Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Bachelor's Project

Analysis of students schedules and checking unusual study plans - collection of web applications

Filip Sivák

Supervisor: doc. Ing. Tomáš Svoboda, Ph.D.

Study Programme: Open Informatics, Bachelor

Field of Study: Software systems

June 9, 2014

Aknowledgements

I would like to thank to Daniel Vecerka for support with virtual server cwtest and Leona Svobodova for providing valuable feedback for the application.

Declaration

Abstract

This work describes usage, implementation and deployment of web application designed to find when students have free time in their schedule, to find empty rooms, find out which minor study fields students of Open Informatics have completed and report on collisions in schedule. The application is implemented in Java using Spring framework for the server and Google web toolkit for the client. The application is loading data from KOSapi web service.

Contents

1 Introduction **1**
1.1 About this work . . . 1
1.2 How to read this work . . . 1

2 Problem analysis **3**
2.0.1 User . . . 3
2.0.2 Timer . . . 3
2.1 Requirements . . . 3
2.1.1 Functional requirements . . . 3
2.1.2 Non-functional requirements . . . 4
2.2 Showing timetable of multiple students . . . 4
2.2.1 Existing solutions . . . 4
2.2.2 Implemented solution . . . 7
2.3 Finding an empty room . . . 7
2.3.1 Existing solution . . . 8
2.3.2 Implemented solution . . . 8
2.4 Check of unusual study plans . . . 8
2.5 Detecting collisions . . . 8
2.5.1 Existing solution . . . 9
2.5.2 Implemented solution . . . 9

3 User guide **11**
3.1 Finding out when students have free time . . . 11
3.2 Finding free room . . . 12
3.3 List of minors of students . . . 13
3.3.1 Definition of minor in application . . . 14

4 KOSapi **15**
4.1 Extracting interesting data from KOSapi . . . 15
4.1.1 The list of students of particular programme or branch . . . 15
4.1.2 Timetable of student . . . 15
4.1.3 Completed courses of student . . . 15
4.2 Limitations . . . 16
4.2.1 Absence of join . . . 16
4.2.2 Fetching data for all semesters at once . . . 17

4.2.3 Support for caching . . . 17
4.2.4 No `last-modified` header . . . 17
4.2.5 CORS is disabled . . . 17
4.3 Problems . . . 17
4.3.1 Inaccessible data . . . 17
4.3.2 Non unique codes . . . 18
4.3.3 Missing pieces of data . . . 18
4.3.4 Using atom:updated field does not speed up fetching process . . . 18

5 Implementation 19
5.1 Architecture . . . 19
5.1.1 Server packages . . . 19
5.1.2 Application context . . . 20
5.1.3 Asynchronous execution . . . 21
5.1.4 Handling GWT RPC calls . . . 22
5.1.5 Handling GWT HttpSession . . . 22
5.1.6 Serving standard web pages using Spring MVC . . . 22
5.1.7 Package `shared` . . . 22
5.1.8 Logging . . . 22
5.2 Domain layer . . . 23
5.2.1 Data sources and strategies . . . 23
5.2.2 Instantiation of proxy . . . 23
5.2.3 Choosing the right strategy . . . 23
5.2.4 Motivation behind proxies . . . 24
5.2.5 Persistence . . . 24
5.2.6 JPA . . . 24
5.2.7 Sending JPA entities through GWT RPC . . . 25
5.2.8 H2 database . . . 26
5.2.9 Building KOSapi query . . . 26
5.2.10 Working with atom feed . . . 26
5.2.11 JAXB unmarshalling . . . 26
5.3 Persentation layer . . . 26
5.3.1 GWT . . . 27
5.3.2 Advantages . . . 27
5.3.3 GWT module architecture . . . 28
5.3.4 User interface . . . 28
5.3.5 Communication with server . . . 28
5.3.6 Showing progress of long running task . . . 30
5.3.7 Using JPA annotations . . . 30
5.3.8 Bug workaround in enhanced compilation of JPA entities . . . 30
5.3.9 Code sharing specificities . . . 31
5.3.10 Solving problems with GWT . . . 31
5.3.11 Thymeleaf . . . 31
5.3.12 Configuration with Spring . . . 32
5.3.13 Bootstrap . . . 32

6 Application **33**
6.1 Finding when students have free time 33
6.1.1 Finding when room is free of lecture 34
6.1.2 Checking unusual study plans 35
6.1.3 Watching for collisions 36

7 Programmer's guide **37**
7.1 Development 37
7.1.1 Obtaining the source code 37
7.1.2 Importing project to IDE 37
7.1.3 Productivity tips 37
7.1.4 Code style 38
7.1.5 Running the application with jetty 38
7.1.6 Speeding up jetty startup 38
7.1.7 Testing with JUnit 38
7.2 Build 39
7.2.1 Makefile 39
7.3 Production requirements 39
7.4 Development requirements 39
7.5 Instalation for development 40

8 Conclusion **41**
8.1 Possible extensions of application 41
8.1.1 Finding empty room in examination time 41
8.1.2 Student's schedule assistant 41
8.1.3 Exam collision alerts 42

List of Figures

2.1 List of study fields 5
2.2 List of courses in single study field 5
2.3 Example timetable of course APO. Hours 3 and 4 was omitted due to the length of image. 6
2.4 Example of support app output 7
2.5 List of rooms 8

3.1 Student filter component 11
3.2 Timetable component 12
3.3 Selection (higlighted with blue color) shows courses and students that has lecture in last two hours of lessons. 13
3.4 Empty rooms 13
3.5 Example of minor record 14

4.1 State diagram showing possible states of course enrollment 16

5.1 Deployment diagram showing structure of project 20
5.2 Activity diagram showing strategy decision algorithm 24
5.3 Using GWT rpc 29

6.1 TimetableSum diagram 34
6.2 Organization of minor listing data 36

Chapter 1

Introduction

1.1 About this work

This work describes web application that finds when students have free time, what rooms are empty and which minor[1] study fields have students of Open Informatics finished. The application is loading data from information system of Czech Technical University in Prague, creates data aggregations and serves them to user in an interactive form. Application loads only the subset of data that belongs to Faculty of Electrical Engineering and uses KOSapi [] web service to obtain the data.

1.2 How to read this work

Work is divided into nine chapters.

Users of the application (faculty clerks) should read chapters 2 and 3 to get an overview of the application and information on how to use it. Programmers that want to implement their own client for KOSapi should read chapters 4 and 6. Programmers that want to extend the application that is described by this work should read chapters 6, 5 and 7.

- Chapter 2, describes what problems the application solves and discuss contribution of this work.
- Chapter 3 serves as user guide, with screen shots of application and possible scenarios of usage.
- Chapter 4 describes used data source and provides useful tips and recommendations for using it in readers own work.
- Chapter 5 provides details on implementation of application. This chapter is recommended for everyone who would like to do changes to application.

[1]Minor is optional study field that is printed on diploma

- Chapter 6 describes how were problems solved.
- Chapter 7 serves as programmer's guide and describes deployment procedure, required toolset and hardware.
- Chapter 8 provides summary on completed work and discusses possible extensions of application.

Chapter 2

Problem analysis

2.0.1 User

Users are administrative workers of faculty, teachers or students. Users are logged in based on FELID[1] authentication, that is provided by server (authentication is not concern of application).

2.0.2 Timer

There are some tasks that are not invoked by user, but by timed event. Such tasks can be re-fetching of data or sending email alerts.

2.1 Requirements

2.1.1 Functional requirements

See chapter 2 to see what problems application solves. Only formal definition of requirements follows.

1. App will allow user to find when specific group of students has free time
   (a) App will allow user to specify group of students by grade, study programme or study stage
   (b) App will allow user to specify group of students by set of courses or their parallels
2. App will allow user to find what rooms are free of lectures by specified time and filter
   (a) App will allow user to filter out rooms by type, department and capacity
3. App will allow to user to list students that have completed minor specialization
   (a) App will allow to filter out students by grade
4. App allows user to control the caching mechanism

[1]FELID is authentication similar to OAuth

2.1.2 Non-functional requirements

1. Application provides feedback for long running operations (such as fetching for kosapi)
2. Application provides fast response for cached data
3. Application can be accesses only by authenticated users

2.2 Showing timetable of multiple students

The application shows timetable of given set of students in single table, so that user can quickly find suitable time for happening or rescheduling a lecture. The user has two options of specifying set of users:

1. By student programme, grade and study type
2. By courses or parallels

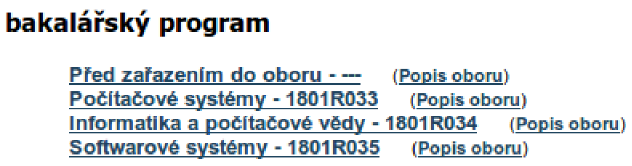
Finding the common schedule of students of given programme and grade allows to plan a meeting for many students. Tough students of same programme have some common courses, their schedule is very diverse and the application must take into an account schedules of all given students. There are some periods in academic year when schedules are being changed very often, so the application must allow to fetch data directly from KOSapi instead of using caching mechanism.

2.2.1 Existing solutions

Finding when students have free time is non-trivial task. Timetables of individual students are very diverse as any student can enroll voluntary course and pick what parallel of lecture he will attend.

For an example, teacher wants to plan meeting for students of second grade that study Open Informatics in bachelor cycle. Teacher can ask students personally during the lecture. However, only few would answer and students that are not present would not have an opportunity to comment. Using online service such as `http://doodle.com` that allows to propose terms and let students to pick the ones that best suits them is better than asking them directly, but not every student is going to give the teacher feedback and the teacher is forced to create a lot of possible terms because he doesn't know when his students are busy.

Another possibility is going to faculty web `http://fel.cvut.cz`, and navigate through links "Studium", "Studijní plány", "Bakalářské - Otevřená informatika", "prezenční" to receive list of possible study fields.



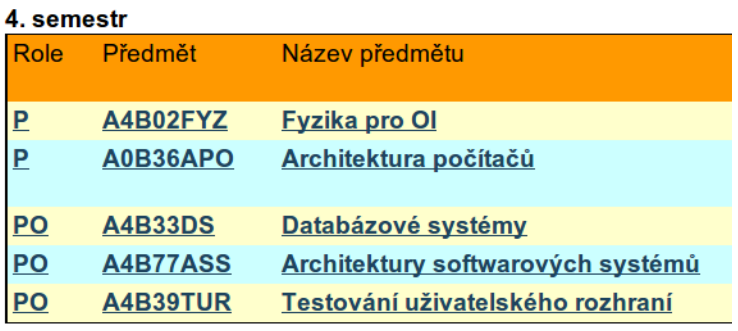
bakalářský program

Před zařazením do oboru - ---	(Popis oboru)
Počítačové systémy - 1801R033	(Popis oboru)
Informatika a počítačové vědy - 1801R034	(Popis oboru)
Softwarové systémy - 1801R035	(Popis oboru)

Figure 2.1: List of study fields

Now the teacher needs to open each study field in the new tab. Per each tab, he needs to do following actions:

1. Choose a language
2. Locate semester he is interested in
3. Compare all three tables to get all mandatory courses of the students
4. Open descriptions of courses in new tabs



4. semestr

Role	Předmět	Název předmětu
P	A4B02FYZ	Fyzika pro OI
P	A0B36APO	Architektura počítačů
PO	A4B33DS	Databázové systémy
PO	A4B77ASS	Architektury softwarových systémů
PO	A4B39TUR	Testování uživatelského rozhraní

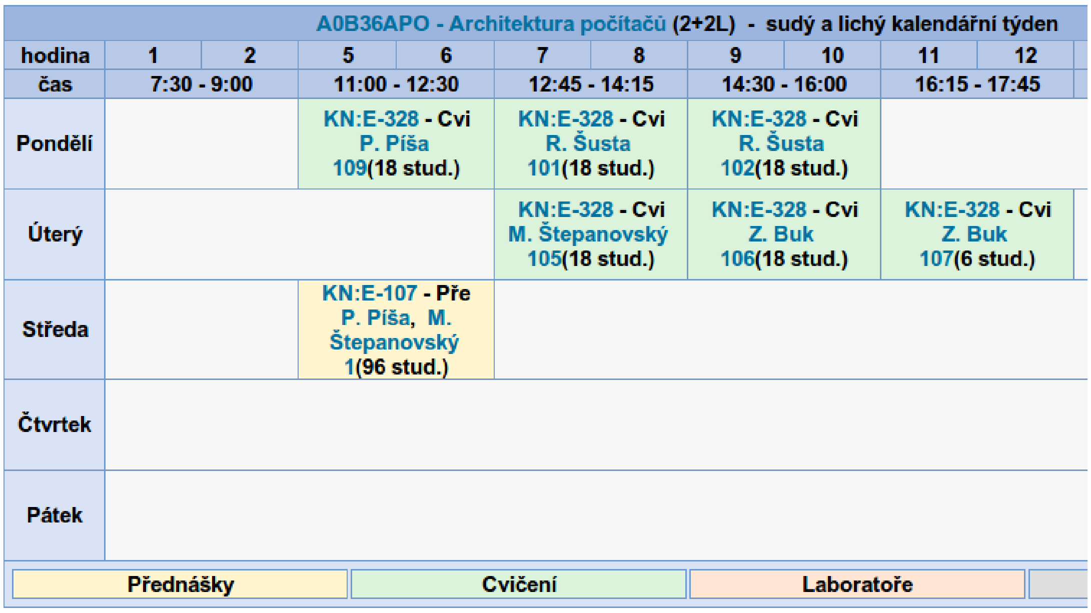
Figure 2.2: List of courses in single study field

There is nine courses in study plan of second grade students of bachelor Open Informatics. For each tab with the course, teacher must click on the code of the course to get to the timetable of the course. Timetables are located at the address `http://www.fel.cvut.cz/cz/education/rozvrhy-ng.B132/public/cz/` where `B132` is code of semester one is interested in. At last, it is now possible to visually compare nine different timetables.

It's good to point out, that such procedure does not yield precise result, as tables of courses are missing:

1. Courses that students enrolled beyond the scope of study programme
2. Courses that students enrolled second time

Moreover, such procedure cannot be applied to summer semesters of last grades as their schedule is not dependent on the study plan.



A0B36APO - Architektura počítačů (2+2L) - sudý a lichý kalendářní týden

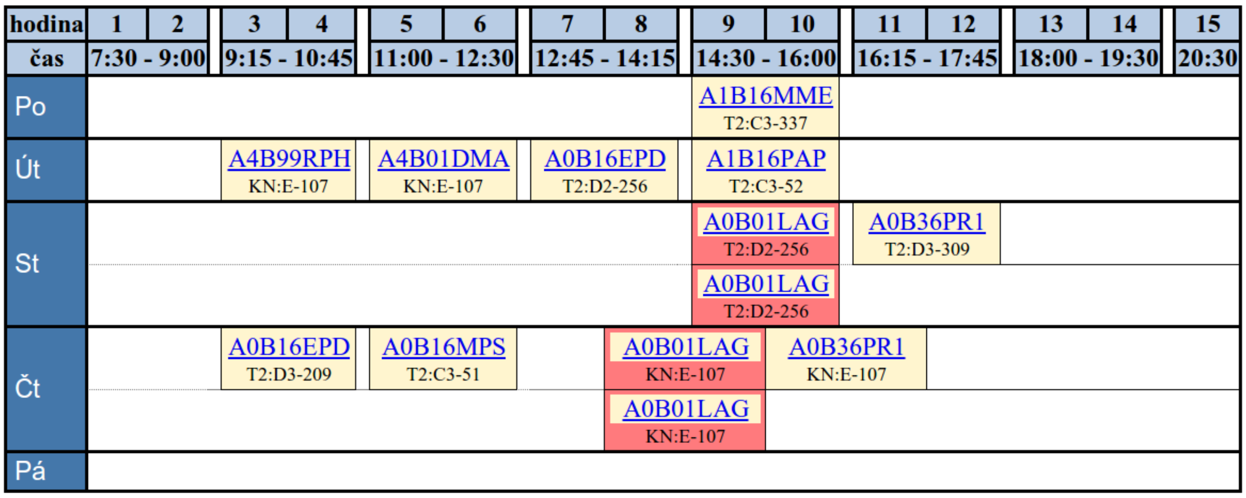
hodina	1	2	5	6	7	8	9	10	11	12
čas	7:30 - 9:00		11:00 - 12:30		12:45 - 14:15		14:30 - 16:00		16:15 - 17:45	
Pondělí			KN:E-328 - Cvi P. Píša 109(18 stud.)		KN:E-328 - Cvi R. Šusta 101(18 stud.)		KN:E-328 - Cvi R. Šusta 102(18 stud.)			
Úterý					KN:E-328 - Cvi M. Štepanovský 105(18 stud.)		KN:E-328 - Cvi Z. Buk 106(18 stud.)		KN:E-328 - Cvi Z. Buk 107(6 stud.)	
Středa			KN:E-107 - Pře P. Píša, M. Štepanovský 1(96 stud.)							
Čtvrtek										
Pátek										

Přednášky	Cvičení	Laboratoře

Figure 2.3: Example timetable of course APO. Hours 3 and 4 was omitted due to the length of image.

Using existing support application

Support application on address https://cw.felk.cvut.cz/support/rozvrhy_predmetu/index.php is capable of showing timetable of multiple courses given by codes. There are useful predefined course groups that can be found on course ware site https://cw.felk.cvut.cz by following navigation: "misc", "radaoi", "ruzne", "rozvrhy_minoru". Main disadvantages are that the data does not show voluntarily enrolled courses.



hodina	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
čas	7:30 - 9:00		9:15 - 10:45		11:00 - 12:30		12:45 - 14:15		14:30 - 16:00		16:15 - 17:45		18:00 - 19:30		20:30
Po									A1B16MME T2:C3-337						
Út			A4B99RPH KN:E-107		A4B01DMA KN:E-107		A0B16EPD T2:D2-256		A1B16PAP T2:C3-52						
St									A0B01LAG T2:D2-256		A0B36PR1 T2:D3-309				
									A0B01LAG T2:D2-256						
Čt			A0B16EPD T2:D3-209		A0B16MPS T2:C3-51			A0B01LAG KN:E-107		A0B36PR1 KN:E-107					
								A0B01LAG KN:E-107							
Pá															

Figure 2.4: Example of support app output

2.2.2 Implemented solution

Application completely automates the process of going throuh schedules. User specifies group that he is interested in and receives single comprehensible table that shows when students are busy and what courses they attend.Instead of long comparison of multiple timetables, user has now possibility to get valuable information using just few clicks. User can navigate to description of course or record of user in user database[2]. After submiting request, url of web application in browser is appended with serialization data of user input so that user can share his url to other user.

2.3 Finding an empty room

The application allows to find an empty room based on search criteria:

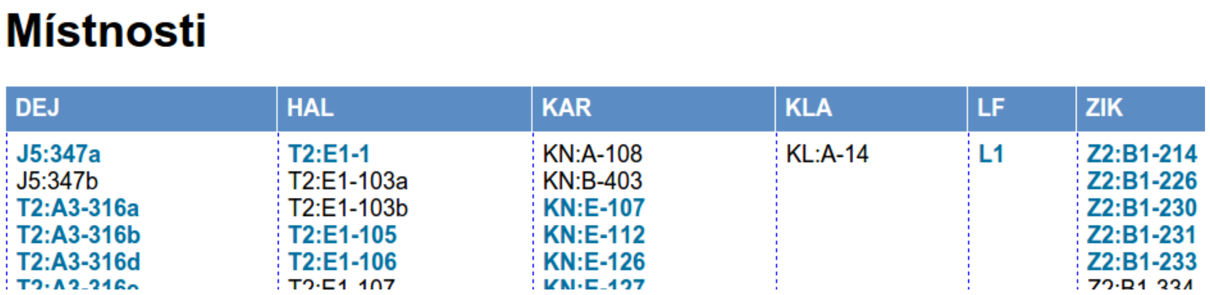
1. Time (lecture hours)
2. Minimal capacity
3. Locality
4. Type of room
5. Division

Application lists rooms in table and allows to display schedule of individual rooms.

[2]`https://udb.fel.cvut.cz/` provides basic information about student and links schedules of active semesters

2.3.1 Existing solution

The teacher can ask administrative clerk of his division to get list of possible rooms. He would then check timetable of each room using the list of rooms http://www.fel.cvut.cz/cz/education/rozvrhy-ng.B132/public/cz/mistnosti/index.html.


Místnosti

DEJ	HAL	KAR	KLA	LF	ZIK
J5:347a	T2:E1-1	KN:A-108	KL:A-14	L1	Z2:B1-214
J5:347b	T2:E1-103a	KN:B-403			Z2:B1-226
T2:A3-316a	T2:E1-103b	KN:E-107			Z2:B1-230
T2:A3-316b	T2:E1-105	KN:E-112			Z2:B1-231
T2:A3-316d	T2:E1-106	KN:E-126			Z2:B1-233
T2:A3-316e	T2:E1-107	KN:E-127			Z2:B1-234

Figure 2.5: List of rooms

The teacher must go though room schedules one by one. Moreover, existing timetable application does not provide any information on capacity or type of room.

2.3.2 Implemented solution

The user can pick suitable time by clicking on timetable that displays when students have time, define restrictions as capacity, type of room and location and by clicking a single button get list of all rooms, that met criteria and doesn't have scheduled lecture.

2.4 Check of unusual study plans

Study programme Open informatics offers possibility to graduate minor field alongside the major. The list of all minors can be found on []. Minor is accredited after completion of defined set of courses. The Application displays list of all students that have finished minor, showing what courses they have finished in what semesters. The user can display unfinished minors to see what courses have students not completed or not enrolled at all. There is no existing solution, apart from asking students what minors have they completed.

2.5 Detecting collisions

Planning timetable is hard task, so timetable is changed only if necessary. That is, when teacher does not have time, there is new course and so on. Rescheduling parallel requires knowledge of students timetables as described in section 2.2.

Collisions are being watched in defined groups. When lectures of two or more courses of given group overlaps, it is consiedered as a collision. When lecture has only one parallel of excercise to enroll, it is also checked for collisions. Course groups that are being watched for collisions are defined in the same way as minors.

2.5.1 Existing solution

Existing support application [] allows to display timetable of given group of courses. It highlights detected collisions of parallels, allowing administrative clerks to reschedule in a way that does not causes collision.

Shortcomings of existing application

1. Application loads data only few times a day, leaving clerks to wait to see feedback to their change
2. Application does not show voluntarily enrolled courses
3. Clerks must check application by themselves

2.5.2 Implemented solution

User specifies course groups to watch and groups of users that watches given course groups. User group is specified by set of emails. Application send emails to specified email addresses whenever collision in their watched group occurs. Such approach allows to do more extensive checks, as they are performed automatically, typacally once a day.

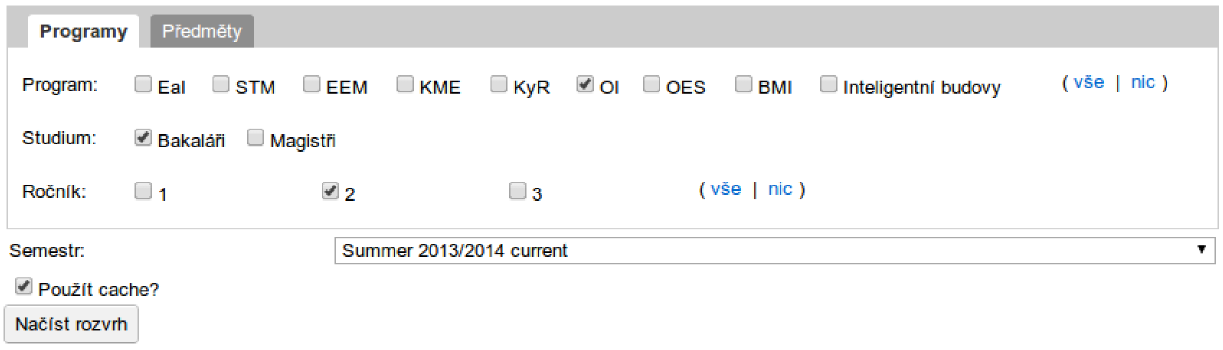
Chapter 3

User guide

The web application is located on address http://cwtest.felk.cvut.cz:8080/rozvrhy_studentu/. Open address in your Internet browser to use the application. It's recommended to use a current Internet browser. The application has navigation menu located at the top of application that user can use to navigate between sections.

3.1 Finding out when students have free time

At first, the user should specify which students he wants to analyze, using the top most form:



Program: Eal, STM, EEM, KME, KyR, [✓ OI], OES, BMI, Inteligentní budovy (vše | nic)

Studium: [✓ Bakaláři], Magistři

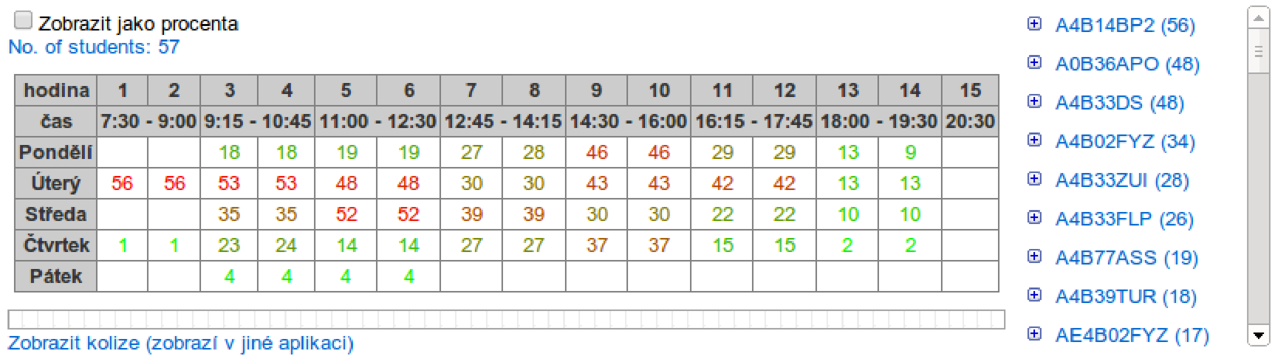
Ročník: 1, [✓ 2], 3 (vše | nic)

Semestr: Summer 2013/2014 current

[✓ Použít cache?]

Figure 3.1: Student filter component

It is possible to choose multiple study programmes, study types and grades. Current semester is always chosen as default, but user can choose different one if he wants to. All the data are loaded from cache (if present in cache) by default. To load most current data, uncheck "Použít cache" button. Application will load current data from server. This operation might take few minutes so progress is displayed and refreshed every second. Once data are loaded, they are shown in a table below.



☐ Zobrazit jako procenta

No. of students: 57

hodina	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
čas	7:30 - 9:00		9:15 - 10:45		11:00 - 12:30		12:45 - 14:15		14:30 - 16:00		16:15 - 17:45		18:00 - 19:30		20:30
Pondělí			18	18	19	19	27	28	46	46	29	29	13	9	
Úterý	56	56	53	53	48	48	30	30	43	43	42	42	13	13	
Středa			35	35	52	52	39	39	30	30	22	22	10	10	
Čtvrtek	1	1	23	24	14	14	27	27	37	37	15	15	2	2	
Pátek			4	4	4	4									

Zobrazit kolize (zobrazí v jiné aplikaci)

- ⊞ A4B14BP2 (56)
- ⊞ A0B36APO (48)
- ⊞ A4B33DS (48)
- ⊞ A4B02FYZ (34)
- ⊞ A4B33ZUI (28)
- ⊞ A4B33FLP (26)
- ⊞ A4B77ASS (19)
- ⊞ A4B39TUR (18)
- ⊞ AE4B02FYZ (17)

Figure 3.2: Timetable component

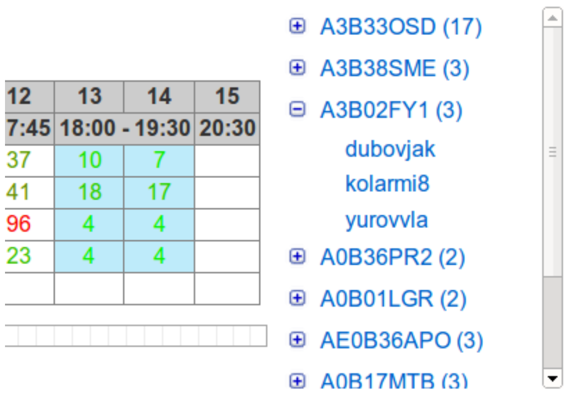
Table shows numbers in different colors. The more red the number is, the less students have time at that hour. There is a text above table that shows total count of students that are in the specified group. Each cell of table represents one lecture hour. The number represents count of students that are at school at given time.

There is a list of courses located at the right of the table. List is sorted by student count. Click on the item to go to description of course that is located on `http://fel.cvut.cz`. Hover with mouse over the link to see tooltip with full name of course. Click on the plus icon to see list of students that are enrolled in given course.

You can refine the list by selecting on cells in the table. To select a cell, simply click on it. Cell will change its color from white to light blue to provide feedback. You can select as many cells you want. List of courses will show courses that take place only at time represented by selected cells (see figure 3.3). To show all courses again, unselect cells or click on the count of all students.

3.2 Finding free room

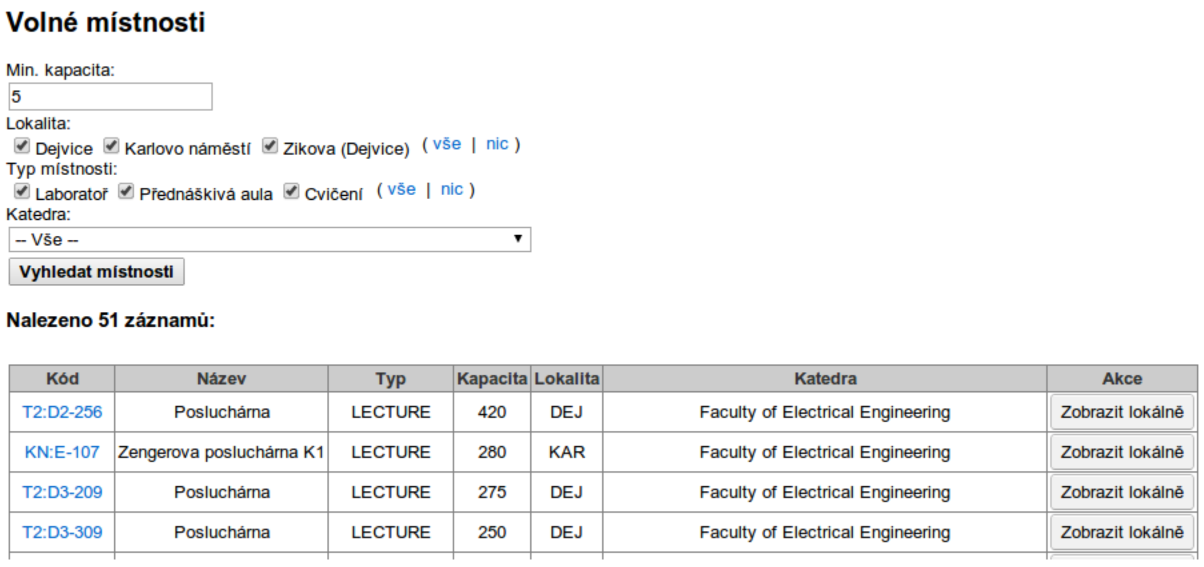
Provide minimal needed capacity, location and type of room by filling the form. Specific division can be selected from drop down list to restrict returned list only to rooms that belong to given division. Clicking the button "Vyhledat místnosti" shows the list of rooms. Note that some rooms has capacity set to 5 while they can provide place for up to 20 students. The user can click on code of room to navigate on schedule web page.



12	13	14	15
7:45	18:00 - 19:30		20:30
37	10	7	
41	18	17	
96	4	4	
23	4	4	

- A3B33OSD (17)
- A3B38SME (3)
- A3B02FY1 (3)
  - dubovjak
  - kolarmi8
  - yurovvla
- A0B36PR2 (2)
- A0B01LGR (2)
- AE0B36APO (3)
- A0B17MTB (3)

Figure 3.3: Selection (higlighted with blue color) shows courses and students that has lecture in last two hours of lessons.



Volné místnosti

Min. kapacita: 5
Lokalita: ✓ Dejvice ✓ Karlovo náměstí ✓ Zikova (Dejvice) (vše | nic)
Typ místnosti: ✓ Laboratoř ✓ Přednáškivá aula ✓ Cvičení (vše | nic)
Katedra: -- Vše --
Vyhledat místnosti

Nalezeno 51 záznamů:

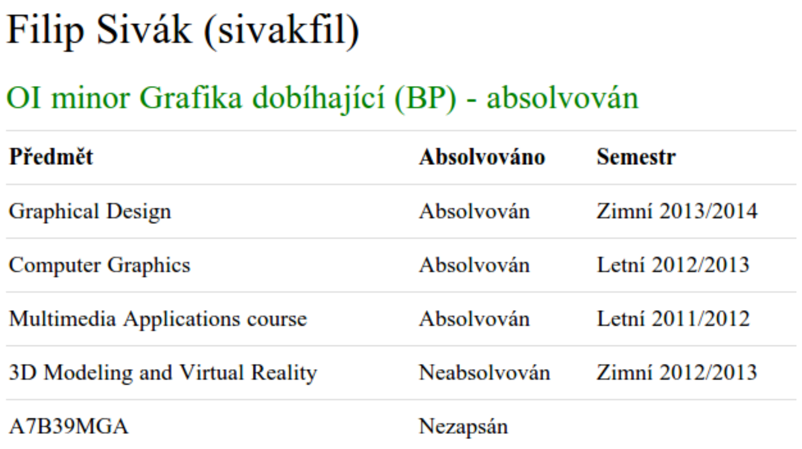
Kód	Název	Typ	Kapacita	Lokalita	Katedra	Akce
T2:D2-256	Posluchárna	LECTURE	420	DEJ	Faculty of Electrical Engineering	Zobrazit lokálně
KN:E-107	Zengerova posluchárna K1	LECTURE	280	KAR	Faculty of Electrical Engineering	Zobrazit lokálně
T2:D3-209	Posluchárna	LECTURE	275	DEJ	Faculty of Electrical Engineering	Zobrazit lokálně
T2:D3-309	Posluchárna	LECTURE	250	DEJ	Faculty of Electrical Engineering	Zobrazit lokálně

Figure 3.4: Empty rooms

3.3 List of minors of students

Click on the “Minory” link located at the horizontal top menu. The list of students that have completed minor is shown. Each student has shown a list of minors he has completed. Click on the link “Zobrazit nedokončené minory” to see all students that have completed or enrolled a course that is a part of a minor. The list bellow shows three possible states of minor course. Note that students that has some unfinished courses in current semester was not examined yet and can complete minor at the end of semester.

1. Finished (absolvován) - course was finished
2. Unfinished (neabsolvován) - course was not yet finished, or was failed
3. Not enrolled (nezapsán) - student did not enroll this course



Filip Sivák (sivakfil)

OI minor Grafika dobíhající (BP) - absolvován

Předmět	Absolvováno	Semestr
Graphical Design	Absolvován	Zimní 2013/2014
Computer Graphics	Absolvován	Letní 2012/2013
Multimedia Applications course	Absolvován	Letní 2011/2012
3D Modeling and Virtual Reality	Neabsolvován	Zimní 2012/2013
A7B39MGA	Nezapsán	

Figure 3.5: Example of minor record

3.3.1 Definition of minor in application

Minors are defined by administrators of application by providing XML configuration files. The files are located at `src/main/webapp/data/coursegroups` directory and can have any name. The application needs to be restarted in order to reload configuration (subjected to change).

```xml
<CourseGroup isMinor="true" minimalCount="3" name="OI minor Grafika dobj (BP)">
    <CourseCode required="true">A7B39PGR</CourseCode>
    <CourseCode>A7B39MVR</CourseCode>
    <CourseCode>A7B39KMA</CourseCode>
    <CourseCode>A7B39MGA</CourseCode>
    <CourseCode>A7B39GRT</CourseCode>
</CourseGroup>
```

Listing 1: Usage of import tag in application context xml configuration.

`CourseGroup` must have attribute `isMinor` set to `true` in order to be considered a minor by the application. Minimal count denotes how many courses from the list must be completed in order to finish a minor. Additional attribute `required` can specify course that is mandatory, thus finishing courses “A7B39KMA”, “A7B39MGA” and “A7B39GRT” does not complete shown minor.

Chapter 4

KOSapi

KOSapi `https://kosapi.fit.cvut.cz/projects/kosapi/wiki` is REST web service written as a interface for KOS information system. It contains information about students, their schedules, enrolled courses, registered exams and more. Kosapi uses XML as it's transfer format. Each item is wrapped in atom feed, which is distinguished by XML namespace `atom`.

4.1 Extracting interesting data from KOSapi

4.1.1 The list of students of particular programme or branch

Students can be filtered by grades, study form, programme or branch. Here is an example of students of first and second grade of programme Open Informatics, that has code `BP4`. Note that supplying parameter specifying study stage (bachelor or magister) is not needed, because bachelor and magister programmes has different codes.
`studentssem=B132&faculty=13000&limit=1000&query=studyForm==FULLTIME;studyState==ACTIVE;(grade==1,grade==2);programme==BP4`

4.1.2 Timetable of student

To obtain timetable of single student run query: `https://kosapi.feld.cvut.cz/api/3/students/sivakfil/parallels?limit=1000` Note TimetableSlot field, that has information about when given parallel takes place. There might be more than one TimetableSlots per parallel!

4.1.3 Completed courses of student

There is a resource in KOSapi called `enrolledCourses`, that allows to find out, what courses have student enrolled and completed. Moreover, using knowledge about processes in university, one might find out whether student have received assessment or attended exam. See the figure 4.1.

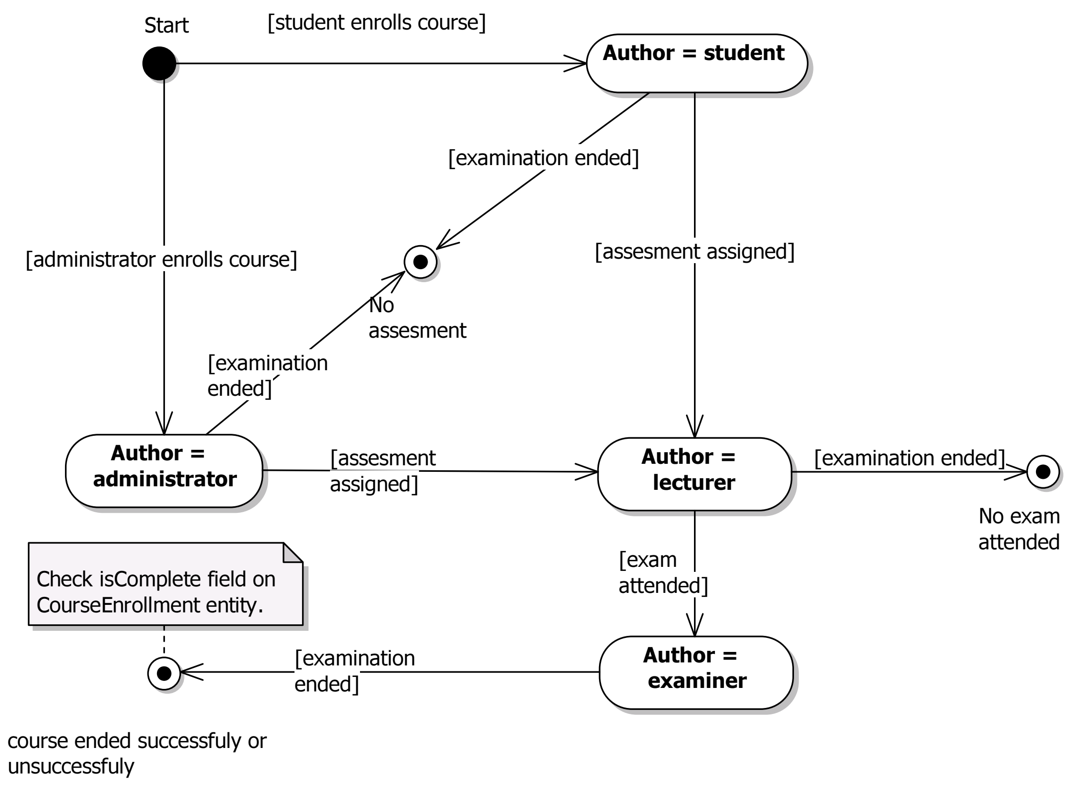


Figure 4.1: State diagram showing possible states of course enrollment

Each KOSapi atom entry has field `author`. That field changes every time when someone alters an entity. Based on author and time, one can deduce what state is course enrollment in. Note that as opposed to parallels, with enrolledCourses, information about what courses was enrolled before timetable was set is availabe. Data source is therefore usable even in pre-enrollment ("předzápis") stage.

4.2 Limitations

4.2.1 Absence of join

REST does not allow to join tables. For example: when fetching students and their parallels, it is impossible to fetch all students and their parallels at once. That results in fetching each student's parallel list separately, which takes a long time. One might accelerate fetching by firing requests asynchronously. Note that this is not possible in PHP. Possible solution would be to allow feeds to return just the relationships between the students and the parallels.

4.2.2 Fetching data for all semesters at once

Data can be fetched only for one concrete semester. It would be great if one could pass special case `all` to `sem` parameter[1] and then retrieve all the data. User of KOSapi would still be able to differentiate semesters in data, as data contains `semester` field.

4.2.3 Support for caching

Some data operations require having all the data stored locally (e.g. in database). For example, when checking what rooms are free to use in certain time, algorithm must go through timetable of all rooms. There are periods of year, when timetables are changed and data must be refetched frequently. Instead of refetching timetables of all rooms, it would be nice to ask KOSapi only of the changed ones. An example url is https://kosapi.feld.cvut.cz/api/3/parallels?updatedAfter=1399733293, where value of `updatedAfter` parameter is unix timestamp. Since KOSapi internally stores all the data in SQL database, it would not be hard to implement.

4.2.4 No `last-modified` header

It would be helpful to provide `last-modified` header to every response. Atom feed has currently field `updated` that corresponds to time, when was feed assembled (i.e. time of request). More helpful would be to assign to updated field the date of oldest entity in the feed. Same date could be supplied to `last-modified` header, so that `HTTP HEAD`[2] [] request could be performed to check availability of new data.

4.2.5 CORS is disabled

Cross-Origin Resource Sharing allows JavaScript clients to perform request to other domains that would otherwise be blocked by browsers due to the security. Internet browsers block requests to other domains, so that potential attacker would not access session of an user. Since KOSapi is web service and there are no security risks, it should provide header `Access-Control-Allow-Origin` with value of '*' (i.e. allow all). Reader can read more in [].

4.3 Problems

There are hidden problems that might surprise programmers of clients. Being aware of those might help one to implement his own client.

4.3.1 Inaccessible data

Some data are not accessible by KOSapi. For example, special case of programme with code `9CIZI` serves as programme for students of other universities. Requesting KOSapi url https://kosapi.feld.cvut.cz/api/3/programmes/9CIZI results in HTTP 500 error

[1]sem is short for semester

[2]HEAD request is like GET request, but returns only header resulting in faster responses

with message: “Well, this doesn’t look good... There’s some stranger behind the door: class org.jboss.resteasy.spi.UnhandledException”. However requesting `https://kosapi.feld.cvut.cz/api/3/programmes?query=code==9CIZI` does not result in error and shows data. Fetching parallels of course `U3V FEL`[3] results in same error.

4.3.2 Non unique codes

Code `9CIZI` is used for multiple programmes. Query `https://kosapi.feld.cvut.cz/api/3/programmes?query=code==9CIZI` returns 7 records.

4.3.3 Missing pieces of data

Some fields are missing, resulting in meaningless data.

1. There are timetable entries without specified day
2. There are course enrollments without specified course

These could be a problem of KOS database itself, but developer must take them in account and never assume he will get all the fields.

4.3.4 Using atom:updated field does not speed up fetching process

KOSapi provides possibility of fetching only listed fields. One could write query: `https://kosapi.feld.cvut.cz/api/3/parallels?limit=500&fields=entry(id,updated)` to receive parallels only with `id` and `updated` field, filter out parallels that have not changed since the last fetching and request only updated parallels by their id. However, **retrieving only few fields is not faster then retrieving them all**. It is also not possible to fetch multiple parallels by their `id`, as `id` is not accessible by query and is passed directly in url.

[3] University of third age

Chapter 5

Implementation

The application is written as web application in Java Enterprise Edition (Spring framework), leveraging Hibernate (JPA 2) and JAXB libraries, GWT framework and Spring MVC for presentation, using Maven build system.

5.1 Architecture

1. `cz.cvut.felk.rozvrhy.server` - server side code
2. `cz.cvut.felk.rozvrhy.shared` - code that is shared between server and client
3. `cz.cvut.felk.rozvrhy.client` - code that is intended for GWT application

There are two applications in project:

1. Server - handles requests from user, makes calls to KOSapi and database
2. GWT application - runs in users browsers, takes input from user and makes calls to server

5.1.1 Server packages

1. `async` - support for asynchronous task execution
2. `config` - loading of xml configuration files
3. `dao` - fetching data from KOSapi and database
4. `engine` - services used to do operations on data
5. `gwt` - GWT RPC call handler and utils
6. `KOSapi` - services that serves direct communication to KOSapi

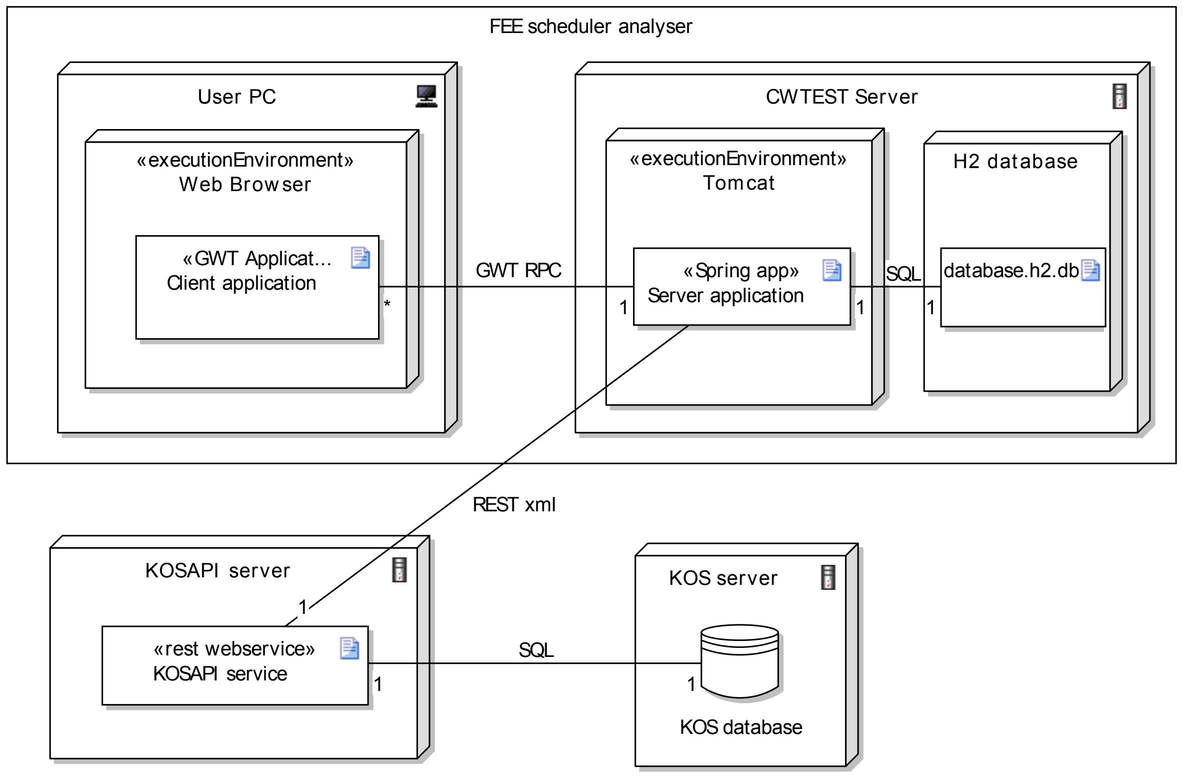


Figure 5.1: Deployment diagram showing structure of project

5.1.2 Application context

Application context is a central provider of configuration for Spring. There are two application contexts, one for production and one for testing, that is used by JUnit. Note that JUnit application context extends production context by using `import` tag.

```
<beans xmlns="...">
  <import resource="applicationContext.xml" />

  <!-- Mocking servlet context -->
  <bean class="org.springframework.mock.web.MockServletContext">
      <property name="contextPath" value="src/main/webapp/" />
  </bean>
</beans>
```

Listing 2: Usage of import tag in application context xml configuration.

5.1.3 Asynchronous execution

The native way
Java provides an elegant way to perform asynchronous execution by providing interface ExecutorService [] that allows to submit Runnable or Callable instances and get Future [] object. Future is `ValueHolder` [], that will have it's value accessible after submitted task (runnable or callable) is finished. Call `isDone()` method to check, whether task is finished or not. Calling `get()` causes blocking of execution until the task is done.

The Spring way
Spring provides convenient way of dealing with asynchronous tasks. Instead of submitting the task directly to executor, it is recommended to let the Spring to handle asynchronous execution on its own. Using own executor outside Spring's awareness might end up with running more executors at once, which would waste resources. To use Spring's task executor, firstly, one must declare task executor in application context:

```xml
<bean id="taskExecutor"
      class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
  <property name="corePoolSize" value="5" />
  <property name="maxPoolSize" value="10" />
</bean>
```

Listing 3: Declaring executor.

Secondly, one must write service that will be called asynchronously with methods annotated by `@Async` annotation. Those methods must return `Future<?>`. Finally, async service is injected into service that calls it.

```java
  @Async
  public Future<Timetable> refetchStudentAsync(String student, String semester) {
    List<Parallel> parallels = parallelKosapi.getTimetables(student, semester);
    return new AsyncResult<Timetable>(new Timetable(student, semester, parallels));
  }
```

Listing 4: Example of async method

Note that it is not possible to call async method directly. Injected instance must always be used, as Spring injects proxy that submits task to task executor [].

Getting progress of executing task
Executed task is basically collection of futures. Some of them might by unfinished, wile others might be already completed. Simply iterate through collection and ask which have been completed using method `isCompleted()`. Divide number of completed tasks by number of all tasks to get percentage.

5.1.4 Handling GWT RPC calls

GWT toolkit comes with RemoteServiceServlet servlet that is capable of handling GWT requests and serving responses. By extending this servlet, we can easily integrate it into our spring application. The extension is called Spring4Gwt and is located in `server.gwt` package.

Spring4Gwt class was based on Spring4Gwt [] project. List of changes that I have done:

1. Added HTTP session support
2. Exchanged commons logging with SLF4J
3. Better handling of exceptions

5.1.5 Handling GWT HttpSession

`RemoteServiceServlet` class that is provided by GWT contains method `getThreadLocalRequest()` that returns instance of `HttpServletRequest`. `HttpServletRequest` contains method named `getSession()` that returns `HttpSession`. Session is then stored into `ThreadLocal` variable that associates session to current thread that is associated to request. Class `GwtSessionValueHolder` is a value holder that associates itself to GwtSession. Any stored value can be restored within one session in different RPC calls.

5.1.6 Serving standard web pages using Spring MVC

Controllers are needed to serve web pages. The controller is specific service that will be delegated when HTTP request is captured and controller fits URL pattern. Controllers are situated in package `web`.

5.1.7 Package `shared`

The root of package contains transfer objects that are generated by application and are transfered over network to client.

1. collisions - JAXB entities and data structures describing course groups and collisions
2. kosapi.dto - entities that are fetched from KOSapi and persisted to database

5.1.8 Logging

SLF4J logging facade is used to log events on server. Log4j implementation is used to perform actual logging. SLF4J provides possibility to exchange Log4j for different logger, while not being forced to change code itself. Static factory `LoggerFactory.getLogger()` is used to obtain instance of a logger.

5.2 Domain layer

Domain layer of application is very specific as it has multiple data sources to get data from. Both data sources share same domain entities, stored in the package: `cz.cvut.felk.rozvrhy.shared.kosapi.dto`. Entities are accessible from GWT application (code is shared) and are transferable through network using GWT RPC. Each entity is mapped to itself using Dozer that removes lazy initialization collections that are not serializable. An alternative would be to hold separate dtos for transfer, but single class representation was chosen for better maintainability.

5.2.1 Data sources and strategies

There are two data sources:

1. Database - serves as cache for entities to allow fast operations over data
2. KOSapi - web service that is primary source of data

Data Access Objects (dao for short) are used to retrieve data from database. DAO is class that has exposed methods for retrieving or persisting data based on given parameters. Instances of DAO are provided by dependency injection using `@Resource` annotation. It is not possible to use `@Autowired` annotation, because proxied instance of DAO is needed.

```
@Resource(name="studentDao")
protected IStudentDao studentDao;
```

Listing 5: An example injection of proxied dao.

Dependency injection will inject an instance of a “Proxy”. Proxy will intercept every call and delegate to given `InvocationHandler` bringing dynamical approach into static world of Java.

5.2.2 Instantiation of proxy

Proxies are instantiated by DaoFactory service. `Proxy.newProxyInstance` method on `java.lang.reflect.Proxy` object is called to create proxy. Second parameter of newProxyInstance is MethodInvocator implementing object.

5.2.3 Choosing the right strategy

DaoStrategyInvocator object is responsible for choosing strategy. See the diagram 5.2.

All methods are delegated on concrete strategy, except for `setCache(boolean)` call, which is intercepted and not delegated. SetCache call serves to set the strategy. If cache is disabled, only KOSapi strategy is used.

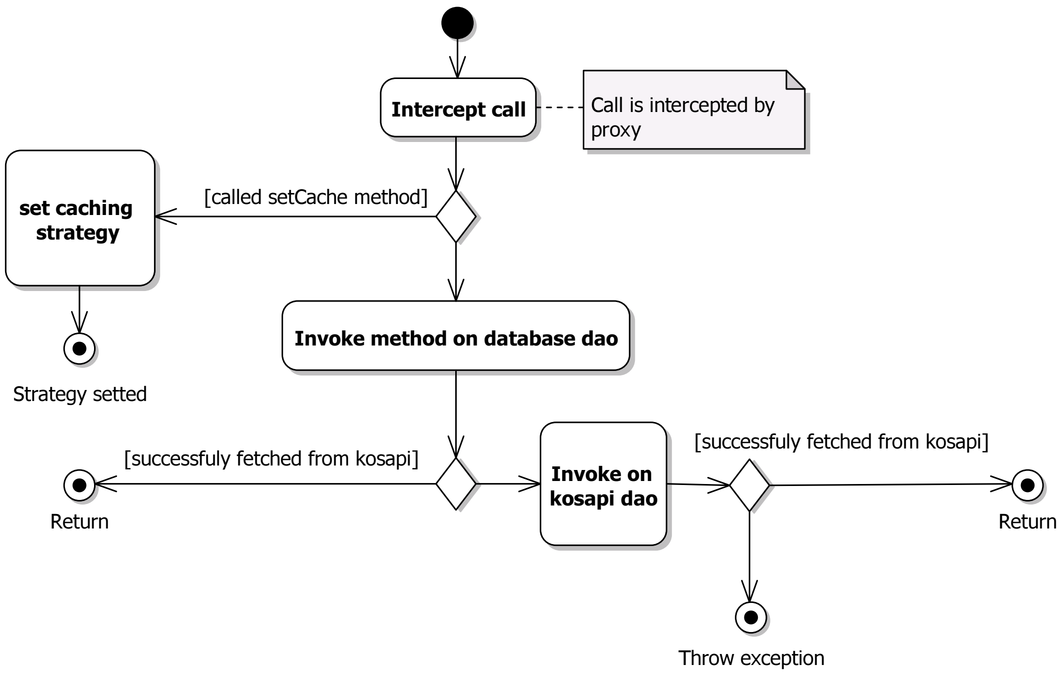

Intercept call
Call is intercepted by proxy
set caching strategy
[called setCache method]
Invoke method on database dao
Strategy setted
[successfuly fetched from kosapi]
Return
Invoke on kosapi dao
[successfuly fetched from kosapi]
Return
Throw exception

Figure 5.2: Activity diagram showing strategy decision algorithm

5.2.4 Motivation behind proxies

Using proxy with `InvocationHandler` allows to have strategy expressed on single place in application. Not using proxy would result into hardly maintainable code as any change to a strategy would require change in all DAO services!

5.2.5 Persistence

Database strategy uses typical persistence schema, where each database DAO extends BaseDAO class that has instance of persistence context (Entity manager)[1]

5.2.6 JPA

Java persistence api (JPA for short) is used to take care about communication with database and object relational mapping. Details about connection are written in persistence unit persistence.xml. Different persistence unit is used in production and different in development

[1]Entity manager provides direct contact to database and is part of JPA implementation provided by third party, in our case Hibernate

mode. Choice is made based on target run in makefile by copying the right persistence unit into META-INF folder.

5.2.7 Sending JPA entities through GWT RPC

JPA entities fetched from database are being proxied in order to provide transparent lazy initialization. Proxy is not serializable, moreover the data are not present in entity until proxy loads them (proxy loads data on getter call).

When GWT servlet receives object to send through GWT RPC, it needs to serialize it. Serializer walks through object, asking each getter to get him data. At that time, no hibernate session is opened, so LazyInitializationException occurs.

General solution is to use object mapper as Dozer and implement CustomFieldMapper to handle not initialized collections [].

```java
public class DozerCustomFieldMapper implements CustomFieldMapper
{
  public boolean mapField(Object source, Object destination, Object sourceFieldValue,
  ClassMap classMap, FieldMap fieldMapping)
    {
        // Check if field is derived from Persistent Collection
        if (!(sourceFieldValue instanceof AbstractPersistentCollection)) {
            // Allow dozer to map as normal
            return false;
        }

        // Check if field is already initialized
        if (((AbstractPersistentCollection) sourceFieldValue).wasInitialized()) {
            // Allow dozer to map as normal
            return false;
        }

        // Set destination to null, and tell dozer that the field is mapped
        destination = null;
        return true;
    }
}
```

Listing 6: Custom field mapper that nulls every non-initialized collection, so that entity can be serialized.

5.2.8 H2 database

H2 database is used as a database to back up persistence. It's embedded database similar to SQLite reader might be familiar with. H2 can run in three regimes:

1. File - used by application
2. In memory
3. Server

Selection of regime is made by specifying JDBC connection url. Application is using file regime, so that no additional interventions of server administrator are needed. H2 comes with own database management application, similar to PgAdmin (used with Postgresql) or PHPMyAdmin (used with MySQL). The management app can be found on H2 database website []. To connect to database file, use same connection url as is defined in persistence unit (but use absolute url). Example JDBC url: `jdbc:h2:/home/username/scheduleApp/database;LOCK_TIMEOUT=10000` Note `LOCK_TIMEOUT` variable that is required, because default lock timeout is too low and causes exception when database is under load. When connected to H2 database file, lock file is created. That means, it is not possible to open database while the application is using it and simultaneously look at the data.

5.2.9 Building KOSapi query

In order to fetch feed from KOSapi, query url needs to be built. For purposes of building complicated queries, `KosapiRequestBuilder` was implemented. KOSapi request builder incorporates Builder [] and Fluent interface [] patterns to achieve comfortable work.

5.2.10 Working with atom feed

KOSapi encapsulates data in form of an atom feed. The atom feed is parsed as a plaintext and then JAXB unmarshalles content of `atom:content` element.

5.2.11 JAXB unmarshalling

JAXB is powerful XML parser that marshalls annotated POJO classes into XML and unmarshalles XML into POJO classes. Instance of JaxbContext is needed to get instance of marshaller/unmarshaller. JaxbContext is accessible by dependency injection. All KOSapi entities that are fetched extends abstract class XmlDto, that holds common data as `atomId`.

5.3 Persentation layer

Presentation layer is responsible for serving data outputs of application to the end user. Modern web applications are now being presented in interactive way. Especially when working with complicated GUI components as a timetable, client code is needed to provide nice and fluent user experience. Instead of creating Flash application, or Java applet, JavaScript language with usage of web browser API is being embraced. However, JavaScript is hard to learn language with it's pitfalls and specificities. There are currently (at the time of writing) several approaches how to simplify JavaScript development:

1. Using abstraction framework such as jQuery
2. Using language that compiles into JavaScript as CoffeScript
3. Using framework that introduces static typing using annotations as Google closure
4. Using tools such as Bower and Grunt to take care of dependency resolution and build
5. Using framework that compiles typed language into JavaScript as Google Web Toolkit (GWT for short)

The application uses both Spring MVC and GWT as a tools of presentation layer. GWT is used for search of student group free time and lecture free rooms. Spring MVC is used in non-interactive reports such as listing students that have completed minor specializations.

5.3.1 GWT

GWT allows to build web application using Java language by compiling Java code into JavaSript that is run on users web browser. It compiles different code for each browser using technique named "deffered binding". GWT is framework developed by Google aimed at Java developers that are familiar with Java environment and know only little about web development.

5.3.2 Advantages

1. Uses typed language for development
2. Allows to share Java code across client and server
3. Programmer can use IDE that he is already familiar with from Java environment
4. IDE can provide autocompletion and type hinting
5. Programmer can use data structures he is already familiar with
6. Programmer can easily automatically refactor code across big codebase
7. GWT provides wast collection of widgets with good extendability support that are very similar to swing components

Disadvantages

1. GWT compilation and development mode startup are painfully slow
2. GWT does not emulate all classes (for example GWT is missing Calendar class)
3. GWT does not have reflection
4. GWT has relatively big learning curve, as some problems are hard to debug

5.3.3 GWT module architecture

GWT applications are divided into modules that can be run individually. Module can be considered to be a single page application. Modules can also serve as a libraries, as one module can extend another. Modules are defined by XML files, that are placed in the root of modules package. Application has single module named `GWT`, that can be found the in package `cz.cvut.felk.rozvrhy.client`. Module has single entry point class named `Entry` (analogy in Java Main class could be found). Entry class is invoked after web page is opened. Once all files are loaded, method `onModuleLoad()` is fired on entry point class.

GWT applications are similar to Swing applications. They use panels and widgets to show GUI, that are translated into HTML. For example, `VerticalPanel` is rendered as `DIV` element. `TextBox` is rendered by appending input with type text into page.

5.3.4 User interface

There are two general ways of creating user interface in GWT. The first is using form designer that comes with GWT plugin for eclipse. GWT Designer [] takes form designers one step further as it successfully tries to understand code. Changing anything in code results in change in designer, which will render components as is expected. On the other hand, GWT designer is slow and clunky. The second way of describing user interface is to use a UiBinder []. In UiBinder, user interface structure is defined in form of XML as would similarly in HTML. Behavior is defined in UiBinder class that is interlinked to the XML file. All components are written in UiBinder, except for `StudentFilterComponent`, that was designed in GWT Designer. To open component in GWT Designer, click on the class with right button and choose GWT Designer in “open with” context submenu.

5.3.5 Communication with server

When the application starts, module needs to load list of semesters from server, so that user can select valid semester. GWT uses own serialization protocol called GWT RPC. GWT RPC uses standard HTTP communication, but encodes data in a special way, respecting structure of data and keeping Java references.

In order to use GWT RPC, following must be defined:

1. Remote interface
2. Asynchronous remote interface
3. Server implementation of remote interface
4. Client implementation of async. remote interface

Sent data must meet following conditions:

1. All sent objects must implement `java.io.Serializable` or `IsSerializable` interface
2. All sent objects must have no args constructor
3. All thrown exceptions must extend java.lang.Exception (not RuntimeException!)

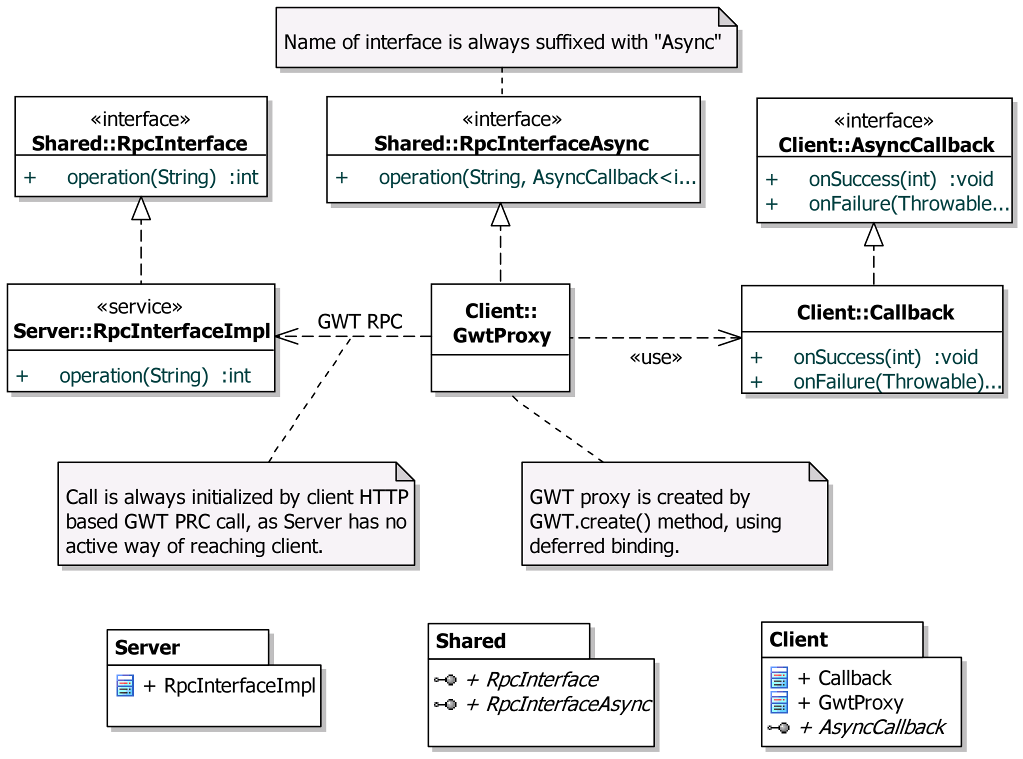


Figure 5.3: Using GWT rpc

5.3.6 Showing progress of long running task

Application allows user to reload all the data from KOSapi instead of reading them from potentially outdated cache. Such operation takes a lot of time and user would have no feedback on progress of operation.

Instead of responding after the fetch form KOSapi is done, server runs fetching asynchronously and responds immediately with `AsyncJobNotDoneException`. When obtaining such exception, client is obliged to repeat his request after some time (e.g. one second). If running job is still not finished, server will throw another `AsyncJobNotDoneException`, setting its percentage property to provide user continuous feedback. After server is finished with computation, client receives data and calls appropriate handler that process it.

Abstract class `client.util.RpcRequest` was written to automate the process. All that is need to do is to fill the template methods [].

5.3.7 Using JPA annotations

In order to use JPA annotations in entity classes, source code of annotations must be provided on classpath of development mode application and compiler. Sources can be downloaded as maven dependencies:

```xml
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-annotations</artifactId>
  <version>3.5.6-Final</version>
  <classifier>sources</classifier>
</dependency>

<dependency>
  <groupId>javax.persistence</groupId>
  <artifactId>persistence-api</artifactId>
  <version>1.0.2</version>
  <classifier>sources</classifier>
</dependency>
```

Listing 7: Maven dependencies of annotation sources.

All that is needed is to include `target/rozvrhy_studentu/WEB-INF/lib` folder on classpath.

5.3.8 Bug workaround in enhanced compilation of JPA entities

Class in GWT is considered enhanced, when containing `@Entity` annotation. Unfortunately, enhanced compilation causes bug in serialization [], so workaround had to be implemented. All JPA entities are annotated normally, except `@Entity` annotation. Entities are specified in `orm.xml` file, that specifies which classes are JPA entities. The file `orm.xml` is located alongside persistence context in META-INF directory.

5.3.9 Code sharing specificities

Not all classes from Java can be used in shared code, as not all of them are emulated in GWT. A good example are regular expressions, where server and client code uses completely different implementations and thus, regular expressions cannot be shared across the platforms.

5.3.10 Solving problems with GWT

When having problem with GWT, developer can do following actions:

1. Set the DEBUG log level
2. Clear gwt-unit cache
3. Restart browser, clear browser cache

5.3.11 Thymeleaf

Thymeleaf is Java library that serves as template engine, providing natural templates[2]. It can be easily integrated in Spring MVC and is very easy to learn.

```
<th:block th:unless="${collisions.isEmpty()}">
  <div th:each="collision : ${collisions}">
      Den: <span th:text="${collision.getDay()}">Pondl</span>
  </div>
  <hr />
</th:block>
```

Listing 8: Example of thymeleaf templating.

Note that thymeleaf requires that templates are valid XML documents and that it is possible to use syntetic element `th:block`, so that it is not required to insert redundant div just to use if statement. Thymeleaf does not contain layouting (template inheritance like Apache blocks or Facelets) out of the box, so “thymeleaf-layout-dialect” additional dialect is used.

[2]Natural template is template that will be correctly displayed in web browser if opened as file without application

5.3.12 Configuration with Spring

In order to use Thymeleaf, maven dependencies must be included in pom and `mvc-dispatcher.servlet.xml` configuration must include:

```xml
<!-- Thymeleaf template engine -->
<bean id="templateResolver"
  class="org.thymeleaf.templateresolver.ServletContextTemplateResolver">

  <!-- Folder where to find templates -->
  <property name="prefix" value="/WEB-INF/templates/" />
  <property name="suffix" value=".html" />
  <property name="templateMode" value="HTML5" />

  <!-- Allows templates to be automatically reloaded when modified (useful in dev mode) -->
  <!-- Set to true in production mode! -->
  <property name="cacheable" value="false" />
</bean>

<bean id="templateEngine" class="org.thymeleaf.spring3.SpringTemplateEngine">
  <property name="templateResolver" ref="templateResolver" />

  <!-- Add dialect to Thymeleaf -->
  <property name="additionalDialects">
    <set>
      <!-- Allows layouts -->
      <bean class="nz.net.ultraq.thymeleaf.LayoutDialect"/>
    </set>
  </property>
</bean>

<bean class="org.thymeleaf.spring3.view.ThymeleafViewResolver">
  <property name="templateEngine" ref="templateEngine" />

  <!-- Allow UTF-8 character encoding in order to add support for check language -->
  <property name="characterEncoding" value="UTF-8" />
</bean>
```

Listing 9: Thymeleaf-spring integration in configuration file mvc-servlet-dispatcher.xml

5.3.13 Bootstrap

Bootstrap CSS framework from http://getbootstrap.com/ was used in Spring MVC pages. Because bootstrap styles are not prefixed by default, bootstrap was recompiled in Less with added prefix []. Bootstrap styles are applied to childs of element with class `bootstrap`, which allows compatibility with GWT styles.

Chapter 6

Application

This section describes how was individual use cases implemented.

6.1 Finding when students have free time

User must specify set of students he wants to process. Set of students can be described as:

1. Combination of grade, programme and type of study (bachelor or magister)
2. Set of courses or parallels

The user uses `StudentFilterComponent` that is part of GWT application (figure 3.1). Once the user fills the data, `StudentFilterComponent` sends it's model `StudentFilter` to server through `GWT RPC` protocol to service named `TimetableRpcService` that implements `ITimetableRpcService` interface.

At first, the service fetches list of the students based on provided `StudentFilter` using `IStudentDao` data access object. `IStudentDao` is obtained by dependency injection in the form of a proxy object, that decides which strategy to use, based on caching settings stored in `StudentFilter` provided by client.

An example url of fetching students of first and second grade of bachelor programme Open Informatics, that has code "BP4": `https://kosapi.feld.cvut.cz/api/3/students/?sem=B132&faculty=13000&limit=1000&query=studyForm==FULLTIME;studyState==ACTIVE;(grade==1,grade==2);programme==BP4`, that fetches timetable is in KOSapi represented by `timetableSlot` field returned in `parallel` entity. An example of query obtaining timetable of student "sivakfil": `https://kosapi.feld.cvut.cz/api/3/students/sivakfil/parallels?limit=1000`. Since there is no way of obtaining timetable of given set of users, application must fetch timetables one by one for each user. Fetching asynchronously significantly improves performance.

Once all timetables are fetched in form of collection, the data are aggregated into structure that the client application can easily process and serve to the user. Such structure is called `TimetableSum`.

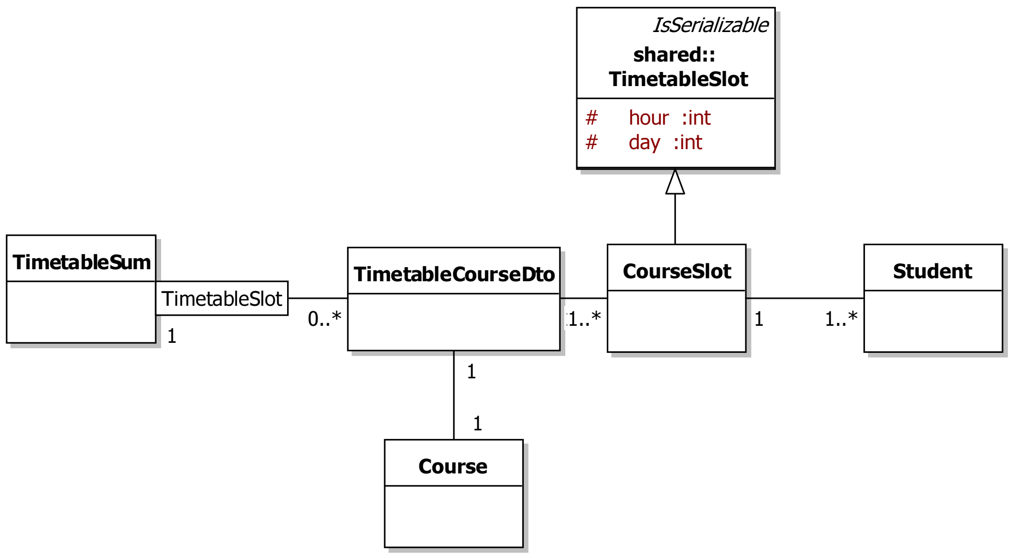


Figure 6.1: TimetableSum diagram

6.1.1 Finding when room is free of lecture

In order to find whether room is free of lecture (i.e. there is no lecture going on in the room in the given time), the application must have knowledge of timetables of all rooms. KOSapi provides information on rooms in feed `rooms` and room timetables in `parallels` (field `room` in `timetableSlot`). In order to get timetables of all rooms, the application must fetch all timetables.

The application stores timetable as collection of `TimetableUnit` entities. `TimetableUnit` is one lecture hour[1] described by hour, day, room and parent parallel entity. Such representation is possible due to discrete division of school day into hours and allows to write convenient queries that are fast. Because it's not possible to fetch all timetables of rooms from KOSapi at once, application fetches the data only from database.

Query joins Room table to itself, asking for all rooms except for those that has timetable unit for given hour. `JOIN FETCH` clause says to JPA to eagerly initialize given relation.

[1]Lecture hour is 45 minutes long

```
SELECT r FROM Room r
JOIN FETCH r.division

WHERE r.id NOT IN (
  SELECT r2.id FROM Room r2
  JOIN r2.timetableUnits tu
  WHERE ((tu.day-1) * :daySpan + tu.hour) IN (:hours)
)

AND r.division.code = :divisionCode
AND r.capacity >= :minCapacity
AND r.locality IN (:localities)
AND r.type IN (:types)
```

Listing 10: Query that finds empty rooms.

6.1.2 Checking unusual study plans

As minors are specific trait of Open Informatics programme, KOSapi has no knowledge about them. KOS itself does not provide support for minors yet. The minors are defined by XML files (see section 3.3.1), that are unmarshalled by JAXB and loaded by service: `cz.cvut.felk.rozvrhy.server.config.CourseFilterLoaderService`.

To determine whether student have finished minor, application must fetch information about completed courses for every semester that student was enrolled to. The required data are located in `enrolledCourses` feed. An example query is: `https://kosapi.feld.cvut.cz/api/3/students/sivakfil/enrolledCourses?sem=B132`. KOSapi allows to fetch only one semester per request, so application needs to load all semesters separately. Note that enrollments of previous years are not going to be changed, so it is pointless to fetch them again. Each course enrollment entity contains field “completed”, which tells information about whether the student successfully finished course or not. This information is used to determine whether the student have finished minor or not. Data are aggregated into map of Students to list of their minors.

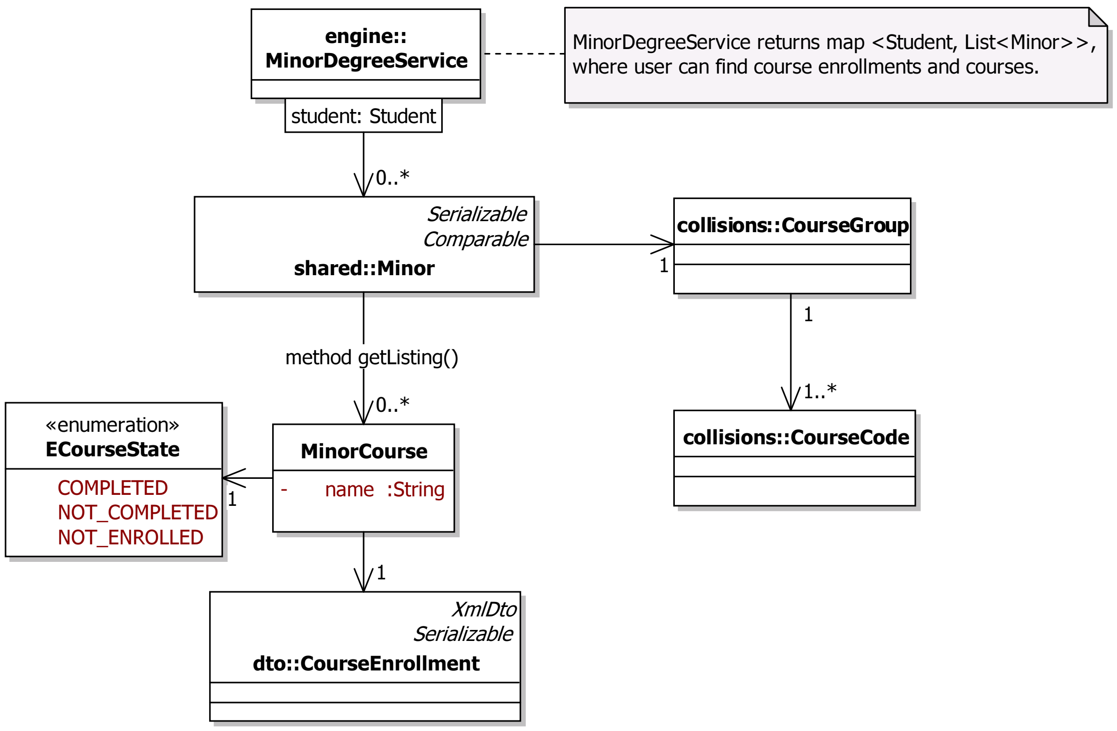


Figure 6.2: Organization of minor listing data

The service returns map of Students to collection of `Minor`. Each `Minor` has collection of `MinorCourse`, that can be at state completed, not completed or not enrolled. If MinorCourse is in state completed or not completed, `CourseEnrollment` is not `null`. Each Minor is associated to `CourseGroup` that defines it.

6.1.3 Watching for collisions

When the user or timed event triggers check of collisions, `CourseCollisionDetectionService` service walks through collection of all course groups and retrieves timetables of defined courses. Each `timetableUnit`[2] is stored into map. If map contains cell with two timetableUnits at the same place that are lectures or exercise parallels that are the only parallels of course, collision have occurred.

[2] one lecture hour

Chapter 7

Programmer's guide

The application is deployed on `cwtest.felk.cvut.cz` virtual server. Tomcat is used as a servlet container serving application. After war archive is compiled, application is deployed using tomcat manager application located at `https://cwtest.felk.cvut.cz/manager/html`.

7.1 Development

7.1.1 Obtaining the source code

Complete source code is located in git repository at `https://gitlab.fel.cvut.cz/svobodat/rozvrhy-support-apps`. You can browse generated javadoc at `http://cwtest.felk.cvut.cz:8080/rozvrhy_studentu/javadoc/index.html`.

7.1.2 Importing project to IDE

In "Eclipse" or "Spring tool suite" (which is extended Eclipse), one can follow these steps:

1. Open new workspace at repository directory (choose workspace at IDE startup)
2. Open File -> Import -> Maven -> Existing maven projects
3. Select the pom.xml of project and click "Finish"

7.1.3 Productivity tips

Google plugin for eclipse[1] is not required for development. However, it is recommended as it can be helpful to beginners. Don't forget to use Source tab to generate getters, setters, constructors, toString methods and even methods based on interface. Static type check is one of biggest advantages of using Java over using dynamically typed language.
List of useful shortcuts:

1. `ctrl+o` - resolves imports

[1] https://developers.google.com/eclipse/?hl=cs

2. `ctrl+shift+t` - finds type (class, enum or interface) by name
3. `ctrl+l` - go to line

Creating junit test: right click on class that is to be tested, "new", "new JUnit test case"

7.1.4 Code style

The code adopts the standard Java conventions. Transient annotations are written with fully qualified names, so that they can be distinguished.Names of enums are prepended with `E`, names of interfaces with `I`.

7.1.5 Running the application with jetty

Maven plugin named "jetty-maven-plugin" is used to run application locally. Use target `make jetty` to run server. To run client application in development mode, use target make `gwtdev`.

7.1.6 Speeding up jetty startup

Jetty is scanning classpath for Serlvet 3 annotations. Project does not use Servler 3 features, so it would be only waste of time. Setting empty regular expression to `WebInfIncludeJarPattern` configuration variable skips scanning and speeds up startup of jetty significantly.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
 <Call name="setAttribute">
  <Arg>org.eclipse.jetty.server.webapp.WebInfIncludeJarPattern</Arg>
  <Arg>^$</Arg>
 </Call>
</Configure>
```

Listing 11: Setting empty regular expression to speed up startup of Jetty

7.1.7 Testing with JUnit

JUnit is powerful testing framework, that is used to run tests. However, application doesn't have 100% test coverage and tests are not design for automated testing, but for programmer to check wheter service returns what it should. Programmer can write test, or look at existing test and run it without starting web server, but cannot run all tests at once. Enabling automated testing would require mocking KOSapi web service, which was out of time frame of this work.

To run a test, use "test" target of "make" and set variable "TEST" to class name of testcase you want to run. You don't need to supply fully qualified name. An example of running test: `make test TEST=IStudentDAOTest`. Note that when test ends with error, your exception is located in "target/surefire-reports" folder, not in console output of maven. In order to use dependency injection in JUnit test, test must extend `BaseTest` class, that is annotated to be run with `SpringJUnit4ClassRunner` class.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"/appContextJUnit.xml"})
public abstract class BaseTest { ... }
```

Listing 12: BaseTest class annotation that allows to use dependency injection in JUnit tests

7.2 Build

7.2.1 Makefile

Makefile is used because it provides interface to the build system that is generally known by programmers. It also allows to define useful targets.
Some of useful targets:

1. all - shows help
2. runserver - runs jetty and gwt codeserver for development
3. jetty - runs jetty
4. compile - compiles GWT application, generates javadoc, compiles server application, produces war that can be deployed
5. clean - cleans all compiled code

7.3 Production requirements

Server application requires at least Java 6 runtime environment and servlet container, such as Tomcat 6. Server with multicore processor is recommended since the application and web server are using concurrent processing. Client application requires modern web browser with CSS3 selectors capabilities, JavaScript and cookies enabled.

7.4 Development requirements

Developer must use unix based operating system [2] with Java 7 JRE (because of jetty), Java 6 JDK, GWTSDK (is downloaded by Makefile), wget (to download GWTSDK) and Maven 3. GWT plugin must be present in web browser for GWT development. Port `8080` must be open for listening.

[2]Ubuntu or MAC OS is fine

7.5 Instalation for development

1. Install Java 7 JDK, or Java 7 JRE and Java 6 JDK, if you don't have them
2. Install Maven 3 and wget packages if you don't have them
3. Checkout project from GIT repository
4. Run command `make runserver` that will do everything that is needed to the point that jetty and GWT DevMode is running
5. Open application in DevMode

Chapter 8

Conclusion

The application allows to find suitable time for meeting for a large number of students and find an empty room suitable for meeting. The application also allows to check completion of unusual study plans such as minor study fields.

The application is implemented as a web application with interactive client. Server is implemented in Java Enterprise Edition and uses current, common and standard technologies such as Spring framework, JPA or GWT and implements generally known design patterns from [] and [] so that other programmers can get familiar with application as fast as possible and reuse as much code as possible. The application leverages Java platform by sharing client and server code, using concurrency and static type system.

The most interesting features of the application are implemented strategy pattern for caching using relational database, interactive client application and usage of many standard libraries and technologies that are being used by other programmers.

8.1 Possible extensions of application

The application can be easily extended with other useful use cases. Here are some of ideas that I have found interesting.

8.1.1 Finding empty room in examination time

There is no possible way to find an empty room after semester is ended. By fetching data about exams, one could find rooms that does not have examinations at given time. This could help to quickly plan additional term for an exam.

8.1.2 Student's schedule assistant

KOS web application that is accessible by students is very user unfriendly. Each semester, students must assemble their schedule. They are forced to either conduct extensive search for courses they might be interest in or sign up for courses that are not so good or related to their field of study.

The solution could be an extension of existing application that would:

1. List courses student can add to his timetable without causing a collision
2. Allow student to move around his parallels in drag 'n drop fashion to discover more optimal schedule configurations
3. Alert student of collisions and traveling between different locations
4. Instantly access score and notes from "student questionare" of given course student plans to include to his schedule

Extension can be easily built on top of existing application, using GWT framework for complex UI and existing KOSapi integration for evaluation. Data access layer is written so that adding another data source would be easy, so fetching data from student survey would be easy. Capability of quickly finding free space in schedule was already proven in finding lecture free rooms use case.

8.1.3 Exam collision alerts

Each exam period, students are facing collisions in exam terms. Extension that would send an email to teachers who plan exam, that would cause major collision in exams might be helpful.