Feed4TestNG	Java	File with test cases and annotations. Data sources (csv or excel files).	Console	Correctness
p-unit	Java	Test cases file. Instructions to start time and memory registration.	Console	Correctness. Memory consumption. Execution time.
ContiPerf 2	Java	Test cases file. Configurations tags.	Console. Csv file.	Correctness. Processes number. Execution time.
PMD	Java, JavaScript, XML, XSL, JSP.	Rules file.	Files: text, xml, html , nicehtml, or xslt.	Style. Complexity.
JavaNCSS	Java	-	Files: XML, XSLT, SVG	Design. Style. Complexity.
Checkstyle	Java	Rules file.	Console	Style
FindBugs	Java	XML file with filters	Files: XML, html, emacs, xdocs.	Style. Debugging.

Additionally during the review there was found an important tool to evaluate criteria, it is Sonar[10]. Its goal is to manage code quality. It supports evaluation of more than 20 programming languages including Java, C, C++, C#, Python, JavaScript, and so on. As criteria to evaluate the quality, it considers architecture and design, comments, coding rules, potential bugs, complexity, unit tests, and duplications. It could be useful for enterprise environments. For the goal of the project, it would be limited by the extensibility regarding programming languages, which depend on updates, and for the lack of support to assign weighting to criteria inside the grading process.

Finally it is worth standing out that some languages do not have tools to evaluate some criteria. It should be taken into account in the area of software testing for future work.

2.3 Technologies for orchestration

Supporting many ways of grading inside the proposed architecture, which will be seen deeply later, implies working with some software components arranged differently each time that a call is done. Every component inside the grading process is a program, which quantify a given metric and allow evaluating a criterion. Then, when a grading process is started, an element to control calls to every program is necessary. This new element is the orchestrator.

The orchestrator will work in every grading process, defining which programs to call, the order of calls, calling the programs, giving parameters to the programs, managing dependencies and so on. A review of which tools could work as orchestrator is helpful to avoid the building of a new one if possible. At least some features of a given tool could help to face issues if the building cannot be avoided. Thus, a brief description of some tools is presented:

- Apache Ant[11]. - It is a Java library that is used mostly to build Java applications. It can build applications implemented in other programming languages as well. In a general way, Ant can be used to support any process that is described as targets and tasks. The target is a set of tasks and the tasks are piece of code, which execute actions on input parameters. The process is described in a XML file called *build.xml*.
  Other advantages include: the possibility of be called as a console program (indispensable to be integrated to VPL), working in a high level (using Java objects), the availability of a set of built-in tasks, and the possibility of extend this set with own built tasks. To build a new task, it is necessary to write a Java class which extends from a given class and so it has to implement some

[10] http://www.sonarsource.org/
[11] http://ant.apache.org/

methods, and after that registering the new created task is necessary. The registration includes some information about the class name, the path to the class, the package name and the arguments in a XML format.

The project recommends to work with Ivy[12], which is a dependencies manager integrated with Ant and that has the same principles.

- Maven[13].- It presents two important goals: the first one is to provide of a tool which manages the building and the dependencies control in Java projects; the second one is allow the quick comprehension about the state of a project development. A Maven project is defined in a XML representation as well. This is known as POM (Project Object Model) and the representation's name is *pom.xml*.

  An important feature is the extensibility through the use of plugins written in Java or scripting languages. The building of new plugins is done based on Mojos, which are simplest Java programs for Maven. Any new Mojo has to extend from a base abstract Mojo and therefore implementing a method. After that, defining the new plugin is necessary. The definition includes information about the version, the identification, the package, the group, the name, and additionally about dependencies (group, id, and version). This information is written in a XML format.
- Gradle[14]. - Its official site says that it is an evolved building tool because it can automate the building, testing, publishing, deployment and more of software projects. It has been built on Ant and Maven. It supports on a DSL (Domain Specific Language) based on Groovy language and provides of declarative language elements. It can work with Java, Groovy, OSGi (Open Services Gateway Initiative), Web and Scala projects. It uses a script written in Groovy to control the process. The XML representation of Ant and Maven projects can be interpreted or converted to a Groovy representation.

  There are a set of already defined tasks, but writing new ones is possible. Additionally, Gradle can be called through the command-line.
- GNU make[15]. – It is a program that can manage processes to generate executable programs, or another kind of files, even programs' installation/uninstallation from a set of source files. Then, it is not a tool only to build applications and it is not limited to a specific programming language. The process is composed of a set of stages. All of them are defined in a file called *makefile.* Every stage defines a target, a set of dependencies and a set of shell commands to execute.

[12] http://ant.apache.org/ivy/index.html
[13] http://maven.apache.org/
[14] http://www.gradle.org/
[15] http://www.gnu.org/software/make/

There are not defined tasks, but new of them can be defined through using command-line calls. So it would be possible to call directly other programs through these calls.

In most cases the goal is to build, deploy or install an application. Although it is not the goal of this project, it is very useful knowing about the main features of this kind of tools, to determine if one of these features can be emulated or used. It can be seen a common fact among these tools and it is the use of a configuration file, which allows managing the process.

In the design chapter there will be deeper information about the decision of using one of these tools or building a new one.

2.4 Virtual Programming Lab

It is a quite mature tool developed by Las Palmas University under GNU/GPL license. Because of its features fulfill most of the requirements gathered (the requirement analysis will be shown in the next chapter) for the project it has been selected as the base tool. Therefore it is necessary to make a deeper review about this tool. The documentation, help and code can be obtained from the official site[16]. The information about the architecture can be obtained from the article written by Rodriguez et al. in (Rodríguez-del-Pino et al. 2012). The different releases have been tested since 2009/2010 academic course. It means that the tool has been tested by many activities and submissions. It has had a good acceptance by academic institutions. The cited article reports about 50 institutions around the world.

2.4.1 Main features

Its main features include:

- Support for many programming languages (a list was given in a former section).
- Application access through Moodle interface. The resultant grade of the grading process can be integrated with the Moodle grades module.
- High level of configurability, it includes physical resource use, actions allowed to students (debug, execute and evaluate their code), files required for evaluation (names and number of files), and so on.
- Customizable grading process, it provides of default scripts to grade every programming language but it can be changed by the teaching staff.
- Safe architecture, it has been separated in the Moodle server and the Jail server; the second one is a safe environment prepared to support effects of intentional or unintentional malicious code.

[16] http://vpl.dis.ulpgc.es/index.php

- Possibility to code in the browser, which avoids the necessity of a compiler installed.
- It provides of a powerful tool for plagiarism detection.
- Easy way to define test cases.
- Support of the tool's creators. It has been released as a Moodle plugin, so it is published in Moodle official site and has a forum to help and collaborate among users of it.
- Because of its license, it can be modified and used as anyone needs.

2.4.2 Architecture

The architecture considers the use of two servers the Moodle and the Jail server and the user machine, with a Java applet. In the most basic physical architecture, only one deployment server will be needed. Figure 1 shows the architecture where both of the servers are software programs, but to guarantee a safe environment every server should be installed in a different physical or virtual server.

The user machine is worth mentioning because the web browser should support a Java applet, which allows online coding. Actually, VPL allow coding, debugging and executing programs using the web browser.

The Moodle server is considered because VPL has been developed as a plugin for this LMS. The plugin has been built as a Moodle activity. This provides all the front-end of the application and allows managing the activity, configuring the parameters, to start the grading process and so on. The submodule for plagiarism detection is inside the plugin as well.

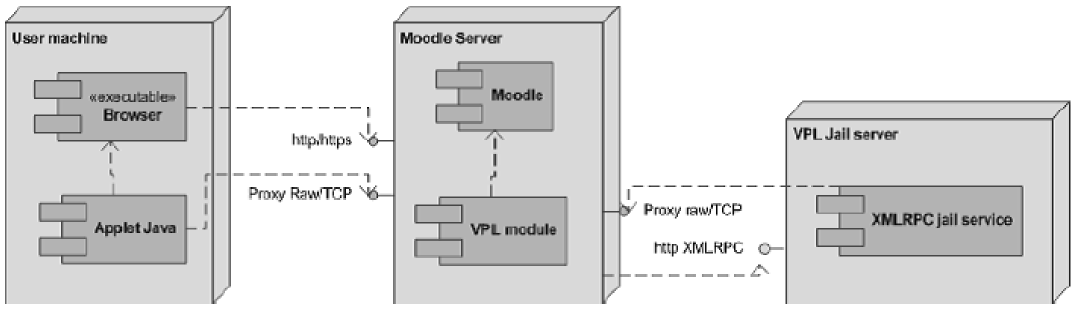


Figure 1.VPL's architecture[17]

The Jail server is quite important, this provides a service to receive programs source code, grade them through a given process, and send back feedback. Considering the grading process, it provides of a safe environment to compile, debug, execute, and test the programs. The environment is dynamically built and then it is destroyed after the grading process has finished.

[17] Retrieved from RODRÍGUEZ-DEL-PINO, J. C., RUBIO-ROYO, E. and HERNÁNDEZ-FIGUEROA, Z. J. *A virtual programming lab for moodle with automatic assessment and anti-plagiarism features*

About the communication means in the servers, the client machine and the Moodle server use HTTP (Hypertext Transfer Protocol) requests and responses. The Moodle and Jail servers use XML – RPC protocol over HTTP to communicate.

2.4.3 Technologies

Knowledge about technologies used to build a tool is very important when considering integration with other systems, or when the goal is to develop an extension, or even to provide an idea about how to face a requirement for building a new tool.

In this case, the used technologies played an important role to choose VPL. The used programming languages can influence directly in the implementation time. The information about extra software is necessary to set up a test environment. The knowledge about the communication protocol can be helpful to determine the right transmission of data and identify possible bugs.

Programming languages for building the tool

VPL has been developed using some programming languages as PHP, C++ and Linux shell scripting. For the front-end it uses web programming languages as HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript. In case of changes, working with the first three will be most probable.

C++ has been used to develop the Jail server. In case of modifications, making changes on the code, compile again, replace the binary server application and restart the service would be necessary.

Linux shell scripting is required to change the default grading process. Currently, there is a shell script program depending on each supported programming language. The grading process uses two shell script programs. They are called orderly by the Jail server. The first one is related to evaluation based on compilation and it generates automatically the second one. The last is related to the execution of test cases. The first script can be modified through the web interface.

PHP was used to develop the plugin. It was because Moodle is developed in this programming language. In case of changes, it would be necessary to use the APIs (Application Programming Interface) provided by Moodle[18].

More details about changes will be presented in the validation chapter.

[18] http://docs.moodle.org/dev/Main_Page

Xinetd (Extended Internet Services Daemon)

It is an open source program considered as a super server, which works on Unix-like systems. It can start other servers to provide a given service over Internet.

The official site[19] shows a list of important features and it provides the source code to download the current release. The most important features include:

- Access control, managing allowed and denied hosts, limiting the number of incoming connections (total and for a given service), binding a service to a specific IP (Internet Protocol) address, and so on. The connections can be TCP (Transport Control Protocol), UDP (User datagram Protocol), or RPC.
- Prevention of DoS (Denial of service) attacks, precisely by limiting the incoming and simultaneous connections, and limiting the number of servers of the same type at a given time.
- Extended logging abilities including configuration of logging level for every service, writing logs in different files, and so on.
- Offload services that use TCP, redirecting the streams to another host.
- IPV6 support.

All the mentioned features define the main advantage of use this program, which is improving the security and reducing the risk of DoS attacks (Raynal 2001).

Raynal in (Raynal 2001) provides of useful information to compile, install, and configure this program. A set of examples and their explanations are shown as well.

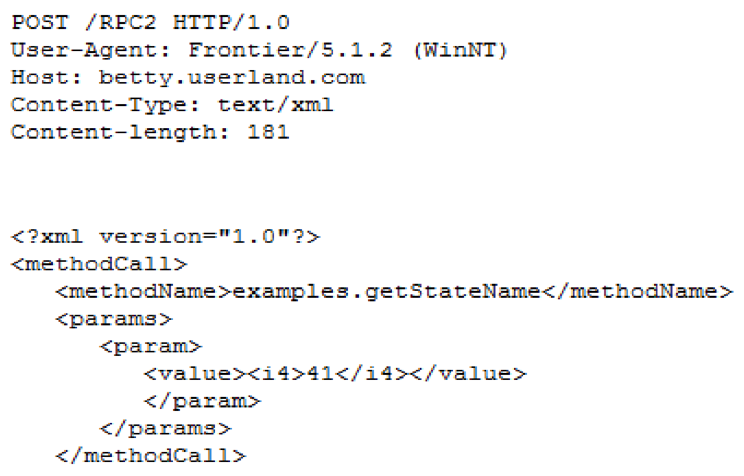
XML – RPC Protocol

The protocol defines a simple schema to implement RPC using XML for encoding the body of the request. The total message is sent on an HTTP-POST request.

The header of the XML-RPC request contains information about the HTTP request type, the HTTP version, the URI (Uniform Resource Identifier), the user agent, the host, the content type and content length. The payload is encoded in XML, which has the procedure's name to call and a set of parameters if needed. Figure 2 shows a basic request example.

The header of the XML–RPC response includes information about the HTTP version, a code when there was not an error, the connection state, the content length and content type, the current date – time, and server data (hostname and agent). The payload is encoded in XML and it contains only one parameter or a fault element. Figure 3 shows a basic response example.

[19] http://xinetd.org/

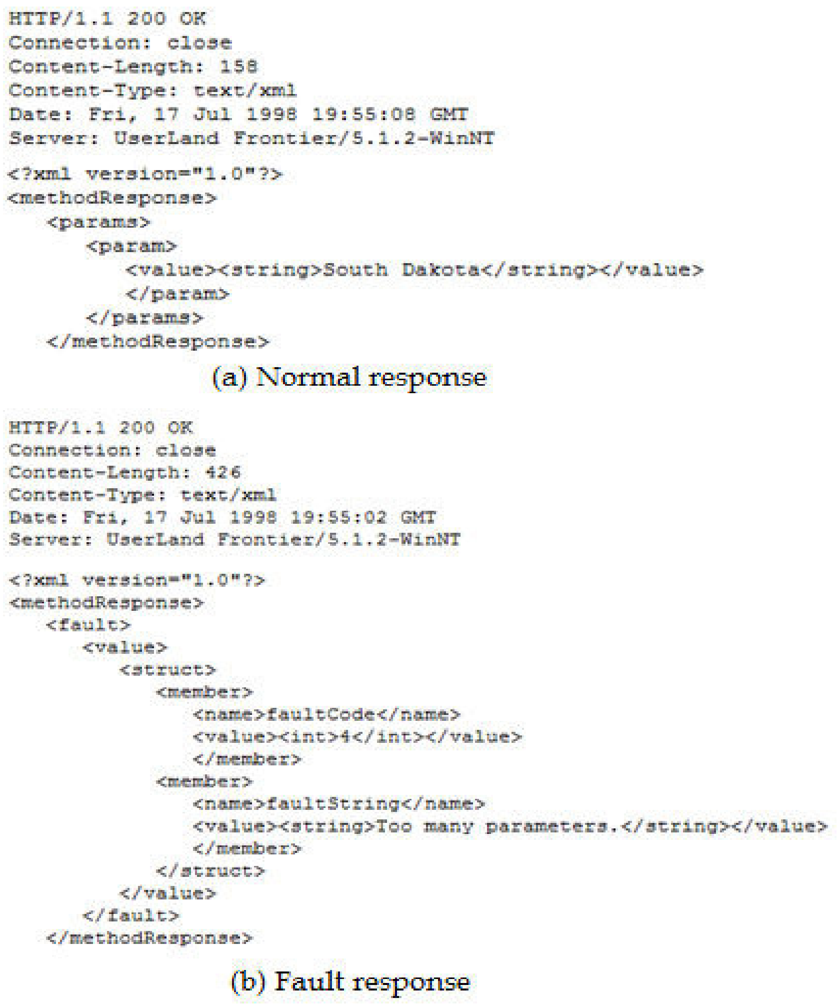


```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181


<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
         </param>
      </params>
   </methodCall>
```

Figure 2. XML-RPC request.

When talking about a parameter or a fault element, both of them have a value. This value can be scalar (int, boolean, string, double, dateTime.iso8601, base64), structure or an array. In the last two types, they can have inside more values willing depending on each case. So, it is possible to send some fields of information although included in one parameter.



```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

(a) Normal response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>4</int></value>
               </member>
            <member>
               <name>faultString</name>
               <value><string>Too many parameters.</string></value>
               </member>
            </struct>
         </value>
      </fault>
   </methodResponse>
```

(b) Fault response

Figure 3. XML-RPC response.

For further information it is highly recommendable to visit the site (Winer 2003), this contains the specification.

2.4.4 The grading process

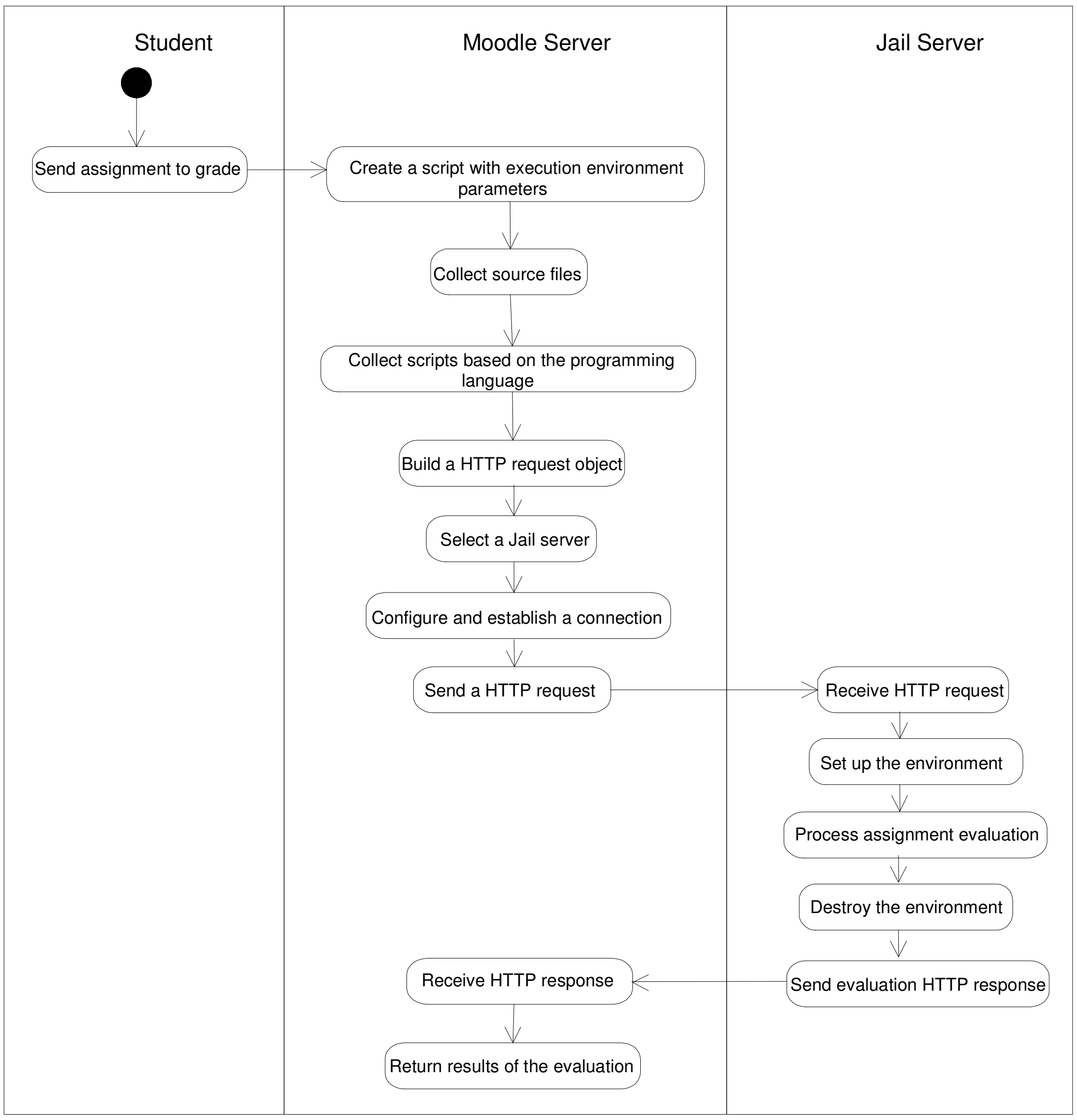
When an activity is created, it could work as a basic process where the most of parameters are set by default. In this case the teaching staff creates the activity and the students can send the source files depending on established requirements. The student could use the code editor to write the programs as well. This scenario does not consider any metric to grade the assignment. So an automatic grading is not possible.

To implement an automatic grading it is necessary to add some test cases. These should be written in a specific syntax that is not complicated. Basically, defining the inputs and outputs for the program is necessary. So the only criterion to grade is the correctness through a metric given by the number of success test cases. It is possible to define more complex grading process, which could consider more metrics and criteria to establish a grade. In this case it is necessary to modify the default script for the activity's evaluation. Additionally and if it is needed, some files should be uploaded. For example files which has test cases or rules files required for a given tool.

In a general way there is a set of files required for an activity, and they are collected by the Moodle server and sent to the jail server. This server builds an environment with parameters received in one of the files. The grading process is carried out and finally the environment is destroyed. The stages can be seen in Figure 4.

The stages since the Moodle server perspective:

- Receiving a signal that indicates the user has started the process.
- Collect of information about parameters for the grading environment. These include maximum limits for execution time, files' size, required memory, and for number of threads to create.
- Creation of a shell script with all data collected previously.
- Collect of source files sent by the student.
- Identification and collection of shell scripts to compile, debug, and execute the program files depending on the programming language.
- Integration of all data collected into packages previous to send them.
- Read of parameters to establish a connection with the jail server. These include maximum time to keep a connection, the servers' list and so on.
- Selection of a jail server and the connection's set up.
- Data stream transmission.
- The server waits for a response and closes the connection after this event.
- It shows the results to the user.



Student
Moodle Server
Jail Server
Send assignment to grade
Create a script with execution environment parameters
Collect source files
Collect scripts based on the programming language
Build a HTTP request object
Select a Jail server
Configure and establish a connection
Send a HTTP request
Receive HTTP request
Set up the environment
Process assignment evaluation
Destroy the environment
Send evaluation HTTP response
Receive HTTP response
Return results of the evaluation

Figure 4. Automatic grading process

As it was said before, the service is deployed in the Jail server using *xinetd*. When the request arrives, a program takes this and executes a set of actions:

- Receive the request and transferred data.
- Create a user and a home directory. Both of them are temporal.
- Set environment's parameters.
- Copy the data inside the directory.
- Set the user as owner of the directory.
- Use the program *chroot* to build the jail.
- Run the grading process and save the results (grade and feedback).
- Send the result to the Moodle Server.
- Clean the temporal environment.

2.4.5 VPL's suitability

There are a lot of requirements gathered from stakeholders' necessities and from the review of previous similar experiences. The whole set of requirements and how much VPL suits them will be explained in the next chapter.

The reasons that made this tool stood out from the others include: the GNU/GPL license, so it is possible to use and modify this regarding the own necessities; the easiness to access the documentation, help and to download the source code; the feature of working as Moodle plugin, it suits a very important non functional requirement for the final goal; its module for plagiarism detection; its security features regarding authentication and working with a safe test environment; its ability to allow defining assessment scripts, it gives the possibility to consider more metrics and criteria to grade; and it can support automatic and semi-automatic processes.

Despite its suitability with many requirements, VPL don't fulfill all of them. The main weaknesses include: the lack of flexibility, and modularity to define a grading process; not providing extensibility for supported programming languages; and the lack of more advanced statistical reports.

Taking into account the previous explanations, VPL has been selected as base tool to support the validation of the proposed architecture but it will be necessary the implementation of a new modules inside it as well.

2.5 Learning management systems

They are software that helps teachers to manage the process of learning. This managing can be applied on users, contents, activities, and so on. They play an important role in e-learning process. Although this kind of tools could be considered as a main mean to teach, usually they are used as supplement to classroom education (Unesco 2008).

There is a big set of tools for learning management. Many of them have been built with commercial goals, but there are several free and open source LMSs as well. Precisely, it is necessary to decide about which one to choose. Some criteria can be used to differentiate and select among all LMSs, for instance user interface, licensing and pricing, services for course building and training, and integration with other on-campus systems such as e-mail and registration, and so on (Unesco 2008).

Although the user interface will be very different among LMSs, the common thing is the use of Web 2.0 resources to make rich and interactive interfaces, and to implement some resources to improve the collaboration, wikis and forums for instance. One important goal aimed by LMSs is trying to get a high level of compatibility, which is reached with standards accomplishments. For example the implementation of SCORM

(Sharable Content Object Reference Model), this is a set of standards and specifications to guarantee interoperability and reusability in content and systems.

They can be sorted considering different scopes. These could be built-in technologies, accomplished standards, and so on. Considering the classification proposed in (Eckstein 2010), some tools are described. They have been sorted as free and open source, and paid-for.

2.5.1 Paid-for

Some of these tools are oriented to a deployment in enterprise environments. However, the most of them can be deployed in much kind of organizations (educational and governmental for instance). There are many of them and they have a big set of features, these can be looked in the official web sites. The goal here is just to mention a few of them to have a general knowledge about their existence.

Blackboard[20] offers mobile support. JoomlaLMS[21] provides a multilingual environment, fills requirements for WAI (Web Accessibility Initiative), gives integration with social learning tools through JomSocial, and supports e-commerce through different subscriptions modes. SabaLearning Suite[22] is full oriented to a business and industrial scope; it provides collaborative learning, social, contents' integration and mobile availability to accelerate the teaching-learning process. SharepointLMS[23] is based on Microsoft Office SharePoint.It offers a multilingual interface, uses Microsoft's tools for multimedia communications and guarantees a high level of security and scalability.

2.5.2 Free and open source

It is a good alternative to take advantage of this kind of tools in the learning process. There are a quite good number of tools in this category, in (Aydin et al. 2010) it is mentioned the existence of fifty FOSS (Free and Open Source Systems) LMSs. The references of many of them can be obtained in the site for free and open source software for e-learning of the Unesco[24] (United Nations Educational, Scientific and Cultural Organization).

Every tool has its own set of offered features, which will depend on the developer's community that supports the tool. In a general way, most of these tools try to offer multilingual interfaces and do a good effort to get certifications for accessibility and interoperability. It is important to consider that although these tools are open source

[20] http://www.blackboard.com/
[21] http://www.joomlalms.com
[22] http://www.saba.com/learning-management-solution/
[23] http://www.sharepointlms.com/
[24] http://www.unesco.org/iiep/virtualuniversity/forumsfiche.php?queryforumspages_id=9

and the most of them express clearly a GNU/GPL license, there are some of them that offer limited version as open source code and the complete version is offered as paid-for.

ATutor[25] has as main features: support for many compatibility (W3C WCAG, ISO/IEC 24751, among others), and interoperability (OpenSocial, SCORM 1.2, among others) standards. It has been released under GNU/GPL license. It integrates features of social networks. It supports the Oauth (Open Authentication Protocol) as well. As a key feature, this tool has a broad documentation to help developers.

.LRN (Learn Research Network)[26], based in the information of its official page, it is the most adopted tool for e-learning and digital communities. It is used for a half million users around the world. The users include higher education, governments, non-profit organizations, and so on. It supports accessibility standards. About used technologies, it makes use of RADIUS (Remote Authentication Dial In User Service), Kerberos, and so on for authentication; it uses RSS (Rich Site Summary) and web services as well. As an additional feature, it has modules for e-commerce.

Sakai[27] aims to accomplish with W3C (World Wide Web Consortium) accessibility requirements for contents. It offers two environments with free access. The first one is an environment for learning and collaboration, and its source code can be downloaded. The second one is an open web environment with a vision of academic collaboration.

Ilias[28] is certified SCORM 2004 and has a GNU/GPL license. It has been certified by NATO (North Atlantic Treaty Organization) as a safe LMS. For authentication it uses RADIUS and mechanisms based on SOAP (Simple Object Access Protocol). The SOAP interfaces can be used to communicate with external applications. It offers flexibility and versatility.

There are more tools as Claroline[29], OpenElms[30], eFront[31], their information can be retrieved from their official web sites.

Finally, in (Aydin et al. 2010) a comparison among more representative LMSs in this category is carried out. The features to define the most suitable tool took in consideration features like the support for many languages, the modular and flexible

[25] http://atutor.ca/
[26] http://dotlrn.org/
[27] http://www.sakaiproject.org/
[28]http://www.ilias.de/
[29] http://www.claroline.net/
[30] http://www.openelms.org/
[31] http://www.efrontlearning.net/

design, and the supported ways for authentication. It concluded that Moodle has advantage in the most of features respect other tools.

2.5.3 Moodle

It is a LMS under GNU/GPL license. Moodle's begin dates from the Dougiamas' PhD thesis (Dougiamas et al. 2003). Nowadays, he is still leading the project and there is a big community of developers supporting this tool.

Moodle is broadly used in learning environments for academics and government institutions, military and health organisms, among others. According to statistics in the official web page, there are more than 75000 registered sites. These sites belong to 215 different countries. Spain is in the second place with 6504 registered sites[32].

Moodle is offered as a tool that can supply of flexible and adaptable e-learning environments. It can be used since just to provide contents until setting up a complete environment customized for collaboration and work among many users with different roles supported by many resources.

The environments can be customized through the creation of contents, activities, tasks, courses, users, roles, and so on. More resources include: forums, wikis, communication resources as chat, and so on. The resources can be extended by the use of additional plugins or by the integration with external tools. The integration can be done because Moodle supports standards for share information (supports SCORM 1.2 for instance) and protocols to communicate with other systems.

Other important advantages include:

- Authentication means supported. These could include authentication with e-mail confirmation, LDAP (Lightweight Directory Access Protocol), registration with e-mail or news servers, and so on.
- The deployment is relatively easy; it only needs of a web server with PHP (HyPertext Preprocessor) support and a SQL data base.
- There is a lot of documentation available. In the official page there is documentation to start with develop, to collaborate solving issues, and references to helpful information.
- This tool is supported by a big quantity of developers around the world, which implies a big number of plugins, and its correspondent actualizations and support.

[32] Information retrieved on February 3, 2012 from http://moodle.org

- Additionally, there are many research projects that include, plugins' development, studies to improve the teaching-learning process, and even on data mining.

2.6 Chapter summary

This chapter has shown information to understand the context of the problem and useful technologies, which will help to solve it. The covered topics include a systematic review of tools for automatic grading of programming assignments, a characterization of criteria to evaluate programming assignments, a quick sight of technologies used to evaluate these criteria, a deep description about VPL as a base tool for the architecture's validation, and a quickly review of LMSs highlighting Moodle.

The systematic literature review shows that this research field is a hot topic. The necessity of tools for automatic grading of programming assignments has been alive for almost fifty years. While new tools were proposed, new gaps were reported as well. The main gaps reported include plagiarism detection, support for grading of GUI programs, the lack of a model to grade, support for web programming languages, meta-testing, security for the host system, LMS integration, the lack of a broadly acceptance for a given tool, and so on. Nowadays, many of them have been considered but there are some of them still waiting for a solution. One of the most important is the lack of a model to grade. The solution maybe is not providing a common model for every case. Probably, the solution is to see in a higher perspective and provide an architecture, which can support many ways to grade.

The lack of a model to grade and the diversity of grading ways applied are intrinsically related. This diversity means considering different grading criteria, and different number of them inside a grading process. The systematic review showed many criteria and associated metrics considered to grade, but there was not a characterization of them. So, a criteria's characterization has been proposed as a first stage to go on with a solution to this issue.

Almost every identified criterion has a tool that can evaluate it. A summary table of this kind of tools has been presented. It allows knowing which criterion or programming language has not been yet supported. The knowledge about input parameters and return values is useful to define a wrapper, which will be used inside the proposed architecture.

To validate the proposed architecture it is necessary to use a tool to grade programming assignments automatically. Using a tool already built would be helpful to save implementation time. VPL has been selected as this tool. Then, its main features and architecture have been shown. The grading process inside VPL has been deeply described. Finally some relevant technologies used within VPL are explained (programming languages, XML-RPC, and Xinetd) as well.

A quickly review of paid-for and open source LMSs has been shown to have awareness of them. But mainly a description of Moodle is done because it is related to a non functional requirement for the final solution.

This chapter is very important inside this work. Its main contributions include: the identification of the specific problem treated in this work, which is the **lack of a model to grade programming assignments**, and **the criteria's characterization** as element to help to think in the architecture to solve the problem.

Some of the contributions described in this chapter have been validated through the publication of a scientific article. This article has as title: **Programming Assignments Automatic Grading: Review of Tools and Implementations** (Caiza et al. 2013). It has been accepted in the **7th International Technology, Education and Development Conference**. The complete document can be seen as an annex.

This chapter helps to get into the context of the problem, as well as to define how much of this project's requirements have been already fulfilled by previous works. Additionally, it helps to define new requirements, which will be analyzed in the next chapter. Finally it gives the introduction of some technologies, which will be used in the next chapters of analysis and architecture design.

3 Problem analysis

The previous chapter provided a context for the problems described in the introduction. They are:

- A general problem, this is the necessity of using a tool to grade automatically programming assignments. It is associated to a final goal, which is to implement a tool for automatic grading of programming assignments. The project SEAPP at UPM aims at achieving this goal.
- A specific problem, this is the lack of a common model to grade programming assignments. It is associated to the general goal of this work (expressed in the first chapter). The goal is to propose and validate a new architecture inside a grading process (which aims to an automatic grading of programming assignments) , which will support many ways of grading.

Considering both problems and following a waterfall software development process, this chapter makes an analysis of these problems to give an approach to fulfill with all the requirements set, and to define a scope and propose a solution.

As a first step, the main actors in the system will be defined. Next, a use case diagram is shown to have a perspective of the system's context. Then, there will be given a complete set of requirements including functional and non functional. Considering the requirements and the existent tools, the most suitable tool will be chosen to be used as a base for further developments (it could be helpful to save implementation time). All these elements will be useful in finding out the most adequate solution to reach the final goal, which will solve the general problem. Finally, the scope to solve the specific problem will be defined. This definition implies selecting a set of requirements to be satisfied.

3.1 Actors

Three actors have been identified: students, teaching staff and administrators. Every actor has its own interests and specific actions inside the system.

The student (S) is interested in sending his programming assignment solution. He is interested in getting back its correspondent grade and feedback as quickly as possible. He gets some advantages of using the tool including the improvement of his skills through receiving a good and quick feedback, the possibility of getting better grades in resubmissions, and the possibility of counting with a better tracing about his improvement.

The teaching staff (TS) includes professors and teaching assistants. They are interested in this kind of tool to provide students with quick feedback (recommendations, comments, errors' explanations and so on). Additionally they can optimize their time to use it in more focused tasks inside the programming learning. Their main tasks include management of assignments and configuration of the grading process. Their responsibilities also include the analysis of reports to see the improvement level of students.

The administrator (A) is in charge of the system's settings management to keep it working. These parameters can include the management of programs and tools required by the grading process.

3.2 Defining the context

A use case diagram allows a high level representation of the system (Cockburn 2001). It permits to think about the context and to define the limits of the system.

To get a better understanding of these diagrams, making a previous explanation about the meaning of some terms used inside them is necessary:

- **Assignment** refers to a programming problem assigned to students. It can include homework, lab exercises and so on.
- **Submission** refers to a set of files, which are the solution for the given problem. An assignment can be associated with more than one submission.
- **Metric** is any type of measure done on a software piece. A metric is quantifiable. The rate of test cases passed by a submission, and the number of tags and comments found on a source code are examples of metrics.
- **Grading-submodule** is an artifact used in this project, which allows evaluating a submission considering one or more metrics for a given programming language. So every grading-submodule has always at least one metric associated. As every criterion can include a set of metrics (refer to the metrics' characterization in the state of the art chapter), every grading-submodule can be related to a given criteria as well. Every grading-submodule has implicitly associated a main action, which will be performed on the source code to evaluate. Then, the grading-submodule will have an associated program to perform this main action. Additionally, every grading-submodule can require a list of parameters to be used by the associated program.

Figure 5 shows a use case diagram, which represents the main operations that actors will require to the system. This representation allows having an idea of which modules the system could have. To maintain traceability, the use cases have a unique code. The prefix of every code is UC, which means Use Case. After the prefix there is a sequenced number.

To get a better understanding of use cases expressed in Figure 5. It is necessary to make an explanation. Thus:

- Manage assignments (UC01). – The teaching staff shall perform CRUD (Create – Read – Update - Delete) actions on assignments.

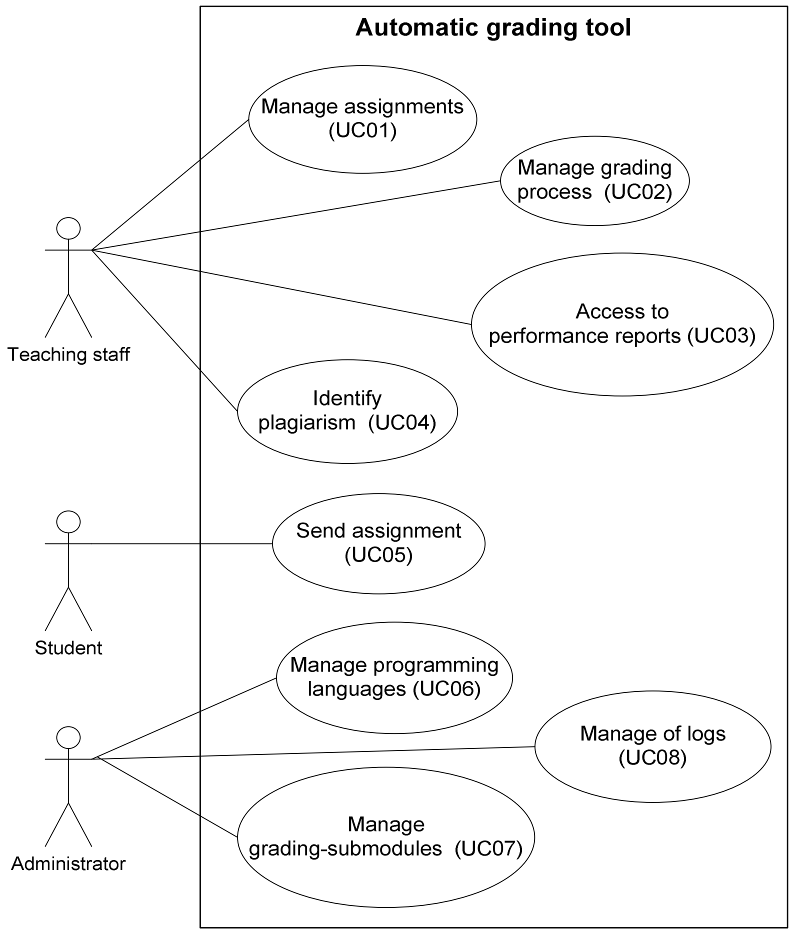

Automatic grading tool
Manage assignments (UC01)
Manage grading process (UC02)
Access to performance reports (UC03)
Identify plagiarism (UC04)
Send assignment (UC05)
Manage programming languages (UC06)
Manage of logs (UC08)
Manage grading-submodules (UC07)
Teaching staff
Student
Administrator

Figure 5. High-level use case diagram

- Manage grading process (UC02). – The teaching staff shall manage the grading process through tasks of configuration and starting the grading process. The teaching staff shall configure the grading process associated to an assignment. The configuration includes choosing different grading-submodules and sorting them in any order, and setting values for parameters associated to every grading-submodule. Additionally, the teaching staff could start the grading process.
- Access to performance reports (UC03). – The teaching staff shall access to reports about performance and improvement of the students.

- Identify plagiarism (UC04). – The teaching staff shall identify plagiarism cases. It will be done comparing current students' assignments and assignments from past courses.
- Send assignment (UC05). – The student will send the assignment's solution and will receive feedback and a grade for that submission. In the background, the grading process will be performed.
- Manage programming languages (UC06). – The administrator shall perform CRUD actions on supported programming languages. These actions are focused to maintain general information about a given language (id, name, description, version, and so on). This functionality will be used to tag or to associate a programming language with a given grading-submodule. Then this tag can help to sort and filter grading-submodules when configuring a process or when getting reports.
- Manage grading-submodules (UC07). – The administrator shall perform CRUD actions on grading-submodules. These actions have associated CRUD actions on required parameters for every grading-submodule as well.
- Manage logs (UC08). – The administrator will manage logs about the system and especially about the grading process. This management includes configuration about parameters to keep logs working, and reports in different levels of detail.

The previous diagram is helpful because it allows thinking about possible modules inside the system; but it is necessary to detail every use case. It is going to be done through a set of requirements in the next section.

3.3 Requirements

It is necessary to go from a higher level, which is the use case diagram shown before, to a lower level of an ample set of requirements. This will permit having a feature-centered perspective of the system. Then, it will be possible to know which of the tools considered in the systematic literature review fulfill the requirements. Next a set of functional and non functional requirements is presented.

3.3.1 Functional requirements

Functional requirements express what the system has to do and they are related with the use case diagram showed before. Every requirement is traceable to a use case. To provide of a unique identification they have their own code. Every code has a prefix (FR - Functional requirement) and a sequenced number. Additionally the description of the requirement is given. Table 5 shows functional requirements associated to every actor. All these requirements have been specified taking into account necessities and future paths reported in previous related works (Refer to the systematic review).

Table 5. Functional requirements

Cod. UC	Cod. Req.	Requirement	Actor
UC01	FR01	The system shall provide a list of the existent assignments.	TS
UC01	FR02	The system shall provide the complete information of an assignment.	TS
UC01	FR03	The system shall allow creating a new assignment.	TS
UC01	FR04	The system shall allow modifying assignment's attributes but for those which are part of the unique identification.	TS
UC01	FR05	The system shall allow deleting an assignment.	TS
UC02	FR06	The system shall allow the teaching staff to add new grading-submodules inside the grading process.	TS
UC02	FR07	The system shall allow the teaching staff to set parameter's values for every grading-submodule inside the grading process.	TS
UC02	FR08	The system shall be able to let the teaching staff sort the grading-submodules in any order inside the grading-process.	TS
UC02	FR09	The system shall be able to let the teaching staff to start manually the grading-process on at least one submission.	TS
UC02	FR10	The system shall be able to let the teaching staff grade manually a submission.	TS
UC02	FR11	The system shall allow the teaching staff to edit manually the feedback and grade.	TS
UC03	FR12	The system shall provide a submissions' historical report.	TS
UC03	FR13	The system shall provide all the information associated to a submission done previously.	TS
UC03	FR14	The system shall provide statistical reports per assignment(s) and per student(s).	TS
UC04	FR15	The system shall be able to let the teaching staff start the plagiarism detection process.	TS
UC04	FR16	The system shall provide reports about plagiarism cases detected.	TS
UC05	FR17	The system shall provide a list of available assignments for students.	S
UC05	FR18	The system shall provide a complete description of the student's assignment.	S
UC05	FR19	The system shall be able to let the student upload and send to grading all files data required by the assignment.	S
UC05	FR20	The system shall provide the grade and feedback to the student.	S
UC06	FR21	The system shall provide a list of supported programming languages.	A
UC06	FR22	The system shall provide the complete information about a supported programming language.	A
UC06	FR23	The system shall allow adding a new supported programming language.	A
UC06	FR24	The system shall allow modifying attributes of the supported programming language but for those which are part of the unique identification.	A
UC06	FR25	The system shall allow deleting a supported programming language.	A
UC07	FR26	The system shall provide a list of existent grading-submodules.	A
UC07	FR27	The system shall provide the complete information about a grading-submodule.	A
UC07	FR28	The system shall allow creating a new grading-submodule.	A
UC07	FR29	The system shall allow modifying attributes of a grading-submodule but for those which are part of the unique identification.	A
UC07	FR30	The system shall allow deleting a grading-submodule.	A
UC07	FR31	The system shall be able to let the administrator add new parameters to a grading-submodule.	A
UC07	FR32	The system shall be able to let the administrator modify parameters of a grading-submodule.	A
UC08	FR33	The system shall be able to let the administrator set the grading process logs' severity required.	A
UC08	FR34	The system shall provide a report of grading process logs sorted by severity.	A

3.3.2 Non functional requirements

There is a set of non functional requirements for this project and a use case diagram cannot show them. So it is necessary to define a way to get this kind of requirements.

Considering that ETSIT (Escuela Técnica Superior de Ingenieros de Telecomunicación) has in charge the SEAPP project and that this centre would be the test environment, it is possible to define a set of non functional requirements. These requirements will be defined considering the current technological infrastructure and trying to have minimum effects on that.

Table 6 shows a set of non functional requirements. They will help to choose a tool from the set given in the systematic review. The selected tool will be the base to develop the solution to validate the new architecture.

Table 6. Non functional requirements

Req. Code	Description
NFR01	The system shall be able to integrate with Moodle. Moodle is the current LMS used in ETSIT.
NFR02	The system shall be able to grade Java programs. It because Java is the programming language taught at ETSIT.
NFR03	The system shall be able to grade source code written in other programming languages. For example JavaScript.
NFR04	The system shall allow adding any metric inside the grading process.
NFR05	The system shall allow sorting grading metrics in any arrangement.
NFR06	The system shall guarantee the normal working of the current infrastructure.
NFR07	The system shall be able to provide an isolated environment to evaluate the source code.

3.4 Solution approach

Considering the whole set of requirements expressed before, considering a solution approach is possible; this approach allows fulfilling all of them.

Functional requirements can help to think about the necessary modules inside the system. There are 9 modules in total and most of them are focused on management. All of them are important but the automatic grading module is highlighted. Figure 6 shows all modules defined. They are:

- Supported programming languages management. - This module will allow performing CRUD actions on any supported programming language. As it has been said before, a supported programming language will be used to tag a given grading-submodule. Certainly, this tag allows associating the grading-submodule to a programming language.
- Grading-submodules management. - This module will allow performing CRUD actions on any grading-submodule. Every grading-submodule will be related to at least one grading metric and to a supported programming language (taking into account the previous module). This module will

provide of modularity and extensibility regards to grading-submodules. Most modularity will be reached when the relationship grading-submodule – metrics is one-to-one. A relationship grading-submodule – criterion is useful as well.

- Assignments management. - This module will allow performing CRUD actions on any assignment.
- Grading process management. This module will allow configuring the grading process for a given assignment. The grading process will have at least one grading-submodule and there won't be a maximum defined. It will allow choosing any grading-submodule to be included inside the grading process. The chosen grading-submodules will be arranged in any order. Additionally, this module will allow setting parameters needed by grading-submodules.
- Assignments submission. This module allows uploading and sending any file required by the assignment. The set of files are named as a submission. Additionally this module triggers the grading process.
- Automatic grading module. This module is the most important considering the goal of the system. It goes through a set of stages to evaluate the submission done and returns a grade and feedback.
- Plagiarism detection. The goal of this module is precisely to detect any copy among students. There is a good research about this field and there are some already built tools.

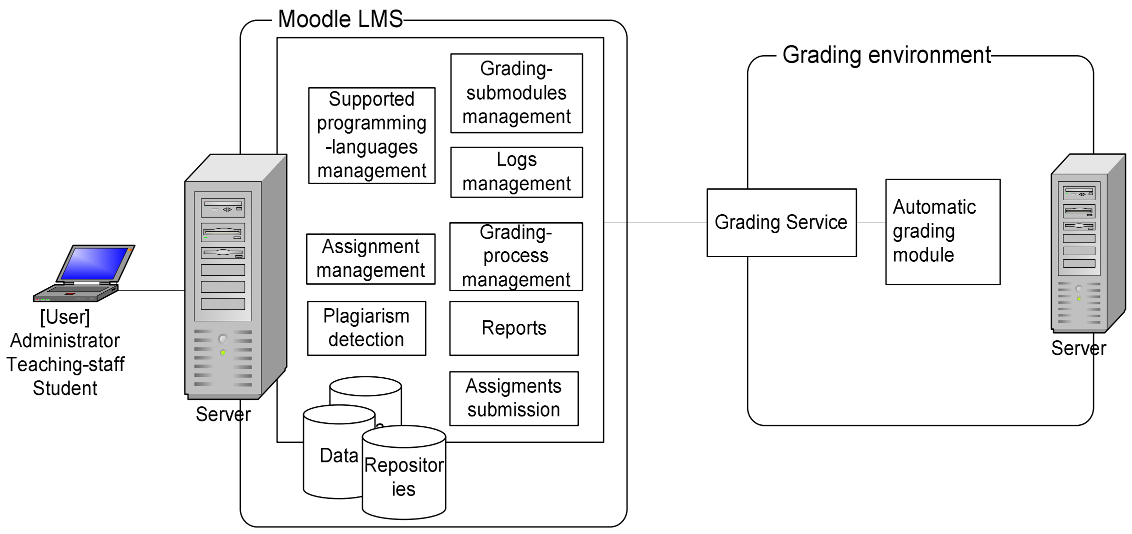


Figure 6. Solution approach

- Reports. This module includes many kinds of reports. The goal here is to do a trace on the students' skills improvement. Additionally reports can help to study the behavior of students while working in programming assignments.

- Logs management. This module allows tracing events when an incident happens inside the grading process.

The separation of the automatic grading module from the others is not only due to its importance. It is considering the non functional requirements as well. The non functional requirements NFR01, NFR06 and NFR07 are very important. Regarding the first one, ETSIT is current working with Moodle LMS and its interest is to avoid the use of many gates to enter the programming learning environment. It implies guaranteeing integration between Moodle and the automatic grading tool. Regarding the other two cases, ETSIT has a current infrastructure working and it has to be maintained in the same way. Then, to prevent and maintain this infrastructure safe, it is necessary to isolate the grading process module. Building the new module in another server is a good option. The use of services can maintain the integration with Moodle and even to integrate with other modules or systems.

The other non functional requirements have been taken into account as well. They are related with some modules defined. Thus, NFR02 and NFR03 related with the module for supported programming languages management and with grading-submodules management (because they are associated to a given programming language). NFR04 and NFR05 are considered by modules grading-submodule management and grading-process management respectively.

3.5 Existent tools' suitability

The systematic literature review carried out in the previous chapter allowed identifying tools that could fulfill with the specified requirements. Thus, in the best case a tool would fulfill with all the requirements and only a stage of deployment would be required. Most often, there would be a tool which fulfills with some, many or most of the requirements. In any case it implies saving implementation time.

It is desirable that a tool fulfills as many requirements as possible. So the mature tools, detailed in the systematic review at Table 1 have been taken because they have more features than the younger tools. To take an already built tool and work on it, its license is important. So it will be the first filter. It is necessary a license which allows accessing the complete code to make changes on this. Implicitly, it is necessary to know if the access to download to the code is possible as well.

Additionally to license and availability, non functional requirements have been considered as comparison parameters. The non functional requirement NFR04 related to grading-submodules extensibility has been adapted to extensibility of grading metrics, which means the support for new metrics for grading. This is necessary to have a fair comparison because grading-submodule is a concept given just in this work. Other parameter to make the comparison refers to modular grading metrics. Due to inexistence of grading-submodule, it is necessary to know if there is a way to provide modularity regards grading metrics (plugin or a given artifact related to a metric could be a possibility). Finally a plagiarism control is a quite important feature, this could be itself a module, and therefore it has been considered as a comparison parameter as well.

Table 7 shows that there is not an already built tool that fulfills with the whole set of requirements. But surely taking one of them to use as base and save implementation time is possible.

Table 7. Automatic grading tools suitability

	CouseMarker	**Marmoset**	**WebCat**	**VPL**	**Tool by Magdeburg University**
License	-	Apache 2.0	GNU Affero	GNU GPL	-
Availability	X	✔[33]	✔[34]	✔[35]	-
Moodle Integration	X	X	X	✔	✔
Support for Java programs	✔	✔	✔	✔	✔
Extensibility for programming languages	X	✔	✔	X	✔
Modular grading metrics	✔	-	✔	X	-
Extensibility for grading metrics	X	X	✔	✔	X
Flexibility in the grading process	-	X	-	–	✔
Safe evaluation environment	X	✔	X	✔	✔
Plagiarism detection	✔	–	-	✔	–

Considering most important parameters, explained previously in this topic, which are: license, availability, and Moodle integration; the most suitable tool is VPL. Additionally VPL fulfills with many of the established parameters and has important features as it can be seen in the state of the art chapter. After the selection of the tool, knowing how much the tool fulfills the whole set of requirements is appropriate. Table 8 allows determining that. It is necessary to define the scope of this work.

[33] https://code.google.com/p/marmoset/source/checkout
[34] http://sourceforge.net/projects/web-cat/files/
[35] http://vpl.dis.ulpgc.es/index.php/es/descargas

Table 8. VPL's fulfillment of all requirements

Cod. Req.	Requirement	VPL
FR01	The system shall provide a list of the existent assignments.	✔
FR02	The system shall provide the complete information of an assignment.	✔
FR03	The system shall allow creating a new assignment.	✔
FR04	The system shall allow modifying assignment's attributes but for those which are part of the unique identification.	✔
FR05	The system shall allow deleting an assignment.	✔
FR06	The system shall allow the teaching staff to add new grading-submodules inside the grading process.	X
FR07	The system shall allow the teaching staff to set parameter's values for every grading-submodule inside the grading process.	X
FR08	The system shall be able to let the teaching staff sort the grading-submodules in any order inside the grading-process.	X
FR09	The system shall be able to let the teaching staff to start manually the grading-process on at least one submission.	✔
FR10	The system shall be able to let the teaching staff grade manually a submission.	✔
FR11	The system shall allow the teaching staff to edit manually the feedback and grade.	✔
FR12	The system shall provide a submissions' historical report.	✔
FR13	The system shall provide all the information associated to a submission done previously.	✔
FR14	The system shall provide statistical reports per assignment(s) and per student(s).	X
FR15	The system shall be able to let the teaching staff start the plagiarism detection process.	✔
FR16	The system shall provide reports about plagiarism cases detected.	✔
FR17	The system shall provide a list of available assignments for students.	✔
FR18	The system shall provide a complete description of the student's assignment.	✔
FR19	The system shall be able to let the student upload and send to grading all files data required by the assignment.	✔
FR20	The system shall provide the grade and feedback to the student.	✔
FR21	The system shall provide a list of supported programming languages.	X
FR22	The system shall provide the complete information about a supp. programming language.	X
FR23	The system shall allow adding a new supported programming language.	X
FR24	The system shall allow modifying attributes of the supported programming language but for those which are part of the unique identification.	X
FR25	The system shall allow deleting a supported programming language.	X
FR26	The system shall provide a list of existent grading-submodules.	X
FR27	The system shall provide the complete information about a grading-submodule.	X
FR28	The system shall allow creating a new grading-submodule.	X
FR29	The system shall allow modifying attributes of a grading-submodule.	X
FR30	The system shall allow deleting a grading-submodule.	X
FR31	The system shall be able to let the administrator add new parameters to a grading-submodule.	X
FR32	The system shall be able to let the administrator modify parameters of a grading-submodule.	X
FR33	The system shall be able to let the administrator set the grading process logs' severity.	X
FR34	The system shall provide a report of grading process logs sorted by severity.	X
NFR01	The system shall be able to integrate with Moodle.	✔
NFR02	The system shall be able to grade Java programs	✔
NFR03	The system shall be able to grade source code written in any programming language.	X
NFR04	The system shall allow adding any metric inside the grading process.	✔
NFR05	The system shall allow sorting grading metrics in any arrangement.	✔
NFR06	The system shall guarantee the normal working of the current infrastructure.	✔
NFR07	The system shall be able to provide an isolated environment to evaluate the source code.	✔

Table 8 shows that VPL does not fulfill with all the requirements (functional and non functional). The requirements not fulfilled refer to:

- Management of grading-submodules (FR26 – FR32). – VPL does not work with artifacts similar to grading-submodules. It means VPL lacks an artifact that is directly related to a grading metric.
- Management of grading process (FR06, FR07, FR08). – VPL allows the teaching staff to write a script to evaluate a submission in a customized way. This script could consider any metric or a set of metrics. But VPL does not work with artifacts to support a modular and flexible grading process. The use of grading-submodules would require of teaching staff just to select and configure the required artifacts inside a grading process without any coding requirement.
- Full statistical reports (FR14). – Currently VPL supports informs about number of submissions and working periods but it lacks more detailed and useful information about submissions to allow tracing the students' improvement. This information could include number of submissions, grade obtained in every submission, time among submissions, and so on. Additionally it could be possible to see the evolution of these factors as long as new assignments are sent.
- Management of logs (FR33, FR34). – Currently VPL provides of logs about which action was done by a user inside the Moodle user interface (visit a page or start a process) but not about events happened inside the grading process.
- Programming language independence (NFR03). – VPL supports a defined set of programming languages for the source code to be graded. But the inclusion of new supported programming languages is limited to new updates of the tool. Then independence or the possibility to extend supported programming languages as the user wants is still required.
- Management of supported programming languages (FR21 – FR25). – Considering the lack of programming language independence in VPL there is a lack of a user interface to manage the supported programming languages inside the system. Additionally, according to the requirements given earlier, this management will allow relating grading-submodules to a supported programming language. This will be helpful to filter or classify grading-submodules.

It is worth mentioning that although VPL supports requirements NFR04 and NFR05, it is not optimal. This is because the teaching staff has to write a script to evaluate an assignment considering one or more metrics. In the worst case, this task would be repeated in every assignment if the method of grading changes. Then these requirements can be considered to be improved.

3.6 Defining the scope

Considering the goal of this work, which is to provide an architecture to support many ways of grading, some of the not fulfilled requirements will be treated. This subset of requirements is directly related to the automatic grading process considering grading-submodules. Table 9 shows them.

Table 9. Requirements considered for this work

Req. Code	Requirement
FR06	The system shall allow the teaching staff to add new grading-submodules inside the grading process.
FR07	The system shall allow the teaching staff to set parameter's values for every grading-submodule inside the grading process.
FR08	The system shall be able to let the teaching staff sort the grading-submodules in any order inside the grading-process.
FR26	The system shall provide a list of existent grading-submodules.
FR27	The system shall provide the complete information about a grading-submodule.
FR28	The system shall allow creating a new grading-submodule.
FR29	The system shall allow modifying attributes of a grading-submodule.
FR30	The system shall allow deleting a grading-submodule.
FR31	The system shall be able to let the administrator add new parameters to a grading-submodule.
FR32	The system shall be able to let the administrator modify parameters of a grading-submodule.
NFR03	The system shall be able to grade source code written in any programming language.
NFR04	The system shall allow adding any metric inside the grading process.
NFR05	The system shall allow sorting grading metrics in any arrangement.

It is worth saying that some requirements are implicitly related and therefore when solving one, another one will be solved as well. Then, the capability of adding any metric inside the grading process (requirement NFR04) will be solved when implementing the addition and configuration of any grading-submodule inside the grading process (requirements FR06 and FR07). To fulfill FR06 and FR07 requirements the implementation of grading-submodule management will be necessary (requirements FR26 – FR32). Likewise the capability of sorting grading metrics inside the grading process in any arrangement (requirement NFR05) will be solved when implementing sort of grading-submodules (requirement FR08). The independence of any programming language (requirement NFR03) could be obtained if the grading-submodule is defined as an agnostic artifact about the language. Then, any programming language could be supported.

3.7 The solution for the defined scope

Taking into account the context, the solution is to implement an automatic grading process considering grading-submodules. This solution will have as features modularity, extensibility, and flexibility in grading criteria and metrics inside a grading process. This solution will help to face the open research field given in the systematic review, which is the lack of a common model to grade. To implement the solution VPL has been selected as base tool to take advantage of most of its features and avoid "reinventing the wheel". The new features to implement and add to current VPL tool are:

- Management of grading-submodules.
- Management and configuration of grading process.
- Automatic grading process considering grading-submodules.

Figure 1 in the chapter 2 showed VPL's architecture. This considers the use of two subsystems: Moodle and Jail (a sandbox environment). Each of them is deployed in a different server. Then, the solution will have to consider these both subsystems inside VPL that will be called as VPL-Moodle and VPL-Jail subsystems. It is worth highlighting that these subsystems belong to VPL and they are not inside the proposed architecture. Figure 7 shows a block diagram, which defines modules already built in VPL and new modules to implement as part of the solution (Grading process management module and grading process module).

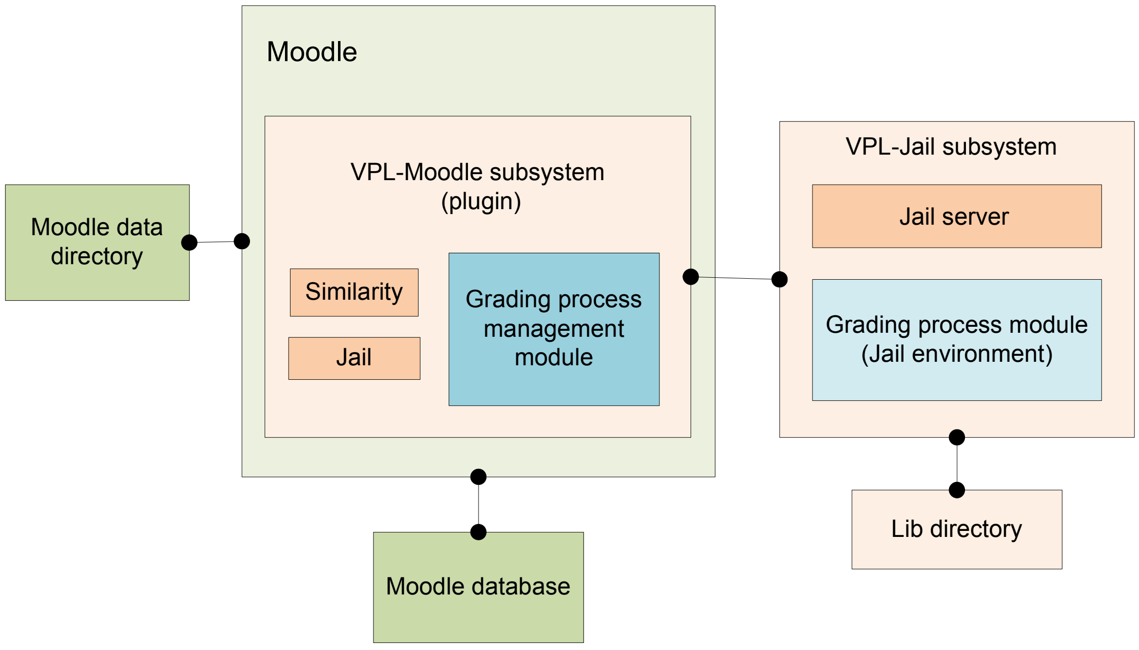

Moodle
VPL-Moodle subsystem (plugin)
Moodle data directory
Similarity
Jail
Grading process management module
VPL-Jail subsystem
Jail server
Grading process module (Jail environment)
Moodle database
Lib directory

Figure 7. VPL-Moodle and VPL-Jail subsystems

3.7.1 Analysis of the VPL-Moodle subsystem

The VPL plugin inside the Moodle server contains all the front-end of the VPL system. This has two highlighted modules (it can be seen in the source code): *similarity*, to detect plagiarism cases, and *jail* to communicate with the VPL-Jail subsystem and to store scripts which will be sent to be executed in the VPL-Jail subsystem as well.

Then, it is suitable to define a grading module, which joins up the management of grading-submodules, the management of the grading process, and means to send data and start the automatic grading process. The same jail module could be used to send data.

The management features will be provided through a web-based GUI. Every user interface will have its correspondent validation of data inputs. Behind them, a communication with the database and with the directories structure will be required as well. To send data, it will be necessary to use a program to package and send that using the XML-RPC protocol (This protocol is currently used by VPL tool). The data will include the source code files, any additional file required by the grading process and some information (which will guide the grading process in the Jail).

3.7.2 Analysis of VPL-Jail subsystem

The VPL-Jail subsystem is quite important because inside it, the grading process is performed. The Jail server, inside the VPL-Jail subsystem, provides its functionality as a service using *xinetd* program (refer to VPL's features in the state of the art chapter). It receives requests to perform the grading process and sends back the result as response.

Figure 8 shows the complete grading process since the evaluation request is received until the feedback is sent back as response.

When an evaluation request arrives, the server receives all data required and creates an isolated environment, the jail. The server places all the required files, needed to perform the grading process inside the jail environment. Then, the server executes a set of defined programs in a specific order to execute or evaluate the source code. This set of programs includes:

- An evaluation script that analyzes if the programming language of the source code is supported by the tool. If so, it calls the compilation program else it outputs an error message. Additionally, it monitors if there is a compilation error or if do not exist a file with test cases. In both cases it outputs an error message. It writes the execution program.
- A compilation script that compiles all the source code files and identify the main program.

- An execution file that sets up the test environment and perform testing of the main program against test cases. It outputs the result of this testing.

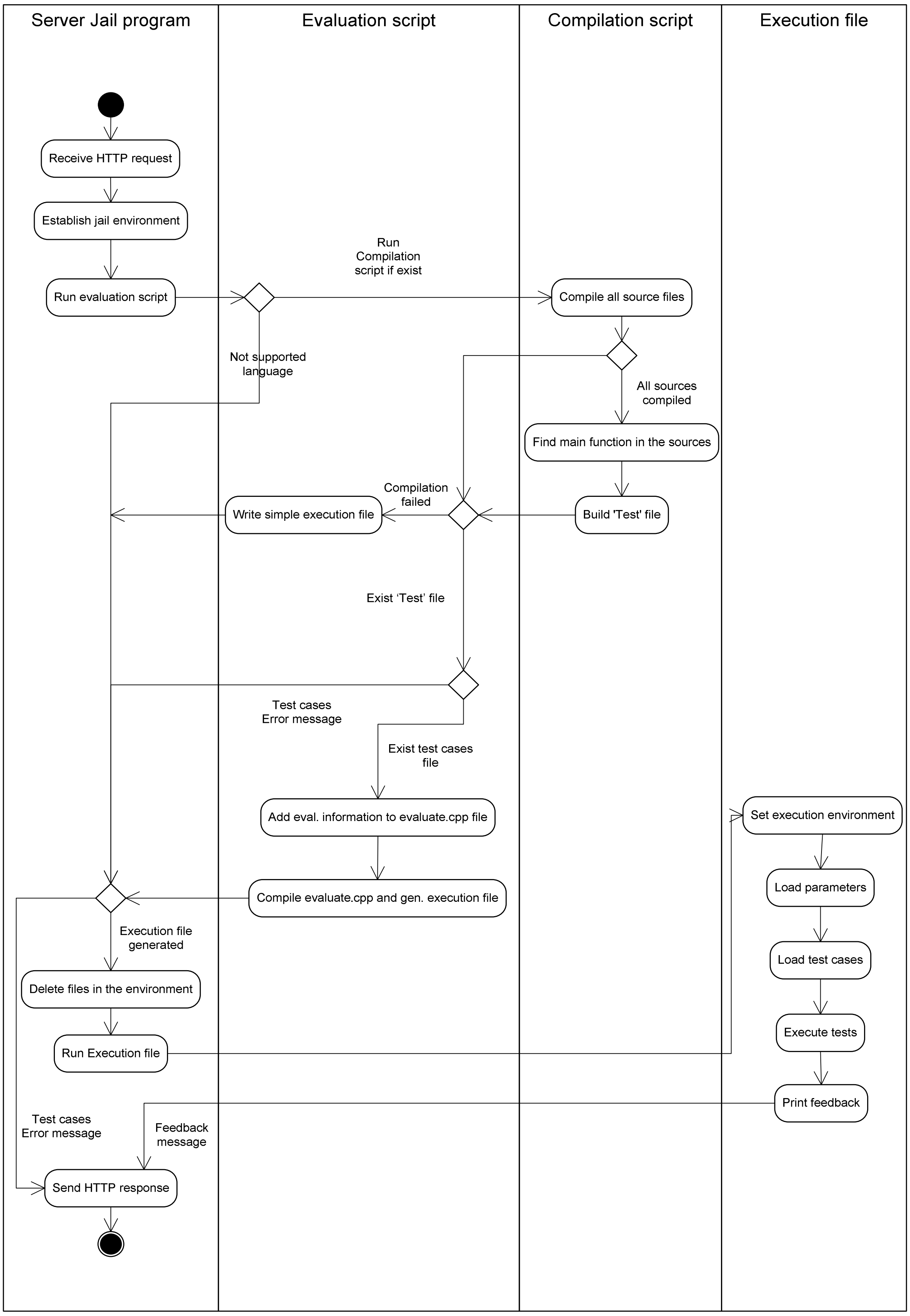


Figure 8. Grading process inside the VPL–Jail subsystem

The Jail server program calls directly the evaluation and the execution scripts. It gets their outputs and prepares a message to send back as feedback.

It is worth standing out that by default VPL uses a file of test cases. These cases are written by the teaching staff in a specific language defined by VPL.

The help of the tool mentions the possibility of using other methods (and other test cases) to grade assignments. It is through rewriting the evaluation script. The evaluation script is a Linux shell script which has to generate (mandatory) an execution file, which has to be another Linux shell script or a binary file. As it can be seen in Figure 8 this file performs the grading itself. Then, evaluation script and execution file are the interfaces to reach the integration between VPL and the new grading process. It has to be considered in the design chapter.

3.8 Chapter summary

This chapter has shown a set of results to describe the scope of this work. They include the identification of actors, the high level view of the system through a use case diagram, the definition of functional and non functional requirements, a solution for the general problem, the search of a suitable tool to be used as base for further development, the definition of requirements to fulfill in this work and the solution proposed considering two subsystems, VPL-Moodle and VPL-Jail (subsystems of VPL).

Every actor has its own interest and allowed actions inside the system. Then, a set of requirements for every one of them has been defined. The requirements include functional and non functional. Based on them, a set of important parameters has been defined to compare a set of mature automatic grading tools. The most suitable tool was VPL. This has some important advantages like its GNU/GPL license, the easy access to the code, its Moodle integration and its module of plagiarism detection. However VPL does not fulfill with all the requirements set in this work and it is necessary to add some new features.

This work will solve a subset of non fulfilled requirements aiming to provide modularity, extensibility and flexibility to the automatic grading process of programming assignments. The three following features have been identified to add to VPL plugin: management of grading-submodules, management of grading process and the automatic process itself.

All the considerations done in this chapter, the specification of a scope and the analysis for the VPL-Moodle and the VPL-Jail subsystems are helpful and allow focusing on the solution to be implemented in this work. Then, the next chapter will present the design of the new grading process architecture.

4 Design

The previous chapter has shown the necessity of implementing additional features to both subsystems inside VPL. The design of the solution for VPL-Moodle and VPL-Jail subsystems are presented here. This design will include an abstract architecture view and different perspectives of the solution. The design aims to define elements and their functionality, which will be helpful in the implementation stage.

4.1 VPL-Jail subsystem

The VPL-Jail subsystem hosts the jail environment. Inside this jail, the grading process will be performed. The jail will have all the necessary data before running the process. Considering this last fact is quite important to understand the proposed architecture.

4.1.1 Detailed architecture

The proposed architecture aims to provide modularity, extensibility and flexibility to the grading process through the use of grading-submodules previously defined (Refer to the analysis chapter).

A layer-based approach will help to get an abstract view of the architecture. To see the improvement provided by the proposed grading process architecture, the current process provided by VPL is shown first.

Current VPL's architecture

Figure 9 shows the current and default architecture in layers of the grading process in VPL. It is related to Figure 8 described in the analysis chapter (refer there to see details of the main programs). There, the grade is calculated only considering success test cases. These test cases are written in a specific language used by VPL.

This way of grading just considers correctness as the grading criterion. So, it is not possible to talk about modularity, extensibility and flexibility for the grading process (default configuration).

One important feature of VPL is the possibility of editing the evaluation script and making a customized grading process. Then VPL can consider more grading criteria inside the grading process. It can be seen in Figure 10.

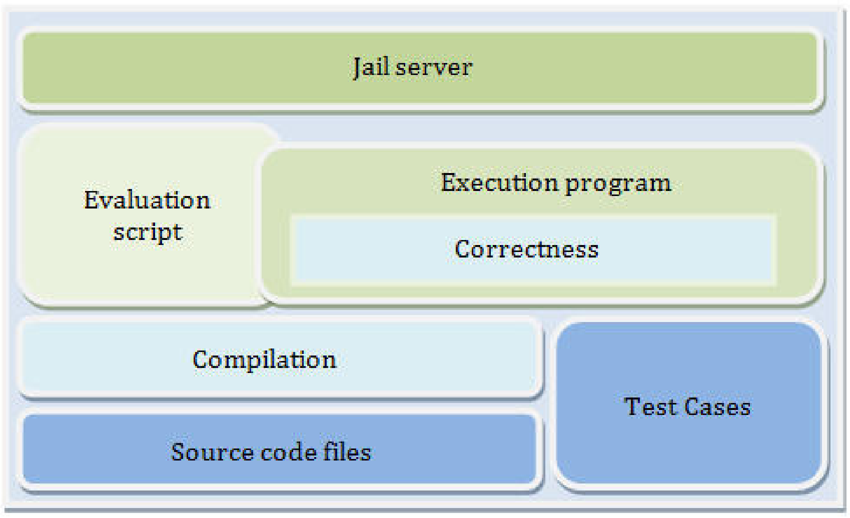


Figure 9. Default architecture for grading in VPL

Figure 10 depicts an architecture that has the execution program block, which considers n criteria inside it (every criterion could consider one or more grading metrics as well). There are two new blocks that are additional files and libraries, which represent files required to perform the grading process considering new criteria.

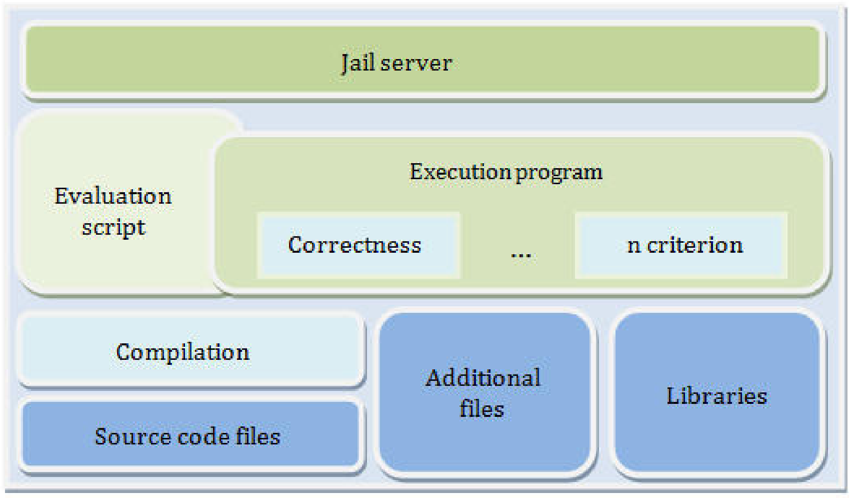


Figure 10. Architecture considering many metrics to grade in VPL

In spite of the ability to consider more grading criteria, which implies extensibility, VPL still lacks modularity and flexibility inside the grading process. This is because all criteria are mixed and statically arranged in a file script.

Proposed architecture

As it can be seen, it is necessary to define an architecture to support many criteria as independent from each other as possible. This independence will help to provide the desired modularity, extensibility and flexibility. Figure 11 shows the proposed architecture.

This architecture considers, as VPL, that all the required files for the grading process will be inside the jail before starting the process. The three bottom levels are completely dynamic. The grading-submodules layer provides of modularity, extensibility and flexibility to the grading process architecture.

It is necessary to make an explanation of every layer. Thus:

- **Environment builder. –** This layer is composed of two scripts, evaluation and execution. They are used to maintain compatibility with the VPL system but their functionality has been changed.
  The evaluation script makes any necessary processing on the files set, charset coding or decoding for instance. More tasks can be defined depending on needs. Additionally, this script has to generate dynamically the execution file.
  The execution file is generated dynamically. This script exports the necessary libraries' paths required by grading-submodules inside the grading process. Then, it calls the orchestrator program.

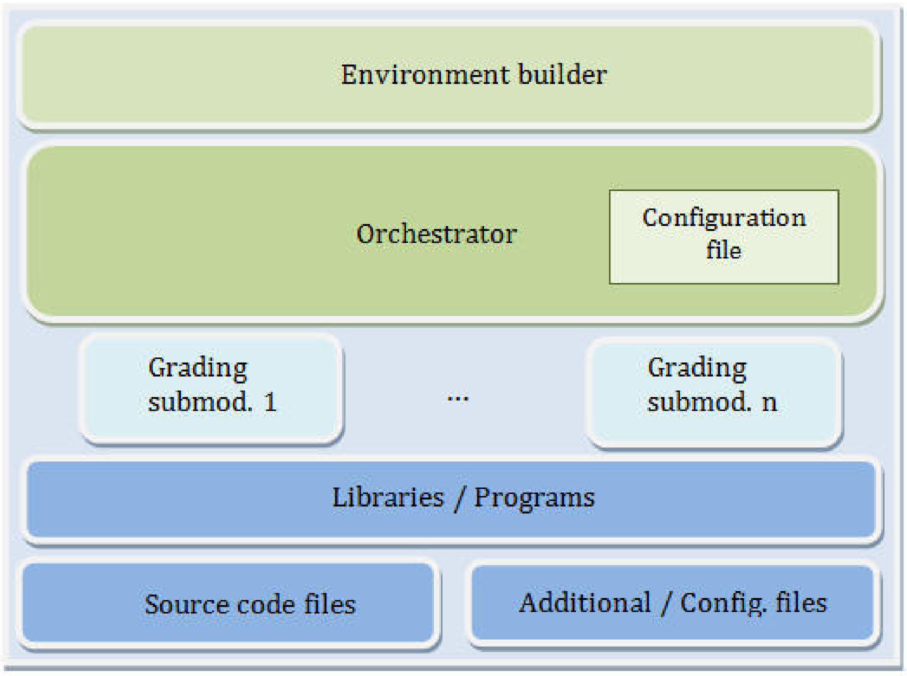


Figure 11. Proposed architecture for grading process

- **Orchestrator. -** This is a program that controls the whole grading process. As first task, the orchestrator loads information about how to perform the grading process. The information has to include information about the submission and the list of grading-submodules, with their parameters, to be performed.
  Based on the list, it calls every grading-submodule associated program. A mechanism for passing data from the orchestrator to this program and for

getting back the execution results has to be established. This mechanism will depend on the programming language used to write the grading-submodules associated programs and on the orchestrator's technology. This will be treated later, in the class diagram and in the implementation section.
After grading-submodules calls are finished, the orchestrator will process every grading-submodule results to calculate the final grade and to establish comments.
Finally, the feedback (grade and comments) will be sent back to the Moodle server.

- **Submission configuration file. -** This file contains submission's metadata and information to be used by every grading-submodule associated program, which will be performed inside the grading process. It could be used to save information about the results of the grading process as well.
The complete description of this file will be shown in the implementation section.
- **Grading-submodule. –** It is a new artifact designed for this work and defined in the analysis chapter. It is associated implicitly to a grading metric or to a grading criterion, and to a programming language (used to write the source code to be evaluated). This artifact has an associated program (which evaluates the source code) that will be executed in the grading process. The number of grading-submodules and its arrangement inside the grading process are depending on the assignment.
- **Libraries and programs.** – It refers to external programs or packages required by the grading-submodules associated programs.
- **Source Files. –** It refers to files written and sent by the students in a submission to accomplish with an assignment.
- **Additional or configuration files. -** Files defined by the teaching staff and required by the grading-submodules associated program inside the grading process, for instance test cases, rules files, etc.

4.1.2 Process perspective

The grading process starts when all the required files are inside the jail and the Jail server executes the evaluation script. In that moment an ordered process starts. Figure 12 shows the whole grading process. This figure is very similar to the architecture shown before but it is helpful to get a better understanding about the responsibilities of the different elements of the proposed architecture in the grading process.

Every grading-submodule is well defined (when it is associated to a one grading-metric or to a one grading criterion), so it provides of **modularity** to the grading process. There is not a limit for the number of grading-submodules used inside the process and they can be added as the teaching staff needs, it implies **extensibility.** The grading-submodules can be arranged in any way, so there is **flexibility** inside the grading process. The number of grading-submodules inside the process, the order and how to call them are defined in the configuration file.

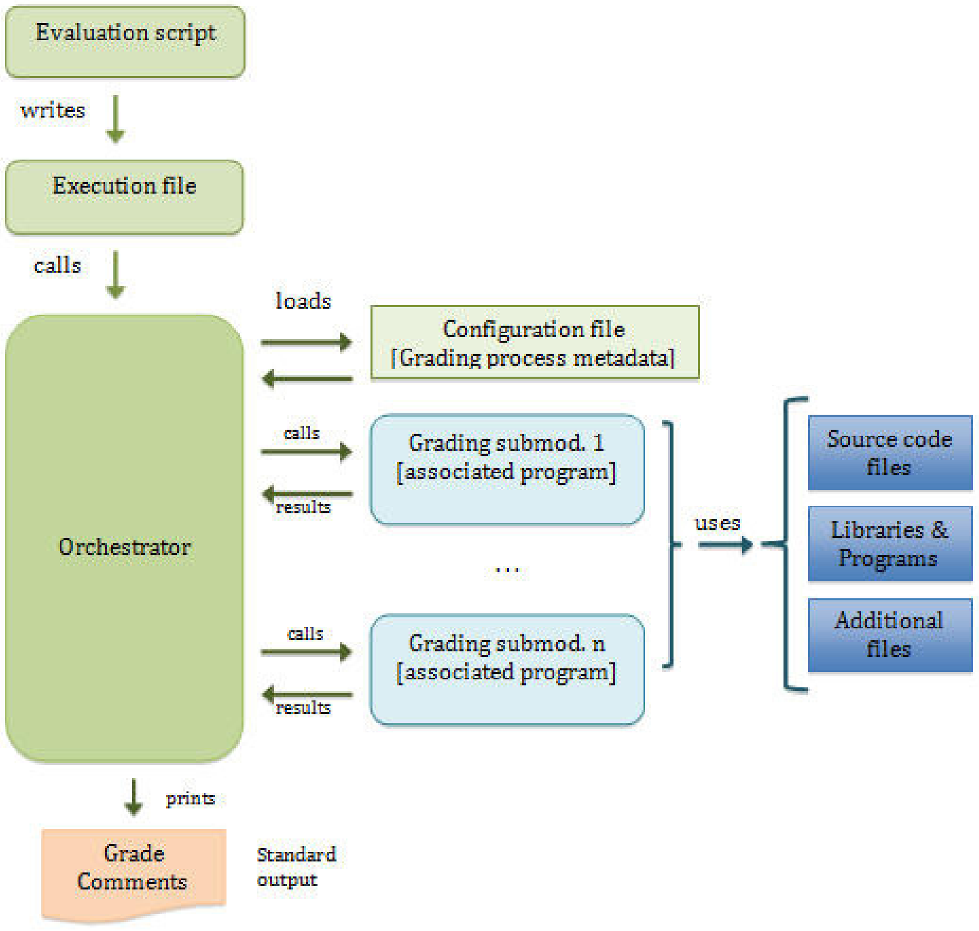


Figure 12. Elements and calls inside the grading process

An advantage of the proposed architecture is that evaluation and execution files are not essential for the well working of the whole architecture. It means that this architecture can be used by other systems just implementing an interface, which sets up the environment and calls the orchestrator. But to test this architecture, VPL is going to be used as base tool. Then, these files will allow maintaining integration with VPL.

The grading process (VPL-Jail subsystem) showed in Figure 8 has changed. Figure 13 shows the new grading process as a sequence of stages.

Considering Figure 13 it is possible to see that evaluation script and execution file perform basic tasks. One of them would be sufficient. But it is seen that Jail server program calls directly these files. So they are needed to guarantee compatibility.

The execution file calls the orchestrator and this last takes in charge the grading process. It calls every grading-submodule associated program (this interaction will be seen later), calculates the final grade, forms the feedback and prints it.

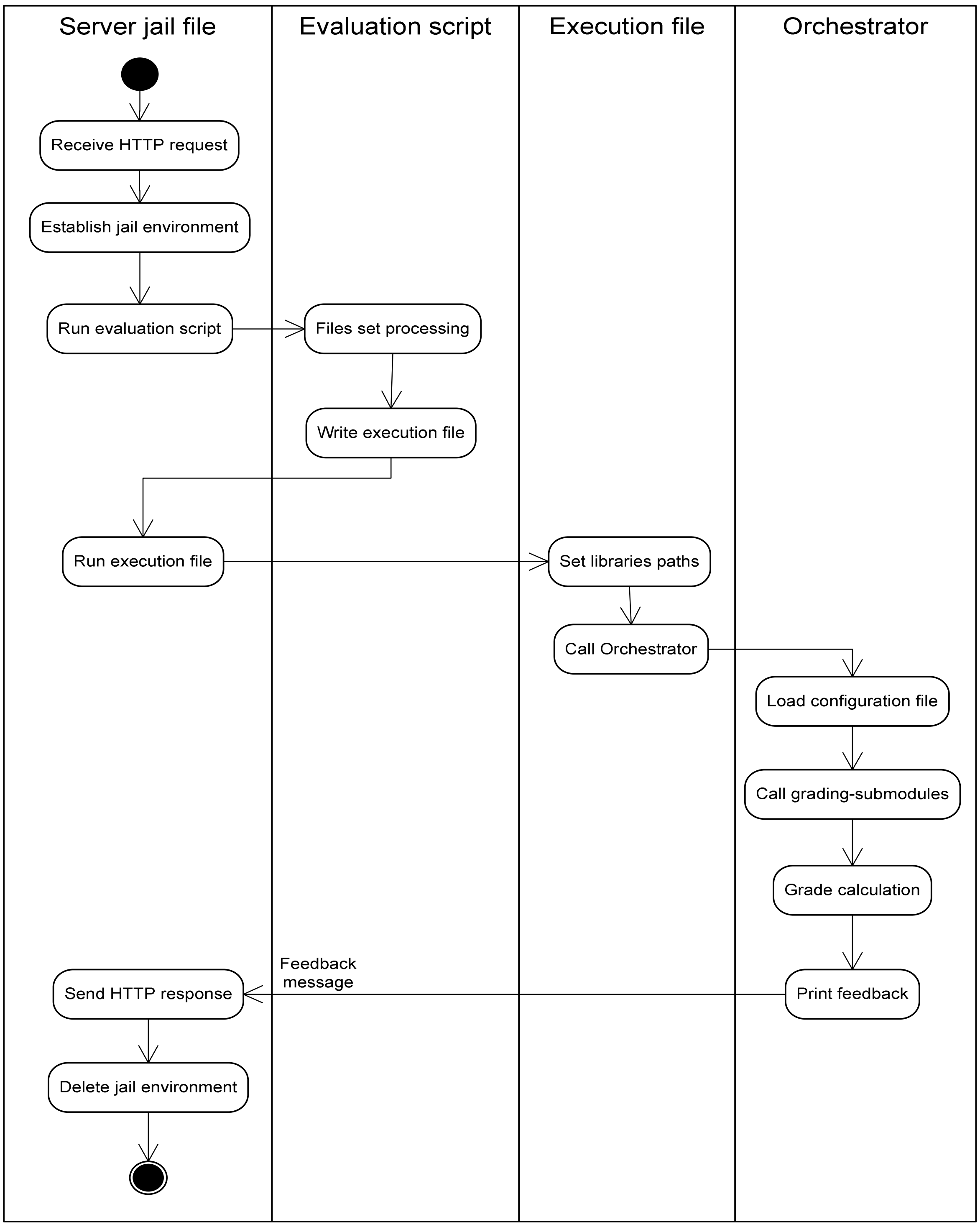
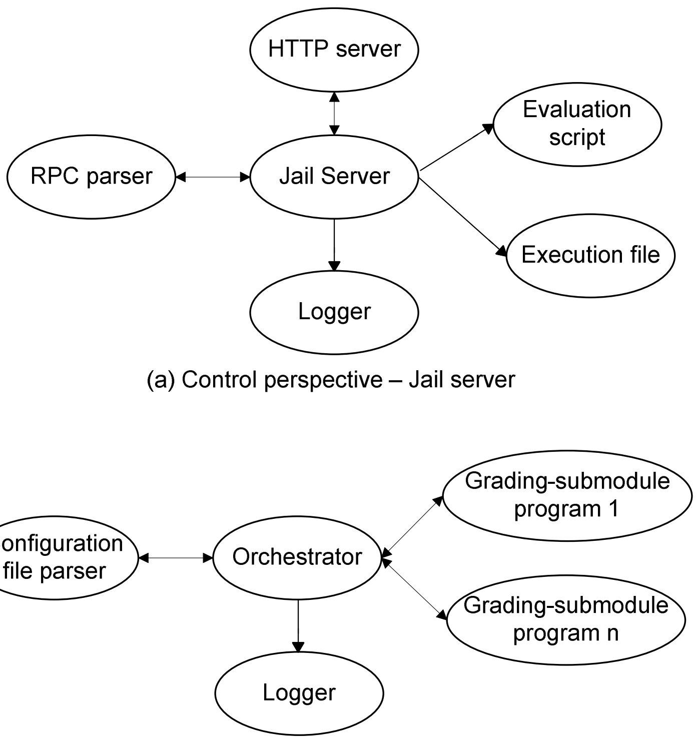


Figure 13. Grading process inside the VPL-Jail subsystem

4.1.3 Control perspective

There are two elements that take a role of controllers inside the VPL-Jail subsystem and they are the Jail server and the Orchestrator. The first one is already implemented in the VPL tool and interacts with: an HTTP server to treat the incoming requests, a RPC parser to interpret the information received through the XML-RPC protocol, the evaluation script, the execution file, and with a logger (which registers every state while setting up the jail environment). The second one, the Orchestrator, is very important considering the new grading process architecture. This interacts with a parser for the configuration file, with every grading-submodule associated program, and with a logger tool. The logger tool is necessary to register every state of the whole grading-process. Figure 14 shows the control perspective.


HTTP server
RPC parser
Jail Server
Evaluation script
Execution file
Logger
(a) Control perspective – Jail server
Configuration file parser
Orchestrator
Grading-submodule program 1
Grading-submodule program n
Logger

(b) Control perspective – Jail [Grading process]

Figure 14. Controllers in the VPL-Jail subsystem

4.1.4 Objects-oriented design

Considering the last section, we now have a concrete idea about the elements related inside the grading process and which their roles are. This section aims to give a more detailed description of every element. The goal is to describe as much as possible that elements because they will be helpful in the implementation stage.

The oriented objects paradigm focuses in representing real life objects. An objects-oriented design provides of features like modularity and encapsulation, extensibility and reusability (Bellas). Two of these features are common to those aimed in the proposed architecture. Then, the objects-oriented approach is a suitable choice to design.

Orchestrator inclusion

Before going on to define required objects, it is worth mentioning that the orchestrator itself could be a tool. Its function is controlling and orchestrating the process, calling a set of grading-submodules, performing calculation and outputting the feedback. Precisely these functionalities can be seen as tasks inside a process. Then, there are some tools already built, which are designed to work with this kind of processes and they could be a good choice to avoid an implementation. These tools include: Apache Ant, Maven, Gradle, and GNU make. They have been reviewed in the state of the art chapter.

Every tool has a set of very powerful features and they mostly aim to build, deploy or install projects and applications. Even though these are not the goal of the grading process, some of these tools mention as feature the ability to support any process that can be represented as a sequence of tasks.

Regarding Apache Ant, it supports any programming language, has a set of already defined useful tasks, and supports the addition of new of them as well. Just this last feature would be helpful for the purpose of adding more grading-submodules as the teaching staff needs. To add a new task it is necessary to code a Java program associated and then to define and register this task using a XML format. It implies an extra step which is the registration; ideally the user should be avoided doing extra steps to create the new task. The process is described in a XML configuration file, but when the process starts there is not a communication between the orchestrator and the tasks, and then to save the task's results it would be necessary another external file.

Regarding Maven, it controls dependencies inside Java projects. It has a set of already defined tasks but supports the definition of new of them. The process uses a XML configuration file as well. The only support for Java projects would be a great problem but it can be solved by implementing a wrapper in the grading-submodule. The issue with this tool is, as with Apache Ant, the necessity of and additional registration stage of the task and the use of an external file to save results.

Regarding Gradle, this works with Java, Groovy, OSGi, and Scala projects and has the possibility of adding new tasks to a set already defined. The language support would be solved with a wrapper as well. The definition of new tasks will imply only writing a script. The process is controlled by a script written in Groovy. It could be a problem; maybe having a XML representation to control the process would be better and easier.

GNU make, has a set of good features, which include: independence of programming languages and every task can be defined with direct calls in the console.

There are some issues, which include: the loss of control in error messages, the stopping of the process when there is a problem in one task, the *makefile* has to be carefully generated (a blank space makes difference), and it would use an external file to save results as well.

At the end, considering the issues of the analyzed tools, it has been decided to build an orchestrator. This orchestrator will have as features: the use of a configuration file to define the process, support for new tasks inside the process (grading-submodules), communication with every grading-submodule while performing the grading-process, avoidance of external and additional files to save results, and the ability to continue with the process in spite of a failure in one task.

Objects' identification

The third and fourth layers of the proposed architecture shown in Figure 11 will require a complete implementation stage. Then, these layers are the place to start identifying objects, which could be considered to be abstracted as classes later to be depicted in a diagram. Thus in a first sight the next objects can be considered:

- Orchestrator, the object that controls the whole grading process. Its functionality has been described in the proposed architecture section.
- Configuration file, the object that has the complete information to perform the whole grading process on a student submission. This will include general information about the submission (final grade, submission identification, and so on) and about every grading-submodule to be performed in the grading process.
- Grading-submodule associated program, this is a program associated to the grading-submodule artifact. This program will act as a wrapper to support the evaluation of any source code written in any programming language.
- Logger, this object will register every event inside the grading-process in a log file.

It is necessary to make a deeper analysis of some of the identified objects.

Regarding grading-submodule associated program. There will be as many of them as grading-submodule were configured inside the grading process. So, they will be added as the teaching staff needs. It is not possible to define all grading-submodule associated programs that will be in the grading process. Then, it is better to define an object that will act as an 'intermediary' between the orchestrator and any grading-submodule associated program created.

Regarding logger, there are already implemented tools of this kind for many programming languages. Even though the programming language is not yet selected, it is very probable that the manual implementation of a logger will not be necessary.

Finally, there have been identified three objects, which will be depicted in the class diagram: the orchestrator, the configuration file, and the grading-submodule associated program (acting as an 'intermediary').

Class diagram

Having the objects defined, it is possible to make an abstraction of them to make a formal representation in an UML (Unified Modeling Language) class diagram. This is shown in Figure 15.

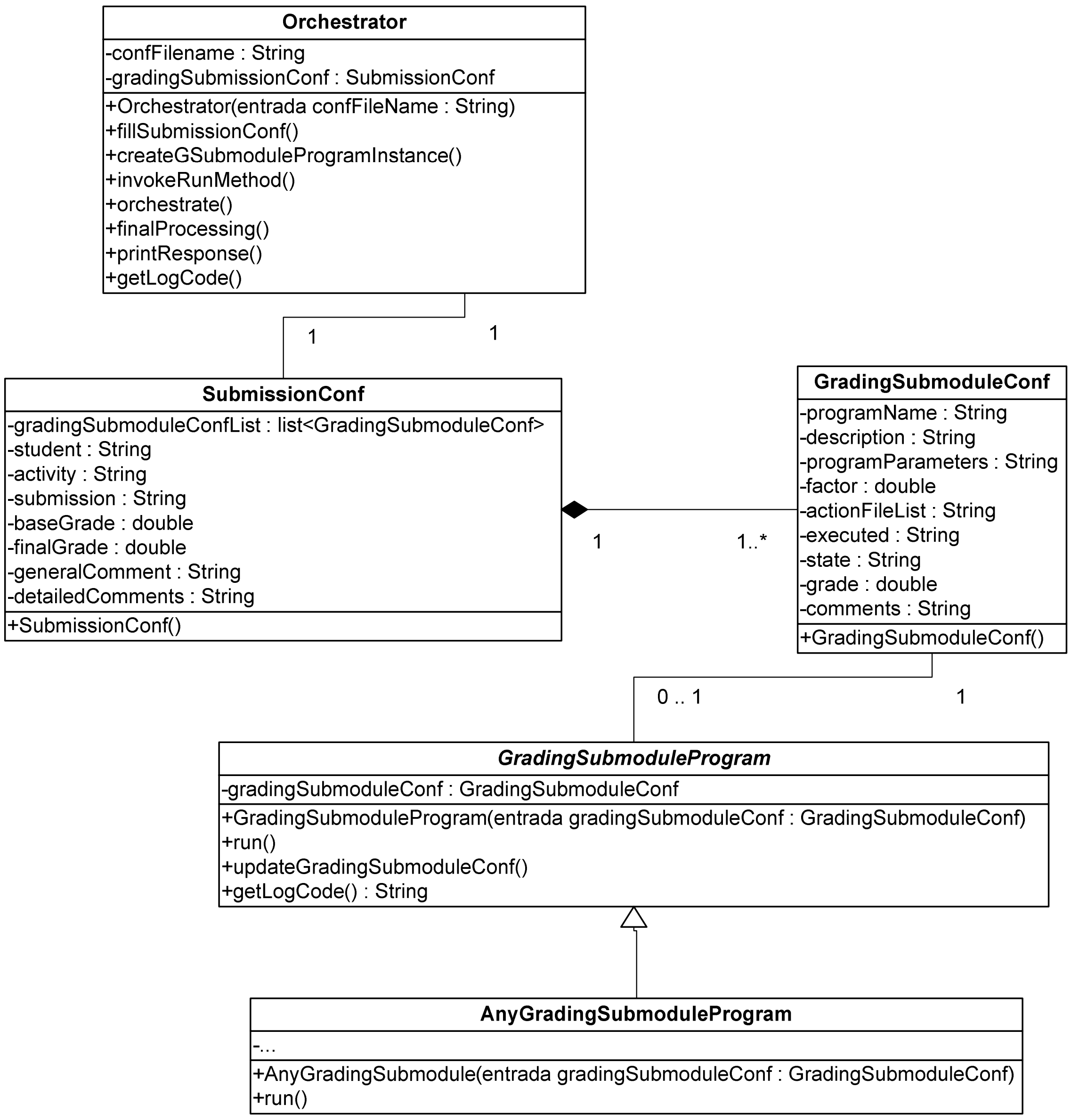


Figure 15. Class diagram for orchestrator and grading-submodules

It is necessary to make an explanation about Figure 15 to get a better understanding of the diagram.

The *SubmissionConf* and the *GradingSubmoduleConf* classes have been abstracted from the configuration file object. The first one includes information about the whole submission and will be used by the orchestrator to start the grading process. The second one represents information to be used for every grading-submodule associated program. A review of the configuration file structure will be helpful to understand this couple of classes (This structure will be shown in the implementation section in the implementation chapter).

The *GradingSubmoduleProgram* class has been abstracted from the grading-submodule associated program object and has been defined as abstract because it acts as 'intermediary' between the orchestrator and any grading-submodule associated program that the teaching staff or administrator will add. This has been defined as an abstract class instead of interface because it counts with already defined operations. This class has as attribute an instance of *GradingSubmoduleConf*, which will be used to receive information from the orchestrator. Then it is required that the constructor receives an instance of *GradingSubmoduleConf*. Additionally, this class forces to every subclass to implement the *run()* abstract operation. This operation should define: how to evaluate a source code (sent by the student) considering a given criterion, how the grade will be calculated and how the feedback will be composed.

The *AnyGradingSubmoduleProgram* class has been depicted to represent any grading-submodule associated program added by the administrator or by the teaching staff that will be considered inside the grading process. It extends the *GradingSubmoduleProgram* class and therefore has to implement the *run()* operation.

The orchestrator requires of an instance of *SubmissionConf* because it has all the information about the submission. To fill any data inside this instance it is necessary to establish the name of the real configuration file; so this is received by the constructor. The Orchestrator's operations are quite interesting and they include:

- Loading the data inside the instance of *SubmissionConf*. For this process it is required to know the name of the configuration file, which will be mapped.
- Orchestrating the process. It refers to iterate the list of *GradingSubmoduleConf* inside the *GradingSubmissionConf* to operate sequentially every grading-submodule. In an iteration, the operation to create an instance of *AnyGradingSubmoduleProgram* and to invoke the run operation inside that will be called.

- Creating dynamically an instance of *AnyGradingSubmoduleProgram*. When performing this operation, an instance of *GradingSubmoduleConf* (It is obtained in a given iteration of the list inside the *GradingSubmissionConf*) will be used as creation's argument. This operation will return the instance of *AnyGradingSubmoduleProgram*.
- Invoking to the *run()* operation defined inside an instance *AnyGradingSubmoduleProgram*. After this invocation, the instance of *GradingSubmoduleConf* passed as argument will have all the resultant information of the evaluation considering the criterion associated implicitly to the grading-submodule.
- A final processing. Considering that the list of *GradingSubmoduleConf* inside the *GradingSubmissionConf* is already updated and contains all the results of the grading process. The list is processed again to calculate the final grade, to collect the individual comments and to establish a general comment of the whole process.
- Outputting the response. The general comment, the detailed comments and the final grade are output in a format to be recognized as response feedback by the Jail server.

Considering that the architecture will be validated through some study cases it is necessary to define a set of *GradingSubmoduleProgram* subclasses. Then, Figure 16 includes an extension for the class diagram shown previously.

The extended class diagram includes four subclasses extended from *GradingSubmoduleProgram*. These new classes are oriented to check the structure of a set of files (*CheckGradingSubmodule*), to compile a set of source code files (*CompilationGradingSubmodule*), to test a set of source code files against test cases (*TestGradingSubmodule*), and to evaluate the style of a source code file (*StyleGradingSubmodule*). Their functionality will be expressed in the *run()* operation and it will be explained in the implementation stage.

While designing these new classes, the necessity of a class which allows executing system commands has appeared. For example, inside the *run()* operation in *CompilationGradingSubmodule*, a compiler has to be called. Then, the *CommandExecutor* class has been defined, which basically has a command attribute and an operation to execute the command. The complete implications will be seen in the implementation stage.

To make possible the access to this class since any *GradingSubmoduleProgram* subclass, a new attribute in the *GradingSubmoduleProgram* abstract class has been added. This new attribute is an instance *CommandExecutor* class. Additionally in the same abstract class there have been defined more operations, which aim to provide of a quick access to execute system commands from the subclasses. These operations include: execution of a string command, getting the execution results through the standard output and getting messages from the standard error.

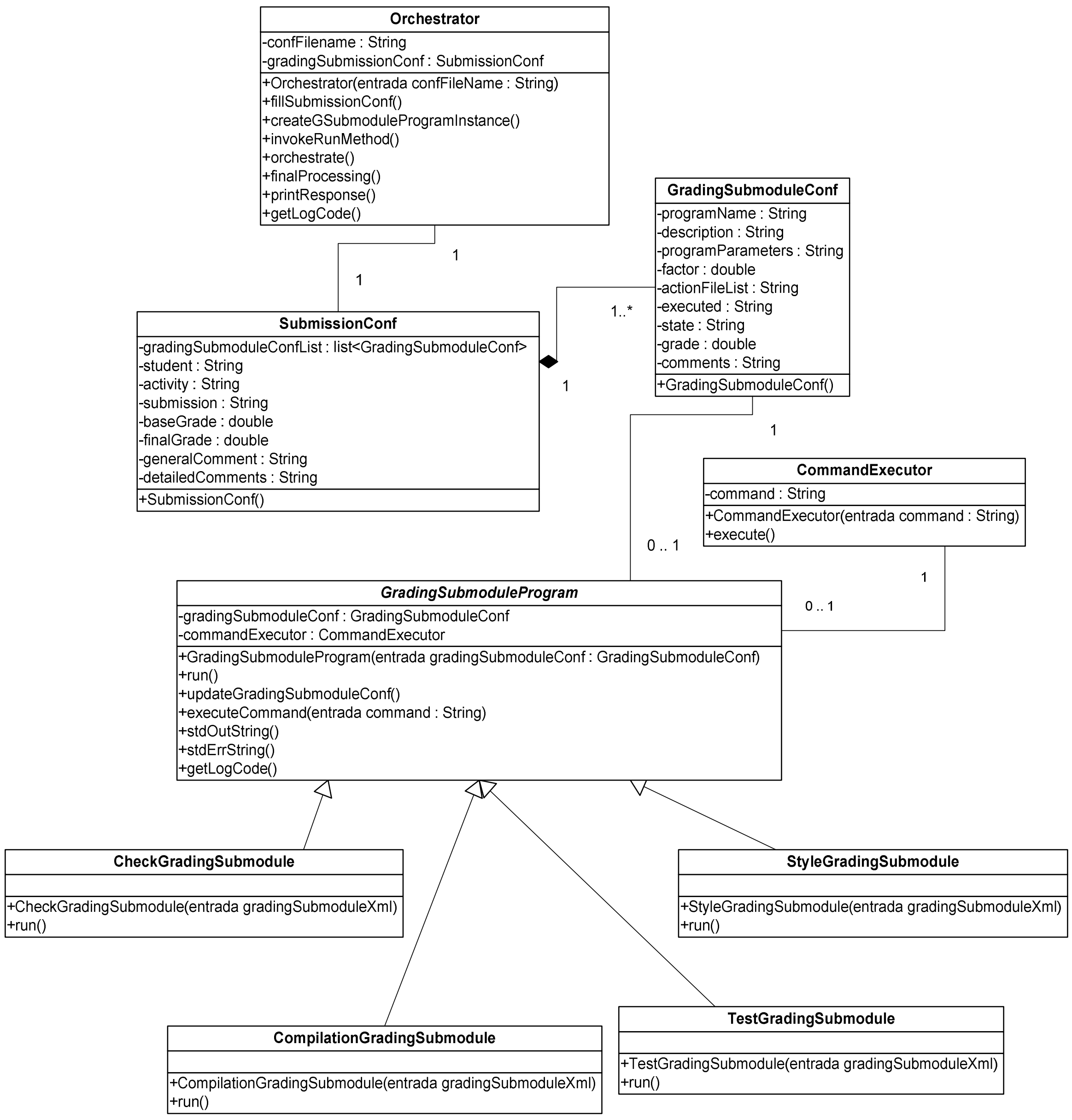

Orchestrator
-confFilename : String
-gradingSubmissionConf : SubmissionConf
+Orchestrator(entrada confFileName : String)
+fillSubmissionConf()
+createGSubmoduleProgramInstance()
+invokeRunMethod()
+orchestrate()
+finalProcessing()
+printResponse()
+getLogCode()
GradingSubmoduleConf
-programName : String
-description : String
-programParameters : String
-factor : double
-actionFileList : String
-executed : String
-state : String
-grade : double
-comments : String
+GradingSubmoduleConf()
1
1
1..*
1
SubmissionConf
-gradingSubmoduleConfList : list<GradingSubmoduleConf>
-student : String
-activity : String
-submission : String
-baseGrade : double
-finalGrade : double
-generalComment : String
-detailedComments : String
+SubmissionConf()
1
CommandExecutor
-command : String
+CommandExecutor(entrada command : String)
+execute()
0 .. 1
1
0 .. 1
GradingSubmoduleProgram
-gradingSubmoduleConf : GradingSubmoduleConf
-commandExecutor : CommandExecutor
+GradingSubmoduleProgram(entrada gradingSubmoduleConf : GradingSubmoduleConf)
+run()
+updateGradingSubmoduleConf()
+executeCommand(entrada command : String)
+stdOutString()
+stdErrString()
+getLogCode()
CheckGradingSubmodule
+CheckGradingSubmodule(entrada gradingSubmoduleXml)
+run()
StyleGradingSubmodule
+StyleGradingSubmodule(entrada gradingSubmoduleXml)
+run()
TestGradingSubmodule
+TestGradingSubmodule(entrada gradingSubmoduleXml)
+run()
CompilationGradingSubmodule
+CompilationGradingSubmodule(entrada gradingSubmoduleXml)
+run()

Figure 16. Extended class diagram with *GradingSubmoduleProgram* subclases

Classes packages

The previously defined classes can be arranged in a package diagram. In spite of the size of the project, the intention is to get a separation among classes based on functionality and semantics. Additionally it will help to have an order in the implementation stage. Ruiz et al in (Ruiz et al.) give some considerations to sort elements inside a package, these include: having strong conceptual relations, and being related through inheritance. Then there have been established some packages, which are depicted in Figure 17.

Figure 17 only considers new classes defined for this work. As it has been said before, a Logger class probably does already exist and this has not been depicted.

The *SubmissionConf* and *GradingSubmoduleConf* are related through a composition, so they go inside the same package called Parsing. *Grading Submodules* package stores the *GradingSubmoduleProgram* class and all its subclasses because they are related through inheritance. Additionally the packages have been depicted to see their relation with every layer of the proposed architecture.

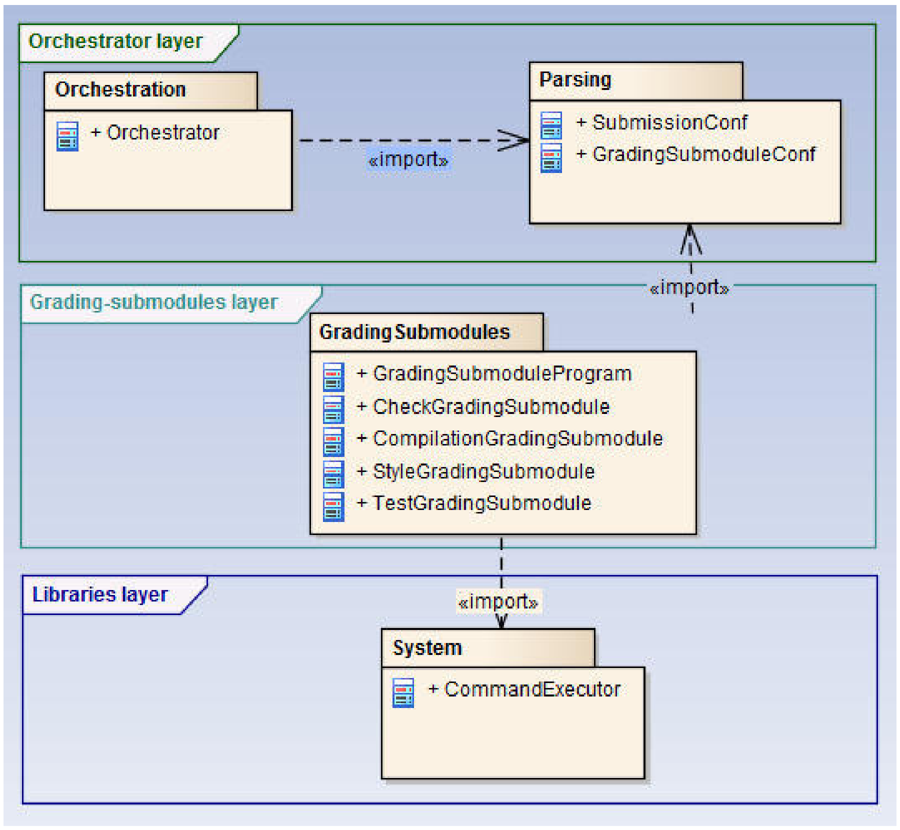


Figure 17. Package diagram of the orchestrator and grading-submodules levels

Finally, it is possible to see that there are two parts inside this solution, one that will be static after its implementation and another one that will be changing depending on more classes' addition. The first group is composed by *Orchestration*, *Parsing* and *System* packages, and will shape the core of the solution. The second group will be shaped by the *Grading Submodules* package, which initially only will have the *GradingSubmoduleProgram* abstract class but eventually will grew when adding new grading-submodule associated programs.

Interactions inside the grading process

The role of every element in the grading process inside the jail has been defined and it helps to highlight the importance of each of them. It is worth noting that there is more interaction among some elements; they are the *Orchestrator* and *AnyGradingSubmoduleProgram*. This interaction increases as more grading-submodule associated programs are considered inside the grading process. Additionally, in cases when a system's command execution is required, the interaction among these elements increases as well.

Then, the sequence diagram shown in Figure 18 is helpful to understand in a better way the interactions. This figure only includes objects and functions performed.

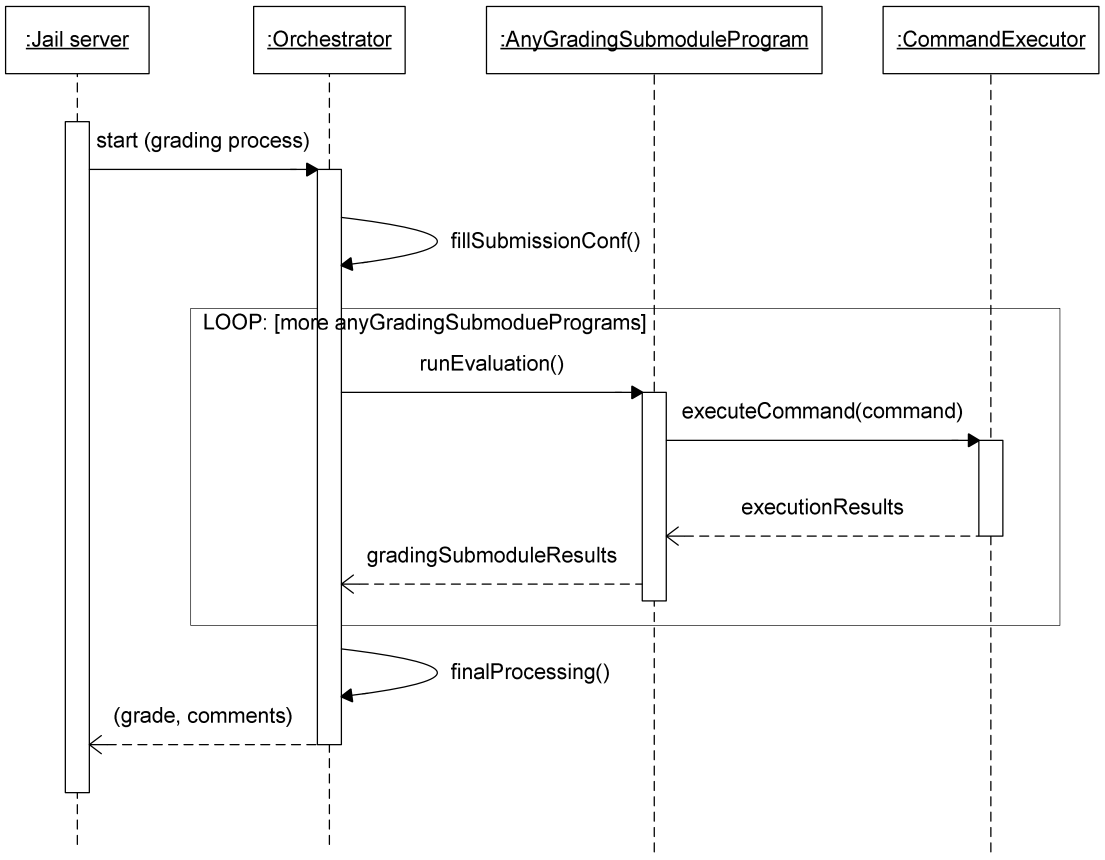


Figure 18. Grading process interactions

There are some interesting facts in this representation:

- Even though the execution file calls the orchestrator, the grading process is started by an instance of the Jail server. The execution file is just an intermediary, which is not an object, and has been maintained to keep the integration with VPL. Additionally the execution file does not care about the response; this last is only used by the Jail server. Finally, the execution file has not been represented.
- The *SubmissionConf* and *GradingSubmoduleConf* classes have not been represented because they only represent the configuration file (parsing files). They do not perform any action as well.
- As it has been said before, it is very probable that system calls will be performed inside instances of *AnyGradingSubmoduleProgram*. Then it is necessary to interact with an object (instance of *CommandExecutor*) to execute

commands in the console system. The number of calls to execute a command is not determined but it could be zero or more.

- There is a loop started in the *Orchestrator* instance. This loop makes calls to *run()* operation inside every *AnyGradingSubmoduleProgram*.
- The *Orchestrator* instance makes calls to operations *fillSubmissionConf()* and *finalProcessing()*.
- Finally, the Jail server starts the process and waits to get the result with the grade and comments to send the feedback.

The new architecture for the grading process has been given. This could be tested without having a formal user interface. The only requirements are to have all the required files inside the jail, and that a program exports the libraries' paths and starts the process by calling a *Orchestrator* instance.

4.2 VPL-Moodle subsystem

The analysis chapter determined the necessity of implementing new features inside the VPL Moodle's plugin, they include: the grading-submodules management, the grading process management and a mean to communicate with the VPL-Jail subsystem; all of them implemented in a new VPL's module called Grading process management module.

To use the grading module, grading-submodules CRUD actions have to be carried out. Then, after the creation of a new VPL activity, the grading process management can be performed. This management includes: the addition or deletion of grading-submodules in the grading process, the sorting of them in any order, and the values assignation to any parameter required by the grading-submodules. When a submission related to a VPL activity is done, the configuration of the grading process is reviewed and then all the necessary data (student's source code files, additional files required by the grading-submodules associated programs) is collected and sent to the VPL-Jail subsystem.

Every new feature will be treated here to provide some design implications, which will make easier the implementation of that features in the next chapter. It is necessary to establish the data model before implementing these new features as well.

4.2.1 Data Model

Moodle's data is distributed in the Moodle's database and in the Moodle's data directory. The second one saves files in a directory structure to avoid an overloading in the database. Other kind of information is saved in the Moodle's database.

VPL is a Moodle's activity plugin and adds some tables to the Moodle's database. Likewise Moodle, VPL uses a directory structure to save files related to a VPL activity.

The new grading module inside VPL will add some new tables to the Moodle's database as well. They will be defined later in this section. Additionally, considering that a grading-submodule has a program associated and this could require of additional files to work in a right way, it will be necessary to define a directory structure for the new module.

Figure 19 shows a representation of VPL-Moodle subsystem, focusing on the structure of the data directories. The new submodule will use two directories inside its root directory. One of them is to store the source code of all the grading-submodules associated programs, and the other one is to store additional files for every VPL activity that has a grading process associated.

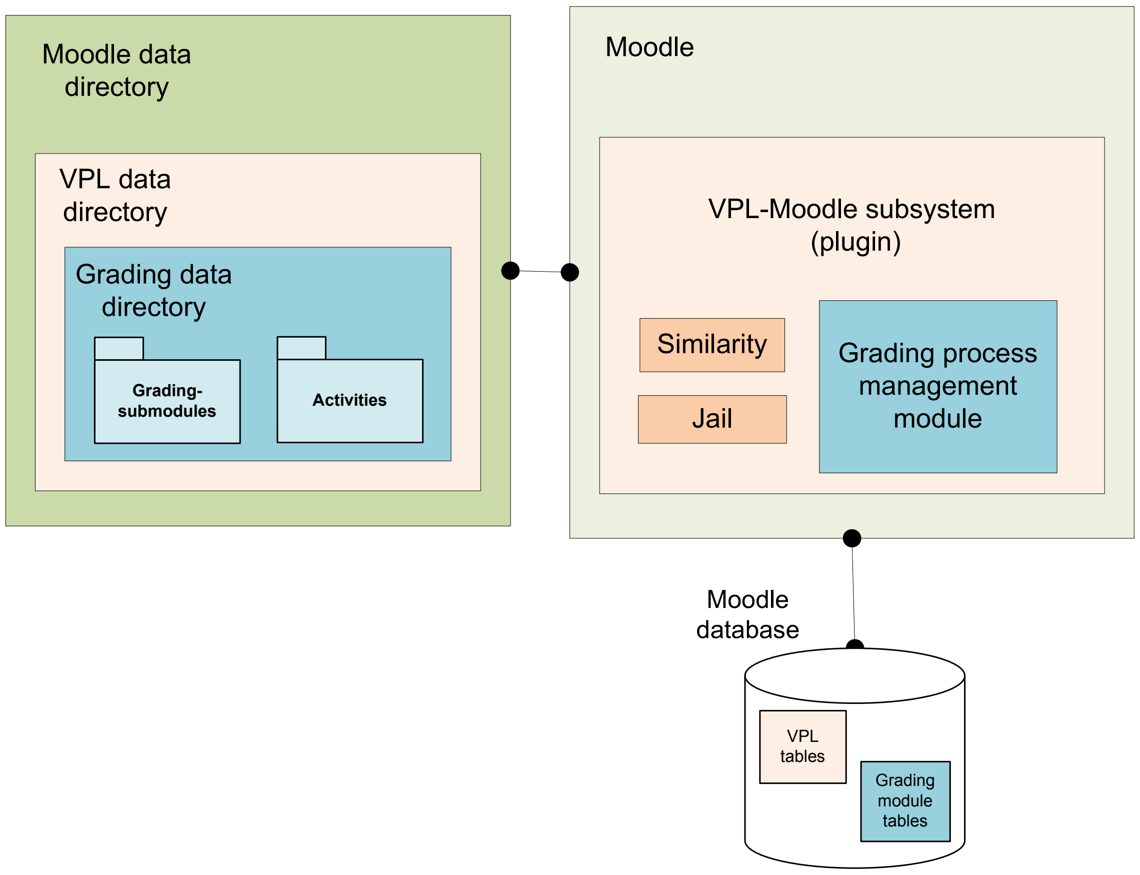


Figure 19. Data directories inside the VPL-Moodle subsystem

This focus of having separated directories helps in maintaining tasks. This avoids the dependency of the Moodle's web interface to manage files. The administrator can go directly inside a directory to look for the file associated to a grading-submodule, or for the files associated to a given VPL activity (which has additional files inside its grading process).

More formally, it is necessary to model the new database's tables. But firstly, it is necessary to know the next implications:

- The Moodle's database lacks of foreign keys which implies that there is no referential integrity. It does not mean that the DBMS (Database Management System) used in the data layer for Moodle lacks of referential integrity support. It means that tables inside the Moodle's database have not been created expressing foreign keys, then they are not "related" and the DBMS is unable to check for referential integrity. This topic has been further analyzed in a forum in the Moodle's official site[36]. In spite of XMLDB (tool provided inside Moodle to define new tables to create in the database) allows specifying foreign keys[37], these foreign keys are not reflected in the database. They are saved as useful information to get a constrained system in the future.
- The official Moodle's documentation says that every table must have as primary key, a field named 'id'[38].

Considering the former points, Figure 20 shows the semantic data model for the grading module.

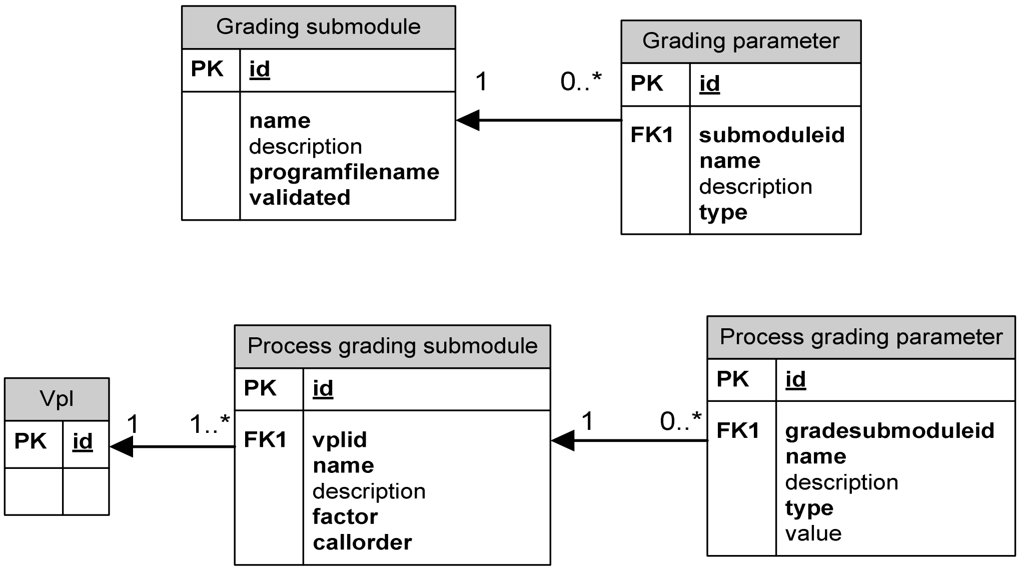


Figure 20. Semantic data model for the grading module

Figure 20 depicts a set of entities, which have been defined thinking analogously to a simulator program, which gives the possibility of adding new simulation elements. It is, on the one hand, there is the possibility of adding more elements through their right definition. On the other hand, when considering the creation of a new simulation environment, it would be possible to depict any simulation element inside the environment and setting its parameters.

[36] https://moodle.org/mod/forum/discuss.php?d=200829
[37] http://docs.moodle.org/dev/XMLDB_defining_an_XML_structure#Conventions
[38] http://docs.moodle.org/dev/Database

It is necessary to make an explanation about every entity defined. Thus:

- Grading submodule. It is the representation of the grading-submodule artifact defined in the analysis chapter. Its attributes *name* and *description* are self described. The attribute *programfilename* saves the absolute path to the location of the program file associated to the grading-submodule. The attribute *validated* indicates if the associated program has no errors and whether it is completely functional.
- Grading parameter. Every grading-submodule associated program could require of parameters to its proper working. For example filenames, paths, and so on. It will depend on criteria considered when designing the grading-submodule. This entity will save the definition of the required parameters.
- Vpl. It is already defined inside VPL's data model. Its *id* field allows linking the grading process to a VPL activity.
- Process grading submodule. Every VPL activity will have a set of grading-submodules to be used inside its grading process. When a grading-submodule is selected to be part of this grading process, this is converted in an element of that process, the new entity is called Process grading submodule. This entity captures some attributes of the original Grading submodule entity but they are completely independent. Additionally this entity has some extra attributes that makes sense only inside the grading process (the call order and a factor to calculate a final grade for a submission for instance).
- Process grading parameter. This entity saves all the values for parameters required by the Process grading submodule entity. As Process grading submodule is to Grading submodule, Process grading parameter is to Grading parameter.

Grading submodules and Grading parameters can persist and be used in any grading process associated to a Vpl activity. Process grading submodules and Process grading parameters only make sense inside the context of a grading process.

4.2.2 An abstract view of the grading process management module

The grading process management module is considered to be inside VPL tool, which is a Moodle's activity plugin. So the new module will be related with VPL and with Moodle. Additionally, the new module will communicate with the VPL-Jail subsystem to send information and data about the managed grading-submodules.

Figure 21 shows an abstract architecture of the grading module. This architecture aims to show the influence of Moodle and VPL on the grading process management module.

It is necessary to make an explanation about every layer of the architecture. Then:

- The VPL layer refers to information and data, which could be used by the grading module. This information and data could include VPL activity's identification (every VPL activity must have a grading process associated), access permissions to the VPL's management (the grading module will belong to VPL tool, so there will be some common permissions), useful modules (the jail module inside VPL can be reused to the communication between the new module and the VPL-Jail subsystem), and so on.

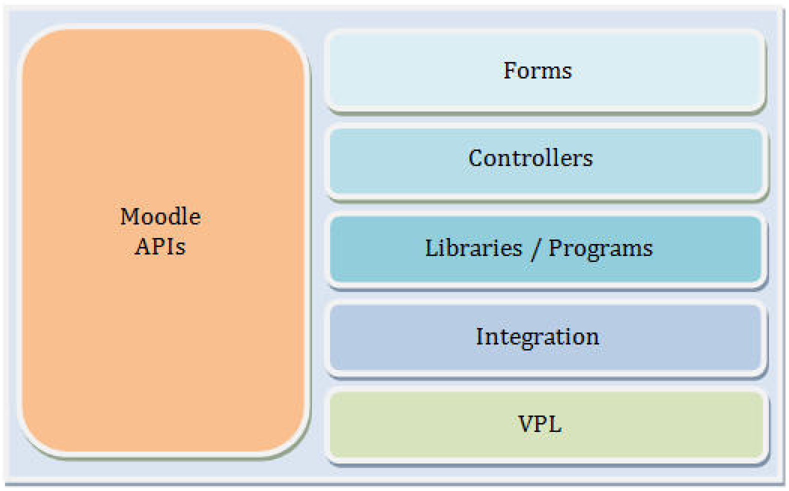


Figure 21. Architecture for the grading process management

- The integration layer is the bridge to receive information from VPL and to provide information to the grading module. This layer makes possible more maintainability in the solution. Because in possible updates of VPL plugin, the changes should be done here. This layer could be represented in a configuration file where the parameters needed by grading module are set with VPL information.
- The libraries and programs layer includes a set of software components to provide of new functionalities to the grading module. These useful functions will help in the implementation of the top layers and will allow the communication with the VPL-Jail subsystem.
- The two top layers aims to: prepare data and information, send it to the user forms and show the user forms. They are defined in a way to maintain compatibility with the Moodle structure.
- The transversal layer refers to a set of Moodle provided APIs [39]. The use of them can help to make quick implementations and to guarantee a complete compatibility with Moodle; but in the other hand, following these guidelines can imply coming into some issues from Moodle. These issues include: using

[39]http://docs.moodle.org/dev/Core_APIs

a combination of structured programming and object oriented programming, and the lack of a mean to separate the logic, the data, and the visual interface.

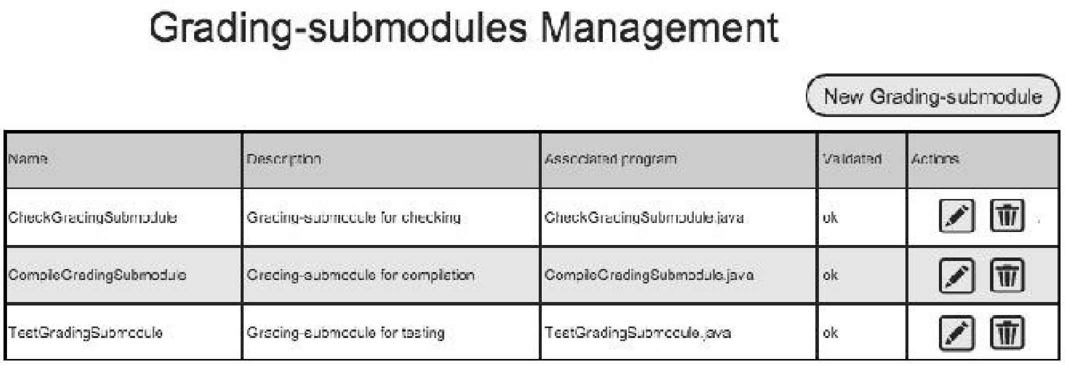
The architecture aims to be as independent as possible of VPL, but guaranteeing its integration. Then, it is possible to think later in an own plugin or in be integrated with another plugins. As it can be seen it is dependent of Moodle because it bases on its API.

4.2.3 User interfaces

Using the Moodle APIs, the user interfaces will be built following a common style. Next, some sketches for the management features are shown.

Management of grading-submodules

This management is focused in allowing CRUD actions for grading-submodules. The Moodle's user management is a good choice to have a reference to start the implementation. The code behind the user's management will be useful to know how to get the desired behavior. Regarding the view, Figure 22 shows the prototype of user interface for the main management page.



Name	Description	Associated program	Validated	Actions
CheckGradingSubmodule	Grading-submodule for checking	CheckGradingSubmodule.java	ok	
CompileGradingSubmodule	Grading-submodule for completion	CompileGradingSubmodule.java	ok	
TestGradingSubmodule	Grading-submodule for testing	TestGradingSubmodule.java	ok	

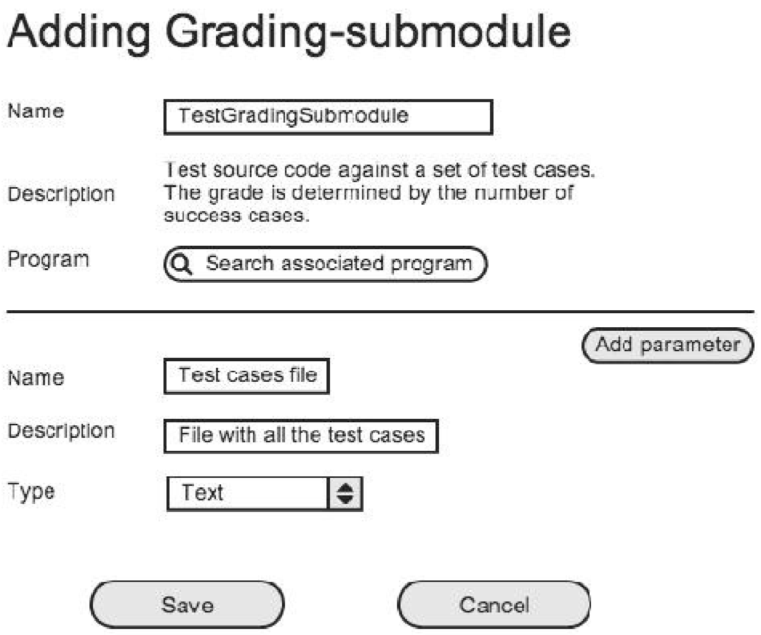
Figure 22. View of grading-submodules management

This user interface shows a list of the existent grading-submodules. This view allows going to add, edit or delete a grading-submodule. It is necessary another view, which allows editing or adding a new grading-submodule. It is shown in Figure 23.

Management of the grading process

This management shows firstly the view of the complete grading process. This will allow setting the order of grading-submodules performance inside the grading process, and the deletion of any of them. It is shown in Figure 24.

Additionally, it is necessary a view that allows: adding new grading-submodules inside the grading process, setting the factor (weighting) of every grading-submodule to calculate the grade, and setting the values for all parameters required by grading-submodules inside the grading process. Figure 25 shows the possible view.

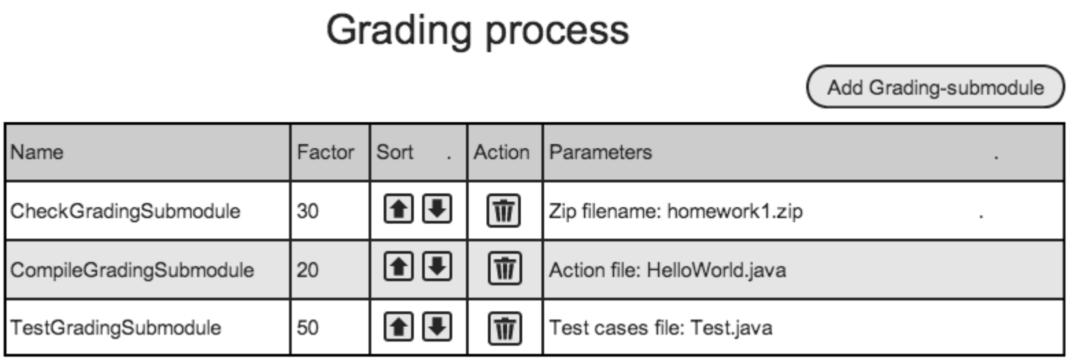


Adding Grading-submodule

Name	TestGradingSubmodule
Description	Test source code against a set of test cases. The grade is determined by the number of success cases.
Program	Search associated program

Name	Test cases file
Description	File with all the test cases
Type	Text

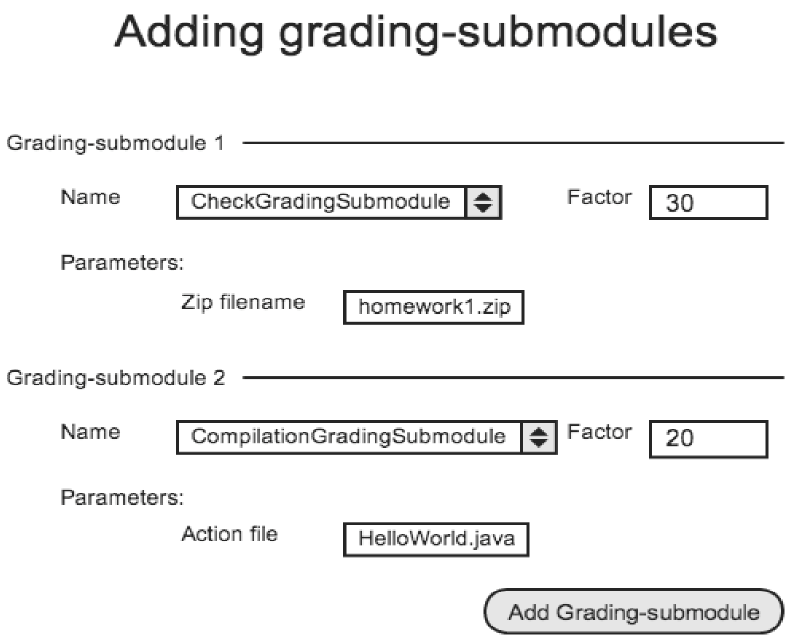
Figure 23. View of grading-submodules creation / addition



Grading process

Name	Factor	Sort	Action	Parameters
CheckGradingSubmodule	30	⬆ ⬇	🗑	Zip filename: homework1.zip
CompileGradingSubmodule	20	⬆ ⬇	🗑	Action file: HelloWorld.java
TestGradingSubmodule	50	⬆ ⬇	🗑	Test cases file: Test.java

Figure 24. General view of grading process



Adding grading-submodules

Grading-submodule 1

Name	CheckGradingSubmodule	Factor	30

Parameters:

Zip filename	homework1.zip

Grading-submodule 2

Name	CompilationGradingSubmodule	Factor	20

Parameters:

Action file	HelloWorld.java

Figure 25. View for configuration of grading-submodules

4.3 Subsystems communication

Additionally to the design of both subsystems, it is necessary to consider the communication between them. It will be done using the same technology as VPL. It means using the server and service already implemented (refer to the description of VPL in the state of the art chapter).

As first step, it is necessary to identify when the communication's establishment will be required.

4.3.1 Grading-submodules management

When a new grading-submodule is created, it is necessary to know if the associated program is correct. So a compilation and a possible addition to a jar library in the VPL-Jail subsystem will be required. Additionally, when the associated program is updated and when a grading-submodule is deleted, a new processing will be necessary in the libraries of the VPL-Jail subsystem. It can be seen in Figure 26.

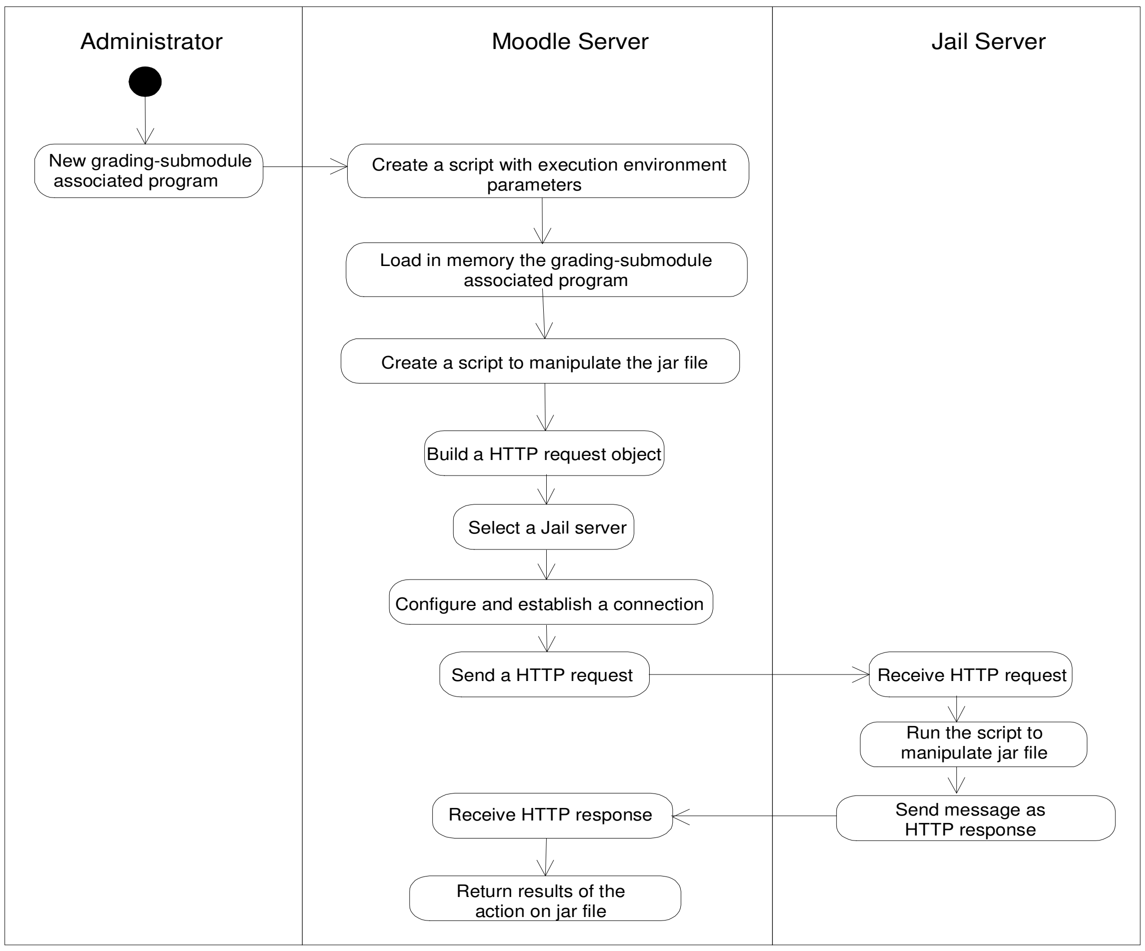

Administrator
Moodle Server
Jail Server
New grading-submodule associated program
Create a script with execution environment parameters
Load in memory the grading-submodule associated program
Create a script to manipulate the jar file
Build a HTTP request object
Select a Jail server
Configure and establish a connection
Send a HTTP request
Receive HTTP request
Run the script to manipulate jar file
Send message as HTTP response
Receive HTTP response
Return results of the action on jar file

Figure 26. Communication between subsystems in the grading-submodules management

As the grading module is new, the implementation of new methods for sending this data will be required. Additionally, these methods have to send new types of request to the Jail server.

In the VPL-Jail subsystem, it will be necessary adding the interpretation for new kind of request coming from the VPL-Moodle subsystem.

4.3.2 Students' submission

The process to send is already implemented as well as the interpretation of the associated request (the process can be seen in Figure 4) but it is necessary to modify the method that prepares the data to be sent to the VPL-Jail subsystem. Now, the set of files has to include the additional files required by every grading-submodule associated program (defined inside the grading process), and the *configuration file* that has the information of the whole grading process.

4.4 Chapter summary

This chapter has focused on the architecture design, which allow going deep in the understanding of the solution proposed, and necessary before the implementation stage. The analysis chapter showed the need of adding new features to the VPL-Moodle and the VPL-Jail subsystems. Each of them has been described separately to apply different design perspectives.

The VPL-Jail subsystem provides a sandboxed environment where the grading process takes place. The architecture is formed by a set of layers with their roles well defined. It is worth highlighting the orchestrator (including the configuration file) and the grading-submodules layers. The Orchestrator takes control of the whole process based on information provided by the configuration file. Additionally the grading-submodules layer provides modularity, extensibility and flexibility to the grading process. Taking advantage of the objects-oriented paradigm, the grading process has been modeled with a set of different classes, which have been described including their attributes, operations and relations. Additionally, the grading process has been described through interactions among instances of the defined classes (Orchestrator, AnygradingSubmoduleProgram, and CommandExecutor).

The VPL-Moodle subsystem is in charge of the management of the grading process. This management requires working with data stored in the Moodle's database and in the Moodle's data directory. Then, a data model and a directory structure for the grading module have been defined. An abstract architecture to show the influence of VPL and Moodle on the grading module has been provided. Additionally, the management user interfaces have been defined through the drawing of some sketches.

All the classes designed and processes explained are helpful to go on with the implementation stage. It is expected that the time and troubles during implementation stage will be reduced.

5 Implementation and Validation

The design chapter had provided of useful classes and elements for the solution's implementation. This chapter shows important considerations and features during the implementation of every element for a proof-of-concept prototype.

The implementation stage considers the two identified subsystems (VPL-Moodle and VPL-Jail). Each of them uses different programming languages for its implementation, and requires a set of different elements for a right working. The implementation of the communication between the subsystems is considered as well.

The architecture's validation is done in two ways. Firstly, it is demonstrated the workability of the proposed architecture. Secondly the new module is validated through two case-studies.

5.1 VPL-Jail subsystem implementation

The VPL-Jail subsystem hosts the jail environment, where the grading process is carried out. The implementation of the architecture's elements will be done from scratch, which is an advantage because it is not limited by a given technology.

To allow the integration with VPL tool, modifications on some existent program files will be done and will be explained as well.

5.1.1 Programming languages

Considering the architecture provided in the section referred to VPL-Jail subsystem in the design chapter (Figure 11), orchestrator and grading-submodules layers contain the main elements to be implemented. These elements have been modeled and depicted in a class diagram in the design chapter (Figure 15). Then, the programming language to implement them has to support objects oriented paradigm.

Java has been selected as the programming language to implement the two layers of the architecture. This programming language has a set of features[40] but in this case, its object-orientation and its portability are the most valued features. Additionally, there are a lot of external libraries already implemented, which could help to save implementation time.

VPL's elements in the VPL-Jail subsystem have been coded using two programming languages. The first one is C++, which is used to code the Jail server program and for default programs when using the default grading process. The other language is Linux

[40] http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

shell scripting, which is used to code the evaluation script and will be used to code the execution file as a script.

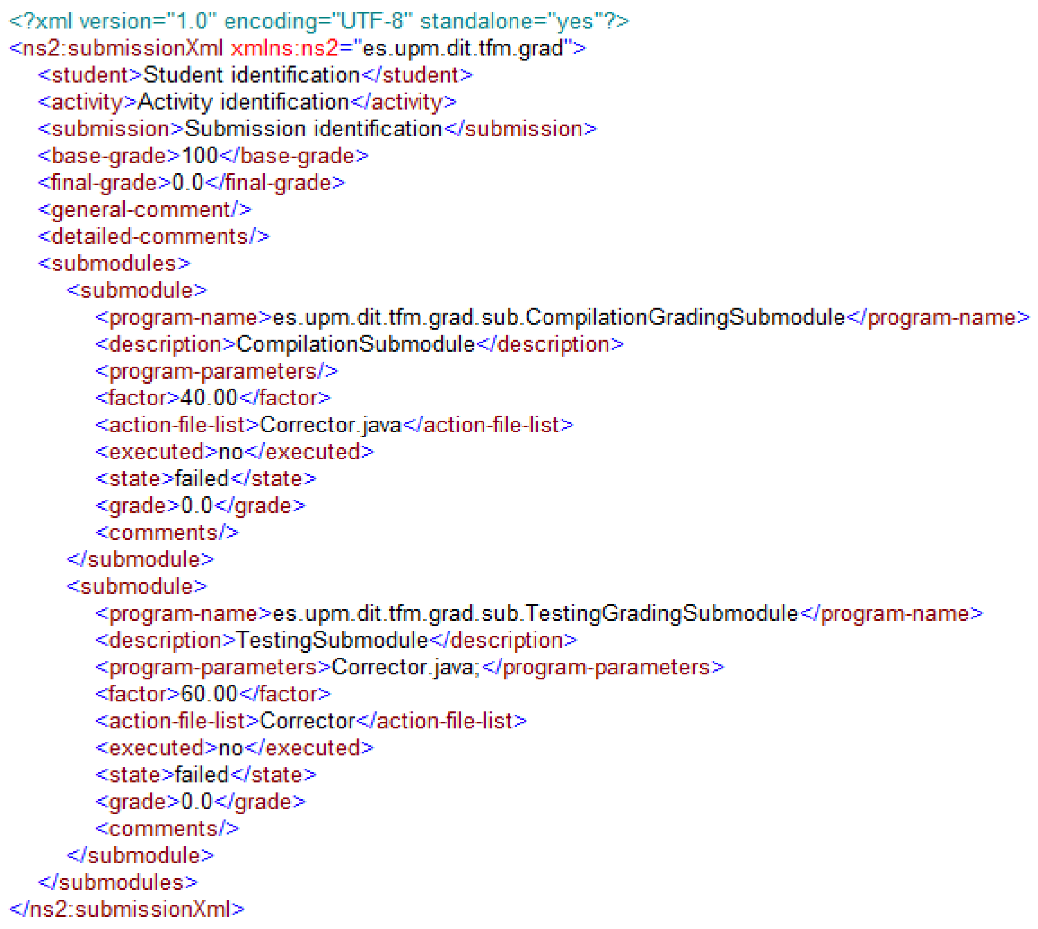
Summarizing, Java will be used to code the new elements (as classes), and C++ and Linux shell scripting will be used to modify existent elements in VPL (necessary to makes an integration).

5.1.2 Configuration file

Before starting to code the new classes, it is necessary to establish the structure and format of the configuration file.

XML has been selected as format to write the configuration file because it has a set of interesting advantages including: the ability to exchange and store data[41] in any kind of applications, the ease of writing XML documents, the ability to be human and machine legible, the ease of writing programs to interpret XML and so on[42].

The structure of the configuration file is highly related to the parser classes defined in the design chapter. The structure is shown in Figure 27.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:submissionXml xmlns:ns2="es.upm.dit.tfm.grad">
   <student>Student identification</student>
   <activity>Activity identification</activity>
   <submission>Submission identification</submission>
   <base-grade>100</base-grade>
   <final-grade>0.0</final-grade>
   <general-comment/>
   <detailed-comments/>
   <submodules>
      <submodule>
         <program-name>es.upm.dit.tfm.grad.sub.CompilationGradingSubmodule</program-name>
         <description>CompilationSubmodule</description>
         <program-parameters/>
         <factor>40.00</factor>
         <action-file-list>Corrector.java</action-file-list>
         <executed>no</executed>
         <state>failed</state>
         <grade>0.0</grade>
         <comments/>
      </submodule>
      <submodule>
         <program-name>es.upm.dit.tfm.grad.sub.TestingGradingSubmodule</program-name>
         <description>TestingSubmodule</description>
         <program-parameters>Corrector.java;</program-parameters>
         <factor>60.00</factor>
         <action-file-list>Corrector</action-file-list>
         <executed>no</executed>
         <state>failed</state>
         <grade>0.0</grade>
         <comments/>
      </submodule>
   </submodules>
</ns2:submissionXml>
```

Figure 27. Structure of the XML configuration file

[41] http://www.w3schools.com/xml/xml_whatis.asp
[42] http://www.w3.org/TR/REC-xml/

This file stores information about the submission and about every grading-submodule that will be considered inside the grading process. The complete submission information will be used by the orchestrator and the information related to each grading-submodule will be used by the grading-submodule associated program.

The information fields related to a submission includes:

- Student, it has information to identify the student. This information can be the name or an id for instance.
- Activity, it has information to identify the activity. It can be the VPL activity's id.
- Submission, it has the submission number or the submission identification.
- Base grade, it is the base over which the final grade will be calculated.
- Final grade, it is the grade for the current submission.
- General comment, it stores a short comment for the submission.
- Detailed comments, it stores the comments of every grading-submodule.

The information fields about every grading-submodule include:

- Program name, it contains the full name of the grading-submodule associated program (including the package). The .class extension is not included.
- Description, it contains a short description for the current submodule. It has to express the main action that the submodule will do.
- Program parameters, it has additional data required by the associated program. It is a string, which includes parameters' values separated by a semicolon and without blank spaces. The parameters can be pathnames, numbers, and so on. If one parameter has many values, they should be separated by commas.
- Factor, a percentage which represents the submodule weight in the final grade calculation. The addition of this field in all grading-submodules has to be 100.
- Action file list, it has a list of filenames over which the main action of the submodule will be executed. The list will be composed of full names (including the package name) or relative names (just the filename) and the file extension depending on every submodule.
- Executed, it shows if the submodule has been executed; it is independent of the success or failed execution.

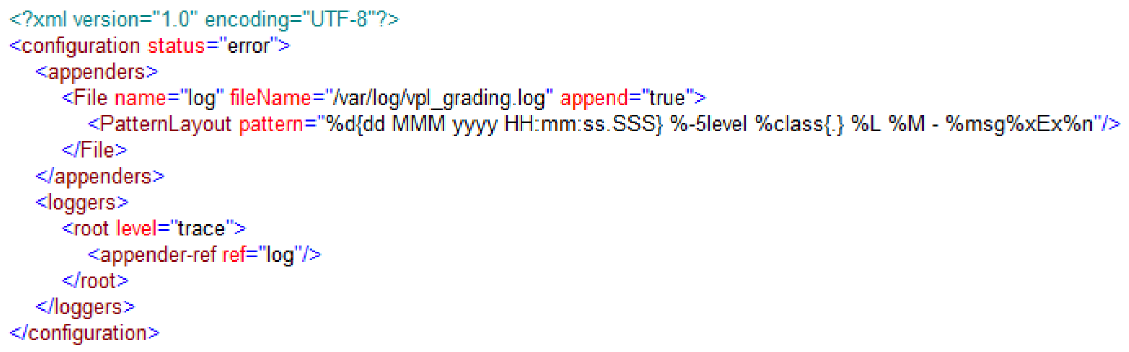
- State, it indicates if the submodule execution finished perfectly (*success*), getting a full grade; or if there were some troubles *(failed)* and a partial grade was obtained.
- Grade, it is the grade for the current grading-submodule. It is a numeric value between 0.00 and 100.00 with 2 decimal places. There always has to be a value in this field since its creation.
- Comments, detailed information about the execution of the grading-submodule associated program.

5.1.3 Logging

Considering that Java has been selected as programming language to code, it is possible to use some already built tools for logging. One of them is Log4J 2[43], which has been selected due to its important features including its ease of installation, and its automatic configuration ability.

The possible implementation was changed by performing a set of few steps, which include: downloading the core and API jar files, setting the logging through defining an XML configuration file, setting the references to the jar and configuration files in the *Java Classpath*, importing the logging classes inside the interested classes, and defining a static logger variable in the same new classes.

Figure 28 shows the configuration file used by the logger in the grading process. This file has to be named as *log4j2.xml* to use the automatic configuration.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="error">
   <appenders>
      <File name="log" fileName="/var/log/vpl_grading.log" append="true">
         <PatternLayout pattern="%d{dd MMM yyyy HH:mm:ss.SSS} %-5level %class{.} %L %M - %msg%xEx%n"/>
      </File>
   </appenders>
   <loggers>
      <root level="trace">
         <appender-ref ref="log"/>
      </root>
   </loggers>
</configuration>
```

Figure 28. XML configuration file for the logger in the grading module

This configuration file defines that the logs will be written in a file named *vpl_grading.log* in the directory */var/log/*. The layout is defined through a pattern and will register events since the TRACE type.

The instruction to define the static variable is:

```java
static Logger logger = LogManager.getLogger(MyClass.class.getName());
```

[43] http://logging.apache.org/log4j/2.x/

This variable references an instance of Logger named “MyClass”.

When working with this Log4J 2, defining log levels is possible; it allows classifying logs by severity and for having granularity. Every log level has been associated to a specific type of information. Thus:

- *Trace*. For information about method enter/exit.
- *Debug*. For information about important system commands executed.
- *Info*. For information send to the student. In this case information about the grades.
- *Error*. For information about exceptions thrown by a given method. It is possible to continue the process.
- *Fatal*. For information related to an action that stops the grading process. In this case when the XML file is not loaded.

5.1.4 Application’s core

The application’s core joins all the classes that won’t change after the implementation. It was mentioned in the class packages section in the design chapter as well. All this classes will be deployed as a single jar file.

The parsing package

Class SubmissionConf [es.upm.dit.tfm.grad.pars]

```
public class SubmissionConf
extends java.lang.Object
```

This class represents the submission's information mapped from the XML configuration file.

Class GradingSubmoduleConf [es.upm.dit.tfm.grad.pars]

```
public class GradingSubmoduleConf
extends java.lang.Object
```

This class represents the grading-submodule’s information mapped from the XML configuration file.

These both classes represent the XML grading configuration file. JAXB[44] (Java Architecture for XML Binding) is the technology used to map the data from an XML representation to an object representation. It helps to improve the developers’ performance because it eliminates the need of writing parsers.

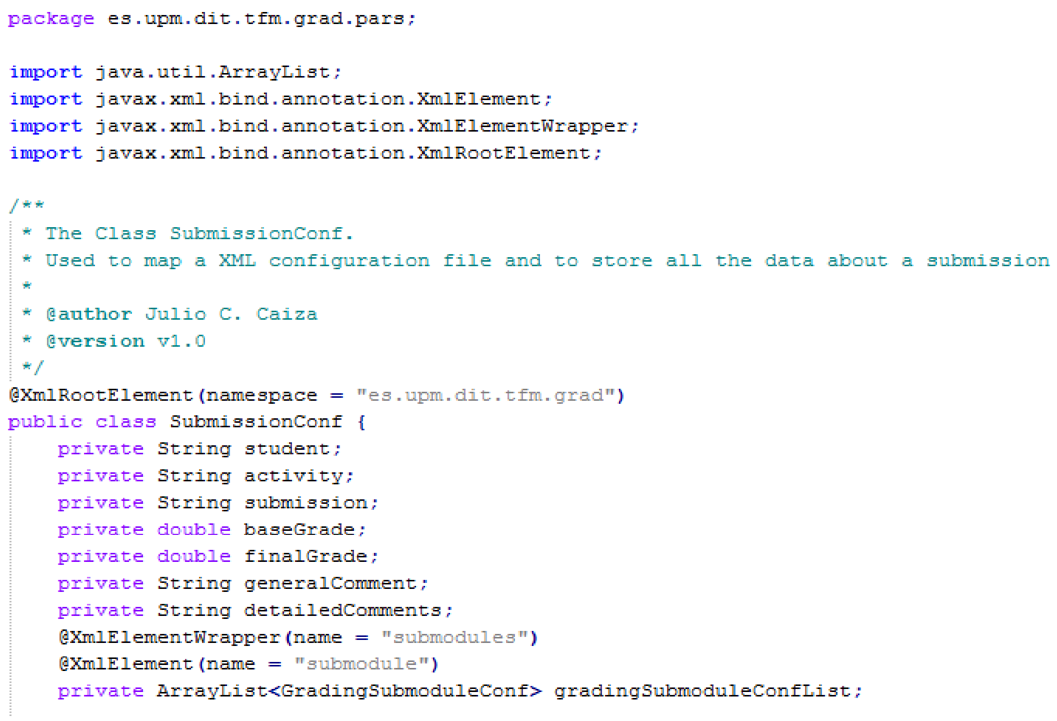
This quick mapping is done through the use of annotations in the Java class. Figure 29 shows a piece of annotated code.

[44] http://jaxb.java.net

The explanations of the annotations are:

- *@XmlRootElemen(namespace = "Namespace name")* allows defining the root element in the XML tree.
- *@XmlElement(name = "xml element name")* allows naming the XML element when its name is different of the attribute's name.
- *@XmlElementWrapper(name = "wrapper name")* allows defining a wrapper for a set of XML elements. It is used to join the grading-submodule elements.
- *@XmlType(propOrder = { "attribute 1", .., "attribute n"})* allows setting the order of the elements in the XML document.

These annotations have been used inside the definition of both classes but there are more of them. It is suggested go to the official page for further study.



```
package es.upm.dit.tfm.grad.pars;

import java.util.ArrayList;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * The Class SubmissionConf.
 * Used to map a XML configuration file and to store all the data about a submission
 *
 * @author Julio C. Caiza
 * @version v1.0
 */
@XmlRootElement(namespace = "es.upm.dit.tfm.grad")
public class SubmissionConf {
    private String student;
    private String activity;
    private String submission;
    private double baseGrade;
    private double finalGrade;
    private String generalComment;
    private String detailedComments;
    @XmlElementWrapper(name = "submodules")
    @XmlElement(name = "submodule")
    private ArrayList<GradingSubmoduleConf> gradingSubmoduleConfList;
```

Figure 29. Piece of annotated code to map the XML configuration file

The orchestration package

Class Orchestrator [es.upm.dit.tfm.grad.orch]

```
public class Orchestrator
extends java.lang.Object
```

This class controls the whole grading process. It has as attributes the name of the configuration file, an instance of *SubmissionConf* class and a static variable for logging. It has to load the XML file configuration data and then it will start to orchestrate the process. The orchestration makes a loop in the list of *GradingSubmoduleConf* inside the *SubmissionConf* instance. In each iteration, the orchestrator instantiates every grading-submodule associated program and it will invoke the run method inside the instance.

Finally, the orchestrator will process every submodule results to calculate the final grade for the current submission, to collect detailed comments, and to define a general comment.

It is worth seeing how to create a new instance of this class and its set of important methods. They are detailed in the annexes.

The system package

Class CommandExecutor [es.upm.dit.tfm.grad.system]

```
public class CommandExecutor
extends java.lang.Object
```

This class allows executing commands in the system console. It acts as a wrapper for the *SystemCommandexecutor* class, which is provided by Alvin J. Alexander through his site[45]. After the execution of a system command, the standard output and standard error are caught and saved inside this class as well.

The constructor and important methods are shown in the annexes.

5.1.5 Application's submodules

It makes reference to the existence of another jar file additional to the core. It is not just an API because it allows adding new programs, which will evaluate a source code considering a grading-criterion.

Initially, this jar contains only one abstract class, which acts as intermediary to add new classes to evaluate the source code. This abstract class forces subclasses to implement the *run()* method and provides them of some useful methods.

The submodules package

Class GradingSubmoduleProgram [es.upm.dit.tfm.grad.sub]

```
public abstract class GradingSubmoduleProgram
extends java.lang.Object
```

This class is the base for new grading-submodule associated programs, which will be built by administrators or teaching-staff. It has as attribute a *GradingSubmoduleConf* instance, a *commandExecutor* instance and logging static variable for logging. The first one will have parameters to be used and updated during the execution of the subclasses associated programs. The second one will execute one line commands in the system console. In addition it provides of methods to make easier the implementation of new subclasses.

The constructor and important methods are shown in the annexes.

[45] http://alvinalexander.com/java/java-exec-processbuilder-process-1

AnyGradingSubmoduleProgram class [es.upm.dit.tfm.grad.sub]

It is possible to establish a frame to create any grading-submodule associated program. The new class has to:

- Belong to the submodules package.
- Extend the *GradingSubmoduleProgram* class.
- Define a constructor as the *GradingSubmoduleProgram* class.
- Implement the *run()* method.

Additionally, a logging static variable can be added to record logs. To complete the frame, it is possible to define a structure for the run method. Figure 30 shows the structure identifying some key sections of code.

```java
@Override
public void run() {
    logger.entry();
    //[MANDATORY]Variables to calculate and update later
    String state="failed", comments = "";
    double grade = 0.0;

    //List of parameters
    String[] programParameters;
    programParameters = this.getGradingSubmoduleConf().getProgramParameters().split(";");
    String firstParameter = programParameters[0];
    String nParameter = programParameters[n];

    //List of files to execute the action.
    String[] actionFileList;
    actionFileList = this.getGradingSubmoduleConf().getActionFileList().split(",");

    // == Block to execute any instruction to evaluate the code ==//
    String command = "Any command to execute in the system console";
    String stdout = "", stderr = "";//Strings to catch the standard output and error
    this.executeCommand(command);
    stdout = this.stdOutString();
    stderr = this.stdErrString();
    // === End of block === //

    // The values to update in the GradingSubmoduleConf instance.
    grade = 100.0;
    comments += "Any comment about the evaluation";
    state = "success or failed";

    //[MANDATORY] Update the associated submodule in process
    this.updateGradingSubmoduleConf(state, grade, comments);
    logger.exit();
}
```

Figure 30. Structure for the *run()* method

The mandatory sections refer to initialize and set the state, comments and grade. Precisely these three indicators will be considered to set the final grade and comment.

The lists of parameters and files sections are oriented to get data from the *GradingSubmoduleConf* instance. This data will be used in the evaluation section.

The evaluation section can have any instruction, which helps to evaluate the code considering a given criterion. It can include calls to execute system commands, reading of the standard output and error, calculations, recording of logs, and so on.

Finally, the state, comments and grade have to be updated in the *GradingSubmoduleConf* instance. This information will be taken by the orchestrator to build the feedback.

5.1.6 VPL's integration

The top layer of the architecture showed in the section referred to VPL-Jail subsystem in the design chapter (Figure 11) allows integration with VPL. There are two files in this level, the evaluation and the execution files. The evaluation file makes any needed preprocessing on the source code files and generates automatically the evaluation file. Both of them have been implemented as Linux scripts.

The evaluation file has been built to decode base64-encoded files (using *base64* shell script function), and to change the charset to UTF8 (UCS Transformation Format -8bit), using *iconv* shell script function. It is worth highlighting that VPL does not support submissions that include binary files[46], so the implementation of support for this kind of files using base64 encoding has been necessary.

The execution file makes a loop inside the lib directory and includes all the jar files in the CLASSPATH environment variable. After that, the absolute path to the logging configuration file is included in the CLASSPATH as well. Finally, it makes a call to the orchestrator program, using the *java* command in the system console.

5.1.7 The lib directory

The architecture considers a layer for libraries and ancillary programs. These libraries and programs have to be stored in a location inside the Jail server. The root directory to save these files has been defined in */usr/share/vpl.* This directory is copied inside the jail environment before to start the grading process, so that the libraries and programs required for the grading process will be available.

Inside the VPL root directory two more directories have been created. One of them called *conf,* which stores the logging configuration file, and another called *libs* that stores jar files of libraries and ancillary programs.

5.2 VPL-Moodle subsystem implementation

This subsystem provides of management features. It can be seen as the solution's front-end. The management includes the implementation of a user interface, which

[46] https://moodle.org/mod/forum/discuss.php?d=154988&parent=939991

allows carrying out CRUD actions on grading-submodules, and the implementation of a user interface to configure a grading process associated to a VPL activity.

As the grading process proposed here is based on the use of a new artifact (grading-submodule), it is not enough with the modification of some user interfaces already built in VPL. In fact, the new management interfaces have to be implemented from scratch.

This section shows some important considerations made during the implementation of the grading module and how the integration with VPL is carried out. It is necessary to consider the diagram showed in the section referred to VPL-Moodle subsystem in the design chapter (Figure 21).

5.2.1 Programming languages

The grading module is considered to be inside VPL tool, which is a Moodle plugin. So to guarantee the new module's integration it is better to use the same programming languages and API used in Moodle.

Then, the programming language used for the implementation is PHP. Moodle's APIs have been used to take advantage of them and to guarantee the integration.

5.2.2 Configuration file

This file defines some parameters, and its values, which will be used by other new PHP programs. Some of the values are set based on values of VPL's parameters. So it is clear its role as a bridge between VPL and the grading module.

At the moment, the configuration file defines the next parameters:

- The capability to manage the grading-submodules. It is based on the capability to manage VPL configuration.
- The capability to configure a grading process. A grading process is always associated to a VPL activity.
- The root directory for grading module data in the directory structure.
- The directory to save the grading-submodule associated program.
- The directory to save files associated to a given VPL activity. These files will be used inside the grading process.
- Parameters to be used in the communication between the Moodle and Jail subsystems. They include: the lib directory in the VPL-Jail subsystem, the namespace for the submodules jar, the name of the submodules jar, the name of the script that will manipulate the grading-submodules associated programs in the VPL-Jail subsystem, the name of the request method to be

sent to the Jail server, and the name of the configuration file for the grading process.

5.2.3 Lib file

This file stores a set of useful functions, which will be used in other PHP files inside the grading module. These functions are oriented to:

- Set parameters to establish layouts for the web forms.
- Make a complete creation or deletion processes when it is necessary to operate with data from the system directory and from the database.
- Operate with files inside the system directory.
- Change the data charset.
- Get lists of registers from the database. They are based on Moodle database API.
- Prepare data before to be sent to the Jail server.

5.2.4 User interfaces

The implementation of the user interfaces follows the structure provided in Moodle's official documentation to create forms[47]. Then, to create a user interface is necessary to code two PHP files.

The first one acts as a kind of controller (it just orders processes) which makes some processes including: setting up the page layout, preparation of data to be sent to the form, instantiation of a form to show, deploying of the web form, treatment of actions sent by the form, and treatment of data sent in the form.

The second one represents the form to be displayed and is implemented as a class which extends from the *moodleform* class. It sets all the HTML elements that the form will contain and will display. It is possible through the use of the *formslib* Moodle API[48].

There have been implemented 3 typical user interfaces which follow the explained way. They were implemented to allow:

- Creating and editing grading-submodules.
- Adding new grading-submodules inside the grading process.
- Setting the parameters of the grading-submodules selected for the grading process.

[47] http://docs.moodle.org/dev/Form_API
[48] http://docs.moodle.org/dev/lib/formslib.php_Form_Definition

Additionally two user interfaces have been implemented and oriented to show an information's summary. They are:

- The list of existent grading-submodules. It is useful to go to perform CRUD actions.
- The list of the grading-submodules inside a grading process. It allows seeing a summary of the grading process with its grading-submodules and the parameters' values. Additionally it allows deleting and sorting the grading-submodules in the grading process.

In total, 13 new PHP source code files have been implemented.

It is worth mentioning the use of the functionality to repeat elements, which was used to work with any number of parameters. This functionality allows repeating elements in the user form as the user wants. For further information it is suggested go to the official documentation[49].

5.2.5 VPL's integration

To get integration with VPL, the creation of a new PHP file and the modification of some already built files inside VPL have been needed.

The configuration file has been built to be the integration layer of the architecture shown in the section referred to VPL-Moodle subsystem in the design chapter (Figure 21). Some of the parameters specified there take values from VPL's variables (These parameters are described in the configuration file's description done previously in this chapter).

There are a set of modified VPL's files. Initially the goal was trying not to alter the existent files but it could not be avoided. The altered files include:

- */mod/vpl/db/access.php* to define capabilities in the system.
- */mod/vpl/locallib.php* to define global variables, which will allow accessing to the capabilities defined previously.
- */mod/vpl/vpl_submission_CE.class* to alter the data to be sent in a submission. It includes the automatic generation of the configuration file for the grading process in the VPL-Jail subsystem.
- */mod/vpl/lang/en/vpl.php* to define the messages to be displayed in the user interfaces.
- */mod/vpl/lib.php* to add new nodes into the Moodle's navigation tree.
- */mod/vpl/settings.php* to add an administration link in the VPL's administration.

[49] http://docs.moodle.org/dev/lib/formslib.php_repeat_elements

- */lib/adminlib.php* to define the element administration link.

5.3 Subsystems communication

After VPL-Jail and VPL-Moodle subsystems' implementation it is necessary to establish a mean to communicate those considering the changes made on both subsystems.

5.3.1 Code reusability

The main idea of using VPL as tool base was to reuse its technology already implemented to save implementation time. Then, the communication technology has been maintained. It includes:

- Maintaining the XML-RPC protocol over HTTP to transport the requests and responses.
- Using the jail communication module. This is inside the VPL-Moodle subsystem.
- Using the Jail server program. This is in the VPL-Jail subsystem.

Even though the reusability has been applied, the implementation of new programs was necessary as well as the modification of some existent files. They are explained in the next sections.

5.3.2 VPL-Moodle subsystem

jailconnection.php file

This file belongs to the grading module inside VPL plugin. It contains the *jailconnection* class that defines methods to make a request and send data to a Jail server which uses XML-RPC protocol. It has as attributes the data to send, the request method to ask in the server, a server instance and an attribute to save the server's response.

This class is based on classes provided by the jail module inside VPL Moodle's plugin. But it could be possible to use other classes which provide of a server's selection, the ability to build HTTP packages, and support the use of XML-RPC protocol.

vpl_submission_CE.class file

This file defines a class to send a submission to the VPL-Jail subsystem. It has methods to prepare the data and to send that, using the jail module. This file has been altered to support collecting and sending files defined as parameters inside the grading process, and to build and send the XML configuration file used by the grading process.

As key point it is worth mentioning the modification done to support the sending of binary files to the VPL-Jail subsystem because it was not implemented in the version 2 of VPL tool. It was done through the use of base64 encoding before sending the data.

5.3.3 VPL-Jail subsystem

jail.cpp file

This file contains the *Jail* class which acts as the server program inside the VPL-Jail subsystem. This receives the requests sent by the VPL-Moodle subsystem, identifies the request associated method, decodes the information received, executes a given process depending on the recognized method, prepares an HTTP response and sends it to the VPL-Moodle subsystem. The basic process can be seen in Figure 4.

This program has been modified to interpret a new request method. This new method is to execute actions when the grading-submodule associated program is added, modified or deleted.

Additionally, this file uses the file *jail.h*. This file has been modified to add the definitions of functions used in the Jail class.

Finally, it is worth mentioning that when binary data (base64-encoded) arrives from the VPL-Moodle subsystem; this server is limited to only copying it inside the jail environment. The decoding of this kind of files is being carried out in the evaluation script inside the grading process. It was a temporal solution but susceptible of being improved.

5.4 Validation

To validate the proposed architecture and its implementation, two case-studies have been considered. They are based on real programming assignments proposed to students at ETSIT. They are going to be configured in the tool from scratch.

5.4.1 Case study 1

The assignment asks the student to code a new Java class called *ReceptorGPS* and a set of methods, which follow a given signature. The complete description of the assignment (in Spanish) can be seen in the annexes section.

There is a set of Java classes (source files) provided by the teaching staff, which has to be used by the student during its assignment implementation. This set includes classes *CoordenadaEsferica*, *CoordenadaCartesiana* and *SateliteGPS*. They are already built and do not need changes.

All the source files have to belong to the package *es.upm.dit.prog.p3* and be compressed in a zip file, which has to be called '*practica3.zip*'. This is the only file to be uploaded as a student's submission.

Grading criteria

The grading criteria include:

1. Checking the directory structure of the source files.
2. Compiling all the source files.
3. Testing the new class and its methods against a set of test cases defined in an additional file.

The test cases file has to be built by the teaching staff but it is not provided to students.

Required grading-submodules

Considering the grading criteria previously given it is necessary to build three grading-submodules. Each of them will be associated to one criterion. They are:

1. A grading-submodule to unpack the zip file inside the jail and to check whether the source code files maintain the directory structure.
2. A grading-submodule to compile the set of Java programs.
3. A grading-submodule to test the Java program against a set of test cases.

Grading-submodule associated programs

It is necessary to implement the grading-submodule associated programs. They will be Java classes, which will extend from *GradingSubmoduleProgram* abstract class.

Class CheckGradingSubmodule [es.upm.dit.tfm.grad.sub]

```
public class CheckGradingSubmodule
extends GradingSubmoduleProgram
```

The main action of this program is checking the directory structure inside a zip file. It means to check the existence of a set of source code files inside a directory structure. The data required from the *GradingSubmoduleConf* instance includes the name of the zip file and the list of files to check their existence. The name of the zip file is obtained with the *getProgramParameters()* method in the *GradingSubmoduleConf* instance. The list of the source code filenames is obtained as a String with the *getActionFileList()* method. These names are separated by commas. Then it is necessary to split the list using the commas as marks. To get a success evaluation, every source code file has to exist. This is done with the execution of a system command. If there is at least one file not found the status will be set as failed and the grade will be set with zero.

Class CompilationGradingSubmodule [es.upm.dit.tfm.grad.sub]

`public class` **`CompilationGradingSubmodule`**
extends `GradingSubmoduleProgram`

The main action of this program is compiling a set of Java source files. The data required from the *GradingSubmoduleConf* instance is the list of files to compile. The list of the source code filenames is obtained as a String with the *getActionFileList()* method. These names are separated by commas. To get just one command for executing, it is necessary to replace the commas by blank spaces. To get a success evaluation, there has to be generated one *.class* file for every *.java* file specified in the list returned before. The existence of every *.class* file is checked with the execution of a system command. If there is at least one file not generated, the status will be set as failed and the grade will be zero.

Class TestingGradingSubmodule [es.upm.dit.tfm.grad.sub]

`public class` **`TestingGradingSubmodule`**
extends `GradingSubmoduleProgram`

The main action of this program is testing a set of Java source files. It uses just one class file (tester file) for the testing process. It can be obtained with the *getActionFileList()* method in the *GradingSubmoduleConf* instance. The test process is done with the execution of one system command. The results of the execution are saved in a text file. This text file has in its first line the total tests number, in the second line the failed tests number and in the additional lines the description of the failed tests. All of them are saved in different variables. The grade is calculated taking into account the total tests number and the failed tests number. To get a success submodule, there has not to be a failed test. If there is at least one failed test, the state will be set as failed but the grade will be set as it was calculated.

Grading-submodules management

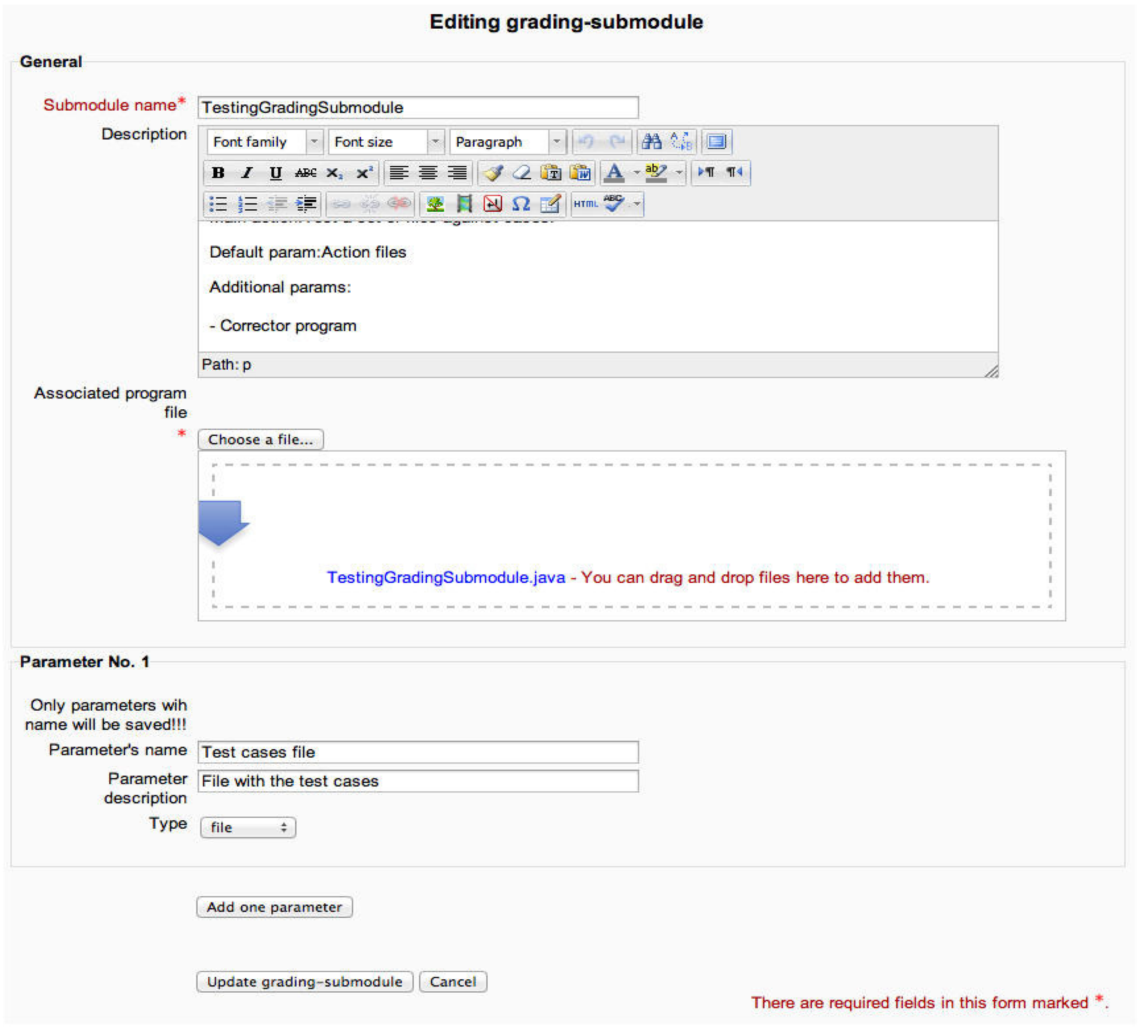
Having the classes already built, they can be registered in the grading module inside VPL Moodle's plugin. Then, it will be possible to associate a grading process (which will use the proposed architecture) to a VPL activity.

The registration of grading-submodules can be done through the VPL's administration inside Moodle. It means going through Moodle navigation tree as: *Moodle/Site administration/Plugins/Activity Modules/Virtual Programming Lab,* a link to the grading-submodules management is provided. The grading-submodules management shows a message that there is not any grading-submodule and provides a link to go to a new web page, which allows adding a new grading-submodule.

Every grading-submodule has to be registered there. Figure 31 shows the user interface to set the data associated to the *TestingGradingSubmodule*. This user interface allows setting the general information, which includes the name, the description and the associated program. The grading-submodule name is used to save the associated program. So this name and the class name in the associated program have to be the same. Additionally this name is unique in the grading module. There is a validation to guarantee it.

This user interface allows adding and deleting parameters that will be required by the associated program as well. The parameters' type can be number, text or file. It will be considered to deploy web form's elements when setting the parameters' values inside the grading process. A parameter to ask for the action filenames (names of the files on which the grading-submodule associated action will be performed) is created by default.

When all the information is ready, it is possible to save the grading-submodule. Immediately a process is performed, this include: storing information in the database, store the associated program in the file system, and sending the program to the Jail subsystem to be compiled and added to the submodules jar file in the lib directory.


Editing grading-submodule

General

Submodule name* TestingGradingSubmodule

Description

Default param:Action files

Additional params:

- Corrector program

Path: p

Associated program file * Choose a file...

TestingGradingSubmodule.java - You can drag and drop files here to add them.

Parameter No. 1

Only parameters wih name will be saved!!!

Parameter's name Test cases file

Parameter description File with the test cases

Type file

Add one parameter

Update grading-submodule Cancel

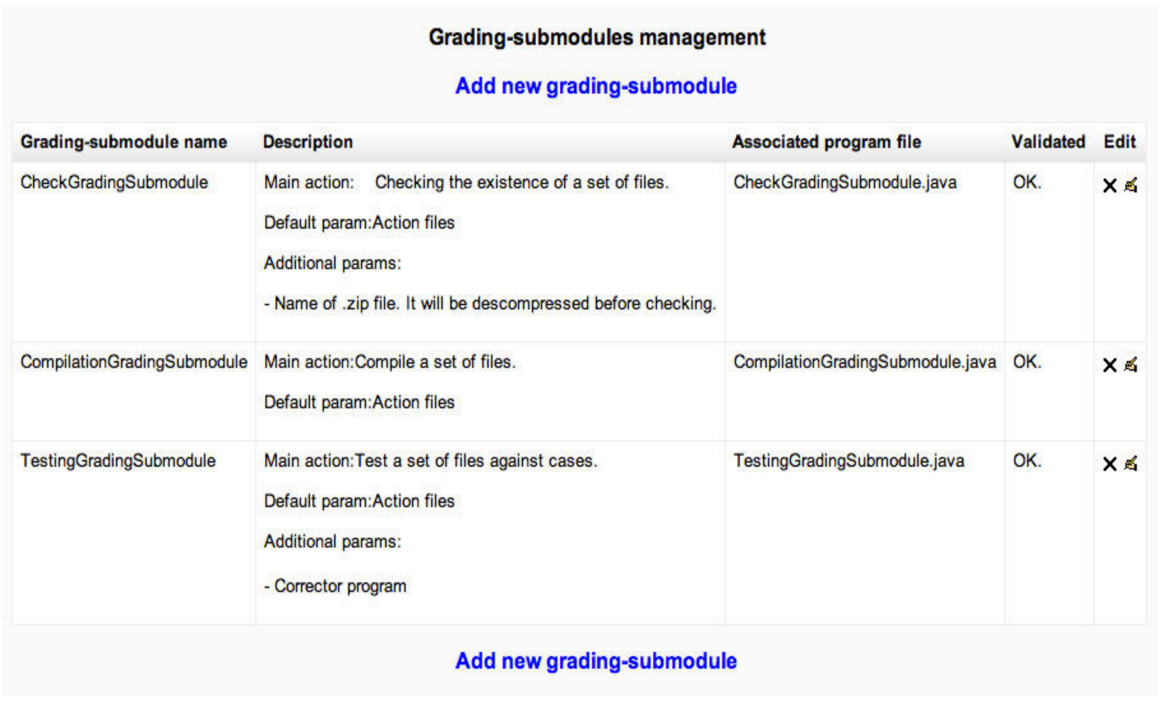
There are required fields in this form marked *.

Figure 31. Adding a new grading-submodule for testing Java programs

After the registration of all grading-submodules, the management user interface shows a summary of the existent grading-submodules. It can be seen in Figure 32. The list of existent grading-submodules provides a quick description of each of them. Additionally, shows information about the validation of the grading-submodule associated program. It means if the compilation and addition to the jar file in the Jail system was carried out without troubles. Links to go to add, edit or delete a grading-submodule are provided as well.

Grading process management

Having grading-submodules already registered it is possible to define grading processes associated to a given VPL activity. So firstly, the VPL activity has to be created and configured.



Grading-submodules management

Add new grading-submodule

Grading-submodule name	Description	Associated program file	Validated	Edit
CheckGradingSubmodule	Main action: Checking the existence of a set of files. Default param:Action files Additional params: - Name of .zip file. It will be descompressed before checking.	CheckGradingSubmodule.java	OK.	✕ ✎
CompilationGradingSubmodule	Main action:Compile a set of files. Default param:Action files	CompilationGradingSubmodule.java	OK.	✕ ✎
TestingGradingSubmodule	Main action:Test a set of files against cases. Default param:Action files Additional params: - Corrector program	TestingGradingSubmodule.java	OK.	✕ ✎

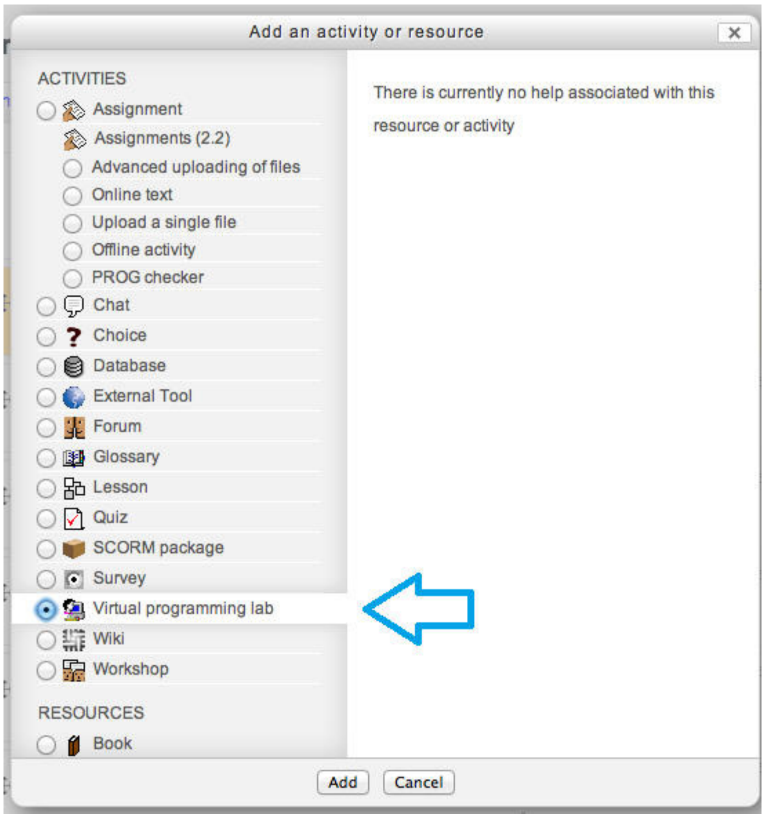
Add new grading-submodule

Figure 32. Grading-submodules management user interface

VPL activity creation and configuration

Figures 33 and 34 show the creation of a VPL activity. The first one shows VPL as an activity type to be created. The second one shows the form to fill the information about the programming assignment. The layout is based on common Moodle's activities. There is a set of fields to fill information about the VPL activity, which is interpreted as a programming assignment. The information about how to fill this information is provided in the online official documentation of VPL[50].

[50] http://vpl.dis.ulpgc.es/index.php/es/pantallazos/capturas-profesor

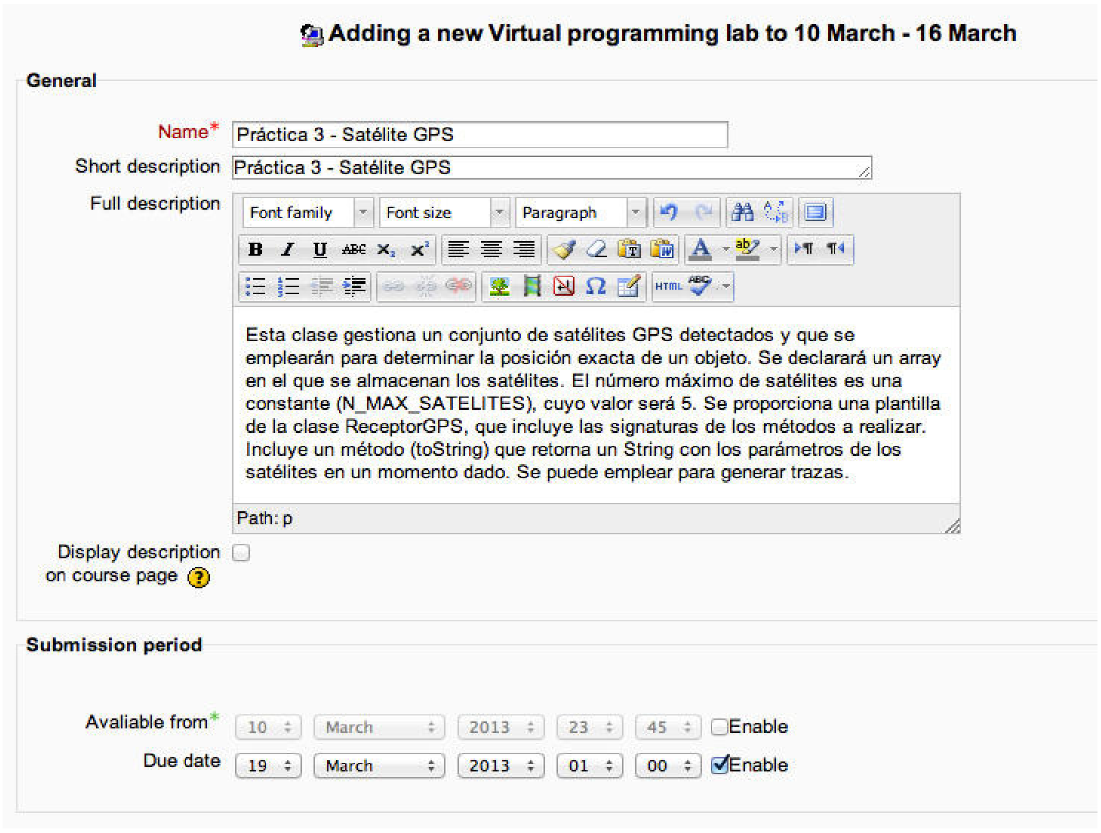

Add an activity or resource

ACTIVITIES
- Assignment
  - Assignments (2.2)
  - Advanced uploading of files
  - Online text
  - Upload a single file
  - Offline activity
  - PROG checker
- Chat
- Choice
- Database
- External Tool
- Forum
- Glossary
- Lesson
- Quiz
- SCORM package
- Survey
- [✓ Virtual programming lab]
- Wiki
- Workshop

RESOURCES
- Book

There is currently no help associated with this resource or activity

Figure 33. VPL activity selection


Adding a new Virtual programming lab to 10 March - 16 March

General

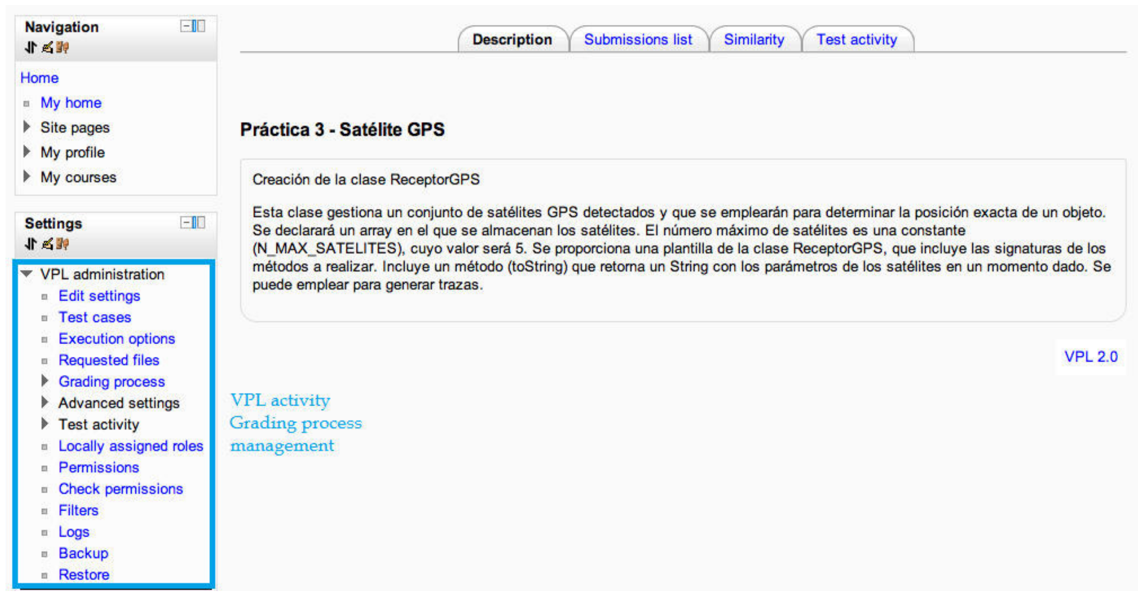
Name*	Práctica 3 - Satélite GPS
Short description	Práctica 3 - Satélite GPS
Full description	Esta clase gestiona un conjunto de satélites GPS detectados y que se emplearán para determinar la posición exacta de un objeto. Se declarará un array en el que se almacenan los satélites. El número máximo de satélites es una constante (N_MAX_SATELITES), cuyo valor será 5. Se proporciona una plantilla de la clase ReceptorGPS, que incluye las signaturas de los métodos a realizar. Incluye un método (toString) que retorna un String con los parámetros de los satélites en un momento dado. Se puede emplear para generar trazas.
Display description on course page	☐

Submission period

Avaliable from*	10	March	2013	23	45	☐ Enable
Due date	19	March	2013	01	00	☑ Enable

Figure 34. VPL activity (programming assignment) creation

After the creation of a VPL activity is done, it is possible to configure that. Figure 35 highlights the management section in the navigation tree. The information about how to set these settings is provided in the project's official site[51]. Almost all the default configuration is going to be used for this study case, but for the execution options and requested files.



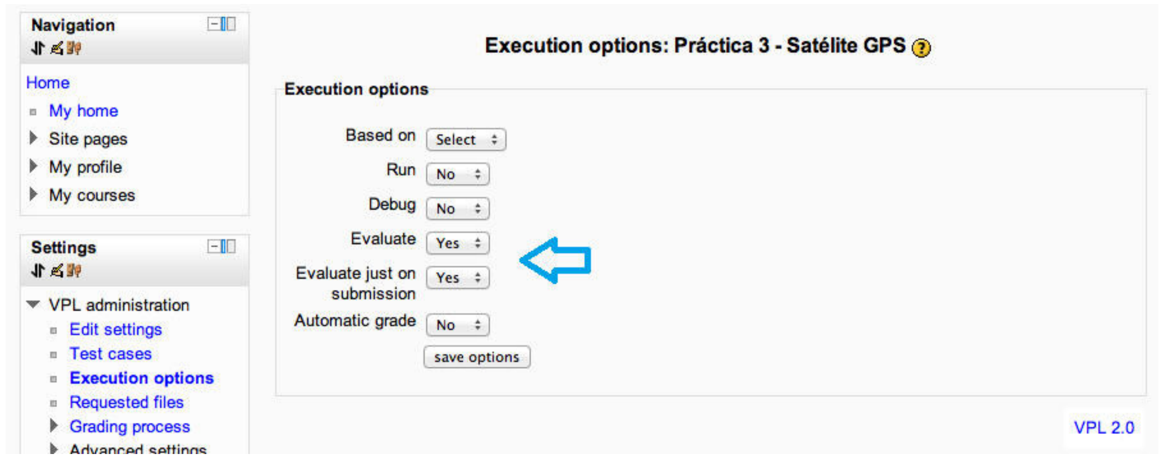
Práctica 3 - Satélite GPS

Creación de la clase ReceptorGPS

Esta clase gestiona un conjunto de satélites GPS detectados y que se emplearán para determinar la posición exacta de un objeto. Se declarará un array en el que se almacenan los satélites. El número máximo de satélites es una constante (N_MAX_SATELITES), cuyo valor será 5. Se proporciona una plantilla de la clase ReceptorGPS, que incluye las signaturas de los métodos a realizar. Incluye un método (toString) que retorna un String con los parámetros de los satélites en un momento dado. Se puede emplear para generar trazas.

Figure 35. Settings' management for a VPL activity

The execution options' setting allows configuring that the VPL tool will evaluate the student's submission and this evaluation will be done automatically. These settings are highlighted in Figure 36.



Execution options: Práctica 3 - Satélite GPS

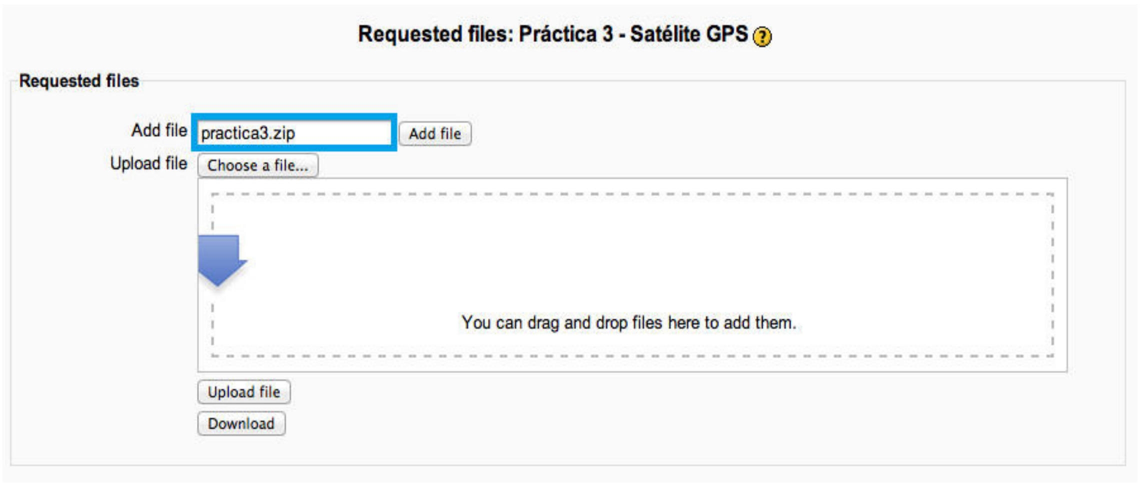
Execution options

Based on	Select
Run	No
Debug	No
Evaluate	Yes
Evaluate just on submission	Yes
Automatic grade	No

Figure 36. Execution options for a VPL activity

VPL provides of other functionality, which is the possibility of set the name of the requested files. In this case-study, that functionality allows guaranteeing that the student uploads a file called "practica3.zip". It can be seen in Figure 37.

[51] http://vpl.dis.ulpgc.es/index.php


Requested files: Práctica 3 - Satélite GPS

Requested files

Add file: practica3.zip

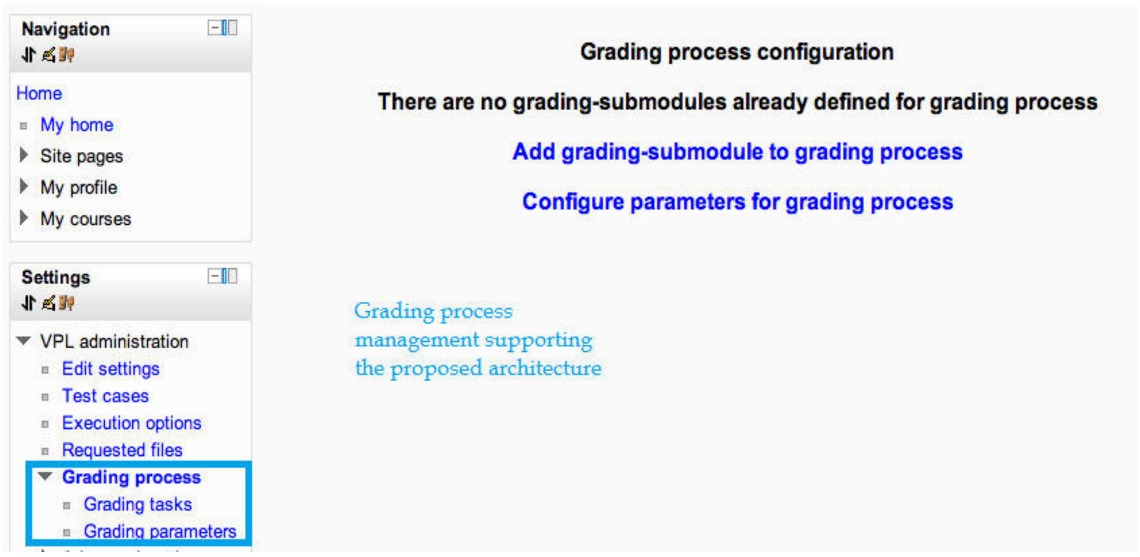
Upload file

You can drag and drop files here to add them.

Figure 37. Requested files for a VPL activity

When the VPL activity is configured, it is possible to manage its associated grading process. To do this, the navigation tree shows 3 nodes, which allow entering to the respective configuration user interface. This can be seen in Figure 38.

The first web form shows the summary of the grading process. Initially it is empty and only the links to add grading-submodules and to configure parameters are shown.


Grading process configuration

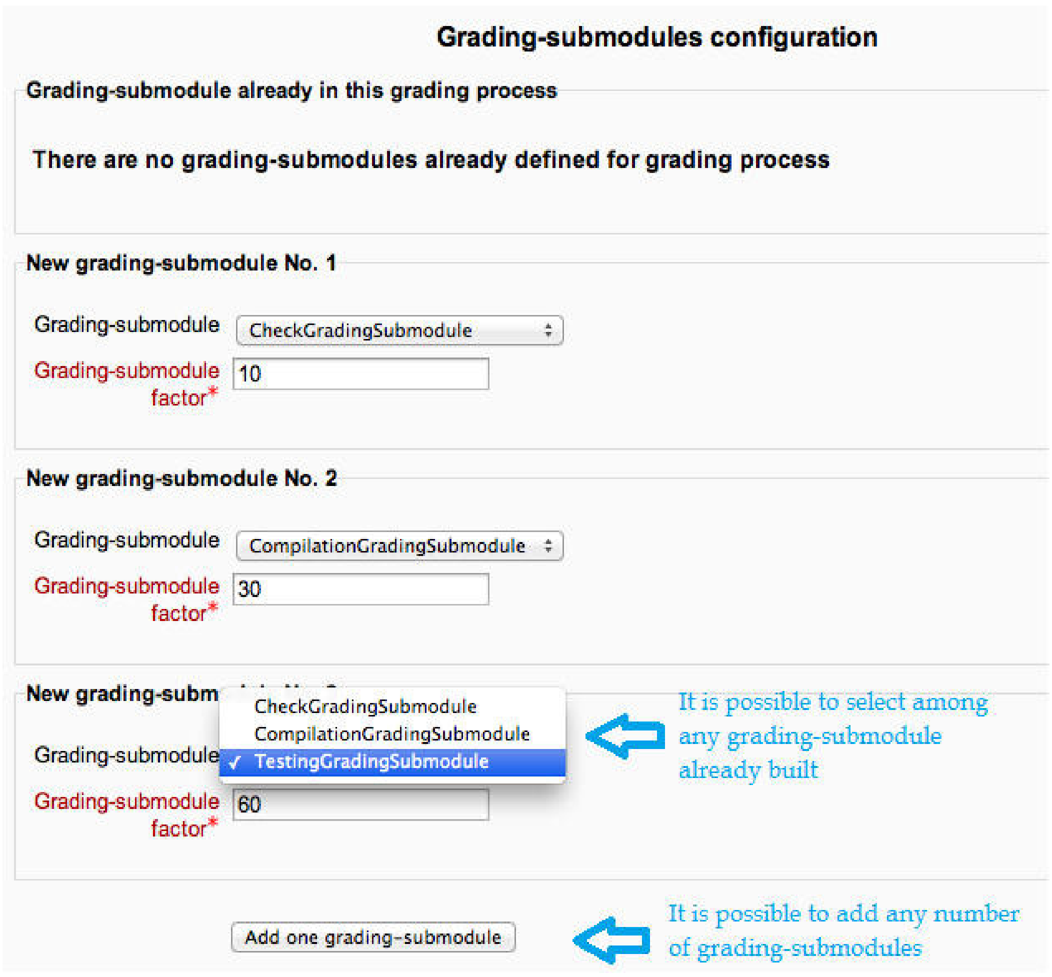
There are no grading-submodules already defined for grading process

Add grading-submodule to grading process

Configure parameters for grading process

Figure 38. Grading process web form

To define the grading process, it is necessary to include some grading-submodules inside that. Figure 39 shows how the user interface to add new grading-submodules looks.



Grading-submodules configuration

Grading-submodule already in this grading process

There are no grading-submodules already defined for grading process

New grading-submodule No. 1

Grading-submodule	CheckGradingSubmodule
Grading-submodule factor*	10

New grading-submodule No. 2

Grading-submodule	CompilationGradingSubmodule
Grading-submodule factor*	30

New grading-subm

Grading-submodule	CheckGradingSubmodule / CompilationGradingSubmodule / ✓ TestingGradingSubmodule
Grading-submodule factor*	60

It is possible to select among any grading-submodule already built

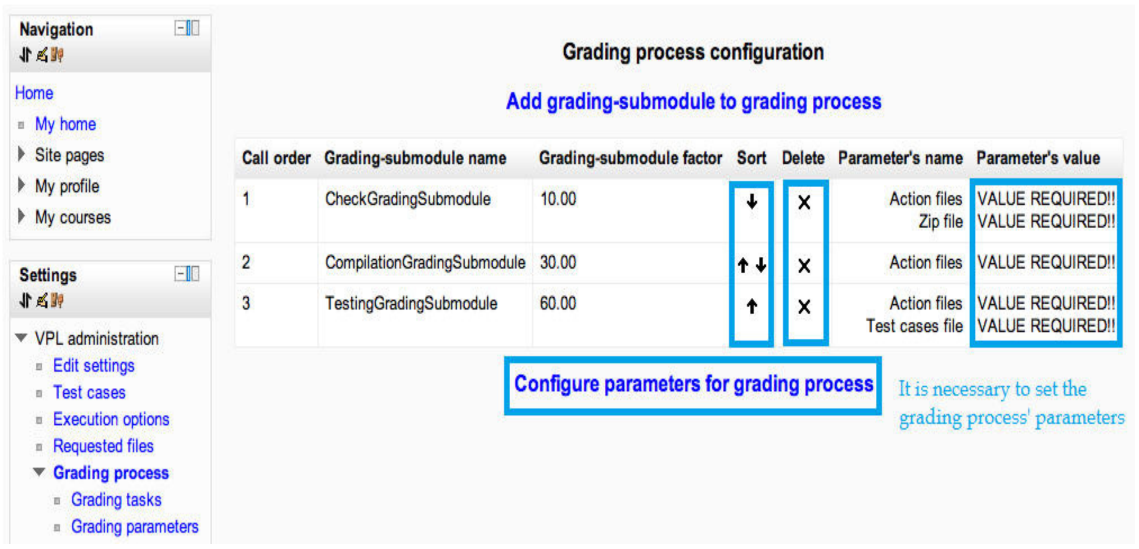
Add one grading-submodule

It is possible to add any number of grading-submodules

Figure 39. Definition of grading-submodules inside the grading process

It is possible to include any grading submodule, registered previously in the VPL system, inside the grading process. It is possible to define any number of grading-submodules as well. Finally, considering that the evaluation of each of them will affect the final grade for the submission, it is necessary to define the percentage of that affectation. The adding of all the defined percentages has to be 100.

After the definition of which grading-submodules will be inside the grading process, it is possible to see them in the grading process summary. Figure 40 shows how this summary looks. In this form, it is possible to sort the order of the grading-submodules as the teaching staff requires. Additionally, it is possible to delete grading-submodules from the grading process. A message, showing that there are parameters without assigned values, is shown as well.



Grading process configuration

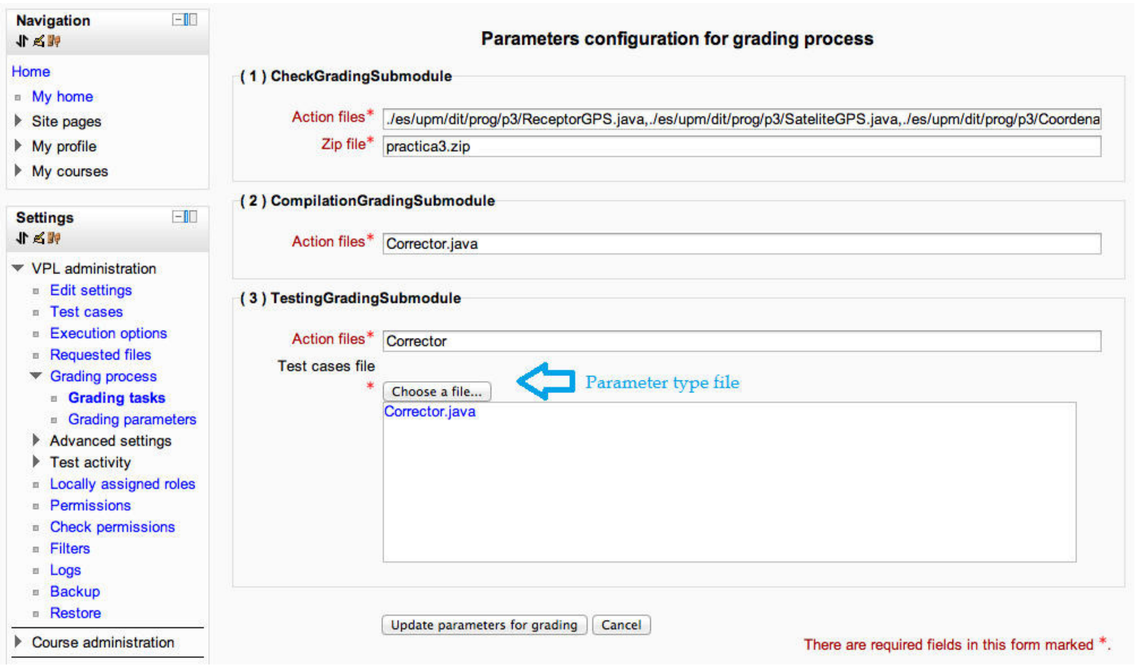
Add grading-submodule to grading process

Call order	Grading-submodule name	Grading-submodule factor	Sort	Delete	Parameter's name	Parameter's value
1	CheckGradingSubmodule	10.00	↓	×	Action files Zip file	VALUE REQUIRED!! VALUE REQUIRED!!
2	CompilationGradingSubmodule	30.00	↑ ↓	×	Action files	VALUE REQUIRED!!
3	TestingGradingSubmodule	60.00	↑	×	Action files Test cases file	VALUE REQUIRED!! VALUE REQUIRED!!

Configure parameters for grading process

Figure 40. Definition of grading-submodules inside the grading process

The next step is to set the parameters' values. Figure 41 shows user interface to set these values. They are separated considering every grading-submodule inside the grading process. The web form element type depends on the parameter's type defined in the creation of the grading-submodule. Thus, the 'zip file' parameter required by the *CheckGradingSubmodule* is a name so it is necessary to use a text field; in the other hand, the 'test cases file' parameter required by the *TestingGradingSubmodule* has been deployed as an element to upload a file. It is worth mentioning that every grading-submodule has a main action implicitly defined, so it is always necessary to define a set of filenames on which the action will be performed.



Parameters configuration for grading process

(1) CheckGradingSubmodule

Action files*: ./es/upm/dit/prog/p3/ReceptorGPS.java,./es/upm/dit/prog/p3/SateliteGPS.java,./es/upm/dit/prog/p3/Coordena

Zip file*: practica3.zip

(2) CompilationGradingSubmodule

Action files*: Corrector.java

(3) TestingGradingSubmodule

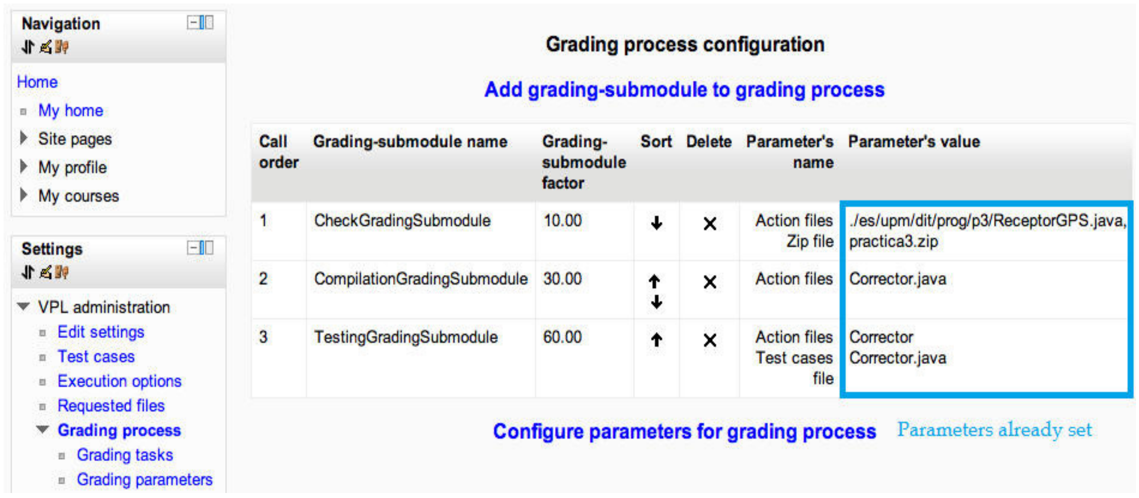
Action files*: Corrector

Test cases file*: Choose a file... Corrector.java

There are required fields in this form marked *.

Figure 41. Configuration of grading-submodules' parameters inside the grading process

Finally, Figure 42 shows the grading process summary already configured.



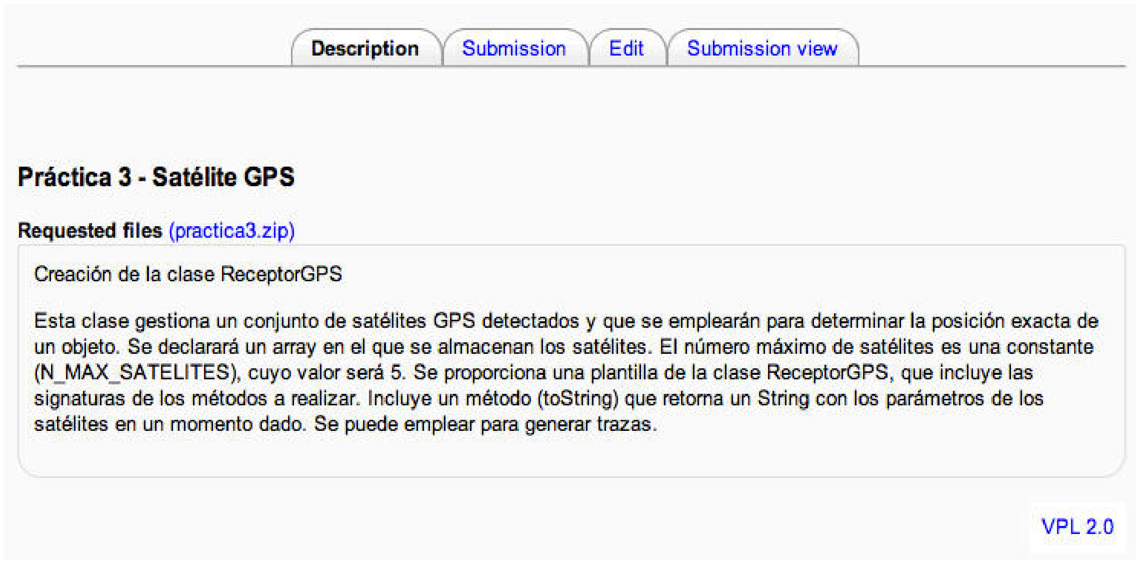
Call order	Grading-submodule name	Grading-submodule factor	Sort	Delete	Parameter's name	Parameter's value
1	CheckGradingSubmodule	10.00	↓	×	Action files Zip file	./es/upm/dit/prog/p3/ReceptorGPS.java, practica3.zip
2	CompilationGradingSubmodule	30.00	↑ ↓	×	Action files	Corrector.java
3	TestingGradingSubmodule	60.00	↑	×	Action files Test cases file	Corrector Corrector.java

Figure 42. Grading process already configured

The student's submission

The user interface for the student is provided by VPL. The student only has to upload its assignment solution as a zip file, send it and see the feedback.

Figure 43 shows the detail of the assignment.



Práctica 3 - Satélite GPS

Requested files (practica3.zip)

Creación de la clase ReceptorGPS

Esta clase gestiona un conjunto de satélites GPS detectados y que se emplearán para determinar la posición exacta de un objeto. Se declarará un array en el que se almacenan los satélites. El número máximo de satélites es una constante (N_MAX_SATELITES), cuyo valor será 5. Se proporciona una plantilla de la clase ReceptorGPS, que incluye las signaturas de los métodos a realizar. Incluye un método (toString) que retorna un String con los parámetros de los satélites en un momento dado. Se puede emplear para generar trazas.

Figure 43. Student interface – assignment description

The student has to upload a file. It is shown in Figure 44. It has been highlighted the name of the requested file, which was configured in the VPL's configuration showed before n this section (Figure 37).

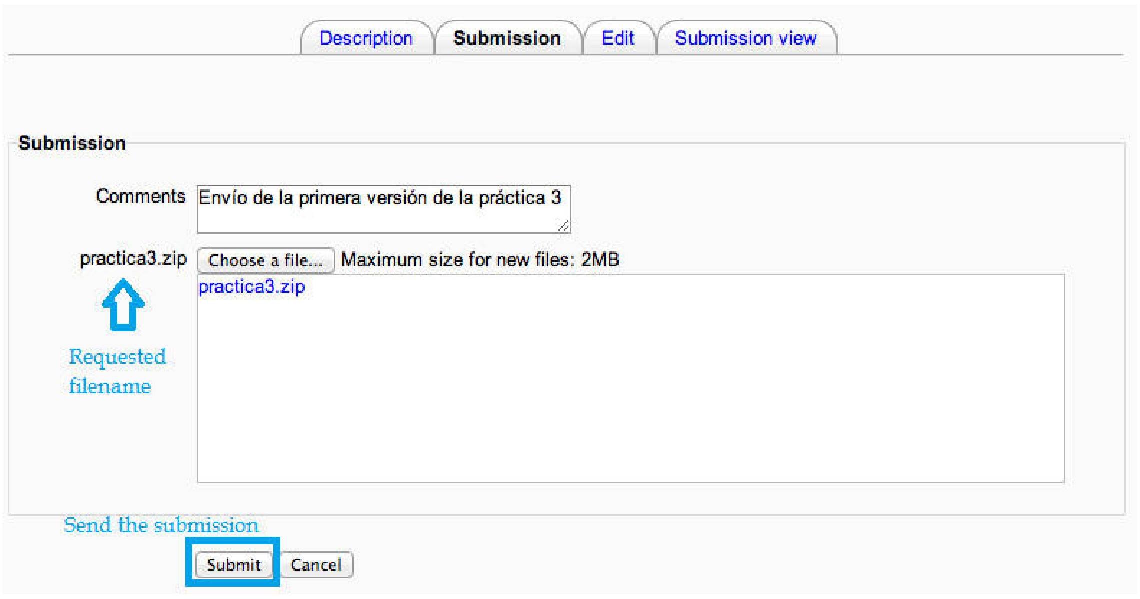
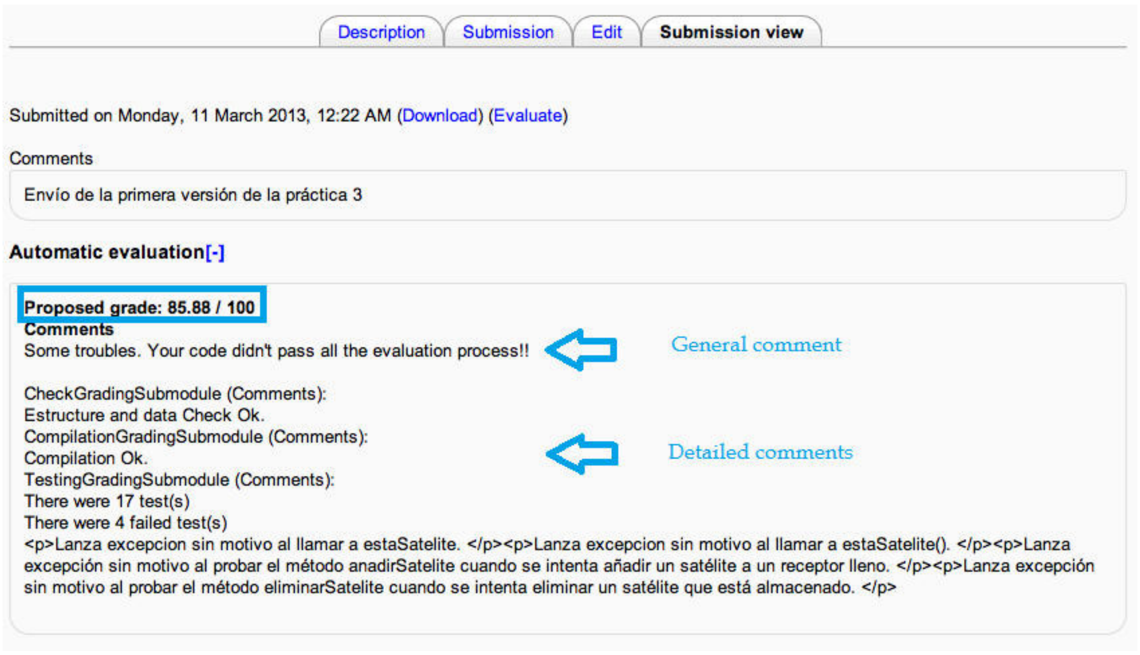


Figure 44. Student interface – uploading the requested file

After the submission is done, the data is sent to the Jail subsystem, which performs the grading process in a sandboxed environment. Then, the feedback is received. This include de proposed grade, a general comment for the process and detailed comments regarding every grading-submodule. It can be seen in Figure 45.



Submitted on Monday, 11 March 2013, 12:22 AM (Download) (Evaluate)

Comments

Envío de la primera versión de la práctica 3

Automatic evaluation[-]

Proposed grade: 85.88 / 100
Comments
Some troubles. Your code didn't pass all the evaluation process!!

CheckGradingSubmodule (Comments):
Estructure and data Check Ok.
CompilationGradingSubmodule (Comments):
Compilation Ok.
TestingGradingSubmodule (Comments):
There were 17 test(s)
There were 4 failed test(s)
<p>Lanza excepcion sin motivo al llamar a estaSatelite. </p><p>Lanza excepcion sin motivo al llamar a estaSatelite(). </p><p>Lanza excepción sin motivo al probar el método anadirSatelite cuando se intenta añadir un satélite a un receptor lleno. </p><p>Lanza excepción sin motivo al probar el método eliminarSatelite cuando se intenta eliminar un satélite que está almacenado. </p>

Figure 45. Student interface – feedback provided

5.4.2 Case study 2

This assignment asks the student to code a new Java class called *NavegadorGPS* and a set of methods, which follow a given signature. The complete description of the assignment (in Spanish) can be seen in the annexes section.

There is a set of Java classes (source files) provided by the teaching staff, which has to be used by the student during its assignment implementation. This set includes classes *CoordenadaCartesiana, POI, Gasolinera y Hotel*. They are already built and do not need changes.

All the source files have to belong to the package *es.upm.dit.prog.p5* and be compressed in a zip file, which has to be called '*practica5.zip*'. This is the only file to be uploaded in the student's submission.

Grading criteria

The grading criteria include:

1. Checking the directory structure of the source files.
2. Compiling all the source files.
3. Testing the new class and its methods against a set of test cases defined in an additional file.
4. Checking the code documentation.

The test cases file and a file with rules to check the code documentation have to be built by the teaching staff but they are not provided to students.

Required grading-submodules

Considering the grading criteria previously given it is necessary to use four grading-submodules. Each of them will be associated to a one criterion. They are:

1. A grading-submodule to unpack the zip file inside the jail and to check whether the source code files maintain the directory structure.
2. A grading-submodule to compile the set of Java programs.
3. A grading-submodule to test the Java program against a set of test cases.
4. A grading-submodule to check the code documentation.

Here, it is possible to reuse the existent grading-submodules. So only the implementation of the last grading-submodule is necessary.

Grading-submodule associated programs

It is necessary only the implementation of a new associated program.

Class StyleGradingSubmodule [es.upm.dit.tfm.grad.sub]

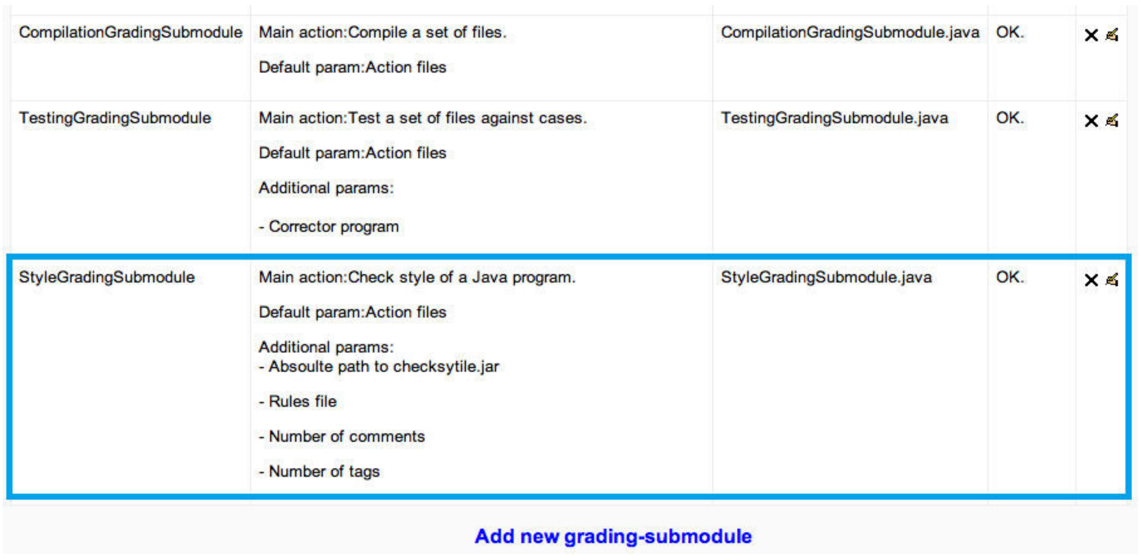
`public class` **`StyleGradingSubmodule`**
extends `GradingSubmoduleProgram`

The main action of this program is checking the style (comments and tags) in a Java source file. The data required from the *GradingSubmoduleConf* instance include, additionally to the file to be checked, the name of the file with the rules to check the style, the comments number and the tags number. They are obtained with the *getProgramParameters()* method in the *GradingSubmoduleConf* instance. All these parameters are obtained as a String separated by semicolons. Then the list is split (using semicolons as marks) to get the parameters in different variables. The file to be checked is obtained using the *getActionFileList()* method in the *GradingSubmoduleConf* instance. The style is checked with a system command execution. The command calls an external program (CheckStyle.jar). The results are received in the standard output. They include missed comments missed and tags. The results are processed to find the number of comments and tags missed. The grade is established taking in account them. A temporal and total grade will be assigned if there is not a missed comment, but this grade will be reduced if missed tags were found. To get a success process, there has not to be missed comments or missed tags. If there is one missed tag or comment, the submodule will be set as failed but the grade will be set with the value calculated previously.

Management and configuration of the process

The case-study 1 showed how to register new grading-submodules, how to create a VPL activity, how to define a grading process and how to set the parameters required by grading-submodules inside the grading process. Then, this section shows only relevant differences between the case-studies.

This case-study has to register a new grading-submodule to check the style of the source code. Its associated program needs a set of parameters including (as well as action files): the absolute path to the checkstyle jar, the rules file to check the style, and the number of comments (the same as the number of methods) and the number of tags that the code should have to. The new grading-submodules management interface is shown in Figure 46.

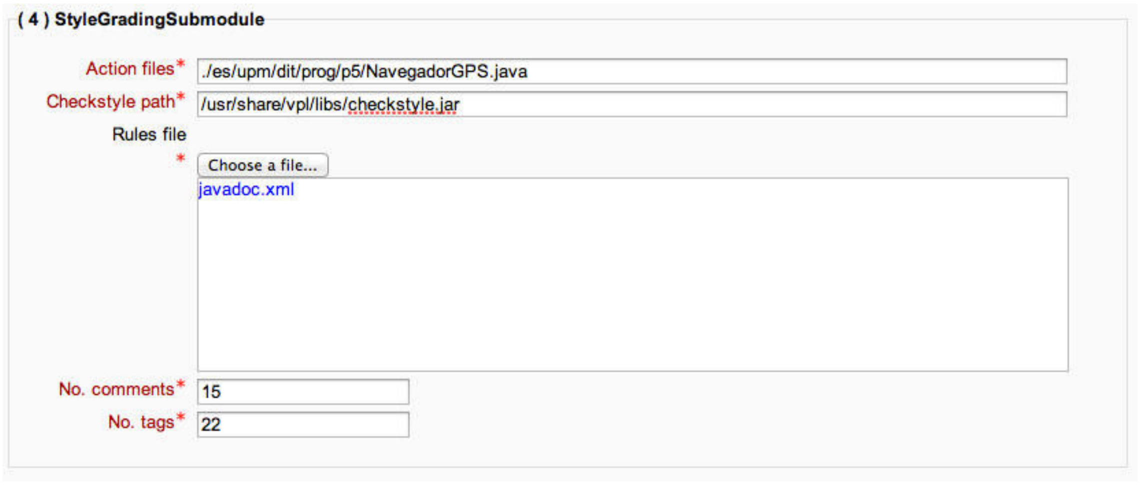


CompilationGradingSubmodule	Main action:Compile a set of files. Default param:Action files	CompilationGradingSubmodule.java	OK.	✕
TestingGradingSubmodule	Main action:Test a set of files against cases. Default param:Action files Additional params: - Corrector program	TestingGradingSubmodule.java	OK.	✕
StyleGradingSubmodule	Main action:Check style of a Java program. Default param:Action files Additional params: - Absoulte path to checksytile.jar - Rules file - Number of comments - Number of tags	StyleGradingSubmodule.java	OK.	✕

Add new grading-submodule

Figure 46. Grading-submodules management considering the StyleGradingSubmodule

The parameters' configuration user interface adds a new section. It can be seen in Figure 47.

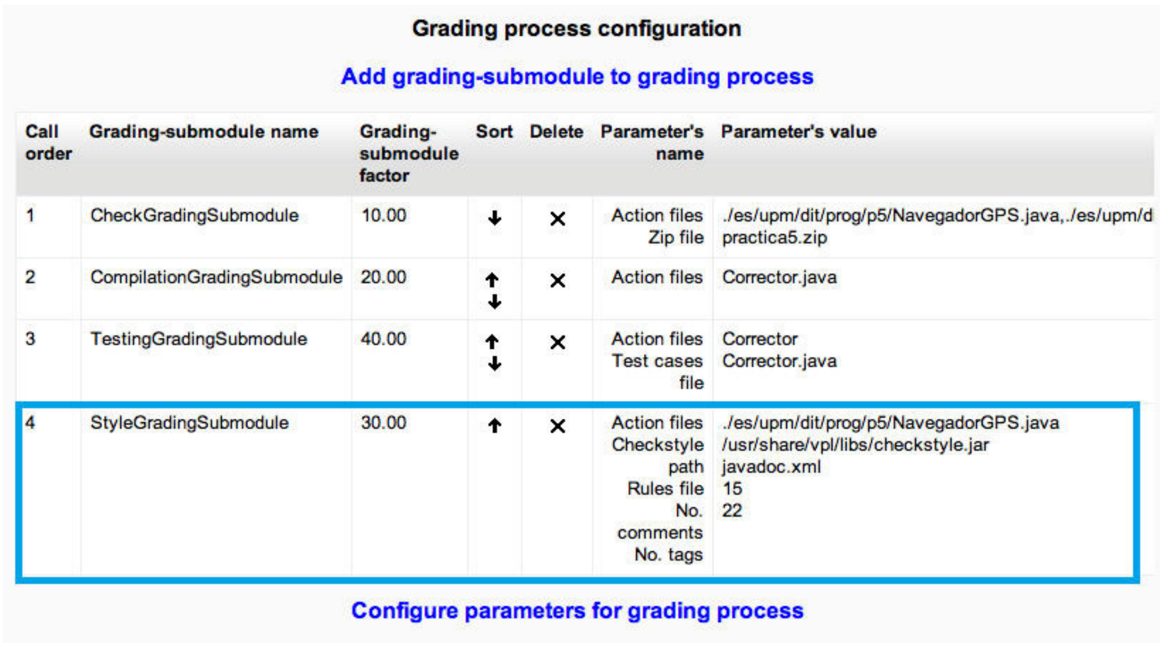


(4) StyleGradingSubmodule

Action files*	./es/upm/dit/prog/p5/NavegadorGPS.java
Checkstyle path*	/usr/share/vpl/libs/checkstyle.jar
Rules file *	Choose a file... javadoc.xml
No. comments*	15
No. tags*	22

Figure 47. Parameters for StyleGradingSubmodule

The complete grading process configured for this VPL activity is shown in Figure 48.



Grading process configuration

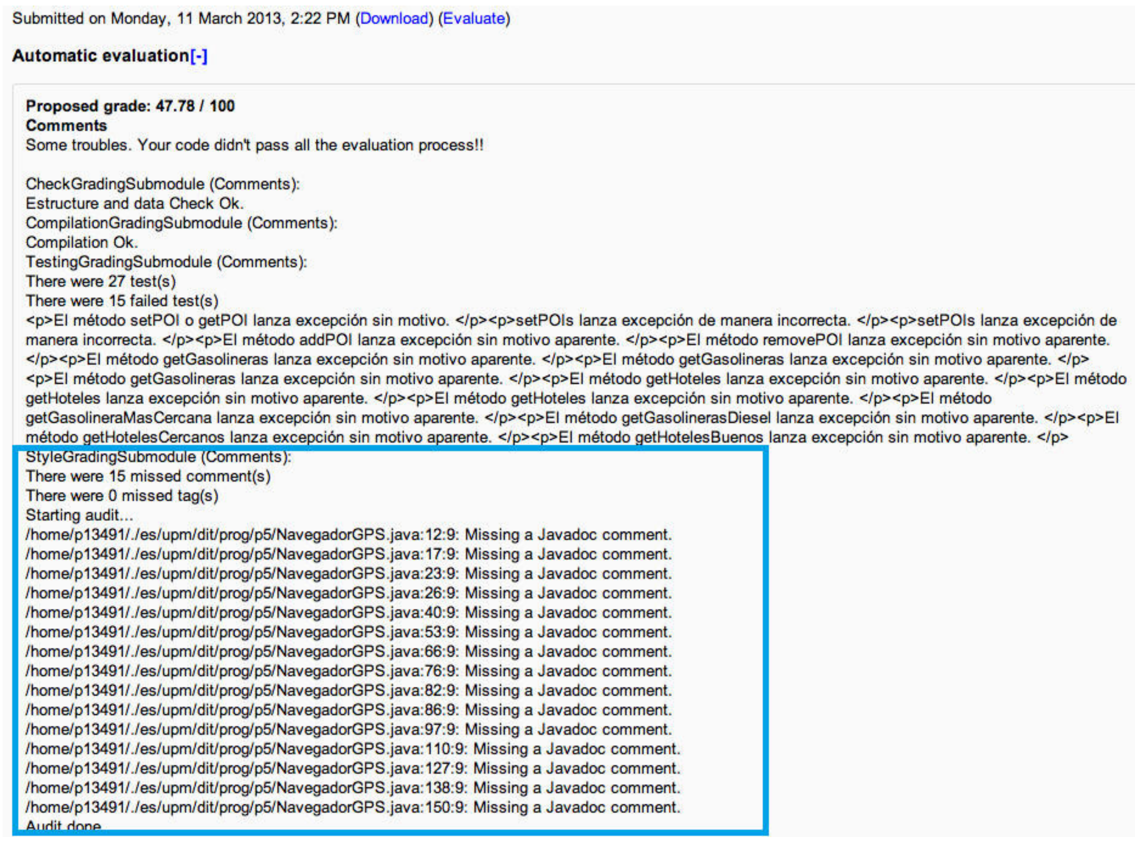
Add grading-submodule to grading process

Call order	Grading-submodule name	Grading-submodule factor	Sort	Delete	Parameter's name	Parameter's value
1	CheckGradingSubmodule	10.00	↓	×	Action files Zip file	./es/upm/dit/prog/p5/NavegadorGPS.java,./es/upm/d practica5.zip
2	CompilationGradingSubmodule	20.00	↑ ↓	×	Action files	Corrector.java
3	TestingGradingSubmodule	40.00	↑ ↓	×	Action files Test cases file	Corrector Corrector.java
4	StyleGradingSubmodule	30.00	↑	×	Action files Checkstyle path Rules file No. comments No. tags	./es/upm/dit/prog/p5/NavegadorGPS.java /usr/share/vpl/libs/checkstyle.jar javadoc.xml 15 22

Configure parameters for grading process

Figure 48. Grading process considering a StyleGradingSubmodule

Finally, the student has not to do anything different at all. He just uploads and sends his file and sees the feedback. The feedback now includes a section for comments about the documentation applied in his code. The final grade depends on the new grading-submodule as well. This can be seen in Figure 49.



```
Submitted on Monday, 11 March 2013, 2:22 PM (Download) (Evaluate)

Automatic evaluation[-]

Proposed grade: 47.78 / 100
Comments
Some troubles. Your code didn't pass all the evaluation process!!

CheckGradingSubmodule (Comments):
Estructure and data Check Ok.
CompilationGradingSubmodule (Comments):
Compilation Ok.
TestingGradingSubmodule (Comments):
There were 27 test(s)
There were 15 failed test(s)
<p>El método setPOI o getPOI lanza excepción sin motivo. </p><p>setPOIs lanza excepción de manera incorrecta. </p><p>setPOIs lanza excepción de
manera incorrecta. </p><p>El método addPOI lanza excepción sin motivo aparente. </p><p>El método removePOI lanza excepción sin motivo aparente.
</p><p>El método getGasolineras lanza excepción sin motivo aparente. </p><p>El método getGasolineras lanza excepción sin motivo aparente. </p>
<p>El método getGasolineras lanza excepción sin motivo aparente. </p><p>El método getHoteles lanza excepción sin motivo aparente. </p><p>El método
getHoteles lanza excepción sin motivo aparente. </p><p>El método getHoteles lanza excepción sin motivo aparente. </p><p>El método
getGasolineraMasCercana lanza excepción sin motivo aparente. </p><p>El método getGasolinerasDiesel lanza excepción sin motivo aparente. </p><p>El
método getHotelesCercanos lanza excepción sin motivo aparente. </p><p>El método getHotelesBuenos lanza excepción sin motivo aparente. </p>
StyleGradingSubmodule (Comments):
There were 15 missed comment(s)
There were 0 missed tag(s)
Starting audit...
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:12:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:17:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:23:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:26:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:40:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:53:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:66:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:76:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:82:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:86:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:97:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:110:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:127:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:138:9: Missing a Javadoc comment.
/home/p13491/./es/upm/dit/prog/p5/NavegadorGPS.java:150:9: Missing a Javadoc comment.
Audit done.
```

Figure 49. Student interface – feedback considering code style

5.4.3 Analysis

Both of the previous case-studies have allowed validating the achievement of extensibility, modularity and flexibility features inside the grading process (it was the goal of the current work).

The modularity is provided by the association of a given criterion to a grading-submodule (it could be possible to associate a grading metric to the grading-submodule as well). The extensibility is achieved because it is possible to register new grading-submodules through the grading module management. This extensibility can be achieved inside the grading process as well, because this process can support the addition of any number of already registered grading-submodules. The flexibility is provided through the possibility of sorting the grading-submodules inside the grading process as the teaching staff needs.

Additionally, the modularity associated to the definition of every grading-submodule makes possible the reuse of them. It can be seen considering the second case-study because it was necessary the implementation of only one additional grading-submodule.

At the end, all of these features make possible the reduction of the required time to define a grading process when there are well designed and implemented grading-submodules.

5.5 Chapter summary

This chapter has described important considerations made during the implementation of the VPL-Moodle and VPL-Jail subsystems, and the subsystems' communication. The implementation has included the creation of new programs and the modification of others already built to integrate the new module with VPL. The implementation itself has been a first way to validate the proposed architecture. The formal validation has been through two case-studies.

The VPL-Jail subsystem implementation has included the architecture implementation. The most of the implementation has been done using Java language due to its objects orientation that has allowed representing the needed elements. Additionally, Java provides interesting libraries which allowed saving implementation time. These libraries provided of functionality to map the XML configuration file, to generate log files, to create instances and pass information at runtime using Java reflection technology, and to execute system commands. The classes implemented have been defined to belong to two different jar files, the core and the submodules. The second one will continue adding new classes as the administrator or the teaching staff defines new grading-submodules. The integration with VPL has been provided through the modification of a set of files (evaluation and execution) written in Linux shell scripting language.

The VPL-Moodle subsystem implementation has focused on building the front-end of the grading module. It has been done through the use of Moodle APIs and working with PHP language. The implementation of well defined files (configuration, library) and the jailconnection class has allowed guaranteeing integration with VPL and at the same time the possibility of the tool's independence.

The subsystems' communication implementation has applied reuse of code. The jail module for communication inside the VPL Moodle's plugin has been used to create a new class, which sends new grading-submodules associated programs to the VPL-Jail subsystem. The Jail server, which is written in C++, has been modified to accept a new request method when it is necessary to operate with grading-submodule associated programs. It is worth mentioning the modification of the communication module inside VPL to support sending of binary files encoded in base64 through XML-RPC protocol.

The validation has been done through the analysis and deployment of two case-studies, which are based on real programming assignments proposed to students at ETSIT. This validation showed the whole process from scratch. It included the analysis of the assignments, the definition and implementation of grading-submodule associated programs, the registration of the grading-submodules and the configuration of the grading process. These case-studies have allowed validating the features of modularity, extensibility, and flexibility in the grading process. An additional feature obtained is reusability.

6 Conclusions

As final part of this work, this chapter shows how much the goals, defined in the introduction of this work, have been achieved. Considering the research field of automatic grading of programming assignments, the main contributions and the future work are highlighted as well.

6.1 Goals achievement

The main goal of this work, which was to propose and validate a new architecture for automatic grading of programming assignments, has been achieved through the performing of a set of software engineering and research stages described in this work.

To make a better explanation about this achievement, the specific goals defined in the introduction chapter are cited.

"To use the knowledge about ***scientific research****, which was acquired in the master course, in a real problem"*

"To make a ***systematic review of related works*** *to get an actual context in automatic grading of programming assignments"*

These goals have been achieved together while carrying out the stage reported in the state of the art chapter. The work done there used techniques learned in the "Methodologies and Scientific Documentation" course. Initially, a systematic review of the last reported related-works was carried out; this included the search of relevant works and the definition of key features to make a comparison. This review helped to get an actual context of the research field, necessary to establish the current problems and research paths, to define a scope for this work, and to provide of a solution.

"To ***disseminate research results*** *through scientific publications in international forums"*

The first and fundamental part of this work, which is systematic review of related works, has been reported as a scientific article. This article has been accepted to be published in the **7th International Technology, Education and Development Conference.** The article has as title: **Programming Assignments Automatic Grading: Review of Tools and Implementations** (Caiza et al. 2013).

"To identify and use the most suitable features of ***software engineering****, which can be applied in this work"*

After getting in the context of the programming assignments automatic grading, some stages of the water fall software development model have been used to propose and validate the solution. These stages include: the requirements specification, the design, the implementation, and the validation. All of them have been considered while carrying out the problem analysis, design, implementation and validation chapters. These stages have been carried out with the use of knowledge learned in the "Architecture and ICT services management" course, and with a deep study about engineering and architecture of software.

"To ***gather a set of requirements*** *based on necessities of the students and the teaching staff inside the teaching-learning process of programming subjects"*

"To ***analyze the requirements and the context*** *to propose a suitable solution for the given problem"*

These both goals have been achieved while working in the stage reported in the problem analysis chapter. This has included the definition of actors, the establishment of use cases and the definition of a set of functional and non functional requirements. All these steps have been carried out considering the information provided by the systematic review and by the ETSIT's context (because ETSIT has in charge the SEAPP project). VPL was selected as a base tool to continue working, after a comparison among some tools for automatic grading. Based on this and on a scope definition an analysis of the solution was provided.

"To apply principles of ***software and services architecture to design a solution*** *for the given problem"*

The design chapter has required using principles of software architecture to design and to provide of software artifacts. The designed artifacts have allowed representing the proposed architecture from a higher level, using a layer-based representation, to a lower level using a class diagram representation. These artifacts have been defined considering two subsystems that compose the VPL tool. This stage allows having a set of useful artifacts that are useful for the implementation stage.

"To ***validate*** *the archi****tecture proposed through the implementation of*** *a working prototype and with the use of it in real case-studies"*

"To evaluate the results for establishing ***conclusions and future works****"*

These goals have been achieved at the end of the implementation stage and with the success results of the case-studies. The implementation and validation chapter has shown firstly that the architecture is workable and secondly, that the architecture works well. The architecture allows the grading process to be modular, extensible and flexible with the use of grading-submodules. Additionally the reusability of the grading-submodules can be obtained when a good design of the artifact and the associated program is carried out.

Finally, the realization of this work has helped to improve the capabilities that the "Master Universitario en Ingeniería de Redes y Servicios Telemáticos" aims to provide to its students[52], standing out those related to software and services architecture, and those related to research training.

6.2 Main Contributions

This work could be helpful to other related works and so it is necessary to make explicit the main contributions. These include:

- **An architecture proposal**, which is based on the definition of an orchestrator and grading-submodules (in any number and any arrangement), which could be implemented with any technology. This architecture can be used by already implemented tools or by new ones.
- **A systematic literature review**, which allows having a current context of the developed solutions in the field of the programming assignments automatic grading. This context and unsolved problems can be useful for other future related works.
- **A grading criteria characterization**, which can be considered as a first stage to define a model of grading programming assignments. Additionally this characterization can be used to get a better understanding when studying automatic grading tools (when different terms are used to express grading criteria).
- **A comparative analysis of grading tools**, which can be useful to get a quick sight about them, and to know their advantages and disadvantages.

It is worth highlighting that the idea of the grading-submodule artifact can be used or improved to define new ways of grading or new architectures. In addition, the elements of the proposed architecture are already implemented and can be reused to going on with new implementations because they have been implemented as open source. It can help to save implementation time in related projects. The mention of used technologies can be helpful to provide a first sight of them and to think about them as possible solutions for issues in other projects with similar functionalities. Finally, the considerations made in the different stages can be useful for other projects quite similar or which follow a similar process that which performed in this work.

[52] http://web.dit.upm.es/~doct/muirst/competencias.html

6.3 Future Work

The present work has validated the proposed architecture. It means that the architecture works as expected but it does not mean that it could not be improved.

Possible improvements are:

- Create of new grading-submodules. The proposed architecture aims to provide to the teaching staff the capability of configuring a grading process that includes many metrics and criteria. So it is necessary to make more tests focused on grading process that includes more grading-submodules and more variable arrangements.
- Measure optimized time in the grading processes definition. After the creation and registration of grading-submodules, the time to define and configure grading processes associated to assignments could be shorter than using other solutions. It could be probed through measuring the time that the configuration of a grading process takes in this solution against the time needed by other solutions' configuration.

Possible improvements maintaining the VPL-Jail subsystem but making changes on the VPL-Moodle subsystem are:

- Define a management module for grading processes. The case-studies have shown that sometimes the grading process could be very similar. Even, the grading process (without the parameters' values) could be the same among different assignments. So it could be possible to define a management of grading processes, it could help to reduce the time of the grading process definition.
- Annotate the grading-submodules. Considering a possible increment in the number of grading-submodules registered and a way to sort and filter them when defining the grading process, it is possible to create tags to make a classification. These tags could be metrics, criteria, and even the programming language associated.
- Improve the deployment of ancillary programs. The current solution supports the use of ancillary programs; these programs have to be placed manually in the *libs* directory. It could be possible to implement a management interface for these programs.

Improvements without considering Moodle's integration as requirement:

- The current solution has been divided in two parts. The main part, which contains the proposed architecture for the grading process, has been implemented in the VPL-Jail subsystem and is quite independent of the other subsystem. It means that the grading process could be used by any tool that creates a sandboxed environment with all the required files and sends a signal to start. On the other hand, the VPL-Moodle subsystem provides of the management interface and stores files required by the grading process. Then, it is possible to think about a solution which can store the data and maintain the jail environment, and provides everything as a service. It means that it could provide a service to access an assignments' repository, a service to copy and to store the data inside that system, a service to start with the grading process, and so on; in this case this solution would be completely independent and could connect any system (just a front-end), which would provide interfaces to connect the solution.

Possible improvements considering changes in the architecture's elements:

- Regarding the grading-submodule associated program, it acts as a wrapper written in Java that can call another libraries or ancillary programs, which have to be located in the *libs* directory. But it is possible to think about the possibility that the wrapper supports calls to other programs in other hosts through services. The idea appeared because there are already built tools which can provide the evaluation of some metrics as a service. In this case the wrapper could be more powerful.
- The XML configuration file is quite important for the proposed architecture because it defines the calls sequence for the grading-submodules (all of them are performed) and defines that the final grade will be calculated considering percentages for every success grading-submodule passed. This file could be changed to support more ways to calculate the final grade and additionally to stop the process if some grading-submodule was not passed. These features could be configurable.

Others possible future works:

- The review of existent tools to evaluate a given metric showed that Java is the programming language with most already built tools. So it is possible to build more of this kind of tools or libraries for other programming languages.

Bibliography

ALLEVATO, A. and EDWARDS, S.H., 2012. *RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications.* ACM.

AMELUNG, M., FORBRIG, P.and RÖSNER, D., 2008. *Towards Generic and Flexible Web Services for e-Assessment.* ACM.

AYDIN, C.C. and TIRKES, G., 2010. *Open Source Learning Management Systems in e-Learning and Moodle.* IEEE.

BELLAS, F. *Introducción a La Orientación a Objetos.* Available from: http://www.tic.udc.es/~fbellas/teaching/ioo/IOO.pdf.

CAIZA, J.C. and DEL ALAMO, J.M., 2013. *Programming Assignments Automatic Grading: Review of Tools and Implementations.* Valencia, Spain ed. IATED, 4-6 March, 2013.

CHOY, M., et al, 2008. Design and Implementation of an Automated System for Assessment of Computer Programming Assignments. *Advances in Web Based Learning–ICWL 2007*, pp. 584-596.

COCKBURN, A., 2001. *Writing Effective Use Cases.* Addison-Wesley Boston.

DOUCE, C., LIVINGSTONE, D. and ORWELL, J., 2005. Automatic Test-Based Assessment of Programming: A Review. *Journal on Educational Resources in Computing (JERIC)*, vol. 5, no. 3, pp. 4.

DOUGIAMAS, M. and TAYLOR, P., 2003. *Moodle: Using Learning Communities to Create an Open Source Course Management System.*

ECKSTEIN, G., 2010. *Guide to Learning Management Systems.* Available from: http://eckstein.id.au/5588/learning-management-systems/lms-review-options/.

EDWARDS, S.H. and PEREZ-QUINONES, M.A., 2008. Web-CAT: Automatically Grading Programming Assignments. *ACM SIGCSE Bulletin*, vol. 40, no. 3, pp. 328-328.

FORSYTHE, G.E. and WIRTH, N., 1965. Automatic Grading Programs. *Communications of the ACM*, vol. 8, no. 5, pp. 275-278.

GARCÍA GONZÁLEZ, A.J., TROYANO RODRÍGUEZ, Y., CURRAL, L. and CHAMBEL, M.J., 2010. Aplicación de Herramientas de Comunicación de la Plataforma Webct en la Tutorización de Estudiantes Universitarios Dentro del Espacio Europeo de Educación Superior. *Pixel-Bit: Revista De Medios y Educación*, no. 37, pp. 159-170.

HIGGINS, C., SYMEONIDIS, P.and TSINTSIFAS, A., 2002. *Diagram-Based CBA using DATsys and CourseMaster.* IEEE.

HIGGINS, C.A., GRAY, G., SYMEONIDIS, P. and TSINTSIFAS, A., 2005. Automated Assessment and Experiences of Teaching Programming. *Journal on Educational Resources in Computing (JERIC)*, vol. 5, no. 3, pp. 5.

IHANTOLA, P., AHONIEMI, T., KARAVIRTA, V.and SEPPÄLÄ, O., 2010. *Review of Recent Systems for Automatic Assessment of Programming Assignments.* ACM.

JELEMENSKÁ, K. and ČIČÁK, P., 2012. Improved Assignments Management in Moodle Environment. *INTED2012 Proceedings*, pp. 1809-1817.

LEAL, J.P. and SILVA, F., 2003. Mooshak: A Web-based multi-site Programming Contest System. *Software: Practice and Experience*, vol. 33, no. 6, pp. 567-581.

MARTÍNEZ, J., 2011. *La Orientación y La Tutoría En El Espacio Europeo De Educación Superior.* Available from: http://www.eumed.net/rev/ced/23/jamg.htm.

MÉNDEZ, I.C., 2008. Mejora de la Acción Tutorial Universitaria a través de las TIC.

PATIL, A., 2010. Automatic Grading of Programming Assignments. Master's Thesis, San José State University.

QUEIRÓS, R.A.P. and LEAL, J.P., 2012. *PETCHA: A Programming Exercises Teaching Assistant.* ACM.

RAYNAL, F., 2001. *Xinetd.* 02/28/2001, Available from: http://www.linuxfocus.org/English/November2000/article175.shtml;.

RODRÍGUEZ DEL PINO, J.C., DÍAZ ROCA, M., HERNÁNDEZ FIGUEROA, Z. and GONZÁLEZ DOMÍNGUEZ, J.D., 2007. Hacia la Evaluación Continua Automática de Prácticas de Programación. *Actas De Las XIII Jornadas De Enseñanza Universitaria De La Informática.Teruel*, pp. 179-186.

RODRÍGUEZ-DEL-PINO, J.C., RUBIO-ROYO, E. and HERNÁNDEZ-FIGUEROA, Z.J., 2012. A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-Plagiarism Features.

ROMLI, R., SULAIMAN, S.and ZAMLI, K.Z., 2010. *Automatic Programming Assessment and Test Data Generation a Review on its Approaches.* IEEE.

RUIZ, F. and LÓPEZ, P. *Arquitectura Lógica Del Sistema (En Desarrollo Orientado a Objetos).* Available from: http://www.ctr.unican.es/asignaturas/is1/is1-t11-trans.pdf.

SOMMERVILLE, I., 2005. Gestión de Calidad. In: Ingeniería del software 7/e Pearson Educación. *Gestión De Calidad*, pp. 587-604 ISBN 9788478290741.

SPACCO, J., et al, 2006. Experiences with Marmoset: Designing and using an Advanced Submission and Testing System for Programming Courses. *ACM SIGCSE Bulletin*, vol. 38, no. 3, pp. 13-17.

Unesco., 2008. *Using a Learning Management System in Education.* 08/25/2008, Available from: http://www.unescobkk.org/education/ict/online-resources/databases/ict-in-education-database/item/article/using-a-learning-management-system-in-education/.

WANG, T., et al, 2011. Ability-Training-Oriented Automated Assessment in Introductory Programming Course. *Computers & Education*, vol. 56, no. 1, pp. 220-226.

WINER, D., 2003. *XML - RPC Specification.* 06/30/2003, Available from: http://xmlrpc.scripting.com/spec.html.

YELMO, J.C., 2012. *La Calidad Del Software.* Lecture Notes of Software Engineering at ETSIT-UPM.

YUSOF, N., ZIN, N.A.M. and ADNAN, N.S., 2012. Java Programming Assessment Tool for Assignment Module in Moodle E-Learning System. *Procedia-Social and Behavioral Sciences*, vol. 56, pp. 767-773.

Annex I: Scientific article about the systematic literature review

PROGRAMMING ASSIGNMENTS AUTOMATIC GRADING: REVIEW OF TOOLS AND IMPLEMENTATIONS

Julio C. Caiza, Jose M. Del Alamo

Universidad Politécnica de Madrid (SPAIN)
j.caiza@alumnos.upm.es, jmdela@dit.upm.es

Abstract

Automatic grading of programming assignments is an important topic in academic research. It aims at improving the level of feedback given to students and optimizing the professor time. Several researches have reported the development of software tools to support this process. Then, it is helpful to get a quickly and good sight about their key features. This paper reviews an ample set of tools for automatic grading of programming assignments. They are divided in those most important mature tools, which have remarkable features; and those built recently, with new features. The review includes the definition and description of key features e.g. supported languages, used technology, infrastructure, etc. The two kinds of tools allow making a temporal comparative analysis. This analysis shows good improvements in this research field, these include security, more language support, plagiarism detection, etc. On the other hand, the lack of a grading model for assignments is identified as an important gap in the reviewed tools. Thus, a characterization of evaluation metrics to grade programming assignments is provided as first step to get a model. Finally new paths in this research field are proposed.

Keywords: Automatic Grading, Programming Assignments, Assessment.

1. INTRODUCTION

The first reference about programming automatic grading comes from 1965 [1]. It has been almost fifty years since it started and the number of students who requires of programming skills is growing. It is not only about computer science or information technology degrees. It includes students of many engineering degrees as well. Nowadays, almost every engineering program includes at least a basic programming course.

Another point to consider is the difficulty of getting programming skills by students. The main path to improve this has been the increment of solved programming exercises. This has to be accompanied with a good feedback. The feedback would be provided by a professor or a teaching assistant (teaching staff). Considering the number of engineering students and a good set of programming assignments, a manual assessment turns into a difficult or even an impossible task. The problem for the teaching staff is the excessive and maybe repetitive workload.

Several researches have reported the development of software tools to automate the process. These tools would give a feedback to orientate the students' learning, and will liberate teaching staff to do more productive work, giving focused help for instance. Almost every tool supports these goals and additionally tries to offer new features based on solve new gaps. These new gaps, among others, refer to plagiarism detection, secure test environment, controlled resource use, the diversity of criteria for grading [2] and the definition of pedagogical models [3].

There has been a good research in the field but now the problem is that there are many tools. If there is an institution which needs to implement a tool of this type or wants to develop a new tool it would be necessary to get a quickly and good sight about the state of the art. A tools' review will be helpful to find important features of already built tools. In addition this kind of work will give new ideas to improve or to build a new tool, which could be used broadly.

This paper reviews a set of mature and recent tools for automatic grading of programming assignments getting and showing key information. The next sections include the revision of related work to get information about the evolution in this research field. Next, a set of important tools is described, taking into account their key features. A comparison and an analysis will be shown to establish the current situation in this field. Then a grading metrics characterization is proposed. The last section includes conclusions and future work.

2. RELATED WORK

Many tools for automatic assessment of programming exercises have been built since the first appearance. A tools' review has been done before. It is necessary to know which the main conclusions of these works were. They will be useful to know if actual tools have already filled all the gaps.

To see an evolution, it is necessary to take a temporal perspective. Douce et al. in [4] make a good and quick characterization of these tools evolution until 2005. It identifies three generations of tools. The first one refers to times when working on operating systems and programming languages was necessary, and the assessment was only made considering a right or a wrong answer. The second generation refers to work with tools, which came with the operating system, to build new tools. C and Java languages were mostly used in development. The third generation is just around the time that this work was done. The main improvements in the reviewed tools are the orientation of using web-based technologies. It reports an increment in support for more programming languages as well.

Douce et al. [4] gave the next steps for automatic assessment of programming assignments. In [5] Ihantola et al. made a work covering tools developed since 2005. Taking these two works, it is possible to contrast them to show the improvement in some issues. These issues can be classified as technical, pedagogical and for a system adoption.

Regarding technical issues, Douce et al. indicated some research paths in [4], which included grading of GUI (Graphical User Interface) programs; meta-testing which refers to qualify applied tests; use and configuration of safe systems to test the programming assignments, the idea is to protect the host system form intentional or unintentional malicious code; integration of systems to avoid overwhelm the user, usually the idea would be integrate the tool with an LMS (Learning Management System), it can be reached using web-services; and support for web programming grading, it was because universities started to teach web programming and then grading this was necessary.

Ihantola et al. in [5] and Romli et al. in [6] had reported improvements in systems integration with LMS and in security for the host system. Then, issues like grading of GUI programs, meta-testing, and support for web programming stayed waiting for more research.

Regarding pedagogical issues, the reported works lack a common grading model. Every institution and even every teacher has his own way to establish a grade. So a reference model could be helpful. In reviews did in 2010, the correctness is reported as the main metric to grade. Some works started to use static and dynamic analysis as well, but in general, every work proposes its own set of metrics to grade. As a result, at that time, there was not a common approach yet; maybe the first step to build a model could be the metrics' characterization.

About feedback, there are some implications: quickly feedback could trigger trial-error practices, how much useful is the automatic feedback, and which is the adequate quantity of feedback. Some works try to provide flexibility to feedback and allow manual and automatic solutions [7].

Some tools have considered the implementation of plagiarism detectors. Usually they are in an additional module but not affect the grading process. Although it is clear that plagiarism detection would imply in a sanction.

With regard to systems' adoption, both works [5] and [6] showed that a big number of tools had been built but they are not broadly used. It is because every tool has been done considering specific requirements. An important way to increase the adoption was to work on open source projects. Some projects have done this and its acceptance has grown [7] but a definitive broadly used tool had not been reached. In [6] is proposed the building of a flexible and parameterizable system and it seemed a good path to reach the goal.

3. TOOLS' REVIEW

The common goal to build or to use these kinds of tools is to improve programming skills in the students, paying special attention to beginner students. The skills will be improved through solving many programming exercises. Students can go on the problems as quickly as they get good feedback. It would help them to understand their mistakes and to improve their skills.

Additionally students get a real benefit, which is to get a fair grade not dependent on personal considerations of the academic staff [2].

Considering the quantity of students in a regular class of engineering and a big number of programming exercises, it is not viable manual grading. Then the idea is not overwhelm the academic staff either, so another goal is to optimize the time of academic staff. The saved time could be used in more productive process like planning and designing the lectures or just giving more personal attention in focused problems.

As the research in this field increases, new goals are proposed. Thus, in [8], [9], [10] and [11] an extra goal is to get the integration with a LMS to improve the performance of the programming assignments assessment process. In [12] is proposed the use of services to reach this goal. In [13] one goal is to collect detailed information to research deeply the students' skill improvement process. More recently, Allevato and Edwars [14] have as a goal to get the interest of students using the popularity of smartphones and mobile applications.

3.1 Analyzed Key Features

The next key features have been defined considering they are important in a deployment case:

- Supported programming languages. It is a very important feature when it is considered to make a quickly implementation. It could define the use or not of a tool.
- Programming language used to develop the tool. This feature has great relevance when there is a set of policies respect of the software used in an institution. In the case of customization or maintenance, it would be a valuable feature to choose a tool.
- Logical architecture. It is an important feature when a modification of the tool is being considered. This architecture will show the modularity, scalability and flexibility level. It could show how the different modules work and how the system could connect with other systems.
- Deployment architecture. It shows how the hardware over which the tool works is. It is helpful to know if a current environment will support the implementation of a tool. In the worst case it will indicate the resources needed and therefore will help to determine the cost of an implementation.
- Work mode indicates if the tool can work alone, for clear implementations; or if the tool can work as plugin, when it is supposed to work with another system, an LMS for instance.
- Evaluation metrics. It displays how the tool can establish a grade. It considers which metrics are being considered inside the grading process. Even, how the grade calculation is done.
- Technologies used by the tool. It is helpful when it is considered to deploy or to build a new tool. In the first case it would help to establish compatibility between the tool and a legacy system. It is useful for future maintenance as well. In the second case it is very helpful to know which technology (standards, protocols, libraries, etc.), could help to face a requirement.

3.2 Tools

3.2.1 CourseMarker

A tool developed in the Nottingham University to avoid the particular criteria of teaching staff. The main advantages are considered being scalability, maintainability, and security [2]. The supported programming languages for grading are Java and C++ and it has been built using Java. Its architecture shows 7 subsystems: login, it controls all the authentication process; submission, it receives the different submissions precisely; course, it stores information about the process; marking, it has in charge the grading process, and the storing of the submitted files and marks gotten; auditing, it has as responsibility to log all actions; and a subsystem to control the communication among the others.

As metrics to establish a grade it considers typography (indentations, comments, etc.), functionality through test cases, programming structures use, and verification in the design and relations among the objects.

It works with technologies like Java RMI (Remote Method Invocation) for communication among the subsystems, regular expressions to verify results and DATsys [15] to verify objects design.

Additional important features include: the capacity to work with feedback levels, the orchestration among subsystems is defined by a configuration file, feedback and grades can be customized, there is plagiarism detector when grading, submissions number and CPU quantity are configurable, and finally there are security considerations which are detection of malicious code and execution in a sandbox environment.

3.2.2 Marmoset

It has been built in the University of Maryland. Its main goal is to collect information about development process to improve the student skills [13]. Its main advantages are to make a complete snapshot about the student's progress, so the student development can be analyzed in detail; the use of different types of test cases (student, public, release, secret); and a personal support through comments' threads on the code.

Originally the paper reported grading of code written in Java, C, Ruby and Caml Objective. Now, the official web page[53] informs that it works with all different programming languages. The architecture includes: a J2EE (Java Enterprise Edition) webserver, a SQL database, and one or more build servers. These last are used in a safe and lonely environment to prevent effects of possible malicious code. The build servers' disposition helps to provide scalability and security. The metrics to establish a grade include dynamic and static analysis. The dynamic analysis is done through test cases.

3.2.3 WebCat

The main features are the extensibility because of its plugins-based architecture and a grading method based on how well students grade their own code [7]. The architecture design provides a set of important features: security, it is provided through means like authentication, erroneous or dangerous code detection; portability, because it has been built as a Java servlet; extensibility and flexibility, it is inherent to the architecture; and support for manual grading as well, it is because the academic staff can check students' submissions and enter comments, suggestions, and grade modifications. The official wiki[54] affirms that it is the only tool that integrates all these features.

The tool supports Java, C++, Scheme, Prolog, Standard ML, and Pascal, but it offers flexibility to support any programming language. The grade is based on code correctness (how many tests are passed), test completeness (which parts of the code are actually executed), and test validity (test accurate-consistent with the assignment). Additionally plugins can provide more metrics for evaluations (static analysis for instance). Additional features include: there are a lot of plugins for Eclipse and Visual Studio .NET IDEs, and it has a GNU/GPL license.

3.2.4 Grading Tool by Magdeburg University

It has a really interesting goal, which is providing a tool which is not forced to work with a given LMS, but avoiding the use of two systems independently [12]. It can be reached using services. It shows a configurable focus. Then there are selectable components like the compiler, the language interpreter, the grading method, and the data set. The submissions' number, and time features are configurable as well.

The tool uses dynamic tests, compilers and interpreters to establish the grade. The supported languages are Haskell, Scheme, Erlang, Prolog, Python, and Java. The architecture is very interesting. It considers three servers: the front-end, it will be an LMS system; the spooler server, it controls the request, the submissions queues and the back-end calls; and the back-end servers, which are the modules to evaluate a programming language. To communicate the servers, XML-RPC (Remote Procedure Calls) has been used.

[53] http://marmoset.cs.umd.edu/
[54] http://wiki.web-cat.org/WCWiki/WhatIsWebCat

3.2.5 JavaBrat

It is a tool reported in [8], and built as a master thesis in San José State University. It gives support for two programming languages, Java and Scala. It uses Java to develop the grader software and PHP to build a plugin for Moodle. The design includes three important modules: a Moodle server with a plugin; a module which contains the graders depending on language and a repository of problems; and the last module is Javabrat which has a set of services to call graders and problems.

Although it can works as a Moodle's plugin, this tool can work alone through a web interface developed as part of the project. This web interface was developed using JSF (Java Server Faces) 2.0. The services are implemented using JAX-RS.

The work was centered in develop the web interface and the problems' repository. Then the grading process is not very complex and is based on correctness, which is determined by test cases. It is a semi automatic tool because it is necessary a revision of the report generated when the grading process is done.

3.2.6 AutoLEP

A tool developed in Harbin Institute of Technology and which is presented in [16]. It has as main feature the combination between static and dynamic analysis to give a grade. The dynamic analysis refers to evaluating the correctness using test cases. The static analysis doesn't need to compile or execute the code. It is just about to make a syntactic and semantic analysis and it is reported as main difference with previous works.

The architecture includes: the client, a computer used by a student, it does the static analysis and can provide of a quickly feedback; a testing server which has to do the dynamic analysis; and a main server which has to control the information of the other components to establish a grade.

3.2.7 Petcha

A tool developed in University of Porto. Its main goal is the building of an automatic assistant to teach programming [11]. An important feature is the coordination among existing tools like IDEs (Integrated Development Environment), LMSs and even automatic graders. It supports the programming languages that IDEs do. The tested IDEs are Eclipse and Visual Studio.

Its architecture is defined as modules for every connected tool. Then, there is a module for the LMS, the IDE, the exercises repository, and for the assessment engine. It relies on some technologies to guarantee interoperability: IMS Common Cartridge as format to build packages with resources and metadata, IMS Digital Repositories Interoperability and bLTI (Basic Learning Tools Interoperability). Additionally it used JAWS (Java Web Start) to build the client interface and it is working with MOOSHAK [17] as assessment engine.

3.2.8 JAssess

It has been built by researchers in two universities in Malaysia, University of Technology and Tun Hussien Onn University [10]. Their goal is to have only one interface to access the assessment process. JAssess is presented as an integrated tool with Moodle.

Their architecture shows the next modules: Moodle server, MySQL Server, JAssess, and JAssesMoodle to communicate Moodle and JAsses.

About supported languages it only supports Java, and precisely it is the language used to build the tool. Then it used libraries as Java File, Java Unzip, Java Runtime, Java Compiler and Java Reflection. About the metrics considered to grade, it is a weakness for the tool because it only depends on compilation. The evaluation process is not completely automatic.

3.2.9 RoboLIFT

The main approach is to get interest of students in programming using the popularity of mobile applications and smartphones [14]. The increasing market of android smartphones and applications makes increase the interest of students. This knowledge will be helpful when they will finish their studies as well. The tool supports grading of Android applications.

The tool is based on WebCat [7], so the architecture will be the same with an additional variation. The variation is the use of Robolectric[55], which is software to accelerate the grading process. The tool uses the development tools for Eclipse provided by Google.

The unit testing is considered as metric to grade. The tests are of two sorts, public and private tests. The students know the first one kind, and the second type is only used in the definitive submission.

3.2.10 Virtual Programming Lab

A tool built in Las Palmas University [9]. The goals of the project include to provide the students with many programming assignments, and to support the managing and grading process. It can be obtained through the integration with a LMS system. The tool supports many programming languages including Ada, C, C++, C#, FORTRAN, Haskell, Java, Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL, and VHDL.

The architecture includes three modules: a plugin for Moodle, which allows the integration with the submission and grades modules in Moodle; a code editor based on browser, which allows coding without the necessity of an installed compiler; and a jail server, which hosts the environment where the assignment will be evaluated. To develop this tool they have worked with PHP to build the Moodle plugin. To implement the jail server, C language has been used. Every language has an associated shell script for evaluation as well. The communication between Moodle and jail servers is done with XML RPC. The jail server gives the services through a Linux program called Xinetd. In addition the jail server implements a safe environment with the Chroot Linux program.

For grading it consider the correctness, evaluated through test cases by default. The test cases are specified in an own and easy syntax. The default scripts, which evaluate the programs, can be changed to improve the evaluation method.

Additionally, this tool has some interesting features like: it is built under GNU/GPL license, it allows automatic and semiautomatic process, it includes a plagiarism control tool, and it provides configurable features for every assignment.

3.2.11 Moodle extension by Slovak University of Technology Bratislava

It is presented in [18]. Its main goal is managing and modeling digital systems using HDL (Hardware Description Language). Then the work reports managing features like assignments managing, and user type definitions. The only language supported is VHDL.

The tool evaluates a submission based on: compilation and syntactic analysis, functionality doing comparisons with a model, and then through a stage to detect plagiarism.

4. COMPARISON AND ANALYSIS

The key features of each tool can be used to identify the real improvements since the last reviews [5] [6] were carried out.

To analyze the improvements through the time, two tables with tool's features are shown. The Table 1 join tools built a few years ago, previous to the work presented by Ihantola [5] which have been updated continuously. Precisely by their maturity, they count with really good features and in some cases with a broad use.

The second table joins more recent tools, which have not been broadly used but that present new features and propose new research lines even.

Firstly it is necessary to take into account which were the pending issues reported until 2010. They were mentioned in an earlier section; technical issues which include lack of a GUI grading tool, meta-testing, and support for web programming; pedagogical issues including lack of a model to grade, trial-error practices, adequate quantity of feedback, and plagiarism; and adoption issues.

As it can be seen most of these issues have been solved. Thus, RoboLIFT has the feature of grade GUI applications because this uses LIFT, a library included in the WebCat project to

[55] http://pivotal.github.com/robolectric/index.html

grade GUIs. Web programming languages have been considered as well, VPL can grade PHP programs for instance. Trial-error practices has not been reported as a serious problem, it is because a quickly solution is the submissions' limitation. Technically it would not be difficult to implement. The amount of feedback has been considered in tools like CourseMarker, which offers the possibility to configure the level of detailed feedback to give the student.

Plagiarism has been seen as an important module inside an automatic assessment tool. Then, some projects have considered it already.

The adoption of a tool depends on some features, which include: how long the tool has been tested, if the tool has been developed as open source, how flexible, scalable, and configurable a tool is. For example many institutions have adopted VPL as it can be seen in its web page[56].

Table 1. Mature tools.

Tool's name	Main Features	Supported Languages	Work Mode	Grading Metrics
CourseMarker	Scalability, maintainability. Security, configurability. Plagiarism detection. Work with levels of feedback.	Java, C++.	Standalone	Typography. Functionality. Structures use. Objects design. Objects relations.
Marmoset	Detailed information. Language independence. Security and scalability for evaluation module. Apache 2.0 license	Any language.	Standalone	Dynamic and static analysis.
WebCat	Extensibility and flexibility based on plugins. Access security. Portability. Semi and automatic process. GNU GPL license.	Java, C++, Scheme, Prolog, Standard ML, and Pascal. Flexibility for any language.	Standalone	Code correctness. Completeness. Test validity. Extensible by plugins.
Virtual Programming Lab	Moodle integration. Customizable grading mode. GNU GPL license. Plagiarism detection. Configurable activities. Jail environment.	Ada, C, C++, C#, Haskell, FORTRAN, Java,Octave, Pascal,PHP, Prolog, SQL, Ruby,Python, Scheme,Vhdl.	Moodle plugin.	Correctness based on test cases. Open for new methods.
Grading Tool (Magdeburg University)	Use of services. Configurable evaluation process.	Haskell, Scheme, Erlang, Prolog, Python, Java.	LMS extension.	Compilation. Execution. Dynamic tests.

Following the temporal comparison, almost all issues reported in the last review have been covered, but for meta-testing. However, it does not imply that everything is worked out, as we will discuss later on the conclusions.

There are a good number of tools for automatic assessment in programming. Then, does it make sense to continue building new ones? Usually the main reason to build a new tool is that the existing ones do not fulfil our requirements. If this is the case then it may be a good idea to get the tool and extend it through a plugin.

[56] http://vpl.dis.ulpgc.es/

Table 2. Recently developed tools.

Tool's name	Main Features	Supported Languages	Work Mode	Grading Metrics
JavaBrat	Use of services. LMS integration.	Java, Scala	Moodle plugin. Standalone.	Correctness.
AutoLEP	Static and dynamic analysis to grade.		Standalone	Static analysis. Dynamic analysis.
Petcha	Coordination among existing programming-support tools. Use of technology for interoperability.	Languages supported by Eclipse and Visual Studio	Standalone	Based on test cases.
JAssess	Moodle integration.	Java	Moodle plugin.	Compilation
RoboLIFT	Grading mobile applications. GUI grading.	Java	Standalone	Unit testing (public and private)
Moodle ext. (Slovak University of Technology)	Oriented for digital systems. Plagiarism detection.	Vhdl.	Moodle plugin.	Compilation. Syntactic analysis. Functionality by comparison.

Table 1 shows important information that supports the reuse of tools; this is based on existing tool features like extensibility, flexibility and configurability. The information in Table 2 shows that Java is the most common supported language by recent tools. Older tools have already supported this language and this cannot be a sufficient reason to build a new tool. If new support for a given language is necessary, it can be done through adding a new submodule or plugin to extensible tools.

An important fact is the use of LMSs in most universities. The ideal thing would be to seamlessly use the automatic tool within the LMS. Some recent tools are considering the integration with an LMS but they do not provide features like scalability, flexibility, and maintainability as the older ones. Maybe the next step to evolve with automatic tools is to add the LMS integration to the features of the more mature tools. Probably it could be reached by a redesign of the architecture of a mature tool. It means, the architecture could consider flexibility for the tool's front-end. This flexibility can be reached through services. The front-end could be a module in the LMS or a module developed in any technology for user interfaces.

Finally, it can be seen that metrics to grade are not normalized. Every tool has considered its own set. It is a real problem, which has come about since the automatic tools have appeared. This problem will be treated in the next section.

5. GRADING METRICS ISSUES

The lack of a common model to grade is still an important problem. First, every institution and even every teacher has his own critter to grade an assignment. Additionally, as Rodriguez in [19] says, it is necessary to recognize that some metrics cannot be measured. The creativity or the right sense of a comment cannot be determined by an automatic tool. In spite of these facts, and taking into account the importance of defining a set of metrics, a characterization of metrics is proposed below as a first step to get a grading model.

The importance of a characterization can be inferred by seeing the Tables 1 and 2. Every tool has its own set of metrics to establish a grade, e.g. one tool just includes the compilation, and another considers extensibility of metrics through plugins. Additionally, some tools refer to the same metric by different names.

Looking at all the metrics expressed in the previous tables as well as criteria from the software engineering discipline; Table 3 shows a characterization for grading metrics.

Table 3. Grading metrics characterization

		Metric
Execution		Compilation
		Execution
Functional Testing		Functionality (system or method level)
Non functional Testing	Specific requirements	Specific requirement for an exercise
	Maintainability	Design
		Style
		Complexity
	Efficiency	Use of physical resources
		Execution times
		Processes number
		File or code size

Every metric could have an associated tool that evaluates it. For compilation, a language compiler; for execution, a language interpreter; for functionality, it can be used a program based on test cases (JUnit in Java for instance); for specific requirement, it will be necessary a particular program; for design, style and complexity, an external program will be needed (Checkstyle for style in Java for instance); for the last four metrics, it could be useful shell script programs.

Some tools offer the possibility of support any grading metric through the building of plugins. But it would be better to consider a complete evaluation process. This evaluation process would have as feature a high level of configurability to support any metric. The goal is not see just a metric; it is to consider the whole grading process.

6. CONCLUSIONS AND FUTURE WORK

This work has reviewed the state of the art of automatic tools for programming assignments assessment. A set of requirements and key features for these tools has been described, and a comparison and analysis of existing tools have been carried out. As a result, a clear snapshot of their current status was provided, showing the lack of a common grading model as the major issue detected. A grading metrics characterization has been proposed as a first step to get a grading model.

The future research lines point towards the consolidation in one tool of several features like LMS integration (without a dependency on a given one), flexibility, scalability, maintainability, portability, and security; and the establishment of a configurable evaluation process where the metrics can be selected as the teacher needs.

7. ACKNOWLEDGMENT

The first author would like to extend thanks to *Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación* of the Ecuadorian Government, which provided him with financial support during his master studies.

In addition, this work has been partially supported by the Universidad Politécnica de Madrid, as part of the "*Ayudas a la Innovación Educativa y a la Mejora de la Calidad de la Enseñanza*" programme.

REFERENCES

[1] Forsythe, G. E., Wirth, N. (1965). Automatic Grading Programs. *Commun ACM,* vol. 8, pp. 275-278.

[2] Higgins, C. A., Gray, G., Symeonidis, P., Tsintsifas, A. (2005). Automated Assessment and Experiences of Teaching Programming. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 5.

[3] Choy, M., Lam, S., Poon, C., Wang, F., Yu, Y., Yuen, L. (2008). Design and Implementation of an Automated System for Assessment of Computer Programming Assignments. Advances in Web Based Learning–ICWL 2007, pp. 584-596.

[4] Douce, C., Livingstone, D., Orwell, J. (2005). Automatic Test-based Assessment of Programming: A Review. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 4.

[5] Ihantola, P., Ahoniemi, T.,Karavirta, V., Seppälä, O. (2010). Review of Recent Systems for Automatic Asessment of Programming Assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research, pp. 86-93.

[6] Romli, R., Sulaiman, S., Zamli, K. Z. (2010). Automatic Programming Assessment and Test Data Generation a Review on its Approaches. Information Technology (ITSim), 2010 International Symposium, pp. 1186-1192.

[7] Edwards, S. H., Perez-Quinones, M. A. (2008). Web-CAT: AutomaticallyGgrading Programming Assignments. ACM SIGCSE Bulletin, vol. 40, pp. 328-328.

[8] Patil, A. (2010). Automatic Grading of Programming Assignments.

[9] Rodríguez-del-Pino, J. C., Rubio-Royo, E., Hernández-Figueroa, Z. J. (2012). A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-plagiarism Features.

[10] Yusof, N., Zin, N.A.M, Adnan, N.S. (2012). Java Programming Assessment Tool for Assignment Module in Moodle E-learning System. Procedia-Social and Behavioral Sciences 56 (56), pp. 767-773.

[11] Queirós, R. A. P., Leal, J. P. (2012). PETCHA: A Programming Exercises Teaching Assistant. Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, pp. 192-197.

[12] Amelung, M., Forbrig, P., Rösner, D. (2008). Towards Generic and Flexible Web Services for E-assessment. ACM SIGCSE Bulletin, pp. 219-224.

[13] Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., Padua-Perez, N. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. ACM SIGCSE Bulletin, vol. 38, pp. 13-17.

[14] Allevato, A., Edwards, S.H. (2012). RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications. Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 547-552.

[15] Higgins, C., Symeonidis, P., Tsintsifas, A. (2002). Diagram-based CBA Using DATsys and CourseMaster. Proceedings of Computers in Education International Conference, pp. 167-172.

[16] Wang, T., Su, X., Ma, P., Wang, Y., Wang, K. (2011). Ability-training-oriented Automated Assessment in Introductory Programming Course. Computer. Education, Elsevier, vol. 56, pp. 220-226.

[17] Leal, J., Silva, F. (2003). Mooshak: a Web-based Multi-site Programming Contest System. Software: Practice and Experience, vol. 33, pp. 567-581.

[18] Jelemenská, K. Čičák, (2012). Improved Assignments Management in MOODLE Environment. INTED2012 Proceedings, pp. 1809-1817.

[19] Rodríguez del Pino, J. C., Díaz Roca, M., Hernández Figueroa, Z., González Domínguez, J. D. (2007). Hacia la Evaluación Continua Automática de Prácticas de Programación. Actas De Las XIII Jornadas De Enseñanza Universitaria de la Informática, pp. 179-186.

Annex II: Description of important methods in the classes Orchestrator, CommandExecutor and GradingSumoduleProgram

Orchestrator class

Orchestrator

```
public Orchestrator(java.lang.String confFilename)
```

It instantiates a new *Orchestrator* given the name of the XML configuration file.

Parameters:

`confFilename` - the configuration filename

---

fillSubmissionConf

```
public boolean fillSubmissionConf()
```

It fills data in the *submissionConf* instance through mapping provided by JAXB and using the *SubmissionConf* and *GradingSubmoduleConf* classes.

Returns:

`true` if the *submissionConf* object was filled successfully

---

createGSubmoduleProgramInstance

```
public java.lang.Object createGSubmoduleProgramInstance(
    java.lang.String className,
    es.upm.dit.tfm.grad.pars.GradingSubmoduleConf gradingSubmoduleConf)
```

It creates a class instance in the runtime given its name as *String* and **using Java Reflection technology**[57]. This instance will receive a *GradingSubmoduleConf* instance used to read, and then to update, data needed by the grading-submodule associated program. The instance created is returned as an object. Its correspondent class extends the *GradingSubmoduleProgram* abstract class.

Parameters:

`className` - the class name

`gradingSubmoduleConf` - the *GradinSumoduleConf* instance

Returns:

The object just instantiated

---

[57] http://docs.oracle.com/javase/tutorial/reflect/index.html

invokeRunMethod

`public void **invokeRunMethod**(java.lang.Object gradingSubmoduleProgram)`

This is implemented using Java Reflection technology. It invokes a method called *run* associated to an object received as argument. This object should be an instance of a *GradingSubmoduleProgram* subclass. The run method is defined in the *GradingSubmoduleProgram* abstract class and has to be implemented in its subclasess.

Parameters:

`gradingSubmoduleProgram` - the object that owns the *run* method to be invoked

---

orchestrate

`public void **orchestrate**()`

It orchestrates the process. Based on the list of *GradingSubmoduleConf* inside the *submissionConf* instance, this will call the *run* method of every grading-submodule associated program. Specifically it will create an instance of a *GradingSubmoduleProgram* subclass and then it will invoke the *run* method inside that instance.

---

finalProcessing

`public void finalProcessing()`

It makes the final processing with the execution results of every grading-submodule program. This includes the calculation of the final grade, the collection of the detailed comments, and the creation of a general comment.

---

printResponse

`public void **printResponse**()`

It prints the response in a format, which allows being interpreted by the Jail server program before send it in an HTTP response. The response includes the final grade, the general comment and the detailed comments previously generated.

---

getLogCode

`public java.lang.String **getLogCode**()`

It creates a log code. This includes a student identificator, an activity identificator and a code for the submission.

Returns:

The log code

CommandExecutor class

CommandExecutor

```
public CommandExecutor(java.lang.String oneLineCommand)
```

It instantiates a new *CommandExecutor* given a system command expressed in 1 line.

Parameters:

`oneLineCommand` - a system command expressed in one line

---

execute

```
public void execute()
              throws java.io.IOException,
                     java.lang.InterruptedException
```

It executes a command list and catches information from the standard output and from the standard error. This information is saved inside the object. This method throws some exceptions in this level, which will be caught by the orchestrator. This last will save a log with the incidences.

Throws:

`java.io.IOException` - Signals that an I/O exception happened.

`java.lang.InterruptedException` - Signals that an interruption exception happened.

GradingSumoduleProgram class

GradingSubmoduleProgram

```
public GradingSubmoduleProgram(
    es.upm.dit.tfm.grad.pars.GradingSubmoduleConf gradingSubmoduleConf)
```

It instantiates creates a new *GradingSubmoduleProgram* subclass.

Parameters:

`gradingSubmoduleConf` - the *GradingSubmoduleConf* instance associated.

---

run

```
public abstract void run()
```

It contains code written by the administrator or the teaching staff, which aims to evaluate source code files considering a grading criterion. It is possible to use any commands but finally it is necessary updating the *GradingSubmoduleConf* instance. It is suggested using the methods of the *GradingSubmoduleConf* class to recover the parameters and the action file list, and the methods defined in this class to save implementation time of new subclasses.

---

updateGradingSubmoduleConf

```
public void updateGradingSubmoduleConf(java.lang.String state,
                                       double grade,
                                       java.lang.String comments)
```

It updates the *GradingSubmoduleConf* associated.

Parameters:

`state` - The state of the program's execution

`grade` - The grade assigned to the source code considering the associated criterion.

`comments` - The comments regarding the evaluation and considering the associated criterion.

---

executeCommand

```
public void executeCommand(java.lang.String command)
```

It executes one line command. It makes automatically an instantiation of *CommandExecutor* and calls the execution method. It catches exceptions and additionally identifies errors from the standard error. The exceptions and error are written in the log file.

Parameters:

`command` - the one line command

---

stdOutString

```
public java.lang.String stdOutString()
```

It gets the standard output of the command execution as String.

Returns:

The standard output as String

---

stdErrString

```
public java.lang.String stdErrString()
```

It gets the standard error of the command execution as String.

Returns:

The standard error as String

---

getLogCode

```
public java.lang.String getLogCode()
```

It gets the log code.

Returns:

The log code

Annex III: Detailed description of the case studies (in Spanish)

PRACTICA 3: CREACIÓN DE CLASES

Objetivos

- Creación de clases, uso de arrays y sentencias de control

Documentos y ficheros proporcionados

Se proporciona:

- Fichero con comentarios de las clases `CoordenadaEsferica`, `CoordenadaCartesiana` y `SateliteGPS`.
- Fichero con una plantilla de la clase ReceptorGPS

Actividades a desarrollar

Creación de las clases CoordenadaEsferica, CoordenadaCartesiana y SateliteGPS

El código de las clases CoordenadaEsferica, CoordenadaCartesiana y SateliteGPS se proporciona al alumno. No es necesario modificar estas clases. En esta primera actividad hay que:

1. Crear un proyecto llamado p3
2. Crear un paquete llamado `es.upm.dit.prog.p3`
3. Incorporar las clases `CoordenadaEsferica`, `CoordenadaCartesiana` y `SateliteGPS`, a partir de los ficheros proporcionados.
4. Leer la documentación de las clases para conocer los métodos disponibles.

Creación de la clase ReceptorGPS

Esta clase gestiona un conjunto de satélites GPS detectados y que se emplearán para determinar la posición exacta de un objeto. Se declarará un array en el que se almacenan los satélites. El número máximo de satélites es una constante (N_MAX_SATELITES), cuyo valor será 5.

Se proporciona una plantilla de la clase ReceptorGPS, que incluye las signaturas de los métodos a realizar. Incluye un método (`toString`) que retorna un `String` con los parámetros de los satélites en un momento dado. Se puede emplear para generar trazas.

Se deben desarrollar los siguientes métodos:

anadirSatelite

Signatura: `public void anadirSatelite(SateliteGPS unSatelite)`

Descripción: Este método añade un satélite. Si ya se ha añadido el número máximo de satélites, se debe descartar el satélite más lejano respecto a la posición del receptor.

Excepciones:

- Si el valor del parámetro es null, se debe elevar la excepción NullPointerException
- Si ya se ha añadido el satélite que se pasa como parámetro, se debe elevar la excepción Exception

getNumeroSatelites

Signatura: `public int getNumeroSatelites()`

Descripción: Retorna el número de satélites almacenados.

eliminarSatelite

Signatura: `public void eliminarSatelite(SateliteGPS unSatelite);`

Descripción: Este método elimina un satélite.

Excepciones:

- Si el valor del parámetro es null, se debe elevar la excepción NullPointerException
- Si el satélite que se pasa como parámetro no se ha añadido previamente, se debe elevar la excepción Exception

estaSatelite

Signatura: `public boolean estaSatelite(SateliteGPS unSatelite)`

Descripción: Este método indica si un satélite forma parte de los detectados por el receptor. Para esta comprobación se utilizará únicamente el nombre del satélite (atributo nombre de la clase SateliteGPS, recuperable con el método getNombre).

Excepciones:

- Si el valor del parámetro es null, se debe elevar la excepción NullPointerException

getDistanciaSateliteMasCercano

Signatura: `public double getDistanciaSateliteMasCercano()`

Descripción: Calcula la distancia a la que se encuentra el satélite más cercano de la posición del receptor.

Excepciones:

- Si no se han añadido satélites al receptor, se debe elevar la excepción Exception

getDistanciaSateliteMasCercano

Signatura:
```
public double getDistanciaSateliteMasCercano
    (CoordenadaEsferica unaPosicion)
```

Descripción: Calcula la distancia a la que se encuentra el satélite más cercano a una posición que se pasa como parámetro.

Excepciones:

- Si el valor del parámetro es null, se debe elevar la excepción NullPointerException
- Si no se han añadido satélites al receptor, se debe elevar la excepción Exception

Evaluación

La evaluación se basará en los siguientes aspectos:

- **Corrección de ReceptorGPS.java**: A la clase desarrollada por el alumno se le pasará una batería de pruebas para comprobar su correcto funcionamiento. La clase desarrollada debe cumplir la especificación de la clase referente a las funciones a realizar y a las excepciones que se deben lanzar.
- Es necesario **seguir las indicaciones** en cuanto a nombres de los métodos de prueba, nombre del fichero a entregar y estructura de directorios. El incumplimiento de estas normas supondrá una **bajada de al menos 4 puntos**. Si la práctica no compila correctamente no se evaluará.

Entrega de la práctica

Instrucciones para la entrega de la práctica:

- La práctica se deberá entregar en el moodle global de la asignatura antes de las 23:59 del día 11 de abril de 2012.
- El nombre de la entrega es “Práctica 3”

- El nombre del fichero que incluya los ficheros requeridos se debe llamar "`practica3.zip`".
- *Código fuente*: hay que entregar el fichero de la clase desarrollada `ReceptorGPS.java`.
- El contendido del fichero comprimido de la entrega debe estar en el directorio relativo: "p3/src/es/upm/dit/prog/p3/".
- El código proporcionado, se debe compilar sin errores. Se valorará la ausencia de avisos (warnings). No se debe modificar la signatura de los métodos proporcionados en la plantilla. En caso contrario, se producirán errores de compilación al pasar la batería de pruebas.
- **El trabajo es individual. La copia de entregas supondrá el suspenso en la asignatura de forma automática, tanto para quien copia como para quien se deja copiar. Se recuerda que está permitido:**

  ✓ Discutir el trabajo con otros;
  ✓ ayudar a otros a depurar su trabajo;
  ✓ usar código publicado en el sitio web de la asignatura;
  ✓ usar código publicado en otros sitios, citando la procedencia.

  Por el contrario, no está permitido:

  ✓ Realizar el trabajo en grupo;
  ✓ copiar el trabajo de otro alumno, ni permitir la copia del propio trabajo, ni siquiera parcialmente.
  ✓ usar código publicado sin citar el origen.
  El trabajo debe ser realizado individualmente y entregado personalmente por el alumno según se ha indicado.

Práctica 5: Relaciones entre clases y colecciones

Objetivos

- Uso de herencia y polimorfismo.
- Uso de bibliotecas: Colecciones, Conjuntos y Listas.
- Uso de arrays.
- Documentación del código: javadoc.

Documentación

Se proporciona:

- Fichero con código fuente y comentarios de las clases CoordenadaCartesiana, POI, Gasolinera y Hotel.

Actividades a desarrollar

En esta práctica el alumno deberá crear una clase llamada ***NavegadorGPS***, para simular algunas de las funciones comunes de un navegador. En particular, nos centraremos en la gestión de una lista de Puntos de Interés (*Point of Interest – POI).*

Un **POI** viene definido por un nombre, que es una cadena de texto con el nombre del establecimiento que representa, y una localización que indica la posición en el espacio del POI, expresada en coordenadas cartesianas.

Hay distintos tipos de POIs, como gasolineras y hoteles. Estas abstracciones se modelan en la práctica con dos clases particulares, que extienden las propiedades de un POI genérico:

- **Gasolinera**: Esta clase almacena información sobre el tipo de combustible que se puede encontrar: hasDiesel y hasSinPlomo, que devuelven un booleano que indica si la gasolinera surte ese tipo de combustible o no.
- **Hotel**: Esta clase almacena información sobre establecimientos hoteleros, su precio y su categoría. En particular, dispone de métodos para conocer las estrellas (getEstrellas) y el precio (getPrecio) del hotel.

Creación de las clases CoordenadaCartesiana, POI, Gasolinera y Hotel

El código de las clases *CoordenadaCartesiana, POI, Gasolinera y Hotel* se puede encontrar en un fichero adjunto. En esta primera actividad hay que:

1. Crear un proyecto Java llamado p5
2. Crear dentro del proyecto un paquete llamado es.upm.dit.prog.p5
3. Incorporar las clases ofrecidas al paquete es.upm.dit.prog.p5 del proyecto p5

Creación de la clase NavegadorGPS

El alumno deberá crear la clase **NavegadorGPS** para gestionar la información sobre diversos POIs. En particular, la clase NavegadorGPS debe disponer, al menos, de los siguientes métodos públicos, con el comportamiento que se detalla a continuación de cada método:

- *public **NavegadorGPS** ()*
    - Crea un objeto NavegadorGPS. La localización del navegador se establecerá por defecto en la CoordenadaCartesiana 0, 0, 0, y no habrá almacenados POIs.
- *public void **setPosicion**(CoordenadaCartesiana p) throws Exception*
    - Fija una nueva localización para el NavegadorGPS.
    - **Excepciones**: El método deberá elevar excepciones de tipo *Exception* en los siguientes casos:
        - El argumento pasado es nulo.
- *public CoordenadaCartesiana **getPosicion**()*
    - Recupera la posición del NavegadorGPS.
- *public void **setPOIs**(POI[] nuevosPOI) throws Exception*
    - Sustituye los POIs almacenados por los pasados como parámetro.
    - **Excepciones**: El método deberá elevar excepciones *Exception* en los siguientes casos. La lista existente de POIs no se debe modificar si ocurre alguna excepción:
        - Alguno de los elementos de nuevosPOI es nulo.
        - El argumento *nuevosPOI* recibido es nulo.
- *public void **setPOIs**(Set<POI> nuevosPOI) throws Exception*
    - Se espera el mismo comportamiento que para el método anterior.
- *public void **setPOIs**(List<POI> nuevosPOI) throws Exception*
    - Se espera el mismo comportamiento que para el método anterior.
- *public void **addPOI**(POI nuevoPOI) throws Exception*
    - Añade el POI *nuevoPOI* a los POIs almacenados.
    - **Excepciones**: El método deberá elevar excepciones *Exception* en los siguientes casos. La lista existente de POIs no se debe modificar si ocurre alguna excepción:
        - El argumento *nuevoPOI* recibido es nulo.

        - El argumento *nuevoPOI* recibido ya se encuentra entre los POIs almacenados.
- *public void* ***removePOI****(POI viejoPOI) throws Exception*
    - Elimina el POI viejo*POI* de los POIs almacenados.
    - **Excepciones**: El método deberá elevar excepciones *Exception* en los siguientes casos. Los POIs almacenados por el NavegadorGPS no se debe modificar si ocurre alguna excepción:
        - El argumento *viejoPOI* recibido es nulo.
- *public POI[]* ***getPOIs****()*
    - Devuelve un array de POIs, con los POIs almacenados en el objeto. Si no hubiera POIs almacenados deberá devolver un array sin elemento, pero nunca un objeto null.
- *public Gasolinera[]* ***getGasolineras****()*
    - Devuelve un array de Gasolineras, con las Gasolineras almacenadas en el objeto.
- *public Hotel[]* ***getHoteles****()*
    - Devuelve los Hoteles almacenados por el navegador.
- *public Gasolinera* ***getGasolineraMasCercana****()*
    - Devuelve la Gasolinera más cercana al NavegadorGPS de entre todas las almacenadas.
- *public Gasolinera[]* ***getGasolinerasDiesel****()*
    - Devuelve un array de Gasolineras con las Gasolineras que ofrecen combustible diesel.
- *public Hotel[]* ***getHotelesCercanos****(int distanciaMaxima)*
    - Devuelve un array de Hotel con los hoteles que se encuentran a una distancia inferior que la pasada como parámetro.
- *public Hotel[]* ***getHotelesBuenos****(int estrellasMinimas)*
    - Devuelve un array de Hotel con los hoteles que tienen un número de estrellas igual o mayor al pasado como parámetro.

La corrección del código desarrollado **se valorará hasta con 8 puntos en la calificación de la práctica**. Puede usar el corrector que está disponible en el área de entrega para tener una idea de la corrección de su código, aunque la calificación obtenida por este método puede no ser definitiva, ya que estará sujeta a una revisión posterior.

Documentación de la clase NavegadorGPS

El alumno deberá documentar de forma adecuada la clase NavegadorGPS, incluyendo comentarios para la clase, los atributos y los métodos desarrollados. Para ello deberá hacer uso de distintas etiquetas javadoc.

La sintaxis de javadoc es estricta. Es necesario seguirla para que la documentación generada tenga la información tal y como la esperan los lectores. Es necesario comentar todos los métodos, incluido el constructor. En particular, se deben incluir las siguientes etiquetas, con el correspondiente formato:

- @author nombre_autor

- @version identificador_versión
- @param nombre_parámetro texto_del_comentario
- @throws nombre_excepción texto_del_comentario
- @return texto_del_comentario
- El comentario de la funcionalidad de una clase, no lleva etiqueta y es aconsejable que acabe con un punto.

A continuación se comentan algunos fallos comunes y la forma de realizarlos correctamente:

- Se deben utilizar las etiquetas tal y como se han definido previamente. El uso de otras (Ej. @autor en lugar de @author) no está permitido. Su uso provocará que se penalice la calificación obtenida por el alumno.
- No se deben añadir caracteres a los nombres de excepción o de parámetro. Ej. No incluir comentarios como: "@param unPunto. texto". Se debe eliminar el punto después del nombre del parámetro.
- Indicar el tipo de la excepción. Un comentario incorrecto sería "@throws texto_comentario". Es necesario poner el tipo de la excepción. Un ejemplo correcto es: "@throws Exception texto_comentario". Lo mismo es aplicable a @param.
- No es recomendable poner varios comentarios "@throws Exception ..." para el mismo método cuando se puede elevar por diferentes causas. Poner una sola etiqueta y explicar las distintas razones que pueden elevarla.
- No utilizar nombres de excepciones que no se usan, aunque sea con fines aclaratorios. Por ejemplo, no incluir comentarios del tipo "@throws Fuera ....", sin que este sea el nombre real de la excepción. De lo contrario el analizador interpreta que se hace un uso incorrecto de la etiqueta, lo que penaliza la calificación del alumno.
- Incluir comentario javadoc en el constructor.

La presencia y adecuación de los comentarios **se valorará hasta con 2 puntos en la calificación de la práctica**. Puede usar el corrector que está disponible en el área de entrega para tener una idea de la corrección de la documentación de su código, aunque la calificación obtenida por este método puede no ser definitiva, ya que estará sujeta a una revisión posterior.

Creación de la clase de prueba PruebaNavegadorGPS

El alumno deberá crear una clase para realizar pruebas que verifiquen el correcto funcionamiento de NavegadorGPS. Su contenido, como en las prácticas anteriores, es un método main con instrucciones para llamar a los métodos de la clase a probar. El resultado de las llamadas a los métodos se puede escribir en la salida estándar para comprobar su correcto funcionamiento.

De forma complementaria, el alumno puede depurar su código para detectar los errores que pudiera haber detectado. En la página de la asignatura, dispone de varios tutoriales que le explican cómo depurar un programa, depuración de arrays, y uso de interfaces y herencia:

- https://moodle.lab.dit.upm.es/moodle/mod/url/view.php?id=4937
- https://moodle.lab.dit.upm.es/moodle/mod/url/view.php?id=5066
- https://moodle.lab.dit.upm.es/moodle/mod/url/view.php?id=5079

La evaluación de la práctica se basará en la ejecución de multitud de pruebas exhaustivas a los distintos métodos implementados por el alumno, utilizando para ello una clase similar. También se revisará la existencia y adecuación de la documentación del código.

Entrega de la práctica

Instrucciones para la entrega de la práctica:

- La práctica se deberá entregar en el moodle de la asignatura **antes de las 23:59 del martes 22** de mayo de 2012
- El nombre de la zona de entrega es "Práctica 5". Esta zona de entrega dispone de una herramienta que le sugerirá posibles fallos en su código.
- **Código fuente:** Hay que entregar los ficheros fuente de la clase *NavegadorGPS*.
- El código fuente debe estar en el directorio relativo "***p5/src/es/upm/dit/prog/p5***". No es necesario entregar los ficheros .class.
- El fichero que hay que entregar se genera como sigue:
    1. Seleccionar el proyecto ***p5***.
    2. Seleccionar la opción de menú "File", luego seleccionar "Export".
    3. En la ventana que aparece, seleccionar "General->Archive File". Pinchamos en 'Siguiente'.
    4. En la ventana que aparece, asegurarse de dar el **nombre** adecuado al fichero de salida: **Practica5**. **No utilice tildes ni espacios en blanco** para el nombre del fichero de salida. De lo contrario el sistema podría no detectar su entrega. Asegurarse también de que está **seleccionado**:
        - El **proyecto *p5***, y dentro del proyecto p5 la **carpeta src**.
        - **salvar en formato zip** y,
        - **crear estructura de directorios**.
- La evaluación se centrará en la corrección de los métodos desarrollados en la clase *NavegadorGPS,* y en la existencia y adecuación de comentarios en esta clase.
- El código entregado debe poderse compilar sin errores. **No utilizar tildes ni ñ para el nombre de los métodos o atributos**. En caso contrario el corrector no funcionará.

<u>AVISO MUY IMPORTANTE</u>: Se recuerda a los alumnos que **el trabajo es individual**, y que **la copia de entregas supondrá el suspenso en la asignatura** de forma automática**, tanto para quien copia como para quien se deja copiar**.

Se recuerda que **está permitido**:

- Discutir el trabajo con otros;

- ayudar a otros a depurar su trabajo;
- usar código publicado en el sitio web de la asignatura;
- usar código publicado en otros sitios, citando la procedencia.

Por el contrario**, no está permitido**:

- Realizar el trabajo en grupo;
- copiar el trabajo de otro alumno, ni permitir la copia del propio trabajo, ni siquiera parcialmente.
- usar código publicado sin citar el origen.

El trabajo debe ser realizado individualmente y entregado personalmente por el alumno según se ha indicado.