Universidad de Talca
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Diseño de Software
Redesigning EDUCANDUS web application

Erik Regla
eregla09@alumnos.utalca.cl

1 de diciembre de 2016

1. Overview of the document

Universidad de Talca uses a Moodle platform to assist via e-learning their students and professors, but the current implementation is heavily patched to the core and lacks most of the desired features for their end users. In this report, we propose a new design to cover those problems detected by the author not considering details not relevant to the modifications design (such as login).

2. Current issues detected by the community and our proposals to tackle them

2.1. Restricted communication between course participants

Interaction between students and teachers is an important aspect of learning, but there is no way for students to communicate each other by using this platform. Currently EDUCANDUS provides a forum system and email system which is synced with the alumni email and with course activities marked for notification.

Moreover, the forum system is readonly for students, hence they cannot communicate between themselves and the notification system tends to fail often. Also, there are times when the students need to work on their homework in groups (sometimes as large as the course itself), and the only way to communicate is to rely on SNSs like Facebook to do so. As the reader can expect, there is no way to provide feedback in any direction for any of the available resources (namely assignments, course materials, etcetera) as now private messaging is disabled.

2.1.1. Our proposal

To solve this problem we need to design the following components:

1. **Direct messaging between course participants**

2. **General forum for the course**
3. **Resource forum**
4. **Group formation for assignments and private forum**
5. **File upload capabilities for forums**

2.2. Lack of versioning of courses and resources

There are two major points on both students and teachers argue against the current platform. Students always see former course contents, which leads to confusion when uploading homeworks and for students which are repeating the course, they get confused with their own submissions. On the other hand, teachers want to check former submissions and former course contents (by example, new teachers or when checking for plagiarism) and the platform erases it's content at the start of the semester. So they have to reupload all the content again and to deal with repeated submissions. When students upload their homeworks, there is no version control for the files that they are uploading to the platform, so, if the student makes an error by overwriting the correct file, there is no way to check if he is saying the truth.

2.2.1. Our proposal

To solve this problem we need to design the following components:

1. **Versioning at course level, including both contents and assignments**
2. **Versioning for the files uploaded to forums**

2.3. Grades not synced with the official grading platform (UTALMATICO)

Class assistants and teachers often get confused about the grading system. The professor can communicate the grades that their students obtain in their assignments by uploading them on EDUCANDUS or UTALMATICO, but in the end only the latter counts to the grade registry used in the university. This means that grading homeworks on EDUCANDUS has no value and there can occur inconsistencies between the two platforms.

2.3.1. Our proposal

To solve this problem we need to design the following components:

1. **API consumer for EDUCANDUS platform, intended to be used with any grading system**
2. **Versioning for the files uploaded to forums**

2.4. Complex and useless assignment creation and management

When creating assignments, there are tons of custom fields that aren't configured by default, like the grading scale used, deadline configuration and so on so forth.

2.4.1. Our proposal

To solve this problem we need to design the following components:

1. **Standardised fields on assignments**

We can summarise some of the most important use cases to be considered during the development stage in the following diagram.

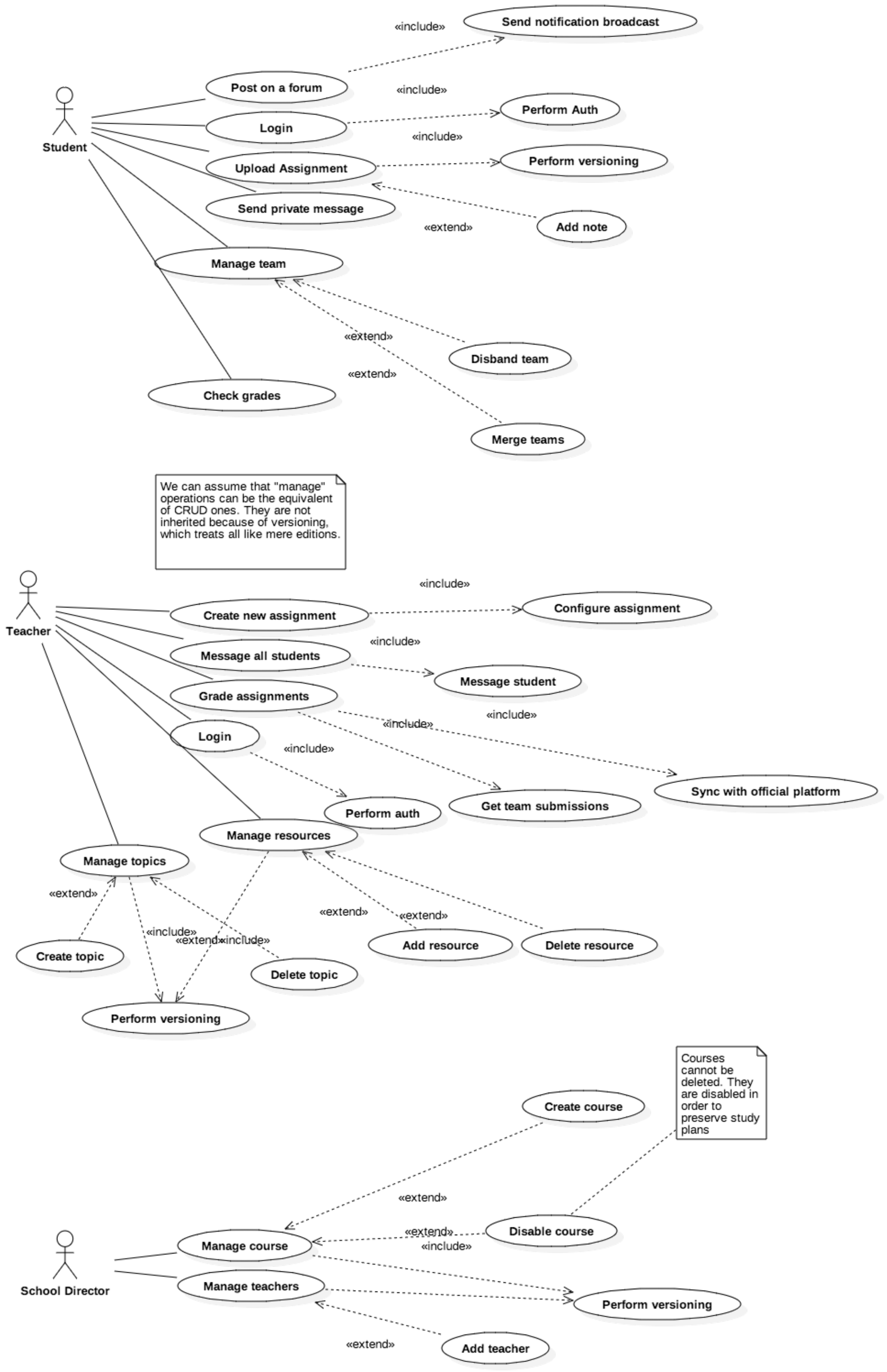

Figura 1: Use case diagram for teachers, students and school directors

3. Domain Analysis

Here we analyse the two most important domains in our design, the course and the forum. As seen, an improvement is the Ad-Hoc creation of teams. By default it is intended to create a team per person, then they can merge themselves to form a composed team. This enables us to implement team grading without changing our logic implementation.

Forums are pretty simple, just a collection of posts. We are aware that forums and realtime chat is already implemented on LMS but here we propose to enable them by default for each resource. This way we expect to motivate students to participate on rich discussions, open to all participants.

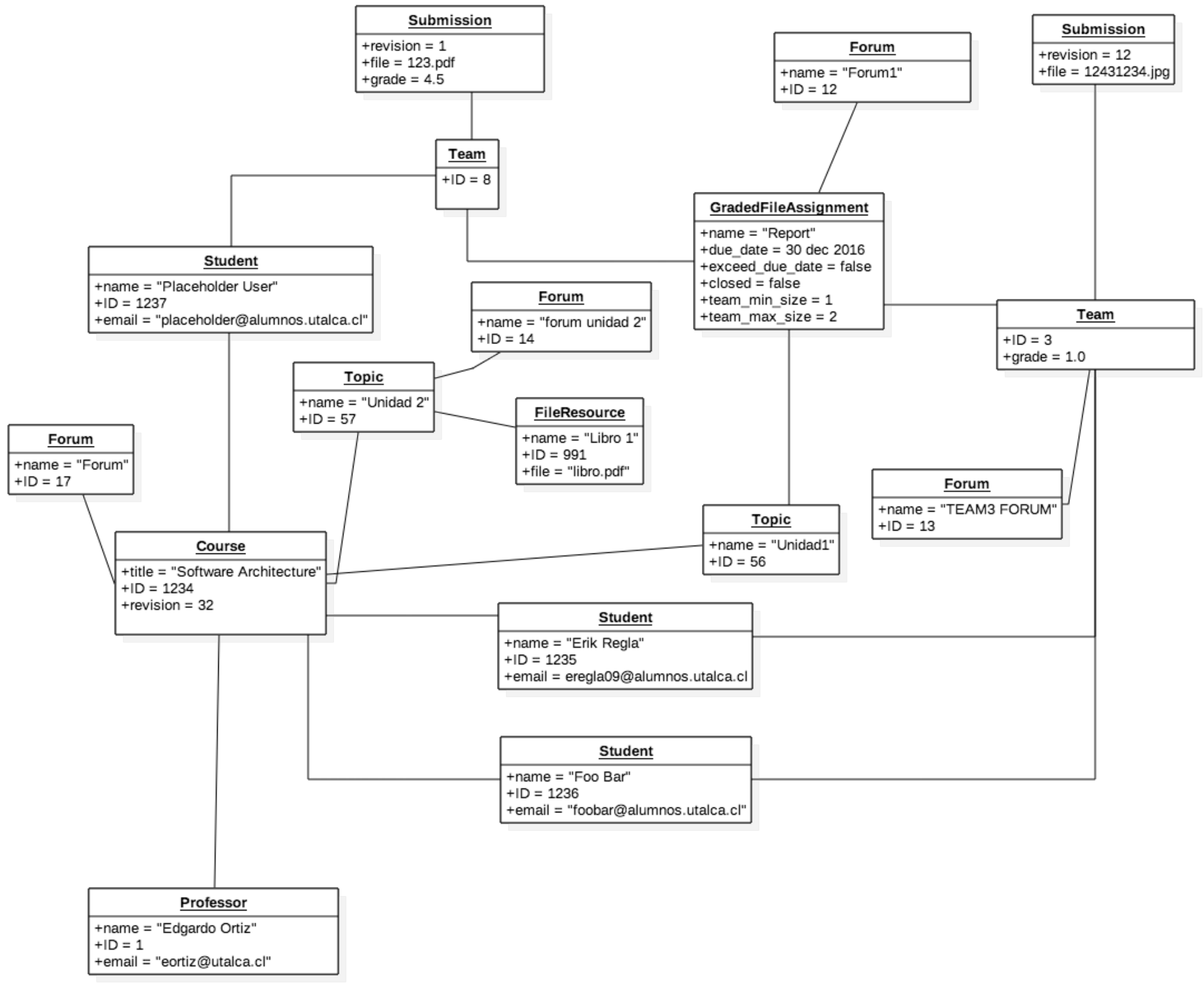

Figura 2: Object diagram for course, students, resources, assignments and teams

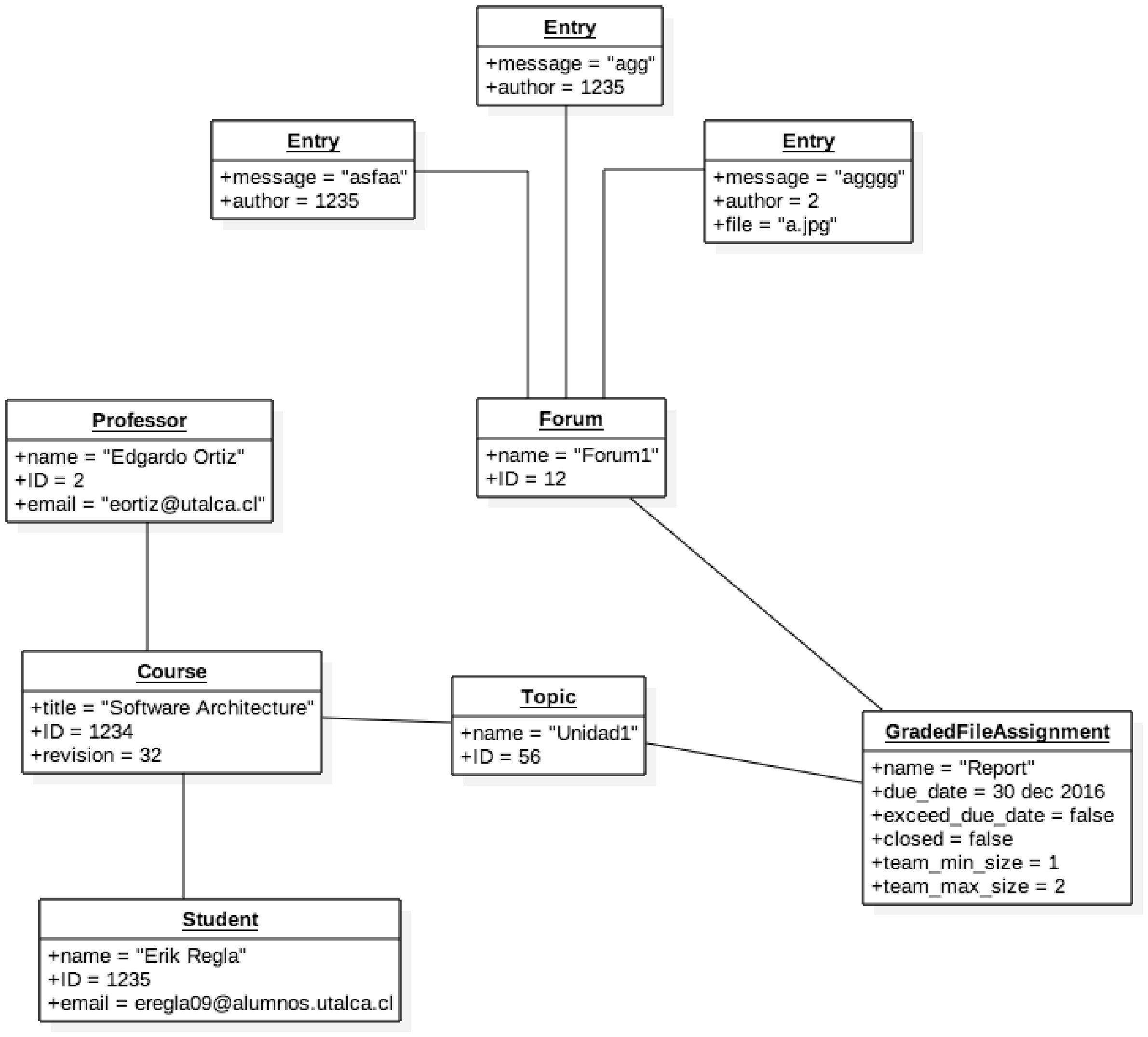

Figura 3: Object diagram for forums and participants

Other diagrams like login, plugins, etcetera are not included as they do not improve the understanding of the problem.

3.1. Users

Users are restricted to three types: School directors, Teachers and Students. At module level, they do not have any extra information, because the login module is in charge to administer all the permisions involved. This is necessary because some students are actually class assistants and they need permissions to edit course materials. When this happens, a petition must be elevated using the management system which is covered in detail later.

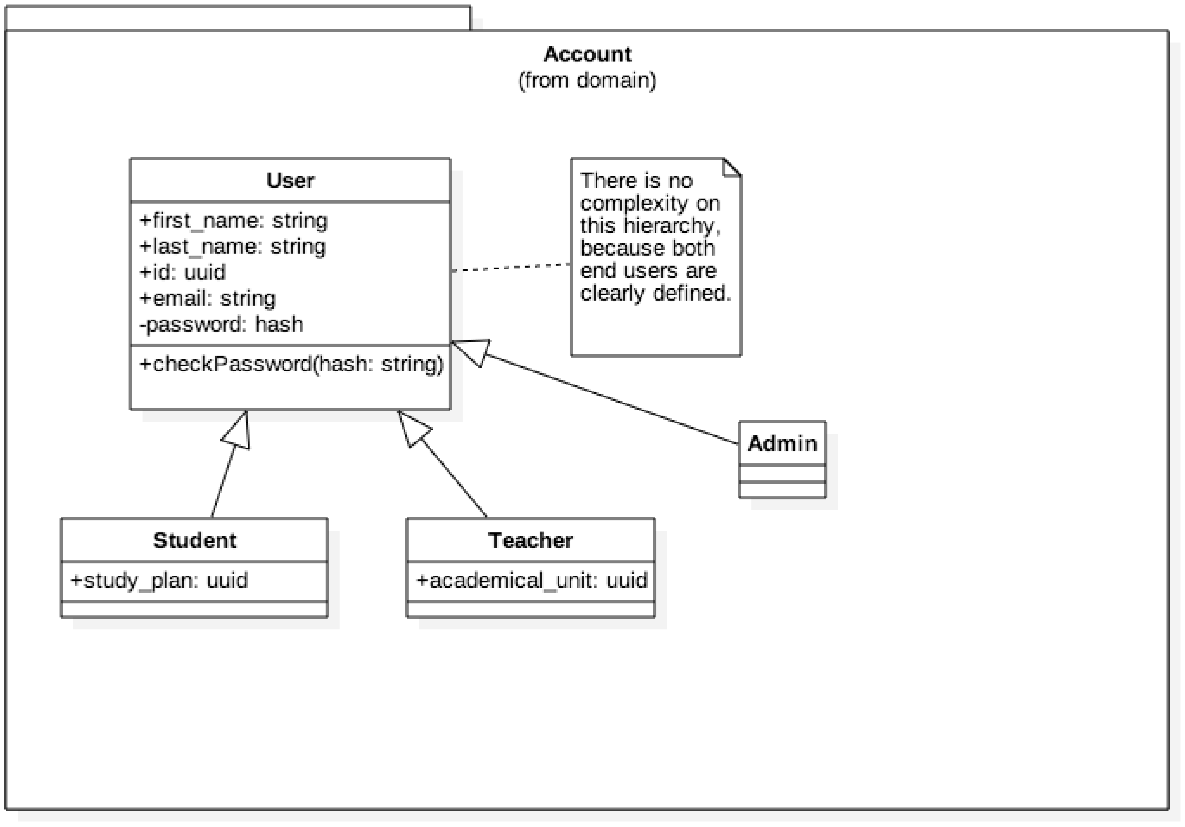

Figura 4: Users package

3.2. Forums

Forums can be seen in two ways, at a resource/conversation level or as a course. We already stated that every resource should have a forum to help public communication and opinion sharing in realtime. By realtime we do not mean to implement a chat system, since it can encourage students to not think before they write and to discuss complex topics that will require images and media can be troublesome.

Forums should reveal a composite design, because we can think a course as a tree of resources more than a list of them, therefore, if one desires to access a course forum, all topics and threads should be seen in the same hierarchy as the course has. Then, aside from the typical implementation of the forum structure, we need an adapter to export the forums to the versioning system and back on. This can be achieved by implementing any serialisation interface as long as the interface remains the same.

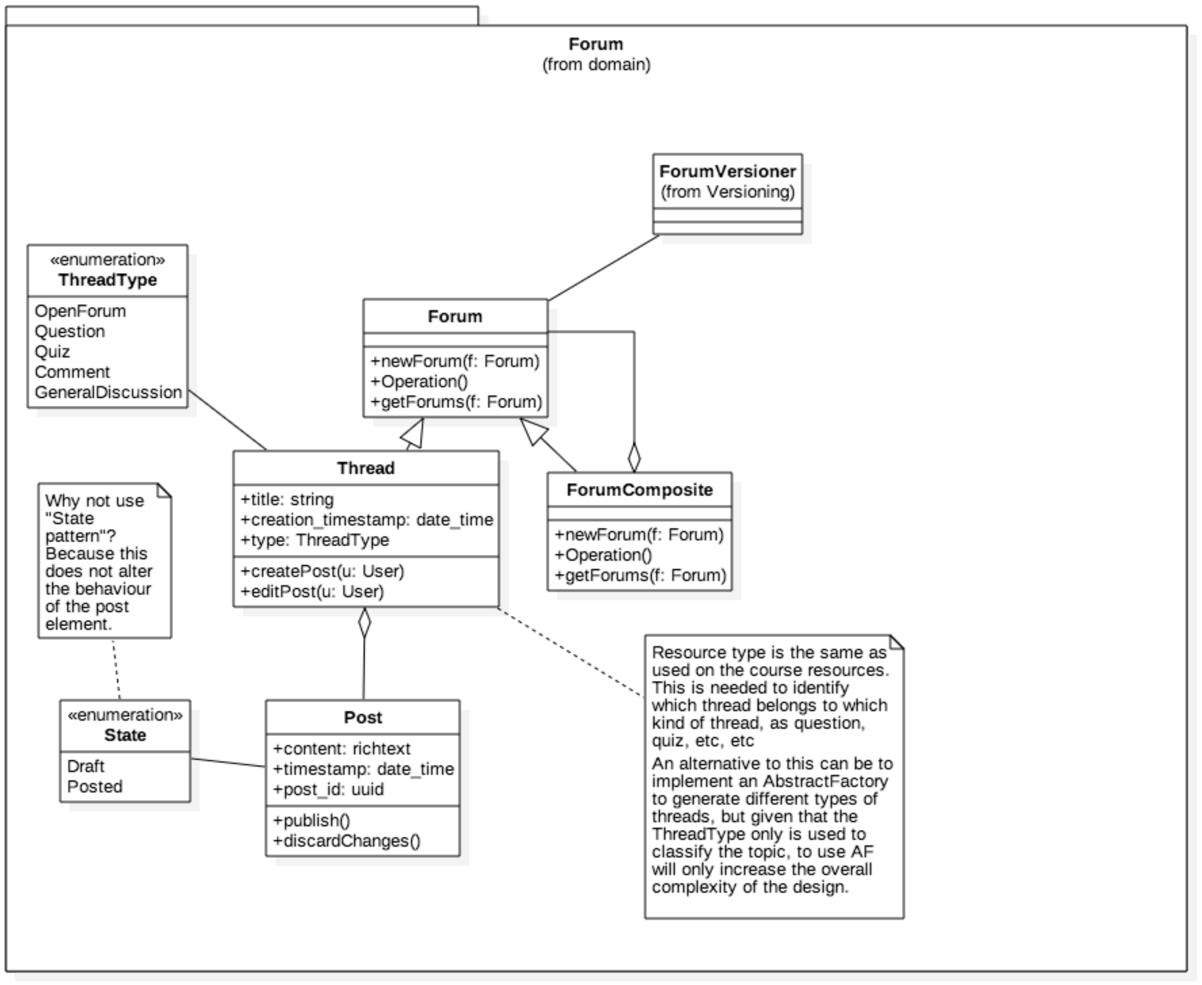
Forum
(from domain)
ForumVersioner
(from Versioning)
«enumeration»
ThreadType
OpenForum
Question
Quiz
Comment
GeneralDiscussion
Forum
+newForum(f: Forum)
+Operation()
+getForums(f: Forum)
Thread
+title: string
+creation_timestamp: date_time
+type: ThreadType
+createPost(u: User)
+editPost(u: User)
ForumComposite
+newForum(f: Forum)
+Operation()
+getForums(f: Forum)
Why not use "State pattern"? Because this does not alter the behaviour of the post element.
«enumeration»
State
Draft
Posted
Post
+content: richtext
+timestamp: date_time
+post_id: uuid
+publish()
+discardChanges()
Resource type is the same as used on the course resources. This is needed to identify which thread belongs to which kind of thread, as question, quiz, etc, etc
An alternative to this can be to implement an AbstractFactory to generate different types of threads, but given that the ThreadType only is used to classify the topic, to use AF will only increase the overall complexity of the design.

Figura 5: Forum package

3.3. Versioning

Versioning is supposed to operate in a different level than forums and users, much alike an adapter or a bridge to another versioning system (which could be SVN, GIT, etc), wihout exposing how versioning should be performed but giving a standard set of operations to version each element of the platform.

The mechamism is much like the same for every element present, the resource is serialised, then passed as a memento to the versioning system. The platform sees each version as a memento, whilst the versioning system is ready to pass this serialised (or maybe not) data onto the underlying versioning platform.

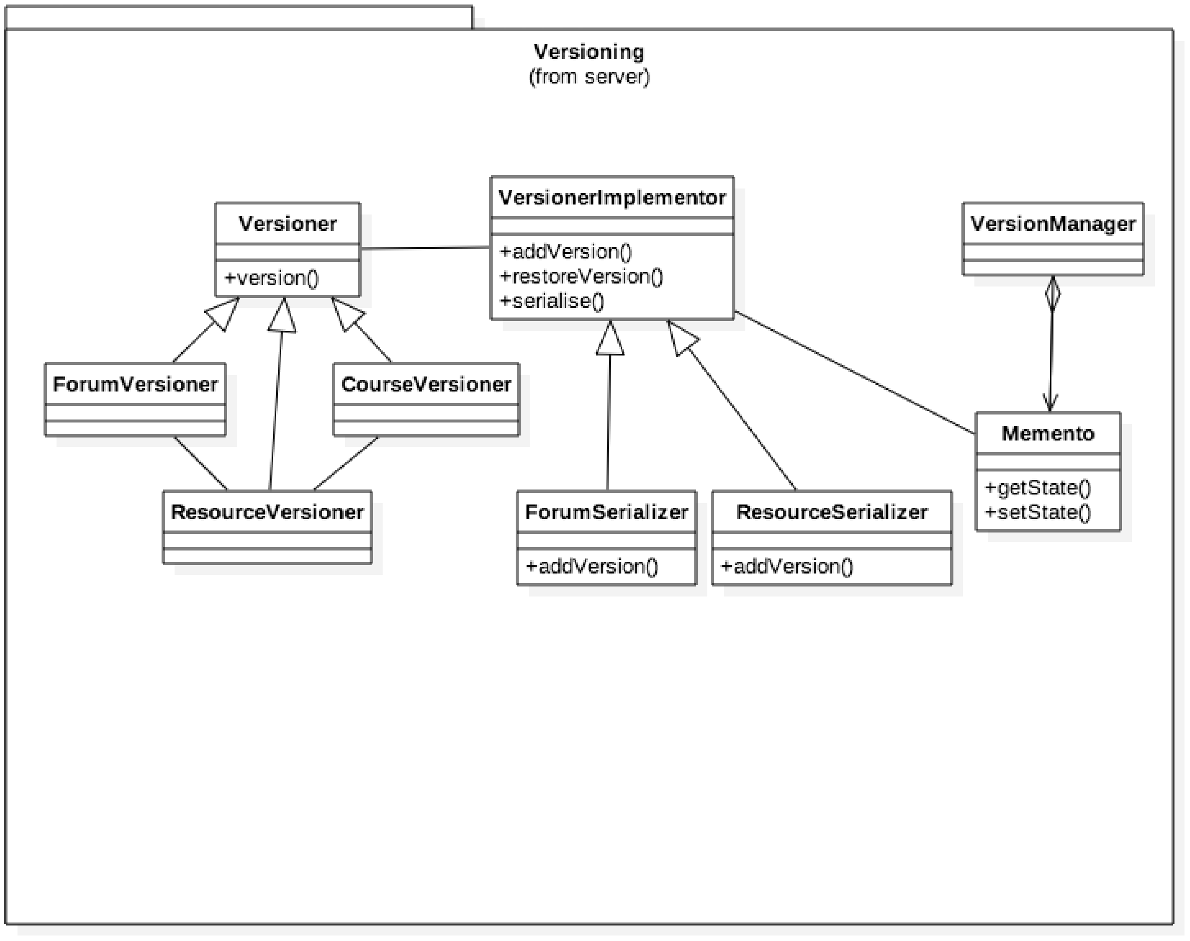

Figura 6: Versioning package

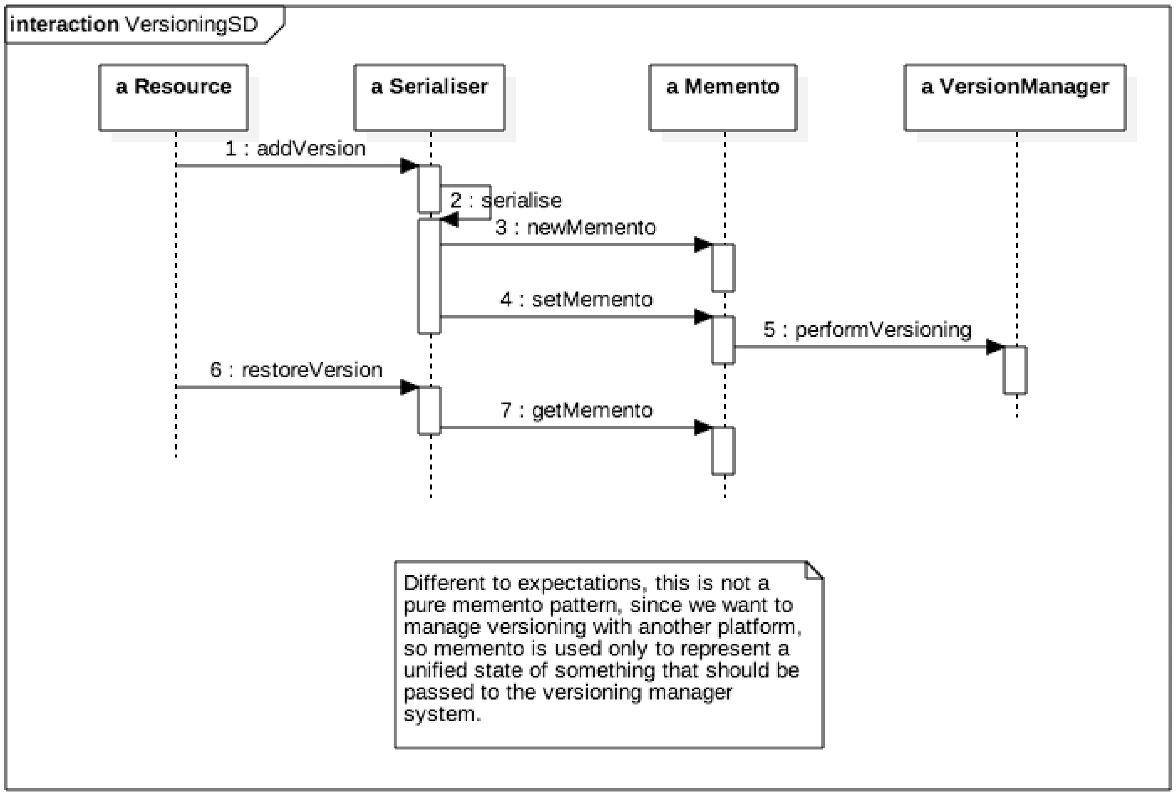
interaction VersioningSD
a Resource
a Serialiser
a Memento
a VersionManager
1 : addVersion
2 : serialise
3 : newMemento
4 : setMemento
5 : performVersioning
6 : restoreVersion
7 : getMemento
Different to expectations, this is not a pure memento pattern, since we want to manage versioning with another platform, so memento is used only to represent a unified state of something that should be passed to the versioning manager system.

Figura 7: Versioning interactions

4. Domain Design and High Level Domain Analysis

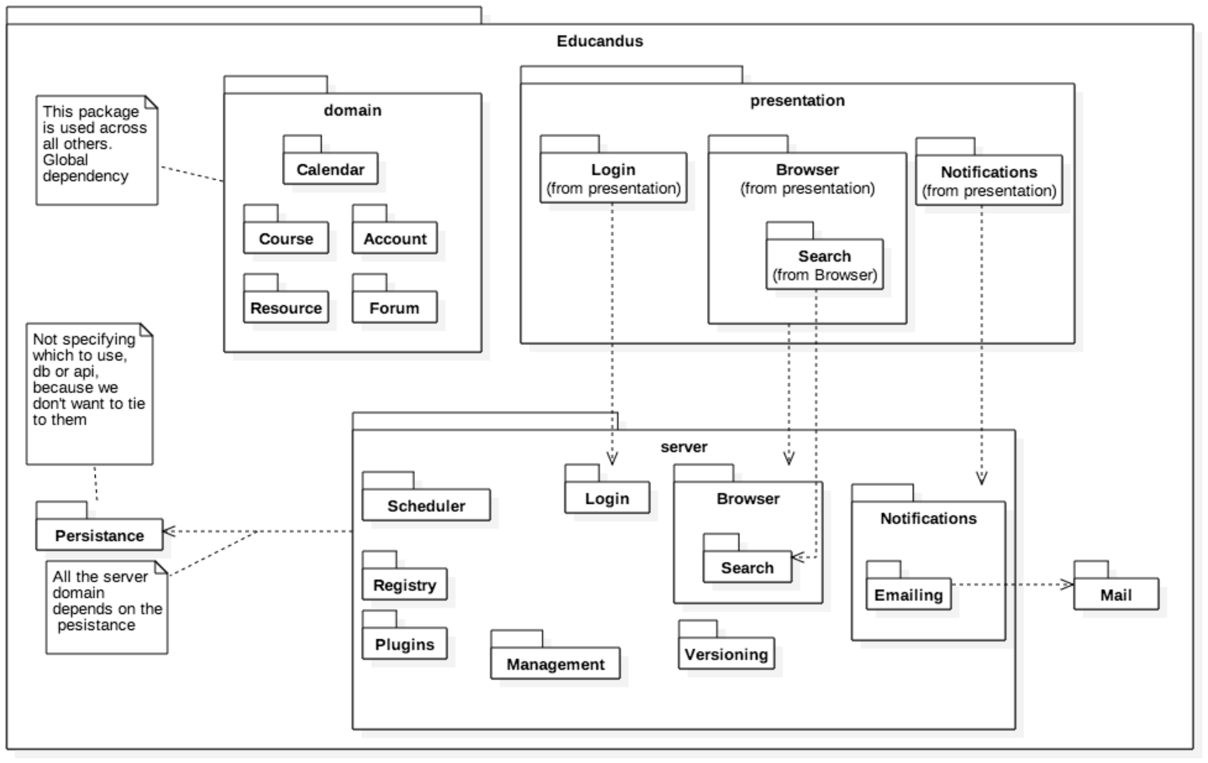
Educandus
This package is used across all others. Global dependency
domain
Calendar
Course
Account
Resource
Forum
presentation
Login (from presentation)
Browser (from presentation)
Search (from Browser)
Notifications (from presentation)
Not specifying which to use, db or api, because we don't want to tie to them
server
Scheduler
Login
Browser
Search
Notifications
Emailing
Mail
Persistance
All the server domain depends on the pesistance
Registry
Plugins
Management
Versioning

Figura 8: Package architecture of the application

We can think of this application as a simple producer-consumer model for the end users, moreover, there are some processes that need to pay close attention. Whilst the view only consumes and triggers commands on the server, the server itself needs to perform several operations by its own. Therefore, we choose to implement a multi-layered architecture on where the main modules are:

1. Presentation layer
2. Serverside layer
3. Mailing
4. The official grading platform API

5. Package and Implementation Architecture