MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Thesis Management System for Industrial Partner Red Hat

BACHELOR'S THESIS

Václav Dedík

Brno, Spring 2013

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Václav Dedík

Advisor: Mgr. Jiří Kolář

Acknowledgement

I would like to express my thanks especially to Ing. Jiří Pechanec and Mgr. Jiří Eischman for their extensive help and consultation, and also to the people from Red Hat who were involved in the initial testing of the Thesis Management System for their expressive feedback that helped to get the project more stable and usable.

Abstract

The aim of the thesis was to design and implement a thesis management system for industrial partner Red Hat. This thesis contains the introduction to the thesis management and the description of the used software development methodology. The second part of the thesis focuses on the used technologies, and mainly the architecture and design of the system.

Keywords

Thesis, Thesis Management, Information System, Grails

Contents

1 **Introduction** . . . 3
2 **Thesis Management** . . . 4
2.1 *Introduction* . . . 4
2.2 *Terminology* . . . 4
2.2.1 Thesis Management System . . . 5
3 **Software Development Methodology** . . . 6
3.1 *Waterfall* . . . 6
3.2 *Incremental model* . . . 7
3.3 *Iterative model* . . . 7
3.4 *Methodology of the Thesis Management System* . . . 7
4 **Technologies Used** . . . 8
4.1 *Groovy* . . . 8
4.2 *Grails* . . . 9
4.3 *Git and GitHub* . . . 10
4.4 *PostgreSQL* . . . 10
4.5 *MongoDB* . . . 12
4.6 *OpenShift* . . . 13
4.7 *Others* . . . 15
5 **Project Architecture and Design** . . . 16
5.1 *Functional Requirements* . . . 16
5.1.1 Use Case Diagram . . . 18
5.2 *Non-Functional Requirements* . . . 19
5.3 *Domain Model* . . . 19
5.3.1 Domain Model of the Thesis Management System . . . 20
5.3.1.1 **`User` entity** . . . 20
5.3.1.2 **`University` entity** . . . 21
5.3.1.3 **`Article`, `Comment` and `Subscription` entities** . . . 21
5.3.1.4 **`Feed` and `Notification` entities** . . . 23
5.3.1.5 **`Tag` entity** . . . 24
5.3.1.6 **`Topic`, `Category` and `Supervision` entities** . . . 24
5.3.1.7 **`Thesis` entity** . . . 26
5.3.1.8 **`Application` entity** . . . 27
5.3.1.9 **`Faq` entity** . . . 28
5.4 *Implementation* . . . 29
5.5 *Evaluation* . . . 29

5.5.1 Functional Requirements 29
5.5.2 Non-Functional Requirements 32
6 **Future Improvements** 34
6.1 *Integration with Universities* 34
6.2 *BPM and BRMS* 34
6.3 *School Projects Management* 35
6.4 *Import and Export of Theses and Topics* 35
7 **Conclusion** 37
A **Diagrams** 40
A.1 *Use Case* 40
A.2 *Domain Model* 41
B **Links** 42
B.1 *Source Code of the Project* 42
B.2 *OpenShift URL of the Project* 42

1 Introduction

In our contemporary society, there is a whole variety of information we need to take care of. Starting with information about people, cars, or real estate and ending with information about furniture or logistics, it is hard to imagine the society developing any further without a means of a fast and easy management of such data. Fortunately, in the age of computer science, it is possible to register and aggregate nearly unlimited amount of information thanks to so-called information systems. The goal of the core project of this thesis is to implement such an information system for the management of theses.

The need for this system was initiated by the Czech subsidiary of the first one-billion dollar open source company [1], Red Hat. The company currently uses an internal wiki – a website that allows its users to add, modify, or delete its content. The disadvantages of such a solution include missing support for integrity constraints (anyone can put any information in the system) and authorization. The current solution also requires its users to do substantial amount of management manually in a rather complicated manner. A specialized information system will not only solve these disadvantages, but also open a window for other features, which are exclusive to this particular domain.

In this thesis, the author will describe the way information is handled in terms of theses, and establish the terminology for the core project. He will also cover the most common methods of software development of information systems including the one used for development of the core project. In the second part of the thesis, the author will focus on the architecture and design of the implemented system. The architecture and design chapter will include the list of the functional and non-functional requirements of the system, and mainly the description of the domain model that was designed. This thesis will also cover most of the technologies that were used for the implementation of the project and the last chapter will discuss some possible improvements that could extend the functionality of the system in the future.

2 Thesis Management

2.1 Introduction

This chapter aims to give a brief description of the thesis management and the related issues. However, the content of this chapter is not to be regarded as a general description of the thesis management, but rather an introduction to the background of the core project of this thesis.

When students fulfill all mandatory subjects and other requirements established by their university, they have to write a *thesis*. To write a thesis, they first have to choose the *topic* of their thesis. There are many ways to choose a topic for a thesis, students can, for example, think of one themselves and contact a teacher at their university. Another way is to choose the topic from a list of topics composed by their university or by some external party (e.g. a company that cooperates with the university). A topic usually consists of a *title*, *description* and a *supervisor* who helps the student with the topic (e.g. clarifies inaccuracies).

Thesis, on the other hand, is the piece of work that is based on a topic and consists of an official title and description, supervisor, abstract etc. It is important to note here, however, that the supervisor must be an academic because they need to understand the policies of the university in question. This complicates the thesis management because if an external party is to offer a list of topics for students, they need one of their supervisors, who can help the student with the topic, and a university supervisor, who can help the student to follow the university policies.

The described problem presents us with the fact that we need to differentiate between topics and theses, but most importantly, it means that we need to manage them separately.

2.2 Terminology

There is a certain lack of terminology in the field of the thesis management. There is, for example, no one-word term for the supervisor who supervises a topic from the point of view of an external party, or the supervisor who supervises the topic at a university. For that reason, the authors of the core project of this thesis established their own terminology, which is described in this chapter.

As there are two kinds of supervisors, the author calls the external supervisor (i.e. the supervisor from an external party, e.g. a company or an

organization) the *leader*. The university supervisor is simply called the *supervisor*. A topic of a thesis is called just the *topic*. We also need to differentiate between the various types of theses. There is, for example, a diploma thesis (which is, in the context of this thesis, the same as master's thesis) and a bachelor's thesis. This is simply called the *type*. The student who is assigned to a thesis, i.e. who works on it, is called the *assignee*. All other terms related to thesis management, like title, description, grade etc, remain the same.

2.2.1 Thesis Management System

The core project of this thesis is to design and implement a web application for thesis management. In the following chapters, the author will refer to it as the *Thesis Management System* as that is the name of the application.

3 Software Development Methodology

A software development methodology is a framework that is used to plan, structure, and control the process of developing an information system. There are many methods to develop an information system and each of them has its advantages and disadvantages, in this chapter, the author will briefly describe such methods including the method used in the development of the Thesis Management System.

3.1 Waterfall

Waterfall is one of the most popular software development methodologies because it is the most simple one. The methodology simply features several processes (or disciplines) in one flow and the project is finished when the last process is successfully completed [2]. The processes with their flow are depicted in Figure 3.1.

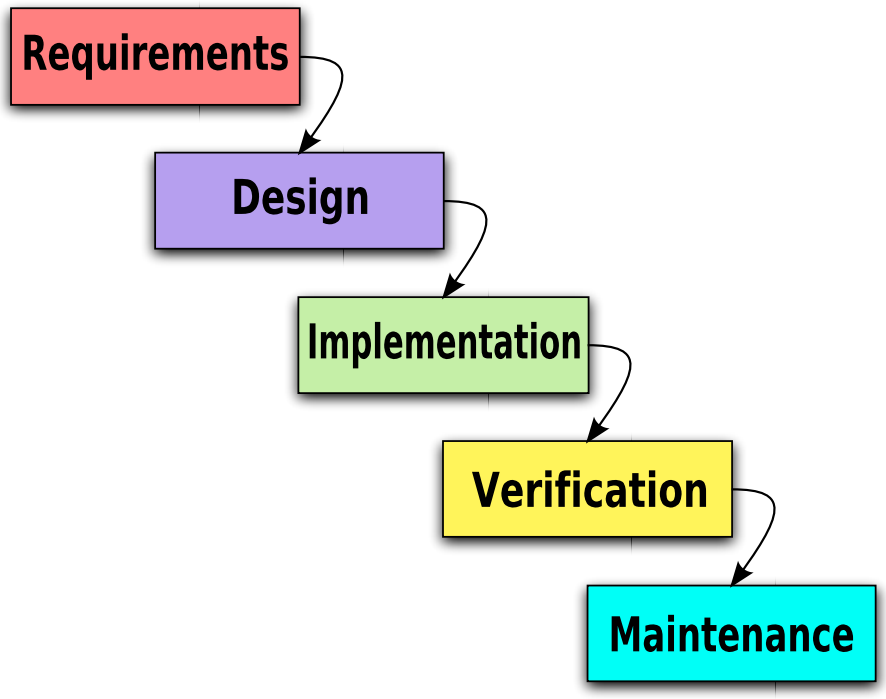


Figure 3.1: Waterfall processes with the flow [3]

3.2 Incremental model

The incremental model is a software development methodology where the project is designed, implemented, tested and maintained incrementally, i.e. a bit of project is added each time. The project is considered done when all requirements are satisfied [2].

The advantage of this methodology is that the project can be developed even when all requirements are not yet known. The disadvantage is that sometime one increment of the project can be hard to integrate with another one.

3.3 Iterative model

In iterative software development methodology, a project is developed in cyclic processes of prototyping, testing, analyzing, and refining. Developing a software application iteratively is similar to building a house, i.e. the architect designs the size of the house, number of rooms and the orientation and the builders then dig out the foundations, when the customer is satisfied, the architect and the builders move to the next phase. Step by step, the building is completed as soon as the last iteration is finished. Waterfall model is the extreme example of the iterative model, i.e. it is an iterative model with only one iteration [2].

The advantage of the iterative development is that the customer gets closely involved in the development process of the project, which results in the requirement changes being caught early. The disadvantage is that the people working on the project can get stuck in a loop, where every iteration introduces new bugs, and run out of time and budget.

3.4 Methodology of the Thesis Management System

The customer for whom the Thesis Management System was developed wished to be involved in the process of development by meeting with the developers every month to assess the implemented part of the system and propose changes if need be. As the system was developed by three students, it was also sliced into several separate functionalities each developed by one student. Furthermore, there were proposed several high risk features that would only be implemented if there was enough time (see chapter 6). The system was therefore developed both incrementally and iteratively.

4 Technologies Used

In this chapter the author will describe the technologies and tools that were used to implement the Thesis Management System. The only requirement of the thesis description was to use platform *Grails* and the requirements of the customer were to use platform *OpenShift* for deployment, *GitHub* for source code hosting and a *GPL* compatible license for licensing.

4.1 Groovy

Groovy is a dynamic object-oriented programming language for the Java platform and therefore "makes modern programming features available to Java developers with almost-zero learning curve" [4]. Groovy compiles to Java bytecode, so it can be seamlessly integrated with all Java classes and libraries [4]. It has a lot of features that make writing code less verbose and more expressive, like:

- Closures – anonymous functions that can be written on one line without making the readability suffer
- Reasonable defaults like public methods, private attributes with default accessors, final classes
- Support for Domain Specific Languages, which makes the code one of the most compact among other programming languages
- Inferred type system that makes the code more DRY[1]
- Null safe operator, Elvis operator

As Groovy features a dynamic type system, it makes the code more readable and less verbose, however, as the type checks happens at run time rather than at compile time, the code is more error-prone since a type error is not thrown until the user accesses the affected code [5].

Groovy is the principal programming language that is used for the implementation of the Thesis Management System since the Grails platform is based on it.

1. Don't Repeat Yourself – A principle aimed at reducing repetition in programming code

4.2 Grails

"Grails is an Open Source, full stack, web application framework for the Java Virtual Machine. It takes advantage of the Groovy programming language and convention over configuration to provide a productive and streamlined development experience." [6]. It is a well documented [7], easy to use and easily extensible platform that is designed according to the MVC[2] paradigm, which divides the application into three components that interact with each other greatly improving the project code's readability and maintainability. The Grails framework uses several well-known Java technologies under the hood, including:

Hibernate – ORM[3] framework that is used as layer for data access, Grails wraps its API with Groovy making it much easier to configure and use.

Spring – The most popular application development framework for enterprise Java [8] and a Java EE competitor.

SiteMesh – Web application template framework which follows the decorator design pattern, Grails uses it for GSP[4] views.

To start with Grails, all you need to do is download and install the Grails distribution and then execute command `grails create-app [name]`, which creates the directory structure and default configuration of your application. You can then write your domain model and, thanks to grails feature called *Scaffolding*, auto-generate the controllers and views for a particular domain class. To start the application, you simply execute `grails run-app` and the application is launched in development environment on Tomcat 7 servlet container. This experience makes first steps with Grails very easy and fun, the only caveat is that the auto-generated code needs a lot of polishing to improve its readability.

Grails was chosen for the implementation of the Thesis Management System because the project is mainly developed for JBoss division of Red Hat – and thanks to Grails compiling to Java bytcode, it is easy to run on any Java application server (e.g. JBoss Application Server). Furthermore, it is based on famous Java and JBoss technologies, which makes it easier to maintain for people familiar with such technologies.

---

2. Model–view–controller
3. Object-relational mapping – technique for converting objects into relational database compatible data
4. Groovy Server Pages

4.3 Git and GitHub

It is nearly impossible to develop a project as big as the Thesis Management System without a version control, therefore you usually need to choose a version control system. There are a lot of options you can choose from including Subversion, Mercurial or CVS. Git is, however, undoubtedly one of the best options as it is the most popular, flexible and scalable VCS[5] there is. It is designed and developed by Linus Torvalds, a principal software developer of the Linux kernel, and the features include very fast branching and a lot of useful commands like `rebase` or `squash` [9][10].

Git is also in comparison with e.g. Subversion decentralized (all repositories are equal), which results in the following properties:

- You can create as many repositories as you want. This means you can for example create a hierarchy of repositories each for a different aspect of the project.
- Most operations are fast because no network is involved due to all commits being stored locally.
- If you want, you can synchronize your commits with another repository with commands `push` and `pull`.
- Allows users to work on complex features without the need to publish it right away.
- No need to worry about system failure in the "central" repository, there are many distributed copies.

GitHub is a source code hosting for projects that use Git as a version control tool. It is free (unless you need a private repository), easy to use web service and with features like forking and code reviewing, it is the most popular code host on the planet [11].

4.4 PostgreSQL

There are many ways you can store data of your application, you can store it in a file or you can use the filesystem hierarchy. These solutions are not, however, scalable enough to handle an amount of work that most applications need and for that reason, more than 30 years ago, Oracle introduced

5. Version control system

the first relational database management system (RDBMS) that stores data in relational tables. Relational databases developed the best mix of simplicity, performance, scalability and compatibility over the years, and therefore they are the predominant choice in storing data even though they are not suitable for storing of objects of current popular object oriented programming languages like Java. There are, however, many technologies that mitigate these drawbacks like ORM or RowSet. ORM frameworks, for instance, usually provide an API that effectively creates a "virtual object database".

To manage data in a relational database, the SQL language was introduced. SQL allows you to select, update, delete or insert data into relational database using an easy declarative syntax. For example, query `SELECT * FROM thesis` returns all data in the `thesis` table and query `DELETE FROM thesis WHERE id = 1` deletes all rows that contain `1` in column `id` from table `thesis`. Modern relational databases support features like triggers, cursors or procedural SQL, which allows you to create the back end of your application entirely in SQL syntax.

One of the most convincing features of relational databases is transaction support. A transaction is a group of operations that have the following properties:

Atomicity – Either all operations occur, or nothing occurs. This prevents occurring of partial updates to the database.

Consistency – No operations violate integrity constraints.

Isolation – Two concurrent transaction are isolated from changes done by the other.

Durability – When a transaction is finished, all data is saved permanently even if the system, for example, crashes.

There are a lot of relational databases, the most popular are MySQL, SQLite and PostgreSQL. OpenShift only supports MySQL and PostgreSQL. PostgreSQL is believed to be more mature and more scalable [12], but for most applications either choice is a good one. For the Thesis Management System, we chose PostgreSQL because is easy to configure, and it has great documentation and support.

4.5 MongoDB

MongoDB is a NoSQL database written in C++ [13] that stores data in JSON[6]-style documents. It features high scalability, master-slave replication, load balancing, file storage and, as well as regular relational databases, it supports indexes on attributes.

In the past 5 years, NoSQL databases have been getting more popular with MongoDB on top [14]. This happens mostly because relational databases are hard to scale and mapping objects on JSON objects is much easier than on tables, even if you use a really good ORM framework. Another reason is that objects in dynamic languages tend to change number of attributes at runtime, with MongoDB, all attributes get stored because MongoDB offers dynamic schemas.

MongoDB, as opposed to relational databases, does not store data in tables. Instead of tables, MongoDB stores data in collections of documents, so a table in a relational database is equivalent to a collection in MongoDB and a row in a table is equivalent to a document. In practice, if you develop an application in an OO[7] programing language, with a relational database, a table usually represents a class (or type) and a table row represents an instance of a class, but if you use MongoDB, a collection represents a class and a document represents an instance.

CRUD[8] operations in MongoDB are much more user friendly for people that are familiar with OO programing languages. Read operations are done via a query with the following syntax:

```
db.collection.find( <query>, <projection> )
```

Where the `<query>` argument corresponds to the `WHERE` statement of an SQL `SELECT` query and `<projection>` argument corresponds to the set of arguments following an SQL `SELECT` query. Write operations follows the same syntax but instead of `find` method, you use either `insert`, `update` or `remove` method.

There are some drawbacks a newcomer from RDBMS must consider, though. If you choose MongoDB for a project, you must manage transactions in your application logic, because MongoDB does not support them. And write operations are atomic only on the level of a single document. You cannot count on integrity constraints either, because MongoDB does not have any apart from the very basic ones. All these drawbacks can be mitigated by a good framework but it is recommended to consider the choice

6. JavaScript Object Notation
7. Object Oriented
8. Create, Read, Update, Delete

of a database storage carefully.

We use MongoDB in the Thesis Management System because we need to store some dynamic data (i.e. data that changes schema) and we also use it to store files.

4.6 OpenShift

OpenShift is Red Hat's Cloud Computing *Platform as a Service* offering [15]. Platform as a Service (PaaS) offerings facilitate the deployment of web-based applications without the cost and complexity of buying servers and setting them up [16]. In short, this means that you can set up your production environment with all software needed (database, application server) in a matter of minutes and you can deploy your application by one command-line command.

Figure 4.1: Platform as a Service overview [15]

Red Hat is an open-source company, i.e. every project developed by Red Hat is open-source – and so is OpenShift. OpenShift is build on RHEL[9], which is a stable Linux distribution with long-term support, it is written in Ruby programming language and it runs your applications on Amazon EC2[10].

Creating your first application on OpenShift is very straightforward. First, you choose a type of application, which is called a cartridge in OpenShift terminology. You can choose from a variety of cartridges including, for example, JBoss Application Server, Tomcat 7, Ruby on Rails and Django, or you can choose the Do It Yourself cartridge and set up the whole environment yourself. Once you have done that, you choose a public URL, a source code repository, gears and scaling. A gear is a container with limited resources that runs your application, i.e. virtual server. By default, you can only choose the default small gear. If you pay a monthly fee, you can choose medium or large gear with more resources at your disposal. Scaling

9. Red Hat Enterprise Linux
10. Amazon Elastic Compute Cloud – cloud computing platform that allows users to rent virtual computers

allows your application to be load-balanced, i.e. if your applications' traffic increases, OpenShift will allocate more gears. When you have your application created, you can do some basic configuration through the web interface or you can add another cartridge to your application, e.g. a database or Cron. If you need to connect to the server where your application runs, you can open a Secure Shell session (supposing you've set up your public RSA key). This overall flexibility decreases time spent managing software and hardware, and it makes it easy to deploy your application without any fuss, so at least from that point of view, it makes sense to use OpenShift even if you have no choice but to use the Do It Yourself cartridge.

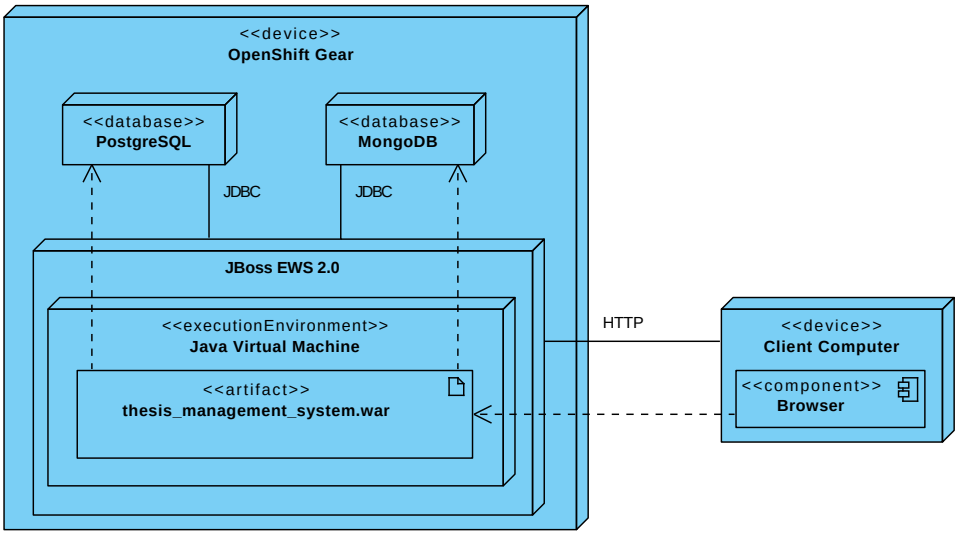


Figure 4.2: Deployment diagram of the Thesis Management System

If setting up a new OpenShift application is simple, deploying one is even more simple. You just push the application's source code to either the repository you specified or the repository that was created for you. You might need to set some properties in your application (e.g. database credentials), though. Fortunately, OpenShift exports all needed properties as environment variables so that you could access them via the API of your programming language and use them wherever necessary. If you need to inject or modify some environment variables at some point (e.g. before or after build), you can use action hooks. These lifecycle hooks allows you to execute any kind of shell script at a certain point in time, for example, you might want to append a property to the `JAVA_OPTS` environment variable

just before the build of your application. To do that, you put the following script in `${project.dir}/.openshift/action_hooks/pre_build`:

```
#!/bin/bash
export JAVA_OPTS="$JAVA_OPTS -DsomeProperty=true"
```

The script above gets executed directly, so you can use PHP, Python, Ruby or any other interpreted programing language.

The Thesis Management System is deployed on medium gear and it runs on JBoss EWS 2.0, which is a servlet container based on Tomcat 7. It also uses the PostgreSQL and MongoDB cartridges. Figure 4.2 illustrates the deployment configuration of the system.

4.7 Others

There are a lot of other technologies used for the implementation of the Thesis Management System, but since some of them were not used by the author and others did not play a crucial role in the implementation of the system, they will be described only briefly.

Spring Security – A Spring framework that provides authentication and authorization for Java.

Searchable – Grails plugin that adds search functionality based on Compass and Apache Lucene [17].

JQuery – The most popular JavaScript framework.

LESS – CSS preprocessor featuring e.g. variables, mixins, operators and nesting.

Twitter Bootstrap – CSS and JavaScript framework and the most popular project on GitHub.

5 Project Architecture and Design

In the following text, the author will describe the architecture and design of the Thesis Management System including functional and non-functional requirements, and the domain model. The author will also depict some design parts with UML[1] 2.0 diagrams that were modeled with Visual Paradigm tool [18], however, the UML diagrams are not to be regarded as blueprints of the system, but rather as sketches, and as such, some information that the author considers irrelevant or obvious are let out. This is allowed as stated by Martin Fowler in his book UML Distilled [19].

5.1 Functional Requirements

1. User management
   The system allows to create, read, update and delete (CRUD) users. User contains fields full name, email and password.
2. Registration
   Only students can sign up and they have to agree to the terms of use.
3. Log In
   Anonymous users can sign in using email and password.
4. Thesis topic management
   The system allows to CRUD thesis topics. Topic contains fields title and description in two languages and it is possible to assign the list of tags and categories to it. The management of a topic is done by its leader and the list of the universities that the topic is offered to is represented by the university list. There is also a field for the list of university supervisors, each supervisor is assigned a university that they supervise at. One topic can have zero or more university supervisors. Lastly, there is a field for the list of thesis types (which can contain e.g. diploma, bachelor) and a flag that represents enabled/disabled state of the topic to disallow students to apply for it.
5. Category management
   The system allows to CRUD categories. Category contains fields title and description.

1. Unified Modeling Language

6. University management
   The system allows to CRUD universities. University contains a field for the name of the university.

7. Application management
   The system allows to CRUD applications.

8. Topics can be filtered
   The system allows to filter topics by university, type, leader, title, tag or category.

9. Thesis management
   The system allows to CRUD theses. Thesis contains fields title, description, thesis topic, assignee, supervisor, abstract, type, university and the list of tags. There is also a field for status that can be either 'in progress', 'finished', 'failed' or 'postponed' and a field for grade that can be one of A, B, C, D, E or F.

10. Supervisors can select universities they supervise at for each topic

11. Supervisors can add private notes to theses

12. Authenticated user can subscribe for topics and theses
    The system sends notifications to all subscribers of a topic or a thesis when a change is made to it.

13. Authenticated user can unsubscribe from topics and theses

14. Theses can be filtered
    The system allows to filter theses by title, supervisor, assignee, status, grade, type, university or tag.

15. File management for theses
    The system allows to upload files to theses.

16. Discussion for topics and theses
    Logged in users can comment on topics and theses and users with certain authority (administrators and leaders) can create comments that are not visible to students and guests.

17. Full text search
    Theses and topics can be searched by a full text search engine.

18. Frequently Asked Questions (FAQ) management
The system allows to CRUD Frequently Asked Questions. One frequently asked question contains fields question and answer.

5.1.1 Use Case Diagram

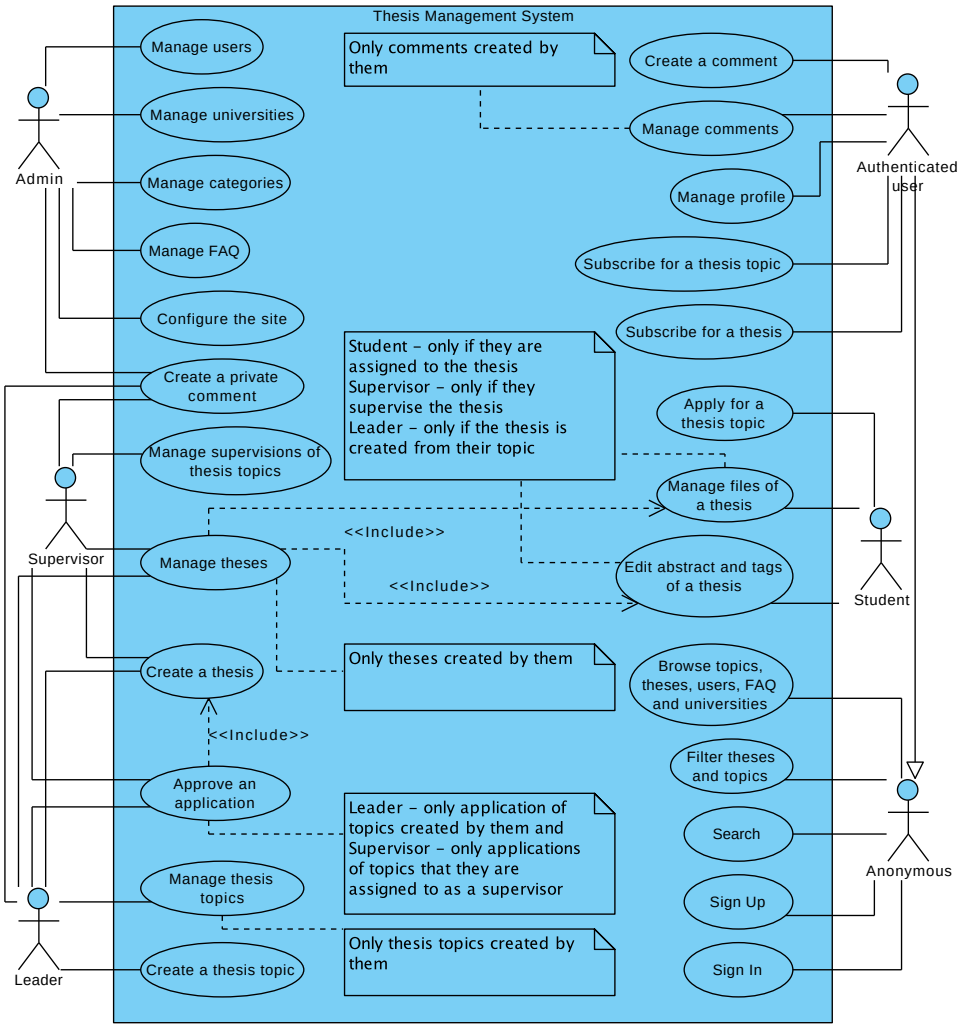


Figure 5.1: Use case diagram of the Thesis Management System

5.2 Non-Functional Requirements

1. Grails platform
   The system is implemented using platform Grails.

2. Authentication and authorization
   The system is secured and there are roles each with certain privileges.

3. Performance
   The system offers reasonable performance at least up to five hundred created topics, theses, users and universities.

4. Deployment
   The system is deployed on OpenShift using any cartridge other than Do It Yourself.

5. License
   The system is licensed under a GPL compatible license.

6. Backup
   There is available a shell script for backing up the database.

7. Internationalization and localization
   The system is internationalized and localized in English and Czech.

8. Software development methodology
   The system is developed iteratively and incrementally.

5.3 Domain Model

Designing the domain model is the most important step when developing an information system. It is also the most difficult one, because you usually cannot easily make changes in the domain model after you implemented the application logic around it. You can, however, add some domain model elements that the already-designed elements do not relate to. For example, if you design an entity `User`, you can implement it and easily add new entity `Group` later on if the `Group` is the owner of the relationship, see fig. 5.2. If the `User` is the owner of the relationship (see fig. 5.3), more refactoring is required to implement such changes (especially in case of object composition), because the fundamental API of the implemented part of the system changed.

Figure 5.2: Example of domain model where `Group` owns the relationship

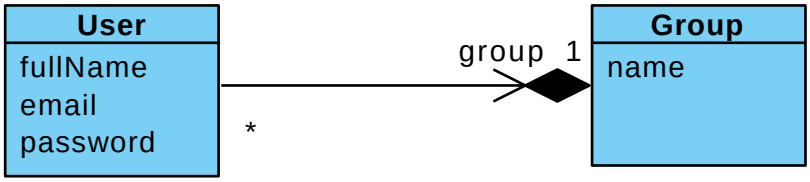


Figure 5.3: Example of domain model where `User` owns the relationship and composition is introduced

Even more difficult can be deleting or editing attributes of an implemented entity. As a lot of refactoring is necessary to cope with such changes, a lot of new bugs is usually introduced, so it is best to avoid such changes by designing the domain model thoroughly.

Note that it is necessary to change the domain model if you implement your application incrementally. You should, however, try to minimize the changes that require the API of the implemented domain model to be changed.

5.3.1 Domain Model of the Thesis Management System

As the domain model of the Thesis Management System is very complex, the author will describe it part by part. The complete domain model can be found in the appendix A.

5.3.1.1 `User` entity

To fulfill the first three functional requirements, we need to register users. For that reason, we introduce entity `User` with fields `email` and `password` that serve as log in credentials and field `fullName`. We also need fields `accountLocked` to allow the administrator to lock a user account and `enabled` for registration confirmation purposes. Field `sendMail` represents the user setting that allows them to disable receiving notifications by

email, and field `dateCreated` represents the date of user's registration. To implement authorization, the `User` entity needs field `roles`, which contains the list of user's authorities. Figure 5.4 illustrates the described model.

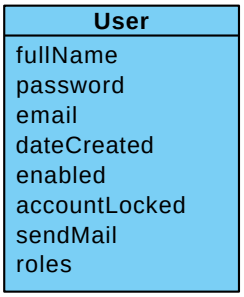


Figure 5.4: Model of `User` entity

5.3.1.2 `University` entity

To be able to keep a record of the universities that a topic is offered to, we need the entity `University` with only one field – `name`, see Figure 5.5.

Figure 5.5: Model of `University` entity

5.3.1.3 `Article`, `Comment` and `Subscription` entities

To allow users to add comments to topics and theses, we need an entity that represents comments. The name of the entity is, for obvious reasons, `Comment` and fields that the entity requires are:

- `content` – Represents the content of a comment.
- `privateComment` – Marks private comments.
- `dateCreated` – The date of creation, this field is used for sorting comments so that we could show the user comments in either descending or ascending order.

There are two association we need to create for `Comment`. First is a many-to-one association between `Comment` and `User` to be able to display the author of a comment. Second is a bit more complicated, because there are two approaches to associate comments with topics and theses. We could either create a many-to-one association between entities `Comment` and `Topic` and between entities `Comment` and `Thesis`, or we could use generalization and move the many-to-one association up from `Thesis` and `Topic` to their parent. We choose the later approach because it allows us to create another "commentable" entity in the future simply by making it another child of the parent entity.

The `Article` entity represents the parent entity described above. We place the field `title` on it for reasons described in the following paragraph.

The `Subscription` is only an association entity with no attributes on it, but for the sake of future improvements, we model it separately. We could, for example, allow users to choose if they want to receive email notification for a particular subscription, or let them set up what changes made to an `Article` they want to be notified about. As far as associating the entity with other entities is concerned, it is very similar to the `Comment` entity, because we need to implement subscription functionality for both topics and theses. We add the field `title` on the `Article` entity because we want to show subscribers the name of the `Article` that changed.

Described model is illustrated in Figure 5.6.

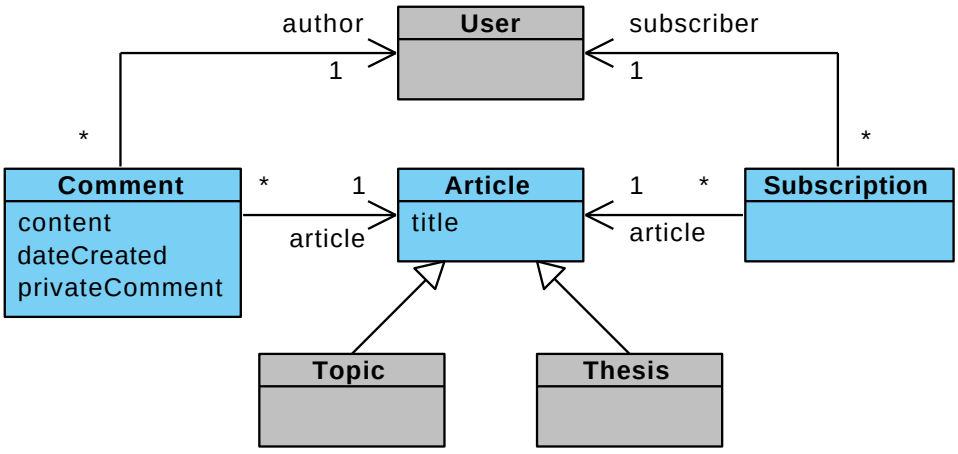


Figure 5.6: Model of `Article`, `Comment` and `Subscription` entities

5.3.1.4 `Feed` and `Notification` entities

Entities `Feed` and `Notification` are closely related to the `Subscription` entity even though they are not directly associated. When a change is made to a topic or a thesis, the subscribers are notified by email and by a notification within the system. Emails are stored by the email hostings, but the notifications need to be stored in the database of the Thesis Management System, because they must be available to users when they visit the system.

Now, there are several ways to implement such functionality, but the most notable ones are the following two. The first approach is to create a `Notification` entity with the `message` field that represents the content of the notification, and associate it with the `User` entity that represents the subscriber. This approach is favorable because it is simple, however, it creates unnecessary redundancy in the database as a new notification with the same `message` is created for every subscriber. The second approach, which is used in the Thesis Management System, is to create another entity `Feed` and move the `message` field into it. The later approach is also more suitable if we want to show the user a notification according to their locale setting.

Figure 5.7 illustrates the model of this part of the system. Since we want the notifications to be localized, the `Feed` entity has fields `messageCode`, which represents the code of the massage to be displayed as a notification, `args`, which represents the arguments of the message, and `dateCreated`, which allows us to sort the notifications to show users the latest notifications first. The `Feed` entity is also associated with the `User` entity, which represents the user who triggered the creation of the feed. This allows us to find user's activity.

The `Notification` entity is basically an association entity between entities `User` and `Feed`. The field `seen` represents status of a notification, i.e. if the user has already seen the notification or not.

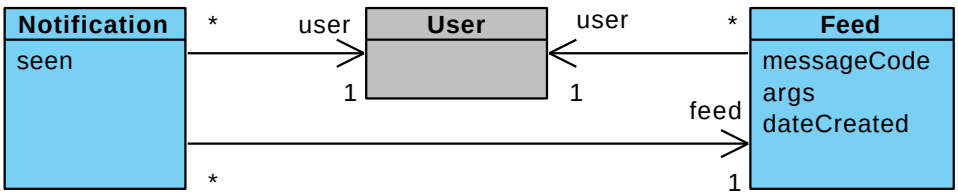


Figure 5.7: Model of `Feed` and `Notification` entities

5.3.1.5 `Tag` entity

The most common way of implementing tags in an information system is to place collection of simple strings in an entity. We chose a different approach for the Thesis Management System, though. We introduced a separate entity `Tag` that contains a simple field `title`. Making the `Tag` entity separate makes it easy to add new fields on it in the future, e.g. `description`, and it is easier to aggregate, for example, the number of all tags used in the system. Figure 5.8 depicts the model.

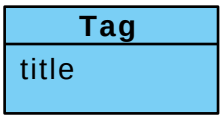


Figure 5.8: Model of `Tag` entity

5.3.1.6 `Topic`, `Category` and `Supervision` entities

`Topic` is the most important entity in the system because it represents the topic of a thesis. It is a child entity of the `Article` entity, which means that the `title` field is included in the `Topic` entity automatically. Description of the other fields follows:

- `secondaryTitle` – Represents the title of the topic in a different language (e.g. Czech, Slovak).
- `lead` – Lead paragraph, is displayed on the page with the list of topics.
- `Description` – Description of the topic.
- `secondaryDescription` – Represents the description of the topic in a different language.
- `dateCreated` – The date of creation of the topic.
- `enabled` – Flag which marks a topic that is enabled or disabled, i.e. students can(not) apply for it.
- `types` – Collection of types of the topic (e.g. bachelor, diploma).

The `Topic` entity also has a many-to-many association with the `Tag` entity and the `User` entity, which represents the leader of the topic, i.e. the

user who created the topic and is in charge of it. The association with the `University` entity is necessary to keep a record of universities that the topic is offered to, i.e. students of which universities can apply for it.

Another requirement of the system is to place topics in categories. Categories are represented by the `Category` entity, which has only two fields – the `name` field, which represents the name of the category and also the `description` field, which is self-explanatory. The `Topic` entity is associated with the `Category` entity in a many-to-many manner. In the first stages of the design, we expected the categories to be hierarchical, i.e. a category could have several subcategories, but we decided to abolish that approach as it would place unnecessary overhead on the database.

The most difficult part of the system was to design the requirement allowing topics to list university supervisors. There are zero or more supervisors in one topic, but one supervisor can supervise a topic at multiple universities, which means that the supervisors cannot be associated with the `Topic` entity directly. Instead, we introduced the `Supervision` entity, which is basically an association entity without any fields representing a supervisor associated with a university.

Described design is depicted in Figure 5.9.

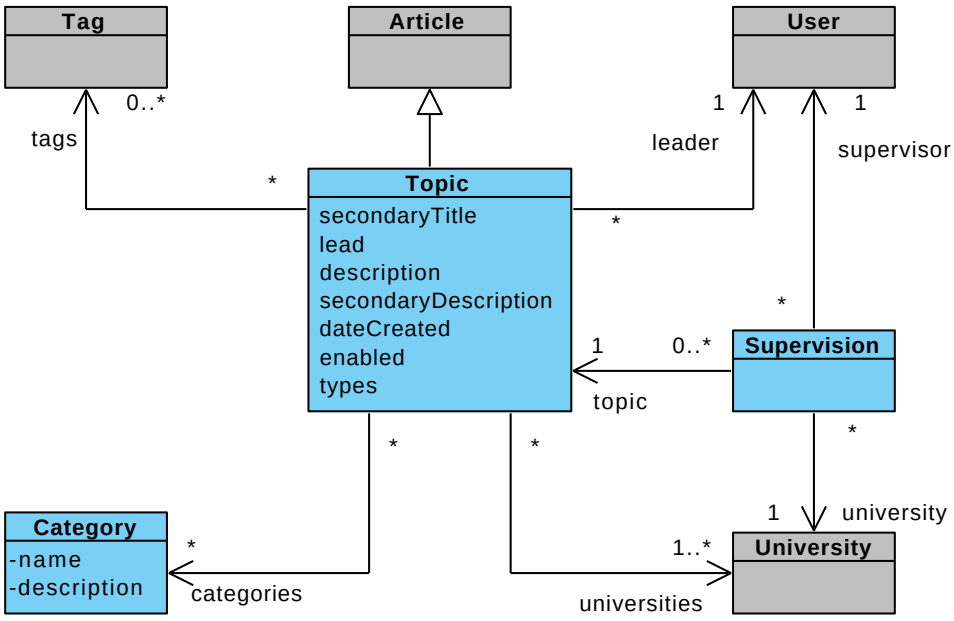


Figure 5.9: Model of `Topic`, `Category` and `Supervision` entities

5.3.1.7 `Thesis` entity

The second most important entity of the system is the `Thesis` entity, it represents an academic project of a student and it is associated with the `Topic` entity. There can be multiple theses created from one topic for several reasons:

- One topic can be intended for several students (for example when the topic is too difficult for one student), which means that a thesis for each student needs to be created.
- The topic is offered in more than one semester.
- The student that was assigned to the thesis of a topic failed to finish with a satisfactory result and the topic has to be offered again.

This means that the cardinality of the association with the `Topic` entity must be a many-to-one one.

The `Thesis` entity is, as well as the `Topic` entity, a child of the `Article` entity, which takes care of the title, comments and subscriptions. Theses are not required to be stored in two languages, which means that there is no need for both the secondary title and the secondary description. There are several other fields required, though:

- `description` – Represents the *official* description of a thesis.
- `status` – Status of a thesis, which can be either 'in progress', 'finished', 'failed' or 'postponed'.
- `grade` – If the thesis is finished, a grade must be awarded. The grade can be one of the following: A, B, C, D, E, F.
- `abstract` – The abstract of a thesis provided by the assigned student.
- `dateCreated` – The date of creation of a thesis.
- `type` – The type of a thesis, which can be for example 'bachelor' or 'diploma'.
- `notes` – A field for the supervisor where they can leave notes related to the thesis.

The student that is assigned to a thesis and the supervisor are represented by the associations with the `User` entity and the thesis university is represented by the one with the `University`. The tags of a thesis are designed the same way as with the `Topic` entity. The reason why the association with the `Tag` entity is not moved up to the `Article` entity is because in the future, there might be added a functionality for management of school projects that does not require tags, but requires the functionality provided by the `Article` entity.

Figure 5.10 illustrates this model.

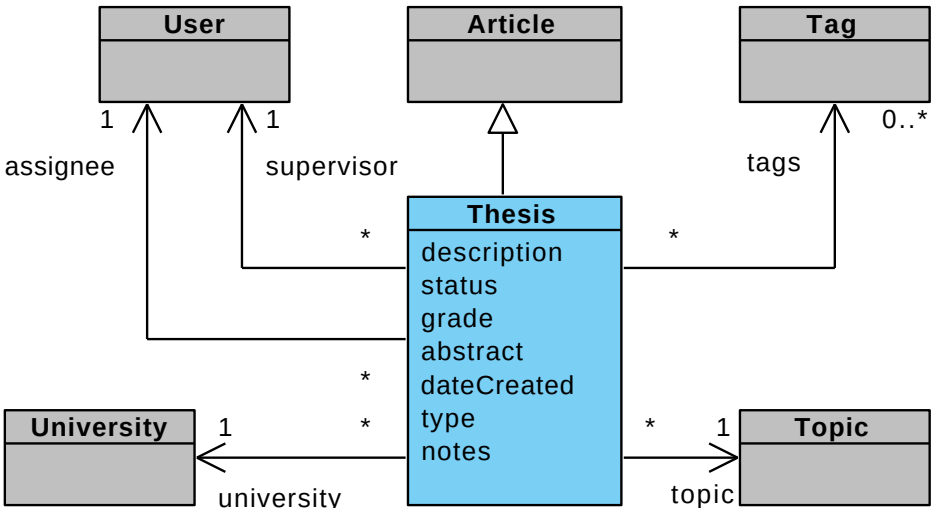


Figure 5.10: Model of `Thesis` entity

5.3.1.8 `Application` entity

When a student applies for a topic, an application must be created to allow the leader or a supervisor to review and possibly approve it. To apply for a topic, a student must choose the type and the university for which they apply. The type is stored in the `type` field and the university is represented by the association with the `University` entity. The student can also leave a note, e.g. the preferred supervisor, which is stored in the `note` field. The `dateCreated` field allows the authority (leader or supervisor) to see what application was created first and the `approved` field marks applications that were already approved. The association with the `User` and the `Topic` entity represents the student who applied for the topic and the topic that

the application is created for, respectively. The association with the `Thesis` entity allows us to display the thesis that was created from an application. Note that this association has a 0..1 multiplicity (in both directions). This is because when a student applies for a topic, the thesis has not been created yet, because it is created as soon as the leader or a supervisor approves it. The multiplicity from the other direction results from the fact that a thesis can be created directly from a topic (i.e. no application is created). This model is depicted in Figure 5.11.

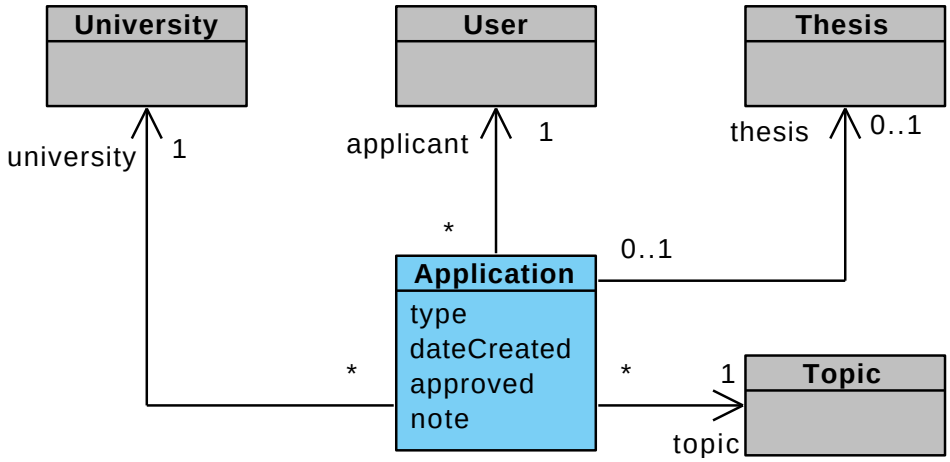


Figure 5.11: Model of `Application` entity

5.3.1.9 `Faq` entity

To allow new users to adapt to a new system more quickly, Frequently Asked Questions are introduced. The FAQ are represented by the `Faq` entity with the following fields:

- `question` – The question to be answered.
- `answer` – The answer to the frequently asked question.
- `locale` – The locale of the users for whom the frequently asked question is displayed.

See figure 5.12 to see the designed model.

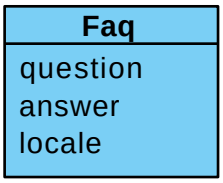


Figure 5.12: Model of `Faq` entity

5.4 Implementation

The core project of this thesis is created by three students, therefore the implementation of it will be described in the thesis of student Jakub Čecháček.

5.5 Evaluation

In this section, the author will evaluate all functional and non-functional requirements.

5.5.1 Functional Requirements

This section contains evaluation of all functional requirements. Please refer to section 5.1 to see the list with the description.

1. User management
   Anyone can access users' profiles (show page). The administrator can create new users on the page with the list of users, when they do, the password is sent to the entered user's email and activation is required. The administrator can also edit and delete users in the users' profiles.

2. Registration
   Anonymous users can sign up on the registration page, but it is only possible to register with email addresses hosted at a domain address that is explicitly entered in the system configuration by an administrator. It is also necessary to agree to the terms of use, otherwise it is not possible to complete the registration. When a user sings up, an activation email is send to their email address. If the user activates their account, the registration is complete, if not, it is not possible to log in and the account is deleted after 30 days.

3. Log In
   Anonymous users can sign in using email and password if their account is activated. If the user does not remember their password, they can request a new password to be send to their email address.

4. Thesis topic management
   List and details of topics can be accessed by anyone. Leaders can also create new topics, and edit and delete topics that they created. Deletion of topics that were already applied to is not possible unless the applications and theses associated with them are deleted. All required fields are available.

5. Category management
   The list of categories is displayed on the page with the list of topics. Only administrators can create, edit or delete categories. It is not possible, however, to delete categories that already contain some topics. Instead, the topics must be either deleted or removed from the category first. The description field is shown as a tooltip text for each category unless it is not filled in.

6. University management
   Universities can be browsed by anyone, but only administrators can create, edit and delete universities. As a safety precaution, a university cannot be deleted if there are already topics, applications or theses associated with it (to delete such university, the associated theses, applications and topics must be deleted first).

7. Application management
   Students can apply for topics by clicking on the button placed on the page showing a particular topic. When they do, they choose their university, the type of their study (Bachelor, Diploma) and optionally they provide a note. A notification is sent to the leader of the applied topic. The leader can decide to approve the application by clicking on the button "Approve", when they do, they are redirected to the page with a form for thesis and they have to create it to approve the application. If the leader approves the application, its student is automatically notified.

8. Topics can be filtered
   The system allows to filter topics by university, type, leader's full name, title, tag or category. All these filter options can be combined,

the title and leader's full name is case insensitive, but unfortunately they are currently sensitive to foreign characters.

9. Thesis management
   Similarly to topic management, the list and details of theses can be accessed by anyone. Supervisors and Leaders can create new topics either from scratch, application or topic. Theses they created can be also deleted or edited by them. All required fields are available.

10. Supervisors can select universities they supervise at for each topic
    There is a "My Supervisions" button on the page with a topic. When a supervisor uses the button, they are redirected to a page where they can choose universities they supervise at for a particular topic.

11. Supervisors can add private notes to theses
    There is a "Notes" button on the page with a thesis. When a supervisor clicks on the button, a popup box with a textarea is displayed. They can enter or delete text as they wish, to save the notes, they have to use the "Save" button.

12. Authenticated user can subscribe for topics and theses
    The "Subscribe" button is displayed on every page of topics and theses. If a user subscribes for a topic or thesis, they are sent notifications about all actions that are made to it, which includes edits, deletes and comment creates.

13. Authenticated user can unsubscribe from topics and theses
    If a user does not with to receive notifications about actions made to topics or theses anymore, they can use the "Unsubscribe" button.

14. Theses can be filtered
    The system allows to filter theses by title, supervisor's full name, assignee's full name, status, grade, type, university or tag. All these filter options can be combined, the title, supervisor's full name and assignee's full name are case insensitive, but unfortunately they are currently sensitive to foreign characters.

15. File management for theses
    Students can upload files to their theses on the thesis page by first selecting the files to be uploaded by the "Select Files" button and starting the upload by the "Start Upload" button. The supervisor of the thesis and the leader of the thesis' topic can also upload files to it.

16. Discussion for topics and theses
    Authenticated users can comment on topics and thesis. They can also edit or delete their comments. The administrators and leaders can create private comments that are visible only to users with those roles. The administrator can edit and delete any comment (not only those created by them), the leader can edit and delete comments that are created in topics created by them.

17. Full text search
    There is a search box available on every page. If a user uses it, they are redirected to a page with the search results.

18. Frequently Asked Questions (FAQ) management
    Administrators can create/delete/edit frequently asked questions. The questions are internationalized and therefore an administrator not only needs to fill in the question and the answer of a FAQ, but they also have to choose the locale of the FAQ. Users see only FAQs created for the locale that they use.

5.5.2 Non-Functional Requirements

This section contains evaluation of all non-functional requirements. Please refer to section 5.2 to see the list with the description.

1. Grails platform
   This requirement was fulfilled by implementing the system using the Grails framework.

2. Authentication and authorization
   The system requires the users to be authenticated to allow them to use most of the functionality. There are four roles – Admin, Leader, Supervisor and Student, and each role has certain permissions, read section 5.5.1 for more information.

3. Performance
   We created a script that put 500 topics, theses, users, applications and universities in the database of the Thesis Management System. The system's performance did not seem to be affected at all.

4. Deployment
   The system is deployed on OpenShift using medium gear and cartridge JBoss EWS 2.0. To deploy the Thesis Management System on a different instance of OpenShift, follow these steps:

(a) Create an OpenShift JBoss EWS 2 cartridge and add PostgreSQL and MongoDB cartridges.

(b) Add remote OpenShift repository to your local project git repository.

(c) Create file `.myenv` on OpenShift in the directory exported in `$OPENSHIFT_DATA_DIR` with email credentials as environment variables like this:

```
export OPENSHIFT_EMAIL_HOST="smtp.example.com"
export OPENSHIFT_EMAIL_PORT=465
export OPENSHIFT_EMAIL_USERNAME="e@exmpl.com"
export OPENSHIFT_EMAIL_PASSWORD="password"
```

(d) Push your local repository to OpenShift.

Note: You need at least medium gear, otherwise grails will not be able to build the application due to memory limitation. Or you can use small gear and build the application yourself.

5. License
We chose the MIT license [20] for the Thesis Management System, which is a GPL compatible license [21].

6. Backup
There is available a shell script `backup.sh` for backing up both PostgreSQL and MongoDB. To create the backup, you just execute command `sh backup.sh` on OpenShift, which will create directory `thesis_backup_mongo` (containing the MongoDB backup) and file `thesis_backup_postgres.sql` (containing backup of PostgreSQL) in directory `$OPENSHIFT_DATA_DIR`.

7. Internationalization and localization
The system is internationalized and localized in English and Czech. The locale is either chosen from HTTP headers or the user can choose one by themselves.

8. Software development methodology
The system is developed iteratively and incrementally, see section 3.4 for more details.

6 Future Improvements

Overall, the Thesis Management System is a rather large project and as it is only a bachelor thesis divided between three students, there are many features that could be implemented in the future. In this chapter, the author will investigate such possible features and describe them accordingly.

6.1 Integration with Universities

The most useful feature would be the integration of the Thesis Management System with the information systems of the universities of which Red Hat is industrial partner. This would cause all changes done in the Thesis Management System to be automatically reflected in the information systems of the universities.

There are two ways this feature could be implemented. The first approach is to use the API of the universities, which requires to implement the integration with each university separately. This is obviously very expensive and hard to maintain, but the advantage is that there is little activity required from the universities. The second approach is to standardize and implement an API in the Thesis Management System and leave the integration to the universities. The API could follow, for example, the REST[1] architecture model. The obvious disadvantage of this approach is that it could be complicated to convince the universities to do their part of implementation, which is because it could require them to do a lot of work as each university uses a different information system based on a different design model.

If this feature were implemented, it would remove a substantial amount of logistics from the shoulders of both the universities and Red Hat, the author cannot, however, see it implemented in any other way than the later one.

6.2 BPM and BRMS

When a students works on a thesis, their supervisor usually requires them to follow a certain workflow and certain rules. This feature can be achieved by integrating the Thesis Management System with a BPM[2] and/or a BRMS[3]

1. Representational State Transfer
2. Business Process Management
3. Business Rule Management System

tool. There is a variety of BPM tools, the most considered one for our project is jBPM because it is a JBoss project and as such it is supported by Red Hat. jBPM is a light-weight, extensible workflow engine written in pure Java that allows you to execute business processes using the latest BPMN 2.0 specification [22]. For the business rule management, the Drools framework would probably be chosen as it is also supported by Red Hat [23].

It is not yet clear how the integration would be done, but as such feature would help both supervisors and students to successfully meet all requirements of their thesis, it is definitely one of the most considered one.

6.3 School Projects Management

Red Hat not only offers thesis topics to universities, but also school projects for subjects taught there. Implementation of such functionality would allow the lecturers to take advantage of the features implemented in the Thesis Management System.

To implement the project management, it would only require a new domain entity to be introduced, but how it should be integrated is not clear at the moment. It could be, as well as theses, associated with the `Topic` entity and a project would represent the concrete work of a student (or students). Another approach could be to implement project as a standalone unit, which would represent both the topic and the students' work. The former approach would present a problem with the `Topic` entity as it does not allow to persist the number of students that can apply for it. But that could be addressed by creating a child entity that would allow such functionality. The later approach, on the other hand, forces the lecturers to create new projects every semester, because a project also represents the topic and is closed at the end of each semester.

To choose the approach, elaboration with the client (Red Hat) and end users would be required. It is, however, a feature worth the effort as it would push the use case of the Thesis Management System another bit further.

6.4 Import and Export of Theses and Topics

One user of the Thesis Management System raised a feature request for the import and export of theses and theses topics in various formats. If it were the only way, apart from coping it field by field, it would allow authorized users to easily import the theses and the topics into another system.

Implementation of such a feature would be rather simple for possibly any format, because it only requires to create a template with values put in it, which is why it is placed quite near to the top of the TODO list.

7 Conclusion

The main objective of this thesis was to implement an information system for thesis management. In this document, the author briefly analyzed and described the thesis management and several software development methodologies. In the second part of the thesis, the focus was moved towards the technologies that were used for the implementation of the system, and its architecture and design in which the author covered the functional and non-functional requirements of the system and mainly its domain model. The last chapter outlined a few possibilities of improvements that could be dealt with in the future.

As with everything else, the development of the system was accompanied with several difficulties, it was, however, successfully finished and the system is currently being used by Red Hat people. Furthermore, it has proved to be reliable, scalable and usable, but only the time will tell how much those properties hold in the long run.

Bibliography

[1] C. Babcock. (2012, March) Red hat: First $1 billion open source company. [Online]. Available: http://www.informationweek.com/development/open-source/red-hat-first-1-billion-open-source-comp/232700454

[2] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, August 2003.

[3] The waterfall model of software development. [Online]. Available: http://www.seowebsitedesign.com/the-waterfall-model-of-software-development/

[4] Groovy homepage. [Online]. Available: http://groovy.codehaus.org/

[5] D. Koenig, A. Glover, P. King, G. Laforge, J. Skeet, and J. Gossling, *Groovy in Action*. Manning Publications, January 2007.

[6] Grails homepage. [Online]. Available: http://grails.org/

[7] Grails framework. [Online]. Available: http://grails.org/doc/latest/

[8] Springsource homepage. [Online]. Available: http://www.springsource.org/

[9] Git homepage. [Online]. Available: http://git-scm.com/

[10] S. Chacon, *Pro Git (Expert's Voice in Software Development)*. Apress, August 2009.

[11] Github features. [Online]. Available: https://github.com/features

[12] J. Rowe. (2008, August) Answer to stackoverflow question: Mysql vs postgresql for web applications. [Online]. Available: http://stackoverflow.com/questions/27435/mysql-vs-postgresql-for-web-applications#answer-27440

[13] Mongodb homepage. [Online]. Available: http://www.mongodb.org/

[14] Db-engines ranking. [Online]. Available: http://db-engines.com/en/ranking

[15] Openshift homepage. [Online]. Available: https://www.openshift.com/

[16] J. Schofield. (2008, April) Google angles for business users with 'platform as a service'. [Online]. Available: http://www.guardian.co.uk/technology/2008/apr/17/google.software

[17] Searchable plugin for grails – documentation. [Online]. Available: http://grails.org/plugin/searchable

[18] Visual paradigm homepage. [Online]. Available: http://www.visual-paradigm.com/

[19] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Addison-Wesley Professional, September 2003.

[20] The mit license (mit). [Online]. Available: http://opensource.org/licenses/MIT

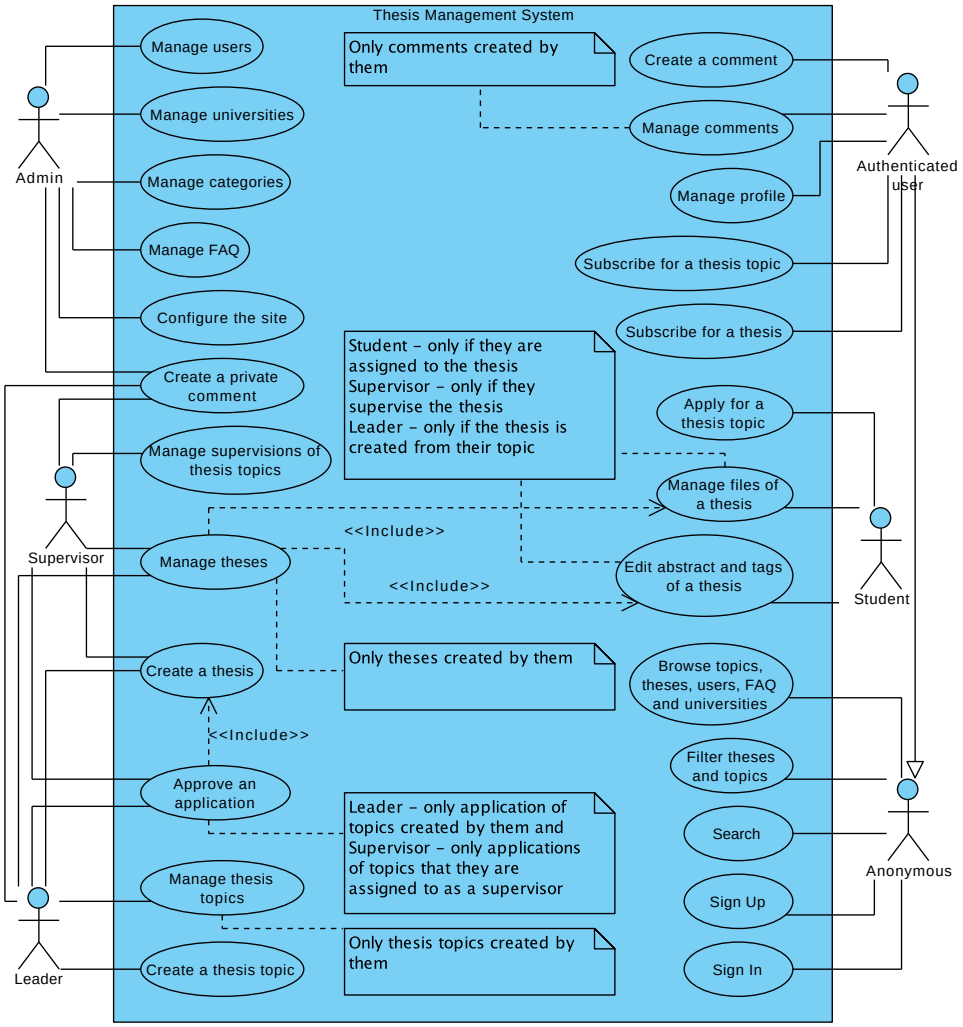
[21] Gpl-compatible free software licenses. [Online]. Available: http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

[22] Jboss jbpm homepage. [Online]. Available: http://www.jboss.org/jbpm/

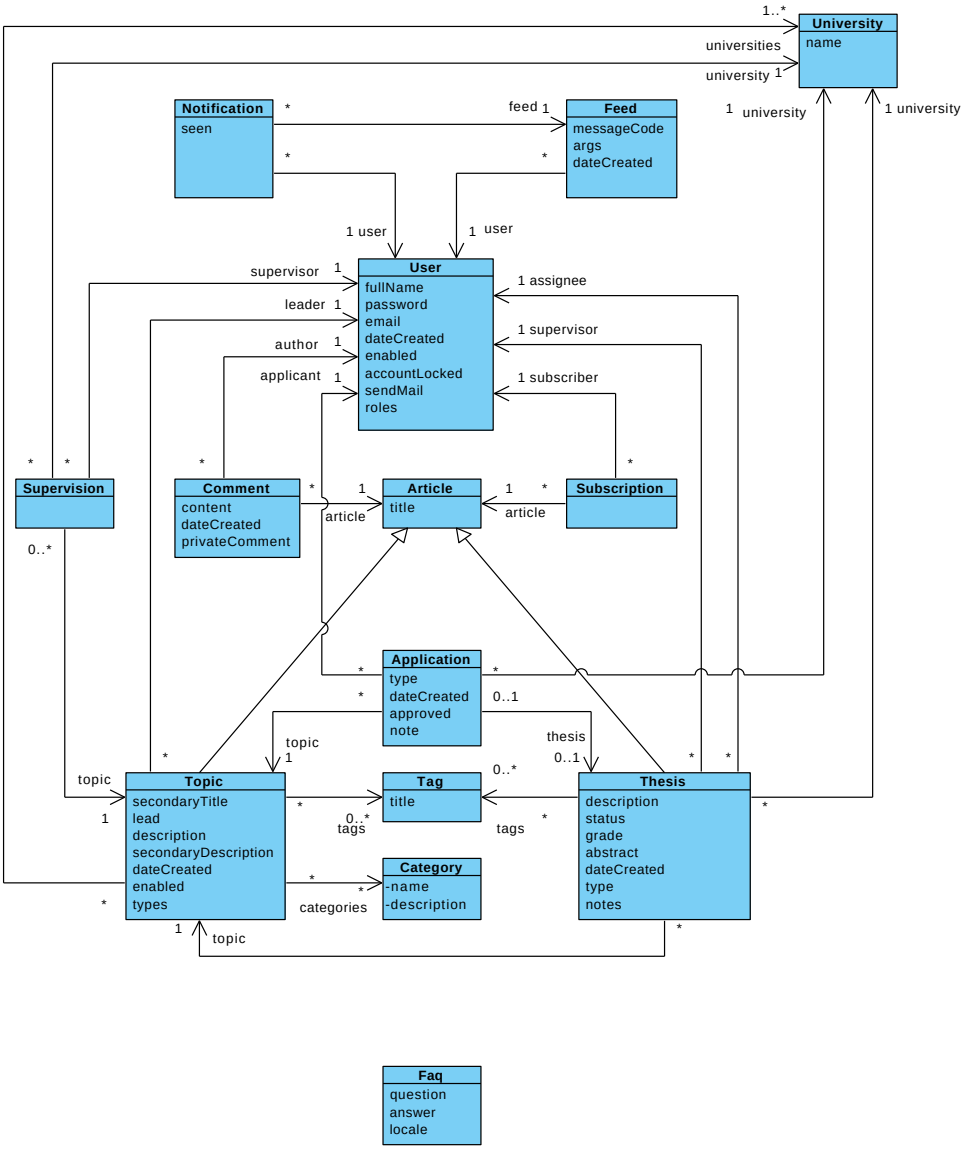
[23] Drools homepage. [Online]. Available: http://www.jboss.org/drools/

A Diagrams

A.1 Use Case



A.2 Domain Model



B Links

B.1 Source Code of the Project

The source code of the core project (Thesis Management System) is hosted on GitHub:
`https://github.com/jcechace/Thesis-management-system`

B.2 OpenShift URL of the Project

`https://thesis-managementsystem.rhcloud.com/`