Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Markku Riekkinen

Integrating Stratum and A+ Functionalities in Moodle:

Architecture and Evaluation

Master's Thesis
Espoo, December 1, 2016

Supervisor: Docent Tomi Janhunen
Advisor: Docent Tommi Junttila

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Markku Riekkinen

Title:
Integrating Stratum and A+ Functionalities in Moodle: Architecture and Evaluation

Date: December 1, 2016 **Pages:** viii + 67

Major: Computer Science **Code:** SCI3042

Supervisor: Docent Tomi Janhunen

Advisor: Docent Tommi Junttila

Automated assessment within electronic learning (e-learning) uses computers for the grading of students' solutions to course assignments, which releases teachers from the burden of manually assessing the submissions and leaves them time for developing other course and teaching activities. The goal of this thesis was to modernize Stratum, an old platform for deploying arbitrary assignments that are automatically assessed and may also be randomly generated in order to provide personalized assignments for each student. The purpose of the modernization was to improve the maintainability and usability of the platform, while retaining its existing core functionality and in particular, its seamless integration in Moodle, a widely used e-learning platform that is also used as the official, university-wide platform at Aalto University, where it is known as MyCourses.

This thesis presents alternative approaches for the modernization and identifies the most suitable one for implementation. The selected approach involves another platform, A+, that outsources the implementation of assignments to external exercise services. In this thesis project, a new Moodle plugin was implemented that replicates A+ functionality so that both A+ and Moodle may utilize the same exercise services and function as a front end to courses with automatically assessed assignments. Furthermore, the exercise service framework used with A+, known as the MOOC grader, was extended to support personalized assignments. The new Moodle plugin was named Astra.

The new platform implemented in this thesis project realizes the requirements set for the modernization of Stratum. Assignments implemented for Stratum may be ported with feasible effort to the new platform as exercise services. In addition, the new platform replicates enough of the A+ functionality so that a typical course may freely select either Moodle or A+ as its front end without further modifications.

Keywords: e-learning, automated assessment, computer-aided assessment, software platform, Moodle plugin

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Markku Riekkinen

Työn nimi:
Stratumin ja A+:n toiminnallisuuksien integrointi Moodleen: arkkitehtuuri ja evaluointi

Päiväys:	1. joulukuuta 2016	**Sivumäärä:**	viii + 67
Pääaine:	Computer Science	**Koodi:**	SCI3042
Valvoja:	Dosentti Tomi Janhunen		
Ohjaaja:	Dosentti Tommi Junttila		

Automaattinen harjoitustehtävien arviointi on sähköisen oppimisen (e-oppiminen) osa, jossa tietokoneohjelma arvostelee opiskelijan tekemän tehtävän ratkaisun. Tällöin opettajan ei tarvitse käyttää aikaa tehtävien arvosteluun käsin ja aikaa voi käyttää enemmän muun opetuksen kehittämiseen. Tämän diplomityön tavoitteena oli uudistaa vanha Stratum-järjestelmä, joka on alusta automaattisesti arvioitavien tehtävien toteuttamiseen. Lisäksi Stratumilla voi toteuttaa satunnaisesti luotavia tehtäviä, jolloin jokaiselle opiskelijalle tarjotaan henkilökohtainen tehtävä. Uudistustyön tarkoituksena oli parantaa Stratumin ylläpidettävyyttä ja käytettävyyttä kuitenkaan menettämättä aiempaa ydintoiminnallisuutta. Erityisesti Stratumin saumaton integraatio Moodlessa haluttiin säilyttää. Moodle on laajasti käytetty sähköisen oppimisen alusta, jota käytetään myös Aalto-yliopistossa: kyseinen alusta tunnetaan Aallossa nimellä MyCourses.

Tämä diplomityö esittelee vaihtoehtoja uudistustyön toteuttamiseksi ja valitsee niistä parhaan vaihtoehdon. Valittu vaihtoehto viittaa myös erääseen toiseen alustaan, A+:aan, joka ulkoistaa automaattisesti arvioitavien tehtävien toteutuksen ulkopuolisiksi palveluiksi. Tässä diplomityössä toteutettiin uusi Moodle-liitännäinen, joka toistaa A+:n toiminnallisuutta siten, että Moodle voi käyttää samoja tehtäväpalveluja kuin A+ automaattista arviointia varten, jolloin opiskelijat näkevät ja palauttavat tehtävät Moodlessa. Lisäksi tässä työssä laajennettiin A+:n käyttämää tehtäväpalvelujen ohjelmistokehystä, "MOOC graderiä", jotta se voi tukea henkilökohtaisia tehtäviä kuten Stratum. Uusi Moodle-liitännäinen nimettiin Astraksi.

Tässä työssä toteutettu uusi alusta saavuttaa uudistustyölle asetetut tavoitteet. Stratumiin toteutetut vanhat tehtävät on mahdollista siirtää uudelle alustalle kohtuullisella vaivalla. Lisäksi uusi alusta muistuttaa riittävästi A+:aa, jotta tyypilliset kurssit voivat valita vapaasti, kumpaa alustaa ne käyttävät kurssialustana ja käyttöliittymänä, Moodlea vai A+:aa.

Asiasanat:	e-oppiminen, automaattinen arviointi, tietokoneavusteinen arviointi, ohjelmistoalusta, Moodle-liitännäinen
Kieli:	Englanti

Acknowledgements

I thank my supervisor, Docent Tomi Janhunen, and my advisor, Docent Tommi Junttila, for their never-ending support and guidance throughout this thesis project and the preceding work. I started working under their supervision in the development of the Stratum computerized learning environment already in 2013, and it continued until the beginning of this thesis project, which replaces the old Stratum with a new and improved platform. I am very grateful for these years and the exciting tasks I have been allowed to undertake.

I thank Professor Petteri Kaski and the Aalto Online Learning (A!OLE) project for funding this thesis project. I thank the development team of the A+ platform in the Learning + Technology group for their advice in designing the new platform developed in this thesis project. Particularly, I thank Teemu Lehtinen from the A+ team for his insight in the development of the MOOC grader framework; he also greatly contributed to the design of the algorithm for the regeneration of exercises.

I wish to thank Professor Pekka Orponen for suggesting the name, Astra, for the front-end component of the new platform. The name reflects that the new platform is influenced by both A+ and Stratum, and it continues the tradition of using Latin words as names. Finally, I thank my family for their support.

Per aspera ad Astra.

Espoo, December 1, 2016

Markku Riekkinen

Abbreviations and Acronyms

AMD	Asynchronous Module Definition
API	Application Programming Interface
CSS	Cascading Style Sheets
DBMS	Database Management System
DVD	Digital Versatile Disc
e-learning	electronic learning
ER	Entity-Relationship (model)
FK	Foreign Key (in databases)
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSAV	JavaScript Algorithm Visualization Library
JSON	JavaScript Object Notation
LCMS	Learning Content Management System
LMS	Learning Management System
LTI	Learning Tools Interoperability
MOOC	Massive Open Online Course
ORM	Object-Relational Mapping
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
PK	Primary Key (in databases)
QTI	Question and Test Interoperability
REST	Representational State Transfer
RST	reStructured Text
SaaS	Software as a Service
SCORM	Sharable Content Object Reference Model
SQL	Structured Query Language
STACK	System for Teaching and Assessment using a Computer algebra Kernel
SVG	Scalable Vector Graphics

URL	Uniform Resource Locator
U.S.	the United States (of America)
VLE	Virtual Learning Environment
xAPI	Experience API
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Contents

Chapter 1

Introduction

Technology is changing the ways how people learn. Technical tools have been employed in the classroom long before the availability of electronic computers: for example, slide rule was used to compute multiplications and divisions in mathematics. *E-learning* (electronic learning) refers to the use of computers or electronic devices, such as smartphones, in learning, education, or training; the learning material or education program is delivered to the learner by electronic means [46]. Nowadays, it is common to utilize the Internet and the World Wide Web in delivering e-learning but other electronic methods could also be used, such as Digital Versatile Discs (DVD). E-learning is not restricted to just delivering traditional learning material in an electronic format, such as printed textbooks converted into digital textbooks, but may also utilize the full possibilities of the modern technology: for example, interactive multimedia presentation in a web page embedded seamlessly in the middle of the written narrative, or highly specialized educational software used in the laboratory work of a science class.

E-learning is changing the methods of teaching in all levels of education, from primary schools to higher education and corporate training. *Traditional instruction* refers to teacher-centric face-to-face classroom learning while *technology-rich instruction* enhances traditional instruction with electronic and software tools, digital textbooks, and web-based content [45, p. 6]. Technology-rich instruction still emphasizes teacher-centric learning in the classroom. *Blended learning* refers to the combination of classroom learning and e-learning so that e-learning is an integral component of the education program: a part of content and instruction is delivered online and the students are given some control over time, place, progression path and/or pace; a part of the learning takes place in a supervised classroom [45, pp. 3–4]. Blended learning emphasizes that e-learning is used in part to deliver content and instruction as well as to provide some form of student control,

while traditional or technology-rich instruction do not provide the student control and e-learning is at most another tool used in the teacher-centric classroom. *Distance learning* refers to the delivery of education program to students separated by a distance; the program and content are prepared by a teacher or an institution, but the students and the teacher do not gather in the same physical location, such as classroom [31]. Historically, distance learning was delivered via mail, radio, and television, but today the Internet and thus e-learning are dominating distance learning [31]. Asynchronous distance learning allows students to set their own pace for studying whereas synchronous distance learning follows the schedule set by the teacher [31]. Synchronous distance learning may utilize video conferences or virtual classrooms to provide teaching sessions similar to physical classroom learning. Ideally, when a course is delivered as distance learning, the student is able to complete the whole course from home, but due to practical matters, final exams are sometimes only organized on-site at the institution [31]. Potential problems in online assessment include verifying a student's identity online or verifying that the student is taking the exam alone without help.

Massive open online courses (MOOC) are a form of distance learning that emerged in 2008 [31] and have since then gained millions of participants in total [22]. These kinds of courses are open to everyone and thus may have thousands of participants simultaneously [31]. Signing up to MOOCs online is easy and not every participant actually completes the course: even less than 10 % complete the course on average [29]. The courses may be offered free of charge or with a small fee; sometimes payment is only required if the student wants to receive a certificate for completing the course [9]. Due to the large scale, MOOCs provide limited interaction between the participants and the instructors. Instead, they may encourage the students to participate in online discussions with each other and assignments are assessed either automatically (quizzes, programming assignments) or by peer review (open essays) [12, pp. 45–46]. Institutions, including highly-ranked universities, often offer their MOOCs through a provider, such as Coursera, edX, and Udacity [22]. The providers supply tools and infrastructure for creating and publishing MOOCs as well as visibility in the web. The institutions may gain publicity and attract new students by organizing MOOCs.

E-learning has grown rapidly in the past decade. The number of students that signed up to a MOOC doubled in 2015 compared to the previous year and reached 35 million [44]. According to a study in 2005, most or nearly all universities at that time were already utilizing a type of a centralized e-learning platform commonly referred to as *learning management system* (LMS) [17, p. 182]. In 2014, 99 % of institutions had an LMS and 85 % of faculty were using it, 56 % on a daily basis [15].

Learning management system is a software application for managing and delivering online courses. Some support a broad definition for LMS that is not limited to just the management aspect, while others like to differentiate LMS, *learning content management system* (LCMS), and *virtual learning environment* (VLE). In that case, LMS concentrates on the management and delivery of courses whereas LCMS is used to create content and learning modules that are imported to the LMS. Virtual learning environment emphasizes the learning activity: students read study material in the VLE, engage in interactive modules, such as answering quizzes, submitting assignments, and receiving assessment. In practice, these terms are partially overlapping and many well-known e-learning platforms call themselves LMSs even though they merge much of this functionality in the same software. This thesis also prefers the broad definition for LMS. Moodle and Blackboard are examples of such feature-rich learning management systems. These systems have a number of features, including but not limited to

- dissemination of electronic teaching material, such as lecture slides, to students;
- integrated, graphical tool for building lessons: web pages that contain text, multimedia, quizzes, and assignments;
- assignments that students submit their solutions to and the teacher grades them with an integrated tool that supports rubrics and annotating the submission directly with comments;
- gradebook that gathers the student's grades in one place;
- discussion forums; and
- calendar of course schedule and deadlines.

Aalto University has a long history with e-learning. The university-wide LMS before 2015 was the **Noppa** system: a simple LMS that did not have much interactive functionality from a student's perspective. Noppa listed the course syllabus and schedule, and teachers added instructions, lecture slides, and assignment descriptions to the course site. The system did not support assignment submissions nor discussion forums, thus teachers had to use other systems or email for those needs. Noppa was replaced by a Moodle-based platform named **MyCourses** in autumn 2015. MyCourses has the interactive features missing from Noppa and as Moodle is an extensible platform that supports plugins, the functionality of MyCourses can be further expanded with Moodle plugins.

A number of novel e-learning systems for various purposes have been developed at Aalto University. **TRAKLA2** is a platform for interactive algorithm simulation exercises [34]. In TRAKLA2, the student "executes" steps of the algorithm by visually manipulating data structures and the system checks if the student is progressing correctly and provides feedback [34]. While the front end of TRAKLA2 is implemented as a Java applet [34], the newer **JSAV** (JavaScript Algorithm Visualization) library is implemented in JavaScript [33]. Hence, using JSAV does not require any web browser plugins like TRAKLA2 does: all modern browsers support JSAV natively and deploying the exercises to students is easier [33]. The JSAV library supports animated slideshows in addition to TRAKLA2-style exercises [33]. **Rubyric** is a system for rubric-based assessment [11]. Students submit their assignment solutions to Rubyric and the teacher evaluates the submissions and writes textual feedback using a rubric prepared for the assignment. The teacher saves time as the prepared rubric includes common feedback phrases, thus the teacher may only select suitable phrases and write additional more specific feedback when necessary. **Goblin** is a system for automatic assessment of programming assignments [21]. Goblin compares the outputs of the student's submitted program and the model solution prepared by the teacher[1]. The submission is scored based on how well the output matches that of the model solution in predetermined scenarios; the input data may be randomized. Mathematics courses have utilized **STACK** system (System for Teaching and Assessment using a Computer algebra Kernel) for providing automatically assessed, mathematical exercises. The STACK system was originally developed by Christopher Sangwin [43], however, a Master's thesis at Aalto University built new modifications and updates to STACK [19]. **A+** is a service-oriented learning management system that is particularly designed to operate with highly specialized, external systems that provide, for instance, automatically assessed exercises or visualizations [32]. **Stratum** is a modular, computerized learning environment that assesses assignments automatically using assignment packages installed in the system [38]. In addition to automated assessment, Stratum can randomly generate personalized assignments so that each student receives a different version of an assignment. A teacher may also set a maximum number of allowed submissions to an assignment and optionally force the assignment to regenerate when the student reaches the limit, i.e., the student starts over with a new, different assignment.

[1]To be more precise, Goblin is a course management system that operates with EXPACA (Experimental XML-based Program for Automatic Code Assessment) framework which is responsible for the automated assessment.

A few of the previously mentioned e-learning systems emphasize automated assessment of the student's work, in particular A+ and Stratum. One major reason that initially triggered the development of automated assessment systems was the substantial increase in the number of students enrolled in information technology and telecommunications degree programmes in Finland in the late 1990s [28]. Automated assessment is profoundly different from manual assessment conducted by a teacher. Automated assessment has several advantages:

- Teachers do not spend time assessing submissions manually, which leaves them time to develop other parts of the teaching [38, p. 12].
- Automated systems easily scale to large numbers of students, even thousands. Doing the same work manually could require infeasible amounts of working hours, especially if the assignments are personalized to each student [34, p. 268].
- Automated systems can be made available via the Internet at any time of the day and in any location with an Internet connection. Students may choose to work at any time and place that best suits them [28, 34].
- Automated systems can provide immediate feedback to the student and the student may correct her errors and learn from them [34, pp. 267–268].
- Automated systems are objective and consistent [20, p. 26]: they do not discriminate or favor any students and they are not prone to human errors (excluding possible software bugs).
- Automated systems may reduce the need for contact learning, that is, students ask less questions from the teachers or assistants when they are using an automated assessment system [39]. Thus, teachers have more time to support struggling students. Alternatively, the contact learning sessions could concentrate on more advanced topics since the students can study the basics independently with the help from the automated system.
- Rudimentary automated checking can ensure that submissions follow the required format, which is helpful even when the assignment is assessed mostly manually. The students are less likely to forget some of the required files or to submit programs that fail to compile. [8]

- Automated systems can integrate monitoring of the student's performance and progression. Such automated analytics can reveal which students are struggling or start working too late, enabling the teacher to take appropriate action. [16]

There are also some disadvantages in automated assessment:

- Automatic assessment may encourage students to submit repeatedly with small changes until the solution receives full points using kind of a trial-and-error approach. This behaviour may be hindered by limiting the number of available submissions, or by increasing the delay between the submission and the release of the automated feedback to the student. Another approach is to provide rewards ("achievement badges") to students for submitting a high-quality solution in the first attempt. [18]

- Developing the automated system and assignments requires time and effort initially. The software system also requires maintenance after deployment.

- Developing automated graders can be very challenging, depending on the nature of the assignment. In computer science, programming assignments are typically automated, whereas automating open essay questions is still not a fully realized field.

- Software platforms for automated assessment add some restrictions on the format and content of the assignment due to design decisions and limitations of the software system. If the teacher assesses submissions manually, the only restriction on the format of the assignment is basically what the teacher is willing to do.

- Automated systems may not give as high-quality feedback to students as humans, depending on the system. It is challenging to design automated systems that could analyze the student's errors and identify misconceptions. [20, p. 28]

- If all students complete the same, identical assignment, copying solutions from other students is easy (plagiarism) [28]. Automated plagiarism detection tools could be integrated in the assessment system in order to detect dishonest students. Another approach is to use an assessment system that can personalize assignments for each student. Thus, copying solutions from others is rendered useless since everyone has a slightly different version of the assignment [28].

1.1 Goals and Requirements

The Stratum computerized learning environment was originally developed as part of a Master's thesis project in 2006. However, the architecture of Stratum has recently become a source of problems that complicates further development of the system. Stratum and its problems are covered in detail in Section 3.1. The goal of this thesis is to modernize Stratum so that the system matches current needs better and may be further developed with feasible effort. Additionally, as Aalto University is currently using Moodle (MyCourses) as the university-wide learning management system, the modernized Stratum must be able to embed Stratum assignments in Moodle so that students can access the assignments directly from the Moodle course instead of a separate, external web site.

The requirements for the modernization of Stratum are as follows:

1. Improve the maintainability of the system and possibility for further development of new features. This implies improving the quality of the source code, e.g., by restructuring software architecture, code organization, and reusability.

2. Improve the usability of the system for both students and teachers.

3. Assignment packages implemented for Stratum must be portable to the new system with reasonable effort.

4. The new system must support the same basic functionality as Stratum, and in particular, the automatic grading and generation of personalized assignments. Regeneration of the assignments must also be supported, i.e., generating the assignment again for a student after she has submitted too many solutions.

5. The new system must support embedding assignments in Moodle (MyCourses) so that students can access assignments directly from a Moodle course.

In order to realize the modernization of Stratum, this thesis shall research other existing e-learning platforms that potentially support automated assessment of assignments and make alternative plans for the implementation of the modernization. The most promising alternative is selected as the basis for the real implementation.

1.2 Structure of the Thesis

This chapter discussed e-learning and automated assessment in general as well as the goals and requirements of this thesis. Chapter 2 describes known e-learning platforms, particularly A+ and Moodle that are relevant to this work. Chapter 3 details Stratum computerized learning environment and explains its problems, which constitute the starting point for this thesis and the motivation for the modernization process. Furthermore, Chapter 3 presents alternative approaches for the modernization of Stratum and identifies the best alternative. Chapter 4 addresses the implementation of the modernized platform, and Chapter 5 evaluates the solution and the realization of the requirements. Finally, Chapter 6 concludes the work and presents some prospects for future work.

Chapter 2

E-learning Platforms

Having introduced e-learning and goals for this thesis in the previous chapter, this chapter describes known e-learning platforms, especially A+ and Moodle as they are relevant to this thesis. Other known platforms are also introduced briefly. Section 2.1 begins by describing open standards in e-learning as many platforms utilize some of the standards. Section 2.2 details A+ with its architecture and grader protocol, while Section 2.3 covers Moodle, the most widely used open source learning management system. Finally, Section 2.4 addresses other platforms. The next chapter discusses another e-learning platform, Stratum computerized learning environment, and how this thesis modernizes it.

2.1 Open Standards for E-learning Platforms

Open standards improve the interoperability of different systems and enable transferring learning content between systems. Otherwise, the transition from a platform to another one could be practically impossible if there is too much valuable content stored in the original system that can not be copied without manual labour. Moreover, the functionality of a non-interoperable platform can not be easily extended by using other external systems when the platform is missing necessary features. In other words, open standards reduce the phenomenon called *vendor lock-in*.

Learning Tools Interoperability (**LTI**) is a specification developed by IMS Global Learning Consortium that enables integrating rich learning applications (Tools) to platforms such as learning management systems (Tool Consumers). The Tools are delivered by remote systems (Tool Providers) via the web [24]. The LTI specification has different versions that vary in their feature set and complexity. The version 1.0 only enables launching Tools: the

user starts from the Tool Consumer (LMS) and clicks a link that redirects her to the Tool Provider. The user authentication is handled by LTI so that the transition to the external tool is fluent for the user. The version 1.1 enables the Tool Provider to return an outcome (grade) of the student's work back to the Tool Consumer. This supports, for example, tools that provide assignments with automated assessment. The version 2.0 is much more complicated and supports web applications that require interoperable communication between the systems for the whole life cycle of the Tool. Version 1.2 is a simplified version of 2.0 with some features removed. [47]

IMS Question and Test Interoperability (**QTI**) is a standard for assessment items and tests, including quizzes and summative tests. Tests in QTI format may be exchanged between systems without losing content [25]. The **IMS Common Cartridge** specifications form a set of standards for packaging learning content so that it may be exchanged between systems and imported to a compliant platform (LMS). Common Cartridge incorporates parts of QTI and LTI [23].

The **Sharable Content Object Reference Model** (**SCORM**) is another standard for packaging learning content. The SCORM standard defines a run-time environment with an application programming interface (API) that enables communication between the content package and the platform (LMS) [7]. The SCORM package may, for example, store grades in the underlying platform using the API. **Experience API** (xAPI, also called Tin Can API) is a newer API that enables sharing data about human performance and activities in a fine-grained micro level [6]. The SCORM and xAPI standards are developed by the Advanced Distributed Learning Initiative, a U.S. government program.

2.2 A+

The A+ platform is a service-oriented learning management system that utilizes external *exercise services* to provide course content, both static study material and exercises. The following description of A+ is based on [32]. Exercises in A+ are usually automatically assessed, as the A+ platform was particularly developed to have common infrastructure and front end for courses that deploy automatically assessed exercises or visualizations, such as most programming courses at Aalto University. The platform manages user authentication and authorization, stores submissions and their grading results (feedback and points), manages course configuration and setup, provides course monitoring (student progress and statistics) and web user interfaces for students and teachers. Additionally, A+ provides a REST API

(Representational State Transfer, Application Programming Interface) that allows other authorized systems to access the data stored in A+ over a network connection. The REST API enables the integration of other specialized systems to A+ that could, for example, compute final grades in a different way than A+, or run statistical experiments with the student data. Implementing these other systems inside A+ would complicate the implementation of A+ and the features would only be useful to some courses.

As the A+ platform is responsible for providing common infrastructure for courses, exercise services concentrate on delivering study material (called chapters) and exercise descriptions to A+ for users to view, in addition to grading submissions to exercises. The exercise description could be just text but it could also be delivered in the form of a visual tool that the student uses to complete and submit the exercise, for example TRAKLA2 or JSAV algorithm visualization exercises. The exercise services are simplified accordingly since they need not manage user accounts or permanent data storage that A+ already takes care of. The A+ platform communicates with exercise services using an HTTP-based (Hypertext Transfer Protocol) *grader protocol*, which additionally allows other systems besides A+ to access the same exercises using the protocol, or the usage of the same exercises in multiple A+ courses. [32]

The A+ platform is quite lightweight compared to other well-known learning management systems, such as Moodle. While A+ does not have, for example, built-in discussion forums, it supports LTI 1.0 protocol which may be used to link the user from the A+ site to an external site that provides functionality missing from A+. For example, some A+ courses have utilized Piazza[1] as the discussion forum for asking questions. Due to the LTI protocol, students can transition from the A+ site to Piazza without a separate login procedure. Furthermore, A+ does not have a built-in calendar but the course schedule can be exported in iCalendar format to other calendar software.

2.2.1 A+ Content Structure and Grading Semantics

The A+ platform divides the content in a course to *exercise rounds*. A round has opening and closing times and consists of *learning objects*. Learning objects are either *chapters* or *exercises*. Chapters provide study material and have no grading associated with them, while exercises accept submissions that are assessed and thus receive points and feedback. Each exercise has a limit of maximum available points set by the teacher and may have minimum

[1]Piazza web site contains a product description: `https://piazza.com/`

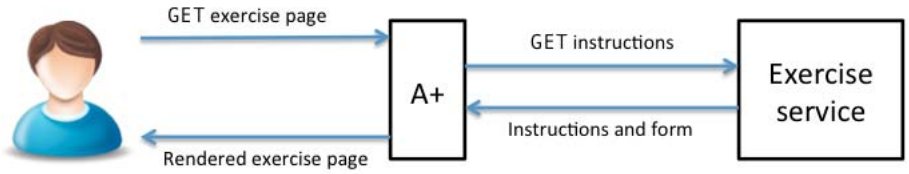

Figure 2.1: Retrieval of exercise description. Image source: [32]

required points that students should earn to pass. Furthermore, the teacher may limit the number of submissions that a student may make in an exercise. Exercise rounds may also have minimum required points that students should earn in total in the exercises of the round.

The platform also has *categories* in addition to exercise rounds and learning objects. Each learning object belongs to a category. The category computes a total sum of the points a student has earned in the exercises of the category across all rounds. The teacher may set the number of points required to pass to categories as well.

A+ supports *group submissions* in addition to normal individual submissions. In a group submission, students form a group and the group submits one solution to an exercise; all group members gain credit for the group submission. The teacher defines the minimum and maximum allowed group sizes for each exercise.

2.2.2 A+ Grader Protocol

As explained in [32], the A+ grader protocol has two features: retrieving exercise descriptions from the exercise service and sending submissions to the exercise service for assessment. Exercise descriptions are retrieved from an exercise-specific service URL (Uniform Resource Locator) set in the A+ configuration. When a student opens the exercise page in A+, the platform retrieves the exercise description from the exercise service by sending an HTTP GET request to the service URL. The exercise service responds with an HTML (Hypertext Markup Language) document that contains the instructions to the student and usually a form for submitting a solution. Figure 2.1 illustrates the retrieval in the protocol.

For assessing submissions, the A+ grader protocol supports both synchronous and asynchronous exercises. Synchronous exercises are assessed during a single HTTP request and are only suitable to exercises that can be assessed in a short execution time, such as multiple choice questions or small programming exercises. This limitation stems from the nature of HTTP: the

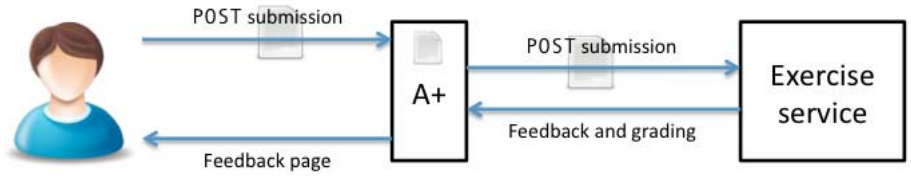

Figure 2.2: Submission for assessment. Image source: [32]

user must wait for the HTTP response to finish before the feedback page loads, and the exercise service server can only manage a limited number of concurrent HTTP requests. Figure 2.2 illustrates the protocol for synchronous assessment. When the assessment requires more execution time, it should be handled asynchronously. In asynchronous assessment, the exercise service needs to send the grading results to A+ once the grading has been finished. Initially, when the exercise service receives a new submission, it responds to the HTTP POST request by only acknowledging that the submission is accepted for grading. Hence, when A+ sends a submission to the exercise service for grading, it also passes an HTTP GET parameter "submission URL" that the exercise service can send the results to by HTTP POST. The A+ platform encodes a submission identifier into the submission URL so that when the platform receives the new grading results, it updates the correct submission in the database. Additionally, the submission URL is signed so that A+ can validate the grading results before accepting them. The platform also checks that the request originates from the exercise service. Thus, students may not cheat by sending HTTP requests to A+ awarding full points to a submission.

2.2.3 MOOC Grader Exercise Service Framework

The A+ platform is accompanied by an exercise service framework called MOOC grader [3]. The MOOC grader provides common functionality needed in exercise services so that it is fast and easy to add new exercises: typically, one needs to only write a small configuration file, a grader program for assessing submissions, and the exercise description. Multiple choice questions can be created with just a configuration file that lists the questions, choices, and correct answers. The MOOC grader is designed to be scalable so that it may run exercise services for large courses (i.e., MOOCs). Hence, the system supports distributing the workload of multiple submissions to multiple servers.

The MOOC grader implements the exercise service side of the A+ grader protocol and provides a queuing mechanism for temporary storage of submis-

sions to asynchronous exercises. Submissions are dequeued and the grader is started when it is ready (typically, there is a limitation on the number of concurrent executions). The MOOC grader provides a sandbox environment for graders that allows safe execution of arbitrary code, which is particularly necessary in programming exercises. Sandboxed execution prevents the student's submitted code from compromising the server or accessing the network.

A teacher writes a course configuration file in MOOC grader that lists the exercises in the course with their deadlines, maximum available points, minimum required points, and other settings. The A+ platform can import the course configuration from the MOOC grader and thus setting up the course in A+ requires no effort once the course has been prepared in the MOOC grader. The Git version control system is supported so that course files can be installed and updated in the MOOC grader using Git operations. Basically, a teacher may change course files, for example, a faulty grader, during the course and push the changes to a private repository and consequently, the MOOC grader automatically updates the files from the repository.

2.3 Moodle

Moodle [37] is a free, open source, online learning management system (LMS) or virtual learning environment (VLE). Moodle allows teachers to create courses with static and interactive content and assignments. Moodle is extensible: plugins can modify and add new functionality to Moodle, while themes change the visual outlook of the site. Activities and resources are the core concepts in a Moodle course. Activities are interactive components, for example, forums in which students participate in discussions or assignments to which students submit solutions. Resources are static material provided by the teacher, for example, a handout as a PDF (Portable Document Format) file.

Moodle plugins are categorized by types: the type defines the purpose of the plugin and which areas in Moodle it modifies or extends. Moodle has approximately 40 different plugin types [36]. For example, activity modules define activities that may be added to courses and plagiarism plugins define services that process submissions from students in order to detect plagiarism. Block plugins define widgets that may be added to sidebars in Moodle course pages. Moodle calls those widgets *blocks* and they are often used to provide navigation or to display information about recent events or actions, such as the headings of the latest announcements the teacher has posted in the announcements forum. *Moodle plugins directory* is a repository of open

source plugins developed by the community. A member of the Moodle core development team must approve a plugin before it is accepted to the plugins directory and thus hosted on the official Moodle site[2]. Moodle and its plugins are written in the PHP (PHP: Hypertext Preprocessor) programming language.

MyCourses[3] is a Moodle-based platform taken into use at Aalto University in 2015. MyCourses has workspaces for all courses held at Aalto, and it is typically used in everyday course work: news announcements, course brochures and schedules, discussion forums, publishing handouts and lecture material, and accepting submissions to assignments. MyCourses is currently the main online teaching platform at Aalto. It is used particularly for blended learning as most courses involve classroom teaching, i.e., lectures and tutorials. Materials for lectures, instructions for self-studying, and assignments can be published in MyCourses.

2.4 Other Platforms

Besides Moodle, there are several other learning management systems widely used in the world that implement similar functionality as Moodle. Some systems are open source, like Moodle, whereas many are proprietary. Open source systems include Sakai and ATutor, while proprietary systems include Blackboard, Canvas, D2L Brightspace, and Itslearning, to mention a few. These systems are next examined briefly. The list does not include all the platforms in the world, but a few of the currently interesting ones. All of these systems support basic LMS functionality, which includes publication of study material, assignments, announcements and news, discussion forums and chat, gradebook to collect all course grades in one place, as well as tests and quizzes in which multiple choice questions are automatically assessed.

Sakai [42] is particularly aimed at the higher education sector and the developers of Sakai work in the higher education sector themselves. Sakai project was started in 2004 when four U.S. universities with an organization called Jasig decided to collaborate in order to combine their different learning platforms into an integrated suite of open source software. They aimed at "providing a compelling alternative to proprietary learning systems" [42]. The new Sakai version 11 utilizes responsive web design and thus the web site works well on both mobile and desktop displays. The lesson tool in Sakai allows the teacher to build pages that contain text, media, and quizzes. Sakai supports project work sites that enable student groups and other user groups

[2]Moodle plugins directory: `https://moodle.org/plugins/`

[3]`https://mycourses.aalto.fi/`

to "coordinate project administration, share resources, schedule activities, develop collaborative work products and track progress towards completion" [42]. Furthermore, Sakai supports IMS standards Common Cartridge and LTI and aims to meet all international accessibility standards.

ATutor [10] is characterized by its support of accessibility standards. All functionality is accessible with just a keyboard without a mouse and works well with screen readers, hence even blind users may administer the system and create content. ATutor is bundled with AContent, which is a tool for developing content packages in standard formats, including IMS Content Packaging, QTI, and Common Cartridge. The content packages may be imported to any learning management systems that support the standards, not only ATutor. The development of ATutor is no longer very active but minor bug fixes are still released[4].

Blackboard Learn [13] is a learning management system developed by Blackboard Inc. In addition to common LMS features, Blackboard Learn offers plagiarism prevention through SafeAssign. Furthermore, the system is accompanied by a mobile application that students may use to access course content, assignments, and grades. The mobile application supplements the main desktop web user interface. Blackboard Learn integrates with other Blackboard software products that support, for example, learning analytics and statistics in order to keep track of the students' progress and retention, or virtual classes and video conferencing easily accessible with a web browser. The mobile application also allows users to join virtual classes. Blackboard Learn may be customized with plugins that are called "Building Blocks". Building Blocks extend the functionality of the system by utilizing the APIs Blackboard offers and they are implemented as Java web applications.

Canvas [26] is an LMS that is only delivered as Software as a Service (SaaS) from the cloud. It is developed by Instructure Inc. The web user interface is supplemented by mobile applications that support most of the basic functionality. Canvas advertises its high usability, user satisfaction, and features that save the teacher's time. For example, SpeedGrader tool allows the teacher to view a student submitted document and grade it in the same page. In particular, the teacher may annotate the document with comments, highlights, and freehand drawings. Grading rubrics are also supported. Canvas provides teachers graphic analytics about student performance in the course. In a classroom, the teacher may start polls to quickly test the students' understanding; the students respond using their own mobile devices. Canvas supports the LTI protocol and has open APIs so that custom tools may integrate with the system or extract data.

[4]ATutor 2.2.2 was released on 1st July 2016.

Brightspace [14] is a SaaS-only LMS from D2L Corporation. The web user interface of Brightspace is accompanied by two mobile applications: one is used to deliver current notifications about deadlines and events, the other enables students to download documents from the course workspace for reading in the mobile device. Brightspace follows web accessibility standards and supports the LTI protocol. Teachers may track student performance with built-in analytics tools and they are notified about at-risk students who are likely dropping out from the course.

Itslearning [27] is an LMS from Itslearning AS, a Norwegian company. Itslearning is deployed only as SaaS. Itslearning assignments easily support peer reviewing amongst students. Personalized learning in Itslearning allows teachers and students to design personal learning goals, tailoring study material to individual needs. A mobile application provides students notifications about deadlines and new assessments. Itslearning has open APIs that enable the integration of custom extensions.

To summarize, the learning management systems covered in this chapter are strikingly similar. The A+ platform stands out with its emphasis on external automated assessment systems. On the other hand, it does not support other common LMS functionality, such as discussion forums, but A+ is not trying to compete against Moodle or other well-established LMSs. Moodle and the other platforms have the same basic functionality, though there are differences in the details of the basic features. Moreover, some systems try to differentiate themselves with additional features, such as course analytics, mobile applications, or enhanced usability. Accessibility or interoperability standards, such as IMS LTI, are also highlighted by some systems. Most systems try to attract users with their modern, graphical user interfaces. When one needs to select one of these platforms, the needs of the institution should be carefully considered and reflected on the features offered by each system in a fine-grained detail. The costs should also be considered: running open source systems on-site entails operational costs and commercial SaaS solutions require licensing costs.

Chapter 3

Modernizing Stratum

This chapter describes Stratum computerized learning environment and its problems in detail that lead to the need for modernizing Stratum. Section 3.1 starts by introducing Stratum and Section 3.1.1 addresses its problems. This chapter then progresses to different alternative approaches for the modernization in Section 3.2 and to the selection of one alternative in Section 3.3.

3.1 Stratum Computerized Learning Environment

The Stratum computerized learning environment was developed in a Master's thesis project in 2006 [38]. The system had already existed before that, but it was rewritten and modularized in 2006. Additionally, the development continued until 2015. In 2009, a new web interface was developed in PHP that enables students to submit solutions in a web browser [48]. Preceding the project reported in this thesis, the author started the development of new features in 2013. The new features from 2013–2015 include, for example, a teacher's web user interface for inspecting assignment submissions, as well as Shibboleth single sign-on authentication [40].

As explained in [38], Stratum can create randomized or personalized assignments to students and grade submissions automatically. Stratum provides common infrastructure for all courses, while the functionality of specific assignments (generation and grading) is implemented in separate assignment packages. The assignment packages must conform to a specific interface so that they can operate with Stratum. The interface consists of two Bash scripts, one for generation and the other for grading, that Stratum executes with certain arguments, for example the file path to the student's submission and personal directory.

The structure of Stratum is described in [38]: it consists of the kernel, course instance directories, and the assignment packages. In addition, Stratum depends on third-party software, for example Apache HTTP server for the web interface. The kernel implements the functionality of the infrastructure and the command-line interface for teachers. The kernel is written in the Python and Bash languages. The course instance directories, stored outside the kernel in the file system, store the data of a course, for example students and their submissions. The course instance also contains the assignment packages used in the course as well as the web templates of the course.

The common infrastructure provided by Stratum includes the following features [38, 40]:

- accepting and storing submissions;
- grading submissions using the grader program in the assignment package;
- storing the grading results, i.e., points and feedback;
- generating new assignments for a student using the generator program in the assignment package;
- regenerating an assignment for a student after she has submitted too many times if the teacher has enabled regeneration;
- allowing teachers to inspect, regrade, or manually grade submissions; and
- displaying statistics about submissions to the teacher.

Students access the Stratum assignments from a course web page in a Stratum server or from a Moodle course page, as explained in [40, 41]: Stratum is accompanied by a Moodle plugin that is installed in a Moodle site and Stratum itself has an HTTP interface that the Moodle plugin uses to connect to Stratum. With the Moodle plugin, students see Stratum assignments embedded in a Moodle course even though the automated assessment is performed outside Moodle in the Stratum server [40, 41].

3.1.1 Problems of Stratum

The goal for this thesis was to modernize Stratum. Stratum has multiple issues that justify such need for modernization. These issues are identified in the following analysis.

First, the web architecture of Stratum is clunky. The core of Stratum is implemented in Python and Bash, while the web component is written in PHP and depends on an HTTP server, such as Apache, to run the PHP code. The PHP code must query the Stratum core for all Stratum functionality and user data, which is implemented by executing subprocesses. This makes developing the web interface awkward for the following reasons. The Python code must implement executable programs for all functionality used by the web interface and this implies building a command-line interface for each feature required by the web interface. The Python code can only send user data to PHP through the standard output stream of the subprocess, which implies a textual format for the data in contrast with the more complex data structures available in Python. If whole Stratum were implemented in Python, these problems would not exist.

Second, the web user interface of Stratum is too simplistic and its usability could be improved. Due to the web architecture, it is difficult to add new content to the Stratum web pages, which hinders the development to improve usability.

Third, Stratum does not employ an HTML template engine. Instead, it writes personalized web pages for each student separately and stores the HTML files in the file system. Using a template engine could simplify the implementation.

Fourth, some parts in the Stratum kernel are implemented in Bash for historical reasons. The Bash components could be implemented in Python like the rest of the kernel. It would improve the interoperability of the code components since Bash can only execute Python code via subprocesses, which leads to the same problems as in the web interface. In addition, file permissions in the operating system may complicate the execution of subprocesses. Stratum has been deployed in a way that uses the user accounts of the operating system to separate the web server process and data from different courses as an additional layer of protection. In order to enable the necessary subprocesses, the file system permissions have to be set carefully; otherwise, the system malfunctions.

Fifth, some parts in the Stratum kernel could be rewritten and reorganized in order to improve code reusability and to avoid duplicating similar functionality in many parts of the code.

Finally, Stratum does not use a (relational) database management system (DBMS). Instead, all of the course and user data are stored in files in the file system. Most of the course and user data could be stored in a database, excluding the contents of submitted files. Using a relational database and object-relational mapping (ORM) middleware could simplify the code and accelerate development time.

3.2 Alternative Approaches for Modernization

In the following, we present three different, alternative approaches for the modernization of Stratum. Only one of the approaches is actually selected for implementation. All three alternative approaches utilize both A+ and Moodle, either simultaneously or by offering different alternative user interfaces. Requirement 5 in Section 1.1 demands that a Moodle interface be included, while A+ is included because many courses at Aalto University have already been built on A+ and many other courses are moving towards A+ in their e-learning arrangements. Therefore, interoperability with A+ is desirable.

3.2.1 Common Changes in the A+ Grader Protocol and MOOC Grader Exercise Service Framework

All alternative approaches share a need to modify the A+ grader protocol in order to support personalized exercises, which is formalized as Requirement 4. As the A+ platform is not responsible for producing exercises or exercise descriptions to users, but exercise services are, it is natural that the exercise service is responsible for storing data needed in personalized exercises. Hence, the A+ grader protocol must provide a unique identifier of the user (student) both when the exercise description is retrieved and when a new submission is sent for grading. Thus, the exercise service may deliver the personalized exercise to each user and grade submissions accordingly. Additionally, in order to support the regeneration of an exercise, the A+ grader protocol must provide the ordinal number of the submission being graded so that the exercise service may change the exercise for a user if the user submits more times than a set threshold. When retrieving the exercise description, the ordinal number must be incremented by one from the previous submission so that the state of the exercise description matches the state of the grader when the next new submission is graded. That is, when retrieving the description, the ordinal number is the same as for the submission that the student makes after reading the description.

Alternatives 2 and 3 utilize exercise services in the same fashion as A+. The MOOC grader exercise service framework needs to be updated to support personalized exercises. A personalized exercise has a number of different instances that in practice are files stored in the file system. A course must define an exercise generator program similarly to grader programs. The generator is executed offline before the start of the course so that sufficiently many exercise instances have been created before students access them. Offline

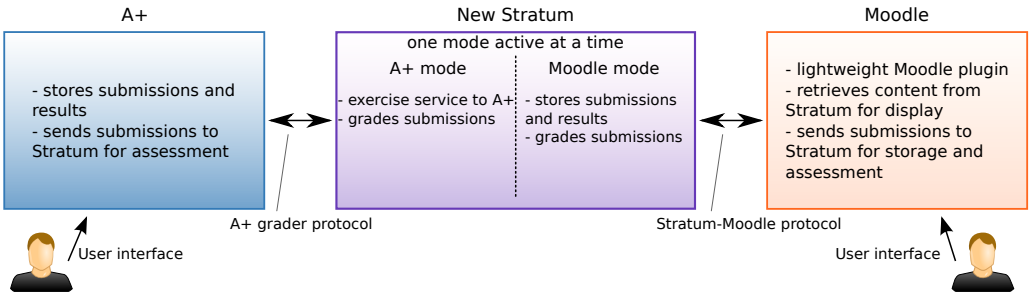

Figure 3.1: Alternative 1: Independent Stratum

generation is used because the generation may require a lot of time and the student should not need to wait while opening the exercise description. Long generation times could also cause timeouts in the HTTP connection. The generated exercise instance files are made available to exercise description templates and grader programs so that students receive their personalized exercise descriptions and their submissions are graded accordingly.

3.2.2 Alternative 1: Independent Stratum

The first alternative, illustrated in Figure 3.1, uses the architecture of old Stratum as a baseline. The implementation source code is rewritten to improve the quality of the code (addressing Requirement 1 from Section 1.1). The new implementation uses a modern web development framework, such as Django[1], in order to ease the development and to avoid repeating the web architecture problems of old Stratum.

The new system has two interfaces that also affect the operation mode: the system can operate as an exercise service to A+ or as an independent system that has a user interface embedded in Moodle by using a lightweight Moodle plugin (a plugin similar to the one for old Stratum). When A+ is used as the front end, it stores the submissions and their results and sends the submissions to new Stratum for grading using the A+ grader protocol. When the lightweight Moodle plugin is used as the front end, new Stratum must store submissions and their results as well as grade submissions. The Moodle plugin only displays a student's submissions and their results by retrieving the data from Stratum and also forwards new submissions to Stratum for storage and grading. The Moodle plugin communicates with Stratum using a simple HTTP-based protocol created for this purpose. A course may not use both interfaces (A+ and Moodle) simultaneously since there is no synchronization of the data stored in the two systems (A+ and new Stratum).

[1] `https://www.djangoproject.com/`

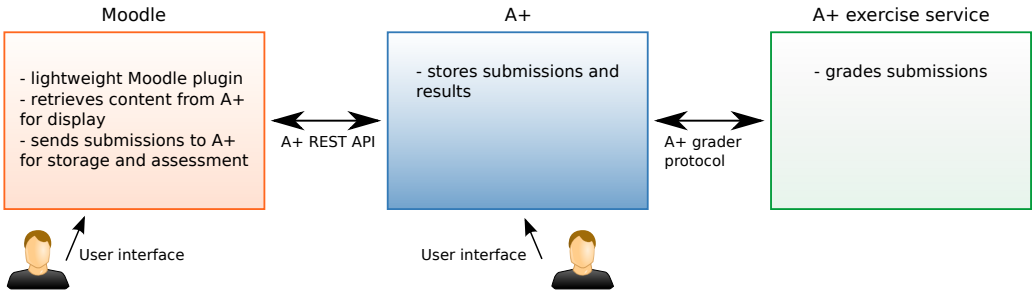

Figure 3.2: Alternative 2: New Moodle Interface in A+

Requirement 2 concerning usability is addressed in the user interfaces of A+ and the Moodle plugin. The A+ platform already has a modern user interface and the Moodle plugin integrates seamlessly into the familiar Moodle environment.

The A+ grader protocol needs to be changed as described in Section 3.2.1 in order to support personalized exercises as stated by Requirement 4. The protocol between Moodle and Stratum is designed with the requirements in mind. The exercise service framework of the MOOC grader is not utilized in this alternative. New Stratum may use mostly the same assignment package interface as old Stratum, thus old assignment packages do not require heavy changes and Requirement 3 is satisfied. The assignment package grader could write points granted for the submission into the standard output stream instead of a file since new Stratum stores data in a relational database instead of recording such information into files.

3.2.3 Alternative 2: New Moodle Interface in A+

The second alternative, illustrated in Figure 3.2, emphasizes A+ more than the first alternative: A+ is either used directly as a front end or as an intermediary to a lightweight Moodle plugin. If Moodle is used as the front end, the Moodle plugin communicates with A+ so that the plugin retrieves content needed in the user interface from A+ and sends submissions to A+ for storage and grading. The A+ platform stores the submissions and grading results and forwards the submissions to an exercise service for assessment. To support the Moodle plugin, the REST API in A+ needs to be expanded so that Moodle can access exercise descriptions, submission results, and course exercise configurations from A+ and submit new solutions to A+ for storage and grading. (The A+ REST API did not support all of these requirements when this Master's thesis project was started. The REST API has since then been under heavy development for other reasons than Moodle integration.)

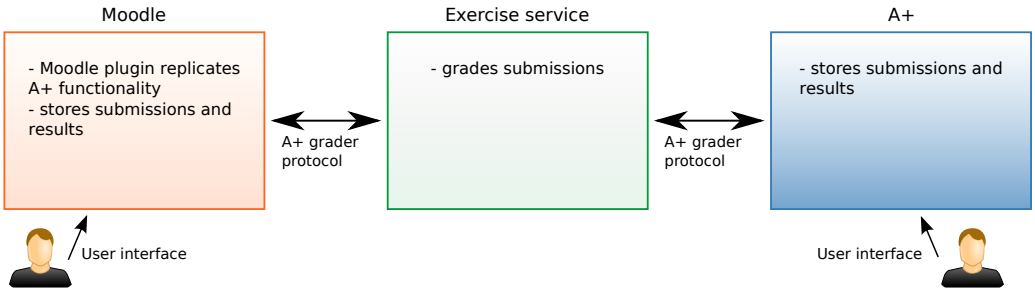

Figure 3.3: Alternative 3: Moodle Replicates A+

The changes given in Section 3.2.1 are needed in order to support personalized exercises. In this alternative, the old Stratum is not utilized as a component in the system besides the assignment packages that are ported to the MOOC grader format so that they can be run in an exercise service. Requirement 3 is satisfied as the porting is estimated to be feasible: the MOOC grader can run arbitrary grader programs as subprocesses similarly to Stratum. Requirement 1 is realized by abandoning the old Stratum code base and relying on A+. User interfaces of the Moodle plugin and A+ address Requirement 2.

3.2.4 Alternative 3: Moodle Replicates A+

The final alternative, illustrated in Figure 3.3, treats A+ and Moodle equally: Moodle plugin implements the same functionality that A+ has and thus Moodle communicates directly with exercise services using the same grader protocol as A+. Moodle and A+ do not connect to each other via a network, hence there is no synchronization between them. A particular course may use only one front end at a time, Moodle or A+. The front-end system is responsible for storing submissions and results as well as forwarding submissions to an exercise service for assessment using the grader protocol.

The changes in Section 3.2.1 apply to this alternative as well in order to realize Requirement 4 concerning personalized exercises. The assignment packages of old Stratum are ported to the MOOC grader format. Requirements 1, 2, and 3 are satisfied in the same fashion as in Alternative 2.

3.3 Justifications for the Selected Approach

The selected alternative actually implemented in this thesis is the third one. Alternative 3 supports both A+ and Moodle as front ends and utilizes the same grader protocol in both cases. This allows the same exercise services

to operate with both A+ and Moodle even simultaneously, even though a single course may utilize only one front-end system at a time. The MOOC grader exercise service framework and the grader protocol are extended as explained in Section 3.2.1 in order to support personalized exercises. This implies that using personalized exercises is not tied to a specific front-end system as long as the front end implements the updated grader protocol.

The first alternative is not chosen due to two reasons. First, the A+ and Moodle modes utilize different protocols and second, new Stratum would have to implement A+ functionality (storing submissions and results) to operate with Moodle, while that functionality would not be required with A+. This approach seems too complicated and redundant.

The second alternative is inappropriate because using the new Moodle interface in A+ would turn A+ into an intermediary that stores data and forwards submissions to an exercise service for assessment. However, A+ is already a complete platform that manages courses and their exercises as well as provides a sound web user interface. In general, Moodle is also a complete platform for managing courses and their content. Having Moodle as a lightweight front end to A+ that acts as a database and a router to exercise services does not seem to be a sensible solution in the long run. Each submission would consume more network bandwidth than in Alternative 3 because there is an additional hop in the network, and moreover, Moodle would need to query A+ each time a student views her results in Moodle. The overall system could perform slower than in Alternative 3 due to the dependence on the network and the large software stacks in Moodle and A+: each request requires accessing the database and processing the data in both Moodle and A+. Even though Moodle does not store the submissions, it still reads user data and course configuration from its database. Finally, Alternative 2 does not match the role that A+ was originally created for, which is to use A+ as a platform and a front end to courses that need specialized tools for automatically assessed exercises and for other kinds of exercises, such as algorithm visualizations [32]. The REST API in A+ is available so that it is possible to integrate other specialized systems to A+ that depend on the data in A+ [32]. Alternative 2 would utilize A+ and its REST API in a way that does not match their original goals.

Chapter 4

Implementation

This chapter describes the design and implementation of the new, modernized e-learning platform based on the approach selected in the previous chapter. The new platform consists of a new Moodle plugin named *Astra* and the upgraded MOOC grader framework. The name Astra reflects that it is based on A+ and Stratum. The Astra plugin is compatible with MyCourses and any other Moodle installation. This chapter begins with a high-level view of the features of the new platform in Section 4.1. Section 4.2 describes the implementation of Astra and Section 4.3 covers the MOOC grader. Finally, Section 4.4 illustrates the most crucial functionality of the platform by an example: accepting and assessing submissions.

4.1 Features of the New Platform

Moodle takes the role of A+ via a new Moodle plugin named Astra and replicates necessary functionality from A+. In order to ensure that the Astra plugin is compatible with A+ and the MOOC grader, it follows the same content structure and grading semantics as A+. That is to say, Astra organizes learning objects, either chapters or exercises, into exercise rounds and each learning object also belongs to a category that organizes learning objects in the course and computes the course total points. Like A+, the plugin displays exercise descriptions to students retrieved from an exercise service using the A+ grader protocol. It accepts and stores submissions to exercises and sends them to the exercise service for assessment. Both synchronous and asynchronous exercises are supported, i.e., the exercise service may either assess the submission immediately or enqueue a task for execution at a later time. Astra uses the extended version of the grader protocol.

To summarize, the Astra plugin includes the following basic features that

closely follow the functionality of A+:

- organization of content: exercise rounds, exercises, chapters, and categories;
- display of exercise descriptions;
- acceptance and storage of submissions to exercises;
- extended A+ grader protocol used to communicate with exercise services;
- presentation of feedback and results for graded submissions; and
- limitation of submissions based on the deadline of the round and the number of submissions allowed in the exercise.

Furthermore, like A+, Astra has multiple features for teachers specifically. Teachers may give personal exceptions to students if it is necessary for some students to exceed the limitations of the deadline or the number of submissions allowed, for example due to a sick leave. Astra supports importing course configuration from the MOOC grader so that setting up the course in Moodle requires no effort. Teachers may also manually set up the course configuration in Astra, however, the automatic setup based on the MOOC grader configuration is preferred because manual changes in the configuration performed in the Astra user interface are not automatically transferred to the configuration file in the MOOC grader server. Teachers may export all of the course data related to Astra exercises and download it to their computers for further processing[1]: the students' results in the exercises are available in a machine-readable JSON (JavaScript Object Notation) format and could be used to compute final grades in the course, while the content submitted in the exercises is also available if the teacher wishes to run some kinds of batch jobs on all the submissions. The submitted content includes files and other data entered in forms, e.g., in text fields and checkboxes. Teachers may browse the submissions received in exercises and inspect their contents; moreover, they may modify the assessment and add feedback in addition to the machine-generated feedback. Students receive Moodle messages about new manual assessments; the message notifications are also sent by email if the user has enabled it in her messaging settings. Astra can automatically grant non-editing teacher privileges to course assistants if their student numbers are listed in the MOOC grader configuration. Assistants require the

[1]The A+ platform supports the export of course data as well, however, in a different format from Astra and the user interface is still under development at the time of writing.

privileges for inspecting and manually assessing submissions in the course. Finally, teachers may easily send a large batch of submissions to the exercise service for new assessments if, for example, there has been an error in the grader. The teachers' functionalities in the Astra plugin are summarized as follows:

- deadline and submission limit deviations (personal extensions in specific exercises);
- automatic course setup from the MOOC grader configuration;
- manual configuration of exercise rounds, learning objects, and categories;
- export of course data: results and submitted form contents in JSON format, and submitted files in a ZIP archive;
- listing and inspecting submissions;
- manual assessment: feedback to a submission from a teacher with Moodle notifications to the student about new feedback; and
- mass regrading: sending a batch of submissions to the exercise service for new assessments.

The upgraded MOOC grader exercise service framework has naturally all of the previous features that it had before this thesis project was started. To summarize, the MOOC grader provides a platform for exercise services: a course only needs to implement its own grader programs and write a configuration file to have a fully functioning exercise service. The course configuration can be imported to Astra or A+ from the MOOC grader so that it is easy to set up the front-end system. In this thesis project, the MOOC grader was extended to support personalized exercises. A personalized exercise requires a generator program that can generate one instance of the exercise, which is stored in files in the file system. The teacher or a server administrator generates a number of exercise instances before the start of the course by running a new command called `pregenerate_exercises` in the command-line of the MOOC grader server. When a student opens an exercise description, she is connected to one exercise instance. If the exercise is configured to regenerate the instance after a student has submitted too many solutions (the threshold is also set in the configuration), the student's instance is switched to another one.

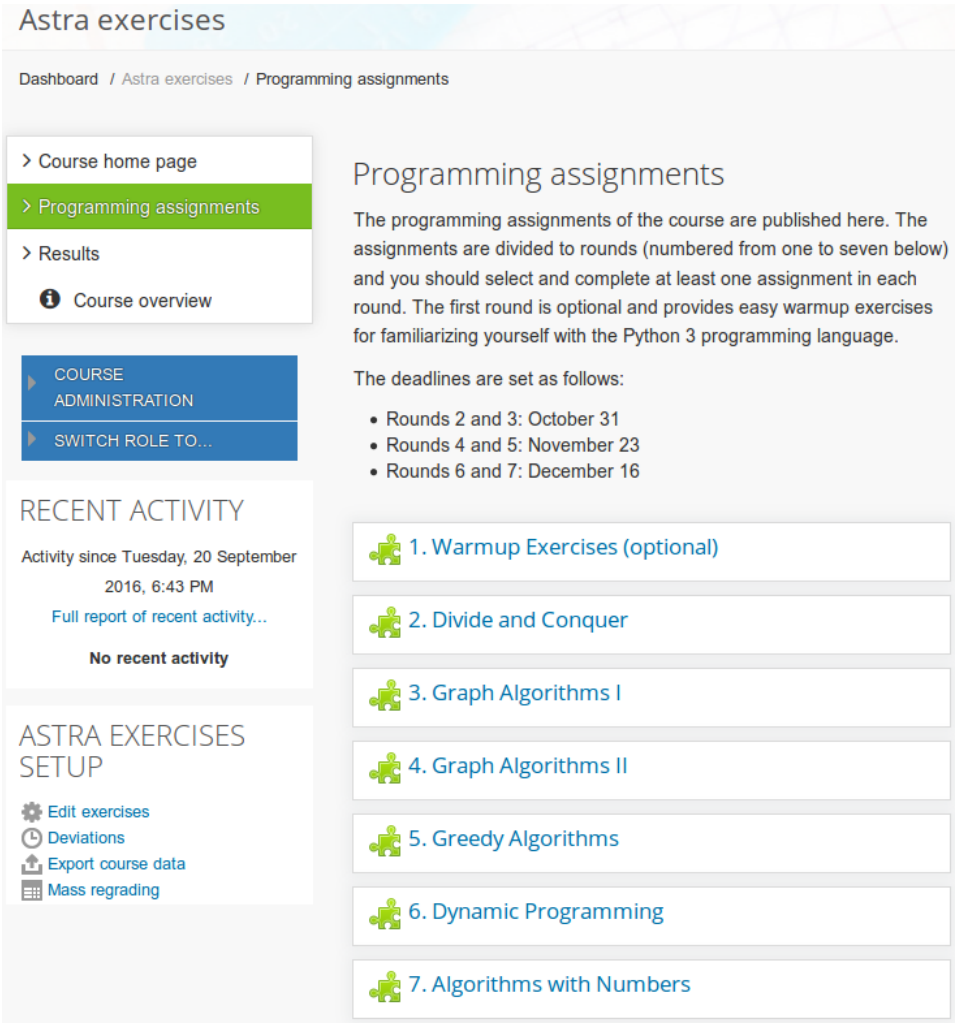

Figure 4.1: A course page in MyCourses. It shows the Astra setup block in the bottom left corner and a few exercise round activities numbered from one to seven.

4.2 Astra

Astra, the Moodle plugin that replicates A+ functionality, is implemented as an activity module plugin. This means that the plugin is displayed in a Moodle course in the same way as other activities, such as forums. Additionally, the module plugin is bundled with a small block plugin that adds a small link box to the sidebar of a course. The box is only visible to teachers as the links point to administrative functionalities. The administrative functions are implemented in the module plugin, however, the block plugin is needed

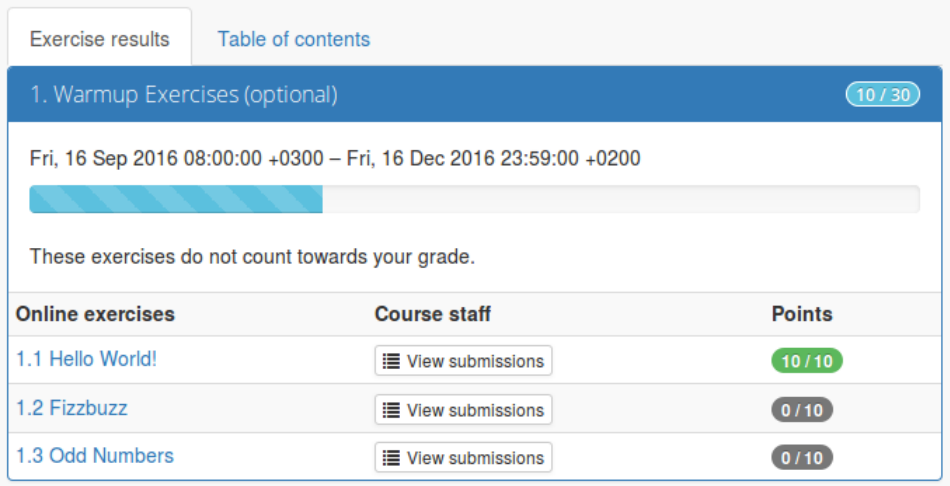

Figure 4.2: Astra exercise round page in MyCourses (without course navigation bars). The course staff column is not shown to students.

to provide teachers easy access to the functions from the Moodle course page. Figure 4.1 illustrates a course page in MyCourses with the Astra setup block and a few Astra activities, exercise rounds numbered from one to seven. This example course is based on the course Principles of Algorithmic Techniques at Aalto University.

One Astra activity instance represents one exercise round. This implies that the Moodle course page lists one activity for each round. This was deemed a better design than having each exercise as a separate activity since the number of exercises could be high. Opening an activity shows an overview of the round with the user's total points and a list of the learning objects in the round. Individual learning objects open in new pages. Figure 4.2 presents the exercise round page in MyCourses: this page opens when the user clicks one Astra activity in the course page. The student has earned points in one of the three exercises in the round.

4.2.1 Architecture

Figure 4.3 illustrates the high-level components of the Astra plugin. The components in the bottom, database and Moodle APIs, form the core of the plugin as they are utilized by almost all functionality, in particular by all of the crucial functionality. The user interface is placed at the top as other components do not depend on it.

Relational **database** stores all the data of the plugin and is thus a crucial component. Moodle provides database-agnostic APIs for defining the

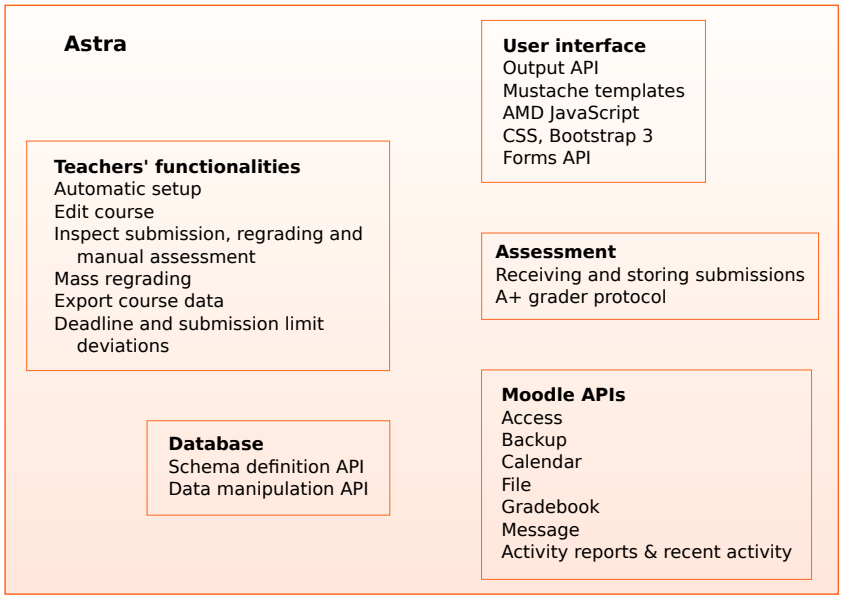

Figure 4.3: Components of the Astra plugin.

database schema and manipulating the stored data, in other words, the database APIs hide differences in the database management systems supported by Moodle so that plugins may operate on any database system. Astra defines tables for storing the configuration of exercise rounds, learning objects, and categories. Submissions as well as deadline and submission limit deviations are also stored in their own tables. In practice a Moodle plugin uses the database via APIs, however, the importance of the database justifies distinguishing it from the other Moodle APIs.

Astra utilizes several **Moodle APIs**. Access API is used for access control based on Moodle user accounts and capabilities required to perform tasks. Backup API makes backups of course content that can be restored or copied to other Moodle courses. Calendar API sets events to the Moodle calendar, which the plugin uses for exercise round deadlines. File API is used to store files that students submit to exercises. The plugin implements the API used by Moodle activity reports and the recent activity block so that they can take submissions to Astra exercises into account. The recent activity block shows new updates in the course since a user's last login, in this case new submissions and newly received asynchronous assessment results. The Moodle gradebook shows a student's grades in one location, hence the plugin fills in the student's best grades for each exercise and round to the gradebook. Message API is used to send messages to students when a teacher or assistant has given feedback to their submission (additional feedback to the machine-

generated feedback) and/or changed the points of the submission. Users may choose in their messaging preferences whether message notifications are also sent by email.

The **assessment** of submissions is a crucial component in the system. It consists of receiving and storing the submissions as well as sending them to an exercise service for assessment. The plugin communicates with exercise services using the A+ grader protocol. Submissions have different states when they are being assessed: firstly, they are *initialized* when they are received, secondly, they are *waiting* when they have been sent to the exercise service but not yet been assessed, and finally, they are *ready* when Astra has received the assessment. The submission is set to an *error* state if the exercise service fails to assess the submission normally. Submissions in the error state are counted as normal submissions when the number of submissions is limited because the submissions should eventually be regraded and thus transitioned to the ready state. Astra does not start regrading automatically, hence the teacher must do it manually once the issues in the exercise service have been resolved. Individual submissions can be regraded from the inspection page and the mass regrading function may regrade all submissions in the error state.

Teachers' functionalities cover operations available to teachers used to set up and manage courses. The operations access and modify the database, as they may change course configuration and the assessment of submissions. These functionalities correspond to the features for teachers introduced in Section 4.1.

The **user interface** consists of several subcomponents. Templates define the content of web pages (HTML) and utilize JavaScript modules written in AMD (Asynchronous Module Definition) format [1], which is supported by Moodle. The templates use Mustache template engine [5] as it is the only template engine Moodle supports. Furthermore, the plugin defines renderable classes as part of the Moodle Output API. The classes prepare the rendering context for the templates, that is, set values to template variables. The plugin also uses CSS (Cascading Style Sheets) to define the layout of the web pages. Particularly, the plugin includes Bootstrap 3 framework [2] as it is also used in A+ and the MOOC grader, which implies that it is simpler to replicate a similar user interface when the plugin uses Bootstrap 3 as well. Moreover, the HTML fragments that originate from the MOOC grader (i.e., exercise descriptions and feedback) use Bootstrap 3 components and the plugin must be able to display them. Astra includes only selected components of the Bootstrap framework because a common base theme in the Moodle core called "Bootstrap Base" also uses Bootstrap, but unfortunately the old version 2. The plugin tries to avoid conflicts between the different

Bootstrap versions; it does not use any JavaScript modules from Bootstrap 3 but instead uses the modules from the version 2. Finally, forms in the user interface are defined and output using Moodle Forms API.

4.2.2 Code Organization

Figure 4.4 on page 34 presents the organization of the source code of the Astra plugin, that is, how the source code is organized into files and directories in the file system. The figure shows the directory structure and some files; most files are not shown to avoid unnecessary detail since the directories already convey the high-level structure. The Astra codebase contains approximately 15 thousand lines of code, including comments and white space. The organization is explained in more detail in the following. Some parts refer to the Moodle cache: updating the cache in production servers requires increasing the version number of the plugin (`in version.php`). In a development server, the caches can also be forced to clear and rebuild. Many parts are directly related to Moodle APIs or the structure that Moodle expects from an activity module plugin: the relevant sections in the Moodle developer documentation [37] should be studied to understand the APIs.

- `amd`: front-end JavaScript code as AMD modules, i.e., the format expected by Moodle. Minified JavaScript modules are located in the `build` directory, whereas the original source is in the `src` directory. Moodle caches the minified modules. Most modules are derived from the A+ source code with no or minor changes in the functionality. The modules are also wrapped in the AMD structure since the original JavaScript code in the A+ platform does not employ AMD modules.
- `assets`: CSS style definitions for the front end, including the Twitter Bootstrap 3 framework and some CSS rules derived from A+ (in `main.css`). There are a few CSS rules that resolve conflicts between the Bootstrap 3 rules and the rules from the MyCourses theme as well as the default Moodle theme ("Clean theme").
- `async`: PHP scripts that receive asynchronous assessment from an exercise service. They are part of the A+ grader protocol implementation.
- `backup`: implementation of the Moodle Backup API.
- `classes`: PHP class definitions that support class autoloading (no require/include statements needed to use the classes). Moodle sets the class autoloading for this directory and expects that subdirectories correspond to PHP namespaces.
  - `autosetup`: implementation of the automatic setup of course exercises based on the MOOC grader course configuration. The

Figure 4.4: Organization of the source code of the Astra plugin.

class parses the JSON-formatted configuration and updates the database accordingly: it creates new objects or modifies existing ones. The MOOC grader configurations for exercise rounds, learning objects, and categories have unique keys that identify the objects so that existing objects may be updated in the database.

- `event`: event classes that are a part of the Moodle Event 2 API and used to add entries to the Moodle logs. Moodle caches the event classes.
- `export`: implementation of the export of course data in the teachers' functionalities. These classes gather course-wide data about the students' results and submissions, process them, and output the collective data in a structured format.
- `form`: form classes for the user interface that use the Moodle Forms API. The forms define the fields of the form and the validation of the input.
- `output`: part of the Moodle Output API; these classes define the values of the template variables for rendering Mustache HTML templates. The renderer class defines which template file is used to render instances of the output classes.
- `protocol`: implementation of the A+ grader protocol. The remote page class issues the HTTP requests to exercise services and parses the responses. The requests are sent using the PHP client URL library.
- `summary`: classes that collect an overview of a user's status and points in exercises using a minimal amount of database queries. The data collected by the classes is used in the user interface, more precisely in the output classes, for instance in the exercise round page.
- `task`: classes for the Moodle Task API, which is used to execute potentially long-running tasks in the background. Astra has only one task, which is used for the mass regrading of submissions.
- `urls`: class that defines the URLs of the Astra plugin in one place; the URLs correspond to the file paths of PHP scripts. If a file path must be changed, updating the reference to the URL in other parts of the code requires only updating the file path in this class.
- Classes directly under the classes directory represent the database relations defined by Astra, for instance, exercise rounds and learning objects. One instance of a class corresponds to a tuple (row) in the database. The classes are used to access and modify tuples in the database and they also implement the Moodle Gradebook and Calendar APIs.

- `db`: a directory that holds files that are used by many Moodle APIs particularly during the installation and upgrade of the plugin.
  - `access.php`: defines capabilities that are used in access control with the Moodle Access API. A page or function of the plugin may require that the user has a certain capability in order to proceed, otherwise the access is denied.
  - `install.php`: installation procedure for the plugin that is executed after installing the database tables. Uses the Moodle Upgrade API.
  - `install.xml`: defines the database schema of the plugin. The API in Moodle for defining database schemas in a database-agnostic way is called XMLDB.
  - `messages.php`: declares message providers and types for the Moodle Message API. Astra declares only one message provider for the notification about new manual assessment and feedback in a submission.
  - `uninstall.php`: uninstallation procedure that is executed before Moodle deletes the database tables of the plugin.
  - `upgrade.php`: upgrade procedure for modifying the database schema. This is needed if the database schema is changed after the plugin has been deployed and installed in production servers.
- `lang`: defines strings for the user interface with the Moodle String API. Moodle caches the strings. The strings may be localized to several languages, although Astra includes only English strings. Each string includes a key and the actual string content; the plugin retrieves strings using the key and Moodle selects the language according to the user's preferences.
- `pix`: icon for the Moodle activity that is displayed in the course page.
- `teachers`: PHP scripts for the teachers' functionalities described earlier. The scripts are mostly part of the front end for the functionalities, while the back end is implemented in the `classes` directory.
- `templates`: Mustache HTML templates that are a part of the Moodle Output API. Most of the Astra-specific components in the user interface are defined in the templates. The majority of the templates are derived from A+ templates. However, A+ as a Django application utilizes a different template language than the Mustache language in Moodle, hence the templates from A+ must be manually converted to the Mustache format.
- `tests`: unit tests based on the Moodle Testing API, which is built on the PHPUnit framework.

- Most of the PHP scripts directly under the `astra` directory correspond to pages in the user interface. For example, `exercise.php` and `submission.php` are used for the exercise description and submission feedback pages, respectively.
- `view.php`: required by Moodle and used to display one instance of the activity. Astra uses the page to present an overview of the exercise round.
- `lib.php`: defines functions that Moodle requires from all activity module plugins.
- `index.php`: Moodle expects this script to list all instances of the activity in the course. Astra utilizes it to show an overview of the course: all exercise rounds, exercises, and categories with the user's current points.
- `mod_form.php`: defines a form that is used to create or edit activity instances, in this case exercise rounds. Required by Moodle. It is not recommended to add exercise round activities in the same way as Moodle activities are normally added to a course, which would be from the "add activity" menu in the course front page with editing mode turned on. Since categories and learning objects can only be added in the teachers' administrative pages, accessed from the Astra setup block, it is recommended to modify exercise rounds from the same place.
- `version.php`: defines the version number of the plugin and its dependencies on other plugins and the Moodle platform. Required by Moodle.

4.2.3 Database Schema

Moodle utilizes a relational database and provides a database-agnostic API for defining the database schema of a plugin: for installation, the schema is defined in XML format, and for post-installation upgrades, the modifications for the schema must be defined in PHP code using the Moodle Upgrade API. This section presents the database schema of the Astra plugin. Since most of the Astra functionality is based on the A+ platform, the schema has similarities with that of A+. However, there are also differences because the platforms are not identical.

Figure 4.5 on page 38 illustrates the entity-relationship (ER) model of the Astra plugin. While there are several different notations for ER diagrams, the notation described in [49, pp. 120–151] is used in this thesis. However, there are some caveats. First, only a few of the attributes for each entity set are shown in the diagram so that it is more readable and fits on a single page. Second, since not all attributes are shown in the diagram, keys are

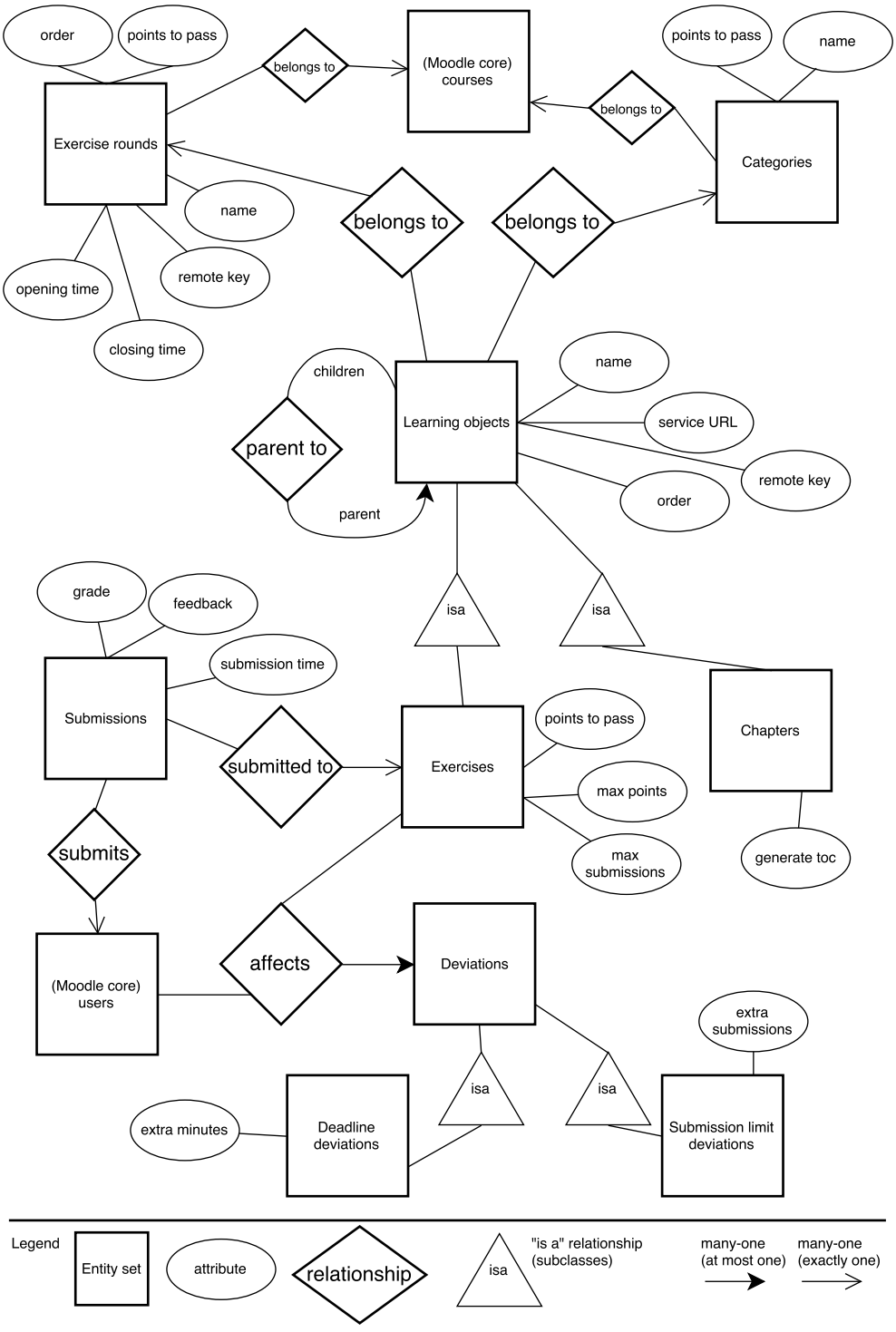

Figure 4.5: Entity-Relationship diagram for the Astra plugin.

not denoted either. Third, the diagram uses a different arrowhead than [49] for referential integrity constraints: the original source uses a rounded arrowhead for many-one relationships from E to F that require that the entity in F related to an entity in E exists, but the diagram uses a sharp, open arrowhead instead. The many-one relationships without any referential integrity constraint ("at most one entity") use the same black arrowhead as the source. Finally, the diagram excludes an entity set for certain course configuration options for space reasons.

The ER diagram in Figure 4.5 presents the relationships between the different types of content supported by Astra. A course consists of exercise rounds that have their own opening and closing times. A round belongs to a course that must exist in the database (the used arrowhead implies referential integrity). A round consists of learning objects that each have their own service URLs and names. The learning objects have an attribute *order* so that they may be displayed in the correct order in the user interface. The objects may be optionally organized into a hierarchy so that an object has child objects that are listed under the parent object. Learning objects are either exercises or chapters. There are no "plain learning objects" in Astra that would be counted as learning objects but would not have a subtype, exercise or chapter. When a chapter page embeds exercises, the exercise objects must be set as children to the chapter object. Each learning object also belongs to a category, while each category belongs to a course. The Astra plugin does not model courses as the Moodle core already does that.

Submissions store the time of reception and eventually the results of the assessment. A submission is made by a single user to a single exercise. Deviations are either deadline deviations or submission limit deviations, i.e., they extend the deadline by a certain time or the maximum allowed number of submissions, respectively. A deviation ties one user and exercise to the deviation rule, and a user may have only one deviation rule of a subtype active in one exercise. The deviation subtypes, deadline and submission limit, do not exclude each other regarding the referential integrity constraint, i.e., a user could have one deviation rule of both types in the same exercise.

Moodle has a few restrictions on the relational database schema. Even though Moodle supports defining foreign keys (FK) in the XML schema, Moodle ignores referential integrity constraints in the database and hence, the database management system does not verify whether the data violates such constraints. Each relation must have an *id* attribute as the primary key (PK) even if other attributes could form the primary key. The id attribute is an integer and incremented serially. The name of each relation must begin with the name of the plugin in lowercase. For other plugin types than activity modules, the names would also include the plugin type. The main relation

of the plugin must be named *astra*, based on the plugin name, and it must store the instances of the activity. Since Astra represents exercise rounds as activities in Moodle, the relation *astra* stores the exercise rounds. The main relation in an activity module is expected to have certain other attributes as well: *course* is a foreign key to the course relation defined in Moodle core, *name* defines a name for the activity instance, *timemodified* holds a timestamp of the last modification, *intro* and *introformat* store a description of the activity instance and its format code (usually HTML), and finally, if grading is involved, attribute *grade* stores the maximum grade for the activity. In the case of Astra, storing the maximum available points of an exercise round is redundant since the maximum points depend directly on the sum of the maximum points in the exercises of the round. However, rounds store the maximum points in order to comply with Moodle.

Figure 4.6 on page 41 illustrates the actual database schema of the Astra plugin, i.e., the defined relations. The conversion from the ER diagram is rather straightforward: entity sets become relations and many-one relationships become foreign keys in the relations. The arrows in the figure only visualize the connections between the relations through the foreign keys: the arrowheads have no special meaning in contrast to the ER diagram. Learning objects are defined in several tables: each object has a tuple in the relation *astra_lobjects* and in one of the subtypes, chapters or exercises. The common attributes to all learning objects are stored in the parent relation; there are several of such attributes and this arrangement avoids repeating the attributes in each subtype relation. It is also possible to add new learning object subtypes as new relations. On the other hand, the relations for deviations do not use a parent relation to store the common attributes: there are only two of them, *submitter* and *exerciseid*. Therefore, both relations for deviations include all attributes that are required for that kind of a deviation rule.

The schema in Figure 4.6 includes some attributes that have not been explained yet. The relation *astra* has *status* so that an exercise round may be hidden from students or set to a maintenance state even if the round has opened according to the opening time. Categories and learning objects may also be hidden. Attribute *remotekey* matches the keys in the MOOC grader course configuration. Exercise rounds have attributes *latesbmsallowed*, *latesbmsdl*, and *latesbmspenalty* for the configuration of late submissions: rounds may accept submissions after the closing time until a separate late submission deadline. Late submissions may be penalized by reducing a percentage of the points. The relation for chapters has attribute *generatetoc* so that an automatically generated table of contents of the round may be enabled or disabled. The relation for exercises has *gradeitemnumber* that is used with

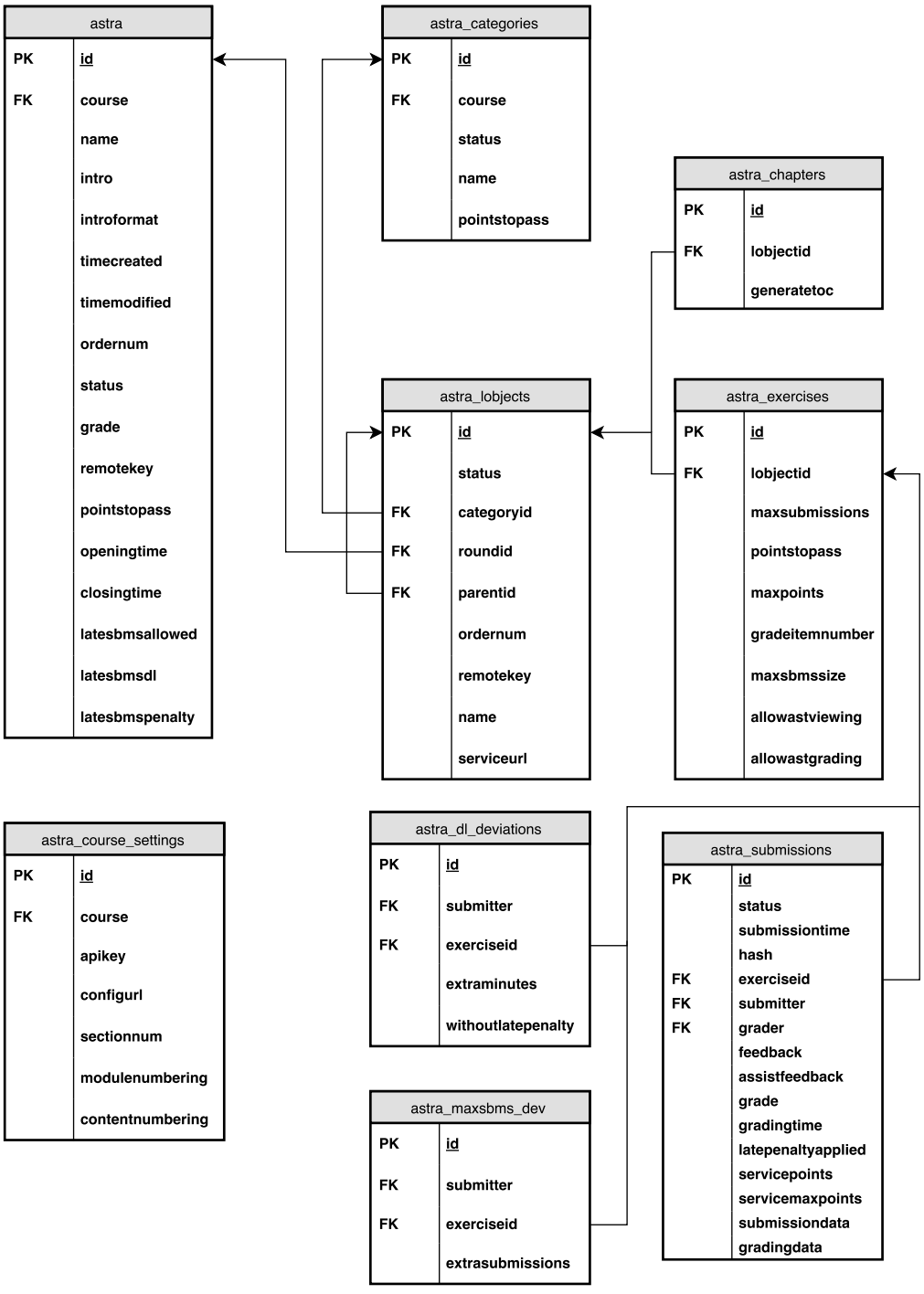

Figure 4.6: Database schema for the Astra plugin.

the Moodle gradebook. The exercises of a round have separate grade items in the gradebook, but they all belong to the same activity instance (the round), hence the exercises need separate item numbers. The item numbers must remain unchanged after the initial creation, therefore they are not based on the ordinal numbers of the learning objects. Exercises also have an option for limiting the maximum accepted file size of submissions. Furthermore, the teacher may limit in which exercises the assistants of the course may inspect and manually assess submissions, assuming the assistants are non-editing teachers in the Moodle course. This provides finer-grained control over what the assistants are allowed to do. Submissions have attribute *hash* for storing a random string that can not be guessed. It is used in asynchronous assessment so that only the exercise service may post grading results to Astra. A submission's status is the state of the submission: initialized, waiting, ready, or error. Submissions also store the original points and maximum points used by the grader (*servicepoints*, *servicemaxpoints*) before they are scaled to the maximum points used in the front end. Attribute *submissiondata* stores the content of the submission (i.e., data entered in a form), excluding files, and *gradingdata* stores potential additional data about the grading. Finally, the relation *astra_course_settings* is used for course-wide configuration. It was excluded from the ER diagram. Attribute *configurl* saves the URL used in the automatic course setup based on the MOOC grader configuration. Attributes *modulenumbering* and *contentnumbering* store the numbering schemes used for exercise rounds and learning objects, respectively, while *sectionnum* is used for the Moodle course section in which activity instances are located. The MOOC grader does not currently employ any kind of API keys for restricting access, however, the attribute was left in the database in case it could become useful at some point.

4.3 Upgraded MOOC Grader Exercise Service Framework

The MOOC grader is upgraded to support personalized exercises. A personalized exercise must define a generator program that generates new instances, which are in practice arbitrary sets of files. The files may, for example, contain data or program source code that is given to students in the exercise description or they may contain hidden data used by the grader, such as a seed for a random number generator or the correct answer. Each instance should have the same uniform structure, i.e., the same number of files with the same names, but the contents of the files may vary. When the MOOC

grader executes the generator, it is given the path to a directory in which one new instance should be generated as an argument. The generator could create the instance randomly, but if the generator should be able to repeat the same instances at another time, the random number generator could be instantiated with the path argument given to the generator program so that each instance can be generated deterministically. The teacher, or a server administrator that has access to the server, creates a number of exercise instances before the start of the course via a new Django `manage.py` command "`pregenerate_exercises`" which executes the actual generator program multiple times (the amount of instances is given as an argument). Creating the exercise instances is simple for those that are familiar with the command line. Since the command is part of a Django application, it starts from the `manage.py` script, but otherwise the teacher need not know the Django framework. Each student is assigned an instance, which is explained in detail later. The MOOC grader is able to identify users with the upgraded A+ grader protocol.

The exercise description (an HTML document) that a student sees in the front end is defined as a template in the MOOC grader. The MOOC grader has default templates that may be used if it is sufficient to fill in some exercise-specific content via configuration options. A course may also define its own templates for each exercise. The exercise description templates may use generated instance files so that the description may include any necessary data that the student needs to solve a personalized instance of the exercise. New template variables allow displaying the contents of the exercise instance files in the exercise description. If it is more appropriate to have the student download the instance file instead of embedding the file content in the description, new template variables also allow adding download links to the instance files. The teacher controls which files may be downloaded in the exercise configuration.

As with the exercise description templates, the assessment process may utilize any generated exercise instance files. The instance files may be copied to the grading sandbox and used during the grading. Assessment (of asynchronous exercises particularly) is configured in the MOOC grader by giving a sequence of actions: the most common actions are the *prepare* action, which is used copy files to the grading sandbox, and the *sandbox* action, which is used to run grader programs in the sandbox. The prepare action is extended so that it has options to copy the generated instance files to the sandbox as well as files from the personal directory of the user. The personal directory is an optional feature that may store user-specific files between grader executions (i.e., different submissions). There is a separate action for copying files from the grader sandbox to the personal directory. It is recommended that

Algorithm 4.1 Assignment of an exercise instance to a user.

```
function SELECT_EXERCISE_INSTANCE(submission_number, userid, exercise)
                                  ▷ The submission ordinal number starts from 1.
                ▷ A personalized exercise has a number of generated instances.
    N ← number of generated exercise instances
    if exercise uses regeneration then
        regen_limit ← exercise.max_submissions_before_regeneration
        instances ← list of integers from 0 to N − 1
        rand ← RANDOM(userid)                                  ▷ Initialize seed
        rand.SHUFFLE(instances)                                ▷ Shuffle the list
        idx ← TRUNC((submission_number − 1)/regen_limit) mod N
        instance ← instances[idx]
    else
        instance ← userid mod N
    end if
    return instance
end function
```

exercises are designed to depend only on the generated instance files so that references to the personal directory can be avoided altogether. Using the personal directory complicates the grading and makes it more error-prone since the personal directory stores a state across all submissions and it may be modified by any grader execution on any of the submissions. Moreover, the same submission could be regraded multiple times.

In personalized exercises, each student is assigned one exercise instance based on the student's user identifier and the number of available instances. The MOOC grader does not store the mapping of users to instances since it can always be computed when needed; the upgraded A+ grader protocol supplies the user identifier. If the teacher has enabled regeneration in a personalized exercise, the student's assigned instance is switched to another one once the student has submitted as many solutions as the set threshold. Pseudo-code for the assignment of personalized exercise instances and the regeneration is given in Algorithm 4.1. When regenerating, the instance is switched in a way that cycles through all possible instances before repeating previously seen instances and the cycles are different to different students. This ensures that students may not easily predict to which instance they are assigned. The cycling of instances is implemented with a random-number generator that is initialized with a seed based on the user identifier. This makes the random number generation deterministic and different to each

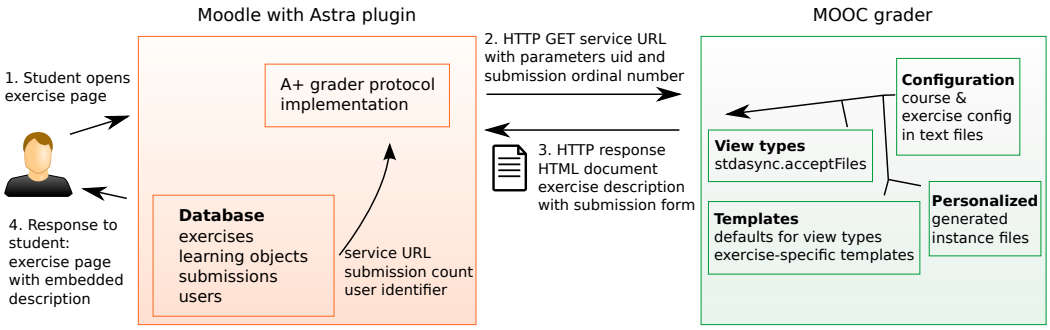

Figure 4.7: Retrieval of exercise description.

user. The random-number generator is used to shuffle the list of all instances so that each student receives a different cycle of the instances. The shuffling must be deterministic for each user so that they are guaranteed to cycle through all possible instances before repeating previous instances.

4.4 Example: Accepting and Assessing a Submission

This section illustrates the functionality of the whole system, including both the Astra plugin and the MOOC grader, when a student submits a new solution for assessment. The example begins with the student opening the exercise page in Moodle, which is illustrated in Figure 4.7. In this example, the exercise is asynchronous as well as personalized and the student prepares a submission in a file. First, the student opens the exercise page in Moodle (part **1** in the figure). Moodle reads exercise configuration and user data from the database, in particular the service URL of the exercise, the user's identifier, and her current submission count. Moodle begins retrieving the exercise description from the exercise service by using the A+ grader protocol. The exercise service runs on the MOOC grader platform. Moodle sends an HTTP GET request to the MOOC grader with parameters that specify the user's identifier (uid) and the ordinal number for the next submission (**2**). The ordinal number is needed if the exercise uses regeneration and it is incremented by one from the current submission count so that the exercise description matches the state of the grader for the next submission. Only the most relevant parameters of the grader protocol are highlighted here. The MOOC grader receives the HTTP GET request and determines the course and exercise from the service URL. It reads the course and exercise configurations and observes the view type and template of the exercise as well as that the

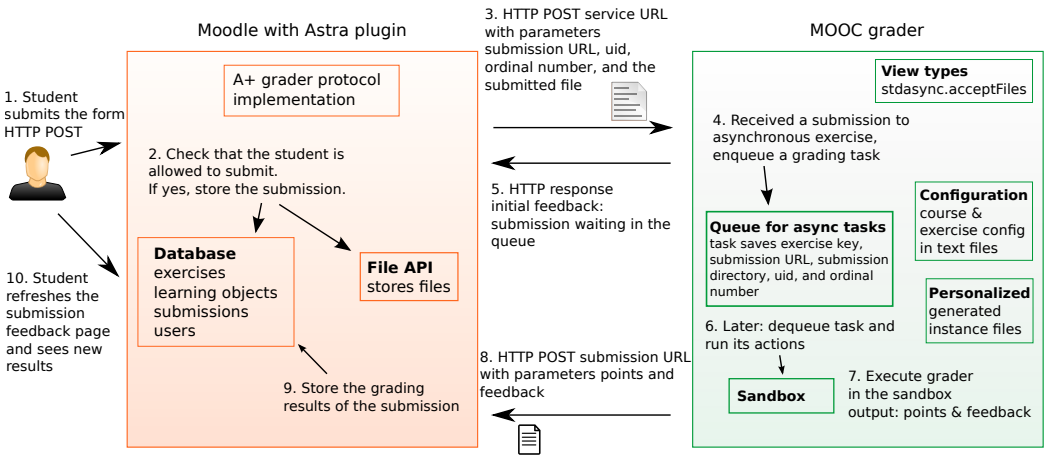

Figure 4.8: Assessment of a new submission.

exercise is personalized. View types define the functionality of an exercise in the MOOC grader. This example uses a view type that expects submissions in the form of files and the submissions are assessed asynchronously. The MOOC grader renders the exercise description template, taking into account the generated files of the exercise instance since the exercise is personalized. It responds to the HTTP request with an HTML document that contains the exercise description and a submission form (**3**). Moodle receives the exercise description with the form and embeds them into the exercise page that also includes other standard components of a Moodle page, such as navigation. The student receives the exercise page in the HTTP response to her original HTTP request for opening the exercise page (**4**).

The student views the exercise description and completes her solution by modifying a file on her computer. For example, the exercise could be a programming task and the student writes program source code to a file. Figure 4.8 illustrates the submission and assessment. The student is ready to submit her solution: she selects the file in the submission form on the exercise page and submits the solution to Moodle (part **1** in the figure). Moodle checks that the student is allowed to make new submissions to the exercise: the exercise round may have a deadline and the exercise may limit the number of allowed submissions (**2**). The submission is accepted and Moodle stores it in the database; the submitted file is handled by Moodle File API. Moodle sends the submission to the MOOC grader for assessment by issuing an HTTP POST request to the service URL of the exercise (**3**). The exercise configuration and user data are read from the database. The POST request includes parameters: submission URL to which the MOOC grader may asyn-

chronously send grading results, user's identifier, the ordinal number of the submission, and the submitted file itself. The MOOC grader receives the submission and reads the course and exercise configurations. The service URL identifies the exercise. The view type of the exercise, "acceptFiles", causes the MOOC grader to store the submitted file in the file system and to enqueue a grading task to the queue of asynchronous tasks (**4**). The MOOC grader responds to the HTTP request with initial feedback stating that the submission will be graded at a later time (**5**). At some point, the MOOC grader dequeues the grading task (**6**). The task contains the data needed to assess the submission: the exercise, the submission URL, the path of the directory holding the submitted file, the user identifier, and the ordinal number of the submission. The MOOC grader executes the grading actions specified in the exercise configuration: in this case, the prepare action copies the grader program and the personalized instance files to the submission directory, which is then copied to the sandbox and the grader is executed safely. The grader outputs the points and feedback for the submission (**7**). The MOOC grader deletes the submission directory and sends an HTTP POST request to the submission URL with the points and the feedback (**8**). Moodle receives the grading results and verifies that they originated from the exercise service by checking the IP (Internet Protocol) address. Moodle updates the submission in the database with the new points and feedback; the submission URL identifies the submission (**9**). The student may refresh submission feedback page at some time and view the new feedback and points for the submission (**10**).

Chapter 5

Evaluation

This chapter evaluates the new e-learning platform that consists of the Moodle plugin, Astra, and the upgraded MOOC grader framework. Section 5.1 begins by assessing the new platform against the requirements set for the project. Section 5.2 estimates whether the new system is dependent on other platforms and how easily this level of integration could be transferred to other platforms. Finally, Section 5.3 discusses the relationship and differences between Moodle and A+.

5.1 Requirements

The requirements for this thesis project were set in Section 1.1. This section evaluates the realization of each requirement. **Requirement 1** addresses the maintainability of the platform as well as its flexibility for future development. The new platform does not employ any code from the old Stratum infrastructure, hence no code quality issues are directly transferred from the old platform to the new one. The new platform consists of two components, Astra and the MOOC grader. First of all, the underlying platforms, Moodle and Python Django web framework for the MOOC grader, are open source and in active development. There is no sign of end of life for the underlying platforms, although previously released versions reach end of life eventually as new versions surpass them. The MOOC grader has a clean, modular implementation and the upgrade for personalized exercises is implemented as a new module. The architecture of the Astra plugin is largely determined by the underlying Moodle framework. The implementation aims at using object-oriented programming when possible and organizing code into modules. Not all Moodle APIs support object-oriented programming, particularly the older APIs do not. Moodle and PHP do not provide a good way for avoiding the

repetition of initialization procedures in PHP scripts that correspond to web pages in the user interface. Such initializations include setting up arguments (usually HTTP GET query parameters), Moodle navigation items for the user interface, and querying the database for necessary tuples. The Moodle Data manipulation API for accessing records in the database is rudimentary: there is no object-relational mapping and the Moodle API supports only simple queries that, for example, retrieve records from a single table with a filter that compares the equivalence between a column value and a given argument. Any more complex query must be written in SQL (Structured Query Language), which arguably makes the code harder to read.

The usage of the Bootstrap front-end framework introduced unexpected issues that affect the quality of the code by a small amount. The A+ platform and the MOOC grader utilize Bootstrap 3, hence the Astra plugin uses it as well to retain compatibility. However, Moodle default themes as well as the MyCourses theme extend a base theme that is dependent on the outdated version 2 of Bootstrap. In order to avoid conflicts between the Bootstrap versions, Astra includes only a subset of the CSS rules in Bootstrap 3 and uses JavaScript from the Bootstrap version 2 in the Moodle core. In addition, Astra includes a little less than 20 CSS rules for fixing visual defects in the user interface that are caused by conflicts between the CSS rules from multiple sources. Since Astra partially includes the Bootstrap 3 framework, it may be sometimes necessary to upgrade the Bootstrap version, i.e., download the latest version from the Bootstrap web site. The web site has a tool for selecting the components included in the download, which also includes a configuration file that lists the selected components; hence, upgrading the partial Bootstrap framework in Astra is simple.

The development of the Moodle plugin requires knowledge of Moodle, its plugin architecture, APIs, and relational databases as well as knowledge of the A+ architecture and grading protocol. The Moodle developer documentation usually provides sufficient information about the APIs, however, it is sometimes necessary to study the Moodle source code. To conclude, the MOOC grader fully realizes Requirement 1, while the Astra plugin has some issues in this regard, however, they are related to the Moodle platform and cannot be completely resolved. Thus, Requirement 1 is realized.

Requirement 2 concerns the usability of the system. From a student's perspective, all course content is integrated in Moodle seamlessly in the normal user interface. Students should be accustomed to interacting with Moodle courses and activities, therefore they find Astra exercise round activities as easily in the Moodle course. Furthermore, students can view their grades in the Moodle gradebook as with any other graded activity. Key elements in the Astra user interface have been replicated from A+, such as the view

of an exercise round with the current points (shown in Figure 4.2), or the layout of the exercise description page that has a dropdown menu for all the submissions and information boxes in the sidebar about the deadline and other metadata. The content of the exercise description is exercise-specific and originates from the exercise service, hence its usability is out of scope for this thesis, which concentrates on the platform, not the exercises themselves. One problem in the user interface is that it does not highlight the index pages of activities well. In general, activity modules must have an index page that lists the instances of the activity in the course. Astra uses the index page to show an overview of all exercise rounds with the points similarly to the page of a single round. The Moodle course pages do not have links to the index pages by default unless the course has enabled the standard "Activities block". Even with the block enabled, the links are easily missed in the sidebar of the page. The teacher could manually add a link to the index page somewhere in the main content of the course page so that students find it more easily.

From a teacher's perspective, most of the Astra user interface is again replicated from A+. The A+ user interface has been found to be functional and Astra utilizes it as a baseline rather than creating something new for the same use cases. The replicated pages include, for example, the pages for inspecting and manually assessing submissions. Teachers need to manually add the Astra setup block to the Moodle course once at the beginning of a course in order to reach the Astra setup pages easily. Setting up the exercises in Moodle is effortless if the exercise service utilizes the MOOC grader as the teacher only needs to enter a URL to run the automatic setup of the course.

Usability from a teacher's perspective is not limited to just the Astra plugin as the teacher needs to set up the exercise services as well, typically in the MOOC grader. The MOOC grader does not have a user interface for creating course content; instead, teachers write text files that define the configuration. The configuration files could be created directly in the MOOC grader server, or more preferably, in a Git repository so that the course may be updated via Git operations. For example, if the central Git repository is in a Gitlab server, the teacher may set a webhook in Gitlab so that Gitlab notifies the MOOC grader when new updates are pushed to Gitlab. The Git manager in the MOOC grader then pulls the updates from Gitlab. Using Git is convenient for teachers that have already previously learned Git, and it simultaneously provides version control of the course. The course configuration files have their own structure that the teacher must study from the MOOC grader documentation. Generally, the configuration consists of key-value pairs: the required keys and accepted values are listed in the documentation. Furthermore, teachers need to develop graders for the exercises

of the course, i.e., the programs that grade a submission automatically and generate feedback for it. The creation of new exercises requires programming skills as the MOOC grader can not create new grader programs by itself. In order to test the graders during their development, a teacher may install the MOOC grader in her computer and submit solutions directly to it without going through the hassle of installing the actual front-end system (Moodle or A+). Finally, teachers may create the content for chapters (learning objects with study material) in the MOOC grader as well. Chapters can be written directly in HTML or in RST (reStructured Text) format that is compiled to HTML afterwards. The latter is more human-readable as it has less syntactic markup cluttering the content. Embedding exercises into chapters requires a small amount of special markup.

To conclude, the user interface of the Astra plugin is more polished than the rudimentary web user interface of the old Stratum platform. For teachers, the Stratum Moodle plugin has a similar automatic setup feature as Astra. The course configuration in the MOOC grader is based on files like in Stratum, though the YAML-based (YAML Ain't Markup Language) format in the MOOC grader is more human-readable. When developing grader programs, testing them is easier in the MOOC grader as the MOOC grader can be easily installed in a personal computer, whereas Stratum can not be separated from its whole software stack, that is to say, making test submissions from a web browser requires web server software with a PHP interpreter in addition to Stratum. Teachers require some technical skills to utilize the MOOC grader effectively, however, if they are to program graders themselves, writing configuration files or using Git should be simple compared to that. Thus, the new platform has improved the usability of the system for both students and teachers, and Requirement 2 is realized. Admittedly, the platform has not undergone any formal usability testing because it does not fit into the scope of the thesis.

Requirement 3 demands that assignment packages from Stratum must be portable to the new platform with reasonable effort, while **Requirement 4** addresses the support for the same basic functionality that Stratum has and, in particular, personalized exercises. The porting of Stratum assignment packages, i.e., grader and generator programs, is interleaved with the support of personalized exercises. The assessment and generation of personalized exercises (or assessment of normal exercises) are implemented in the upgraded MOOC grader framework. The MOOC grader sets few limitations on the structure of the grader or generator programs: they may be implemented in any programming language and the invocation of the program may include command-line arguments set in the course configuration. Stratum requires that the programs are written in Bash, though the Bash scripts

may invoke other subprocesses. Stratum also allows setting arguments in the configuration. Grader programs in the MOOC grader must output the points and feedback of the assessment by printing them to the standard output stream, whereas Stratum expects them to be written to files. Stratum also allows the grader to produce additional files that are stored in the student's personal directory, so that the home page of the student could link to them. These files could be, for instance, generated images that complement the textual feedback. The MOOC grader does not currently support such additional files in the feedback. The upgraded MOOC grader supports personal directories for students in which the grader could store files, however, the directory is not accessible from the web, which would be required when a feedback HTML page links to generated images. The MOOC grader is able to disseminate files to the web, but currently the grader programs may not add or modify such files in the server that could be disseminated to the web. However, teachers using Stratum have rarely utilized such additional files in the feedback and hence, the feature has low priority.

The MOOC grader supports generator programs that create one exercise instance in a given directory. The instance may consist of several files and the files are never modified after creation. In Stratum, the generators likewise create one instance at a time but store it directly in the student's personal directory in the file system. In contrast to the MOOC grader, the generated files in Stratum are expected to consist of one file that is displayed to the student and additional hidden files for the grader. The grader programs may modify the generated files, but usually they do not. The upgraded MOOC grader includes a possibility to store personal files for a student after grading a submission, and the personal files may be utilized in subsequent assessments of submissions. The use of the personal directory is discouraged, however, as it was implemented in case it is challenging to port some Stratum assignments to the MOOC grader by only utilizing the initially generated exercise instance files. Furthermore, the MOOC grader supports the regeneration of an exercise by switching the instance assigned to the student to another one. The MOOC grader generates all exercise instances beforehand, while Stratum executes the generator at the time when a student's assignment must be regenerated. The outcome is the same, thus regeneration is supported in the MOOC grader.

The course Principles of Algorithmic Techniques (PAT) was the first one that was ported from the old Stratum to the MOOC grader. The porting was easily accomplished, as it only required the creation of new configuration files for the MOOC grader and small modifications in one Python script that is used to start all graders in the course. The modification concerned the part that outputs the feedback and points for the submission: they must

be printed to the standard output stream instead of writing to files. The porting of PAT was straightforward because the graders for all exercises share a uniform structure and they are thus started by the same script that is also responsible for outputting the results. In addition, there were no hidden dependencies on the old Stratum platform, such as a grader program that assumes that the file paths given in arguments follow a specific pattern used in Stratum. We conclude that Stratum assignment packages are portable to the MOOC grader format with reasonable effort and that Requirement 3 is fully realized. Likewise, the upgraded MOOC grader supports the grading and generation of personalized assignments, as well as their regeneration. The support of other basic Stratum functionality in the new platform is discussed next.

The features of the Stratum infrastructure that are not directly related to the assignment packages are implemented in Moodle in the Astra plugin. Most of the Stratum functionalities are supported by Astra in a similar or identical fashion. Astra accepts and stores students' submissions, sends them to an exercise service (the MOOC grader) for assessment, and stores the grading results. Astra provides the user interface for students for completing the exercises, and the user interface for teachers to inspect, regrade, or manually assess submissions. However, Stratum can collect overall statistics of a course and present them in a table. The statistics include for example, the average number of submissions and the average points in each exercise. Astra does not have identical statistics, though a teacher may browse through a list of submissions in an exercise and view the submissions and their results. A teacher may also export course data in Astra and process the data with self-written scripts. Advanced processing of course statistics could be implemented in Astra in the future. The statistics feature in Stratum was rarely used and hence implementing a similar feature in Astra had a low priority. Furthermore, Stratum has a feature that keeps track of students that submit before a bonus deadline that is separate from the normal deadline. The teacher may reward such students by some means, such as additional exam points; Stratum does not add any bonus points to any assessments. The bonus feature is not implemented in Astra since it was seldom utilized in Stratum. A teacher may again process the exported course data from Astra in order to compute bonus, if necessary. To summarize, Astra supports the same relevant functionality as Stratum and thus Requirement 4 is realized.

Requirement 5 states that the new platform must enable students to access assignments directly from Moodle. This is obviously realized by the Astra plugin. The upgraded MOOC grader supports both Astra and the A+ platform as front-end systems. Astra integrates seamlessly into Moodle and a student can hardly notice that the exercises originate from an external

service. If an exercise description has links to images or archives of base code that are disseminated from the exercise service (which is supported by the MOOC grader), then the student may see from the links that they refer to another system. The exercise services are responsible for exercise descriptions in the A+ architecture, hence Astra or A+ as the front-end system is not supposed to store parts of the exercise description, such as images. Therefore, storing and disseminating such files from the exercise service is in accordance with the architecture even though it reveals to the student that the exercise is not completely hosted in Moodle.

5.2 Platform (In)dependence

Aalto University decided to a deploy the Moodle-based MyCourses platform as the official university-wide learning management system (LMS) in 2015. The decision triggered the search for means to embed Stratum assignments in MyCourses, which resulted in the first Moodle plugin for the old Stratum platform. The experiences with the Moodle plugin guided the modernization of Stratum, the project reported in this thesis. Integration with Moodle was set as a requirement for the modernization due to MyCourses and the positive outcomes with the first Moodle plugin. However, MyCourses might not permanently remain as the official LMS at Aalto. It is hard to predict how long MyCourses will remain in use and what kind of platform could be taken into use after it.

The new system implemented in this thesis project replicates the overall architecture from the A+ platform: the front-end system is responsible for the common infrastructure and the implementation of the exercises is outsourced to exercise services. The exercise services are simplified since they may concentrate exclusively on the assessment of submissions (and the generation of personalized exercises). The front end communicates with the exercise service using the A+ grader protocol. The MOOC grader framework provides the infrastructure for exercise services, including for example the implementation of the grader protocol, which further simplifies the implementation of exercises. Exercises built on the MOOC grader framework may operate with any front-end system that implements the grader protocol, hence the exercises are independent of the platform used as the front end.

The Astra plugin integrates A+ style exercises in Moodle in such a way that the user does not notice any difference between accessing normal Moodle content and Astra exercises: the Astra exercises are seamlessly embedded in Moodle like any other Moodle content and the user is oblivious of the distributed architecture of the platform. The platform leaves an impression

that the exercises are implemented inside Moodle even though they reside in an external exercise service[1]. This is the high degree of integration that the Astra plugin implements in Moodle. Astra is not a stand-alone system: it can not operate without Moodle. The implementation of Astra shows that it is thoroughly dependent on Moodle: Astra is built on Moodle APIs and structured around requirements that Moodle sets for plugins. Reimplementing Astra without any Moodle dependencies would be an arduous task. Of course, the A+ platform is already an independent, stand-alone system that operates on its own without any other LMS, such as Moodle.

Should Aalto University decide to replace MyCourses with a different platform, possibly with another well-known LMS besides Moodle, the integration of Astra exercises in the new platform requires that the new platform implements the A+ grader protocol and other necessary functionality, such as the storage of submissions and grading results. The new platform should also provide the user interface for students and teachers: students access exercises and submit their solutions, while teachers manage the course and monitor the students' work. The replication of all A+ functionality is a non-trivial task, as this thesis project shows, though depending on the needs of the target audience, it is not necessary to replicate all of the A+ functionality with every detail. The A+ grader protocol is the crucial component that is required to retain compatibility with exercise services. Otherwise, the new platform does not have to strictly comply with the A+ implementation. It could even be possible to separate some of the functionality: the front-end system could communicate with exercise services using the A+ grader protocol, but outsource the storage of submitted files to another system that co-operates with the front-end system. This example merely shows that there is room for different kinds of approaches in implementing A+ functionality in a new platform. Currently, there is no sign of Aalto planning to switch MyCourses to some other platform, hence we assume that the Astra plugin could be in active use in MyCourses for at least a few years.

5.3 Moodle versus A+

The Astra plugin does not implement all features of the A+ platform. Group submissions are supported by A+: students may submit as a group to an

[1]Section 5.1 noted under Requirement 5 that a student may see links in an exercise description that refer to the exercise service. Even in that case, it is irrelevant from which server the student is downloading a file, such as an archive of base code, when the link to the file is in the exercise description that the student views in Moodle. The student does not need to navigate outside the exercise description in order to download the file.

exercise and each group member receives the points and feedback for the submission. Astra does not support groups: students submit to exercises only individually. Support for group submissions could be implemented in the future. The A+ platform has a REST API for accessing course data over a network, whereas Astra has none. The teacher may export course data in Astra from the user interface; the same data could be offered through a REST API, but it has not yet been implemented. Moodle has APIs that support the development of REST APIs in plugins, which should ease the development. Finally, A+ supports the version 1.0 of the LTI protocol as a tool consumer, which may be used to, for instance, link to external discussion forums, such as Piazza. The Astra plugin has no support for LTI, however, the official Moodle release includes a separate activity module that supports the LTI version 2.0 (as a tool consumer). Furthermore, there is another third-party plugin for Moodle that functions as an LTI tool provider [4]. Therefore, adding LTI support directly to the Astra plugin seems unnecessary since Moodle already supports it outside Astra.

The Astra plugin is separate from the A+ platform: the two platforms do not communicate nor co-operate together in any way. Astra is developed independently, even though it aims at replicating core functionality from A+ so that it is compatible with A+ and particularly the exercise services. The ultimate goal is that a course could select either one of the front-end systems and it would work correctly without any special adaptations. However, as the platforms evolve on their own, there is a risk that they become progressively different and separated from each other. At any rate, the two platforms do not directly benefit from each other concerning the development effort: if a feature is modified in A+ or a new feature is implemented, similar updates need to be developed in the Astra plugin. Moodle and A+ are implemented with thoroughly different web server technologies, hence program source code can not be directly copied from one system to the other. There are now two systems that offer similar features, however, both require considerable development effort: the overall required development effort has been doubled. If all efforts were targeted at a single platform, it would progress faster and have more features.

The A+ platform is built with Django, a modern web development framework in Python, while Moodle is built with PHP. As a programming language, PHP has problems that hinder development, whereas Python and Django provide a more advanced framework for web development. For example, Django has an object-relational mapper that enables the developer to access the database via Python classes, whereas the database API in Moodle is low-level and forces the developer to manually write database queries for any non-trivial query that does more than just retrieves records from a single

table. Python and Django generally offer a higher-level abstraction to web development that accelerates the development, whereas PHP and Moodle provide a lower-level abstraction that impedes the development.

Chapter 3 sketched three alternative approaches for the modernization of the Stratum platform. The selected approach was to replicate A+ functionality in Moodle, which separates the two platforms, A+ and Moodle, leading to increased development effort even though the platforms aim at providing similar or identical functionality. Chapter 3 presented Alternative 2 that pursued a lightweight Moodle plugin similar to the one for Stratum. The development effort of such a plugin is smaller than that of Astra. The lightweight Moodle plugin in Alternative 2 provides only a user interface and instead of replicating all A+ functionality in Moodle, it outsources the functionality to A+ by using the A+ REST API. Implementing a client for a REST API is simpler than natively implementing the same A+ functionality in Moodle. In Alternative 2, the Moodle plugin may also benefit from new updates in the A+ platform as the new functionality could be available in Moodle by just updating the REST API. Alternative 2 eases the Moodle plugin development and concentrates the development effort mostly on one platform, A+.

Chapter 6

Conclusions

E-learning encompasses any kind of learning or education that is delivered to the learner electronically by using computers or other devices. This thesis concentrates on the field of automated assessment within e-learning, i.e., the use of computers to automatically grade a learner's solution for a given exercise. A few such automated assessment tools have already been developed at Aalto University, for instance, for programming exercises and visual algorithm simulations. Stratum and A+ are platforms for running arbitrary automatically assessed exercises. They provide common infrastructure that is typically needed in all courses so that implementing new exercises requires only the development of grader programs, writing the exercise descriptions (instructions to students), and other materials to students if necessary, such as base code for programming exercises. Stratum also supports the generation of randomized or personalized exercises. The A+ platform is newer than Stratum and utilizes modern web technologies. Many courses at Aalto have started to use A+, while Stratum has fallen behind with regard to its technical implementation. However, Stratum supports embedding assignments in Moodle via a Moodle plugin developed for that purpose, which is important since Aalto University has started to use a Moodle-based learning management system, named MyCourses.

The main goal of this thesis project was to modernize Stratum so that the quality of the code and the feasibility for further development are improved, while still preserving the support for embedding exercises in Moodle. Three alternative approaches were presented for the modernization: the selected approach was to replicate A+ functionality in Moodle so that Moodle functions as a front end to *exercise services* external to Moodle. The exercise services implement the functionality of specific exercises and they are compatible with both Moodle and A+. The front-end system, Moodle or A+, is responsible for the storage of course data, such as submissions and their

results, and the user interface for both students and teachers. In addition, the existing exercise service framework, known as the MOOC grader, was extended in this thesis project to support the generation and assessment of personalized exercises. Therefore, the new platform developed in this thesis project that replaces the old Stratum consists of a new Moodle plugin, named Astra, and the upgraded MOOC grader framework.

The new platform is separate from the old Stratum as the old code base was not reused. However, the new platform implements similar functionality so that courses may be ported from Stratum to the new platform. Stratum assignment packages are ported to grader and generator programs in the MOOC grader, while Astra provides the user interface and teachers' administrative functionalities, such as manual assessment of submissions. The teacher defines the course configuration in the MOOC grader, which may be easily imported to Astra for the setup of the Moodle course. Furthermore, Astra replicates enough of the A+ functionality so that a typical course could be easily transferred from the A+ platform to Moodle without complications.

In summary, the new platform fulfills the requirements set for the modernization of Stratum: old courses may be ported from Stratum with feasible effort, the quality of the new implementation is higher than that of Stratum, the usability has been improved compared to Stratum, and Astra embeds exercises seamlessly in Moodle so that students can hardly notice that the exercises originate from an external service. Each requirement is realized at least adequately well and the new platform is stable for production use, i.e., real courses. In addition, the implementation of the exercises has been separated from the front-end system into exercise services. Should Aalto University decide to replace MyCourses with another learning management system, it is possible to embed the exercises in the new system by creating a plugin that connects to the exercise services using the same grader protocol.

6.1 Prospects for Future Work

There are some features in A+ that are not currently supported by Astra. Such features could be implemented in the future: they include at least group submissions and a REST API. Group submissions would allow students to form groups and submit solutions together if the teacher has enabled groups in the exercise. The implementation must take into account that

1. students may have different groups in different exercises,
2. the group should be formed before opening the exercise page (retrieving it from the exercise service) since the exercise may be personalized for

an individual or a group,

3. all group members must be able to view the submission and earn points from it, and

4. possible deviations should be considered: what if only one group member has an extended deadline, then what is the deadline for the group?

Students should probably be restricted to one group per exercise, in which case switching the group for the exercise is disallowed after the student has submitted once alone or in a group. Switching groups after submitting could be abused if a group decides to share its complete solution with others by forming new groups with other students.

Implementing a REST API in a Moodle plugin is straightforward. Astra already has a feature for exporting course data, thus the data needed in the REST API is available. The plugin must define a web service using Moodle APIs, which concern for example, the expected arguments, return values, and their types as well as access control restrictions of the web service [35]. Moodle manages such common procedures once the plugin has declared its web service functions. The implementation of the actual web service functionality can be compact (i.e., few lines of code), especially when the implementation is based on functions already implemented elsewhere in the plugin.

Section 5.1 discussed code quality issues in the use of the Bootstrap front-end framework. As long as the Moodle core utilizes the outdated version 2 of Bootstrap and using Bootstrap is required due to the MOOC grader, it is impossible to circumvent the issues completely. However, if the Moodle core is updated to utilize Bootstrap 3, the Astra plugin should no longer need to include its own version of Bootstrap and the extra CSS rules that were used to fix visual defects could be removed. Furthermore, there are likely possibilities for improving the user interface and the usability of the Astra plugin since the system has never undergone any usability testing and we have not yet had much experience with the system in real courses.

Section 5.1 also stated that the MOOC grader does not support graders that generate additional files, such as images, to complement the textual HTML feedback. Currently, there is no location accessible to the web in which the grader could store additional files during the assessment of a submission. The HTML feedback can only contain text; images can be added by linking to images that have been stored in the web beforehand, and by client-side JavaScript code that creates the image at runtime (e.g., from Base64-encoded [30] image data that the grader embedded in the HTML document),

an inelegant workaround. Images in the Scalable Vector Graphics (SVG) format are an exception since they are stored in a textual XML format and modern web browsers can render them, hence they can be naturally embedded in the HTML feedback. The MOOC grader could be upgraded to support additional files in the feedback. As the grader programs are usually executed in a sandbox, they can not directly create files in a permanently stored directory. A new grading action could be defined for the exercise configuration file that moves files from the sandbox directory to a web-accessible directory and, in addition, the grader must be able to form URLs to the new files so that they may be embedded in the HTML feedback.

The Astra plugin could be improved by adding support for statistics and learning analytics. Their purpose is to provide information about student performance to the teacher, for instance, the number of submissions students perform on average in an exercise and how close to the deadline they start working. The teacher could use such information in the development of the course and, particularly, the teacher could base decisions on real evidence instead of guessing or collecting feedback from the students afterwards.

The MOOC grader supports synchronization from a Git repository: when a developer modifies the course files and pushes the update into a Git server, the MOOC grader can update itself automatically by pulling the update from the Git server. However, if the course settings are modified, for example, the maximum points of an exercise are changed, the Astra plugin does not update the settings automatically. Instead, the teacher must visit the Moodle course page and run the automatic course setup in the Astra settings. This step could be potentially removed in the future so that when the MOOC grader updates itself from Git, it connects to Astra and instructs it to download the new course configuration and make changes accordingly.

Teachers are recommended to utilize the MOOC grader for building exercise services, in which case they create a course configuration file and eventually import the configuration to Moodle with the automatic setup. The Astra plugin also allows the modification of the configuration in the Astra user interface, however, the changes are not transferred to the MOOC grader. If the teacher modifies the configuration in Astra and then runs the automatic setup again, the modifications made in Astra are lost and replaced by the configuration in the MOOC grader. This problem would not occur if the configurations in Astra and the MOOC grader were automatically synchronized with each other. This matter is related to the Git synchronization in the MOOC grader discussed earlier since it modifies the configuration there. When the teacher modifies the settings in Astra, Astra could connect to the MOOC grader and upload the modifications, after which the MOOC grader would save them in the configuration file. However, this introduces a new

issue: should the MOOC grader automatically commit changes it receives from Astra to Git? If not, there would be Git merge conflicts when new updates are pushed to the course Git repository and the MOOC grader tries to pull them. This synchronization issue could be avoided by using another approach: when a teacher manually modifies the configuration in Astra, Astra does not connect to the MOOC grader and update the configuration there, but it saves the manual modifications separately. When the configuration is updated in Astra by the automatic setup, Astra can ask the user if the previously set manual modifications should be reapplied.

If exercises are personalized, the generated instances must be created before the start of the course in the MOOC grader. Currently, the generation requires access to the command line in the server, which could be problematic for teachers that are not familiar with the command line. Furthermore, the administration of the server is not necessarily the responsibility of the teacher, thus mainly the administrator should have access to the command line. If the generation of the exercise instances could be started from a web user interface, the command line access would no longer be required and the teachers could generate the instances easily at their own discretion. At this stage, the MOOC grader has a simplistic web user interface for some testing and administrative purposes, and its user authorization is based on a predefined list of accepted client IP addresses. If the functionality of the web user interface is extended, one should consider whether the access control model is too simplistic and restricting. Another approach for the creation of the exercise instances is that what if a teacher prefers to create the instances on her own computer and to upload them to the server (instead of running the generator in the server)? In that case, the MOOC grader should be able to copy the instances from the course directory into the correct paths when the course files are updated by Git.

Bibliography

[1] Asynchronous Module Definition API. https://github.com/amdjs/amdjs-api. [Online; accessed 22-September-2016].

[2] Bootstrap — the world's most popular mobile-first and responsive front-end framework. http://getbootstrap.com/. [Online; accessed 22-September-2016].

[3] MOOC grader source code. https://github.com/Aalto-LeTech/mooc-grader. [Online; accessed 10-August-2016].

[4] Moodle plugins directory: LTI Provider. https://moodle.org/plugins/local_ltiprovider. [Online; accessed 25-September-2016].

[5] A Mustache implementation in PHP. https://github.com/bobthecow/mustache.php. [Online; accessed 22-September-2016].

[6] Advanced Distributed Learning Initiative. Experience API. https://www.adlnet.gov/adl-research/performance-tracking-analysis/experience-api/. [Online; accessed 14-August-2016].

[7] Advanced Distributed Learning Initiative. SCORM. https://www.adlnet.gov/adl-research/scorm/. [Online; accessed 14-August-2016].

[8] Ala-Mutka, K., and Järvinen, H.-M. Assessment process for programming assignments. In *IEEE 14th International Conference on Advanced Learning Technologies* (2004), IEEE Computer Society, pp. 181–185.

[9] Anders, G. Coursera flirts with diplomas: Online 'specialization' is $250. http://www.forbes.com/sites/georgeanders/2014/01/21/coursera-flirts-with-diplomas-online-specialization-is-250/#1ad4308e6d85, January 2014. [Online; accessed 9-August-2016].

[10] ATutor. ATutor learning management tools. `http://www.atutor.ca/`. [Online; accessed 3-August-2016].

[11] Auvinen, T. Rubyric. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (2011), ACM, pp. 102–106.

[12] Bali, M. MOOC pedagogy: gleaning good practice from existing MOOCs. *Journal of Online Learning and Teaching 10*, 1 (2014), 44–55.

[13] Blackboard, Inc. Blackboard Learn. `http://www.blackboard.com/learning-management-system/blackboard-learn.aspx`. [Online; accessed 3-August-2016].

[14] D2L Corporation. Brightspace learning environment. `https://www.d2l.com/products/learning-environment/`. [Online; accessed 3-August-2016].

[15] Dahlstrom, E., Brooks, D. C., and Bichsel, J. The current ecosystem of learning management systems in higher education: Student, faculty, and IT perspectives. Research report. Louisville, CO: ECAR, September 2014. Available from `http://www.educause.edu/ecar`.

[16] Douce, C., Livingstone, D., and Orwell, J. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC) 5*, 3 (2005).

[17] Elgort, I. E-learning adoption: Bridging the chasm. In *Proceedings of ASCILITE* (2005), pp. 181–185.

[18] Hakulinen, L., Auvinen, T., and Korhonen, A. Empirical study on the effect of achievement badges in TRAKLA2 online learning environment. In *Learning and Teaching in Computing and Engineering (LaTiCE), 2013* (2013), IEEE, pp. 47–54.

[19] Harjula, M. Mathematics exercise system with automatic assessment. Master's thesis, Helsinki University of Technology, 2008.

[20] Helminen, J. Jype – an education-oriented integrated program visualization, visual debugging, and programming exercise tool for Python. Master's thesis, Helsinki University of Technology, 2009.

[21] Hiisilä, A. Kurssinhallintajärjestelmä ohjelmoinnin perusopetuksen avuksi (Course Management System for basic courses in programming). Master's thesis, Helsinki University of Technology, 2005.

[22] ICEF Monitor. Mooc enrolment surpassed 35 million in 2015. `http://monitor.icef.com/2016/01/mooc-enrolment-surpassed-35-million-in-2015/`, January 2016. [Online; accessed 9-August-2016].

[23] IMS Global Learning Consortium. Common Cartridge background. `https://www.imsglobal.org/activity/common-cartridge`. [Online; accessed 14-August-2016].

[24] IMS Global Learning Consortium. Learning Tools Interoperability background. `https://www.imsglobal.org/activity/learning-tools-interoperability`. [Online; accessed 14-August-2016].

[25] IMS Global Learning Consortium. Question and Test Interoperability and Accessible Portable Item Protocol background. `https://www.imsglobal.org/activity/qtiapip`. [Online; accessed 14-August-2016].

[26] Instructure, Inc. Canvas learning management system. `https://www.canvaslms.com/`. [Online; accessed 3-August-2016].

[27] Itslearning AS. Itslearning learning platform. `http://www.itslearning.eu/`. [Online; accessed 3-August-2016].

[28] Janhunen, T., Jussila, T., Järvisalo, M., and Oikarinen, E. Teaching Smullyan's analytic tableaux in a scalable learning environment. In *Kolin Kolistelut/Koli Calling. Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education* (2004), pp. 85–94.

[29] Jordan, K. Initial trends in enrolment and completion of massive open online courses. *The International Review of Research in Open and Distance Learning 15*, 1 (2014).

[30] Josefsson, S. The Base16, Base32, and Base64 data encodings. RFC 4648, RFC Editor, October 2006.

[31] Kaplan, A. M., and Haenlein, M. Higher education and the digital revolution: About MOOCs, SPOCs, social media, and the Cookie Monster. *Business Horizons 59*, 4 (2016), 441–450.

[32] Karavirta, V., Ihantola, P., and Koskinen, T. Service-oriented approach to improve interoperability of e-learning systems. In *13th IEEE International Conference on Advanced Learning Technologies (ICALT)* (2013), IEEE, pp. 341–345.

[33] Karavirta, V., and Shaffer, C. A. JSAV: The JavaScript Algorithm Visualization Library. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (2013), ACM, pp. 159–164.

[34] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., and Silvasti, P. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education 3*, 2 (2004), 267–288.

[35] Moodle community. Web services API. `https://docs.moodle.org/dev/Web_services_API`. [Online; accessed 27-October-2016].

[36] Moodle community. Moodle plugin types. `https://docs.moodle.org/dev/Plugin_types`, 2016. [Online; accessed 2-August-2016].

[37] Moodle Pty Ltd. Moodle — open-source learning platform. `https://moodle.org/`, 2016. [Online; accessed 4-August-2016].

[38] Nykopp, J. STRATUM – Yleiskäyttöinen automaattinen koneisharjoitusjärjestelmä. Master's thesis, Helsinki University of Technology, 2006.

[39] Odekirk-Hash, E., and Zachary, J. L. Automated feedback on programs means students need less help from teachers. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (2001), ACM, pp. 55–59.

[40] Riekkinen, M. New features of Stratum. Dept. of Computer Science, Aalto University, December 2015. Technical memorandum.

[41] Riekkinen, M. Stratum MyCourses-Moodle plugin guide for teachers. Dept. of Computer Science, Aalto University, December 2015. Technical memorandum.

[42] Sakai project. Sakai. `https://sakaiproject.org/`. [Online; accessed 2-August-2016].

[43] Sangwin, C. J., and Grove, M. STACK: addressing the needs of the "neglected learners". In *Proceedings of the Web Advanced Learning Technologies Conference and Exhibition, WebALT* (2006), pp. 81–96.

[44] Shah, D. By the numbers: MOOCS in 2015. Class Central, https://www.class-central.com/report/moocs-2015-stats/, December 2015. [Online; accessed 10-August-2016].

[45] Staker, H., and Horn, M. B. Classifying K-12 blended learning. Tech. rep., Innosight Institute, May 2012. Available online: http://www.innosightinstitute.org/innosight/wp-content/uploads/2012/05/Classifying-K-12-blended-learning2.pdf.

[46] Stockley, D. E-learning definition and explanation (elearning, online training, online learning). http://www.derekstockley.com.au/elearning-definition.html, 2003. [Online; accessed 6-July-2016].

[47] Tibbetts, J. LTI2 introduction. https://www.imsglobal.org/lti-v2-introduction. [Online; accessed 14-August-2016].

[48] Tukiainen, N. Raportti. Dept. of Information and Computer Science, Helsinki University of Technology (now Aalto University), August 2009. Technical memorandum about the new web interface in Stratum.

[49] Ullman, J. D., and Widom, J. *A First Course in Database Systems: Pearson New International Edition*, 3 ed. Pearson Education Limited, 2014.