Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Báza znalostí a zručností študentov

Dokumentácia k inžinierskemu dielu

Tím č. 9
Vedúci projektu: RNDr. Valéria Šimáková
Predmet: Tvorba softvérového systému v tíme
Študijný program: Softvérové inžinierstvo
Ak. rok: 2008/2009

Bc. Bálint Farkas
Bc. Michal Holub
Bc. Juraj Kollár
Bc. Vojtech Villaris
Bc. Martin Virik

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Účel dokumentu

Predkladaný dokument predstavuje technickú dokumentáciu k softvérovému systému Znalec, ktorý je výsledkom študentského projektu v predmete Tvorba softvérového systému v tíme na tému Báza znalostí a zručností študentov.

Prehľad dokumentu

Opis riešeného problému sa nachádza v kapitole 1. Obsahuje základný prehľad problémovej oblasti a ciele projektu. Pojednáva aj o postupe riešenia pri napĺňaní jednotlivých cieľov.

Druhá kapitola obsahuje biznis analýzu. V tejto kapitole predstavujeme podklady, z ktorých pri riešení projektu vychádzame. Analyzujeme tu výsledky, ktoré tímy pri riešení tejto témy dosiahli v minulých rokoch. Ďalej analyzujeme všeobecné trendy pojednávaním o zaujímavých prácach zasahujúcich do oblasti znalostí a modelov používateľa. Takisto ponúkame stručný prehľad fakultných systémov, z ktorých by sme mohli čerpať údaje pre náš systém. Následne sa zamýšľame nad rôznymi typmi údajov a ich atribútmi, ktoré môžu ovplyvniť znalosti používateľa.

V kapitole 3 uvádzame špecifikáciu požiadaviek na vytváraný softvérový systém. Tá obsahuje špecifikáciu funkcií systému spolu s určením ich priorít, špecifikáciu správania systému a špecifikáciu údajov v systéme. Funkcie systému sú tu špecifikované ako jednotlivé prípady použitia. K vybraným prípadom použitia uvádzame aj sekvenčné diagramy na presnejší opis správania systému. Údaje v systéme sú špecifikované uvedením logického modelu údajov. V tejto kapitole tiež uvádzame špecifikáciu nefunkcionálnych požiadaviek.

Štvrtá kapitola obsahuje hrubý návrh architektúry systému. Návrh je spracovaný v diagrame zobrazujúcom jednotlivé moduly systému. Ich funkcionalita je následne opísaná slovne.

V piatej kapitole sa venujeme prototypovaniu. Opisujeme tu ciele prototypovania a vybrané časti systému, ktoré sme sa rozhodli zahrnúť do prototypu. Následne uvádzame dosiahnuté výsledky.

Šiesta kapitola pojednáva o implementácii navrhnutého systému. Nachádza sa tu opis implementácie jednotlivých konceptov systému, implementácie rozhrania prezentačnej vrstvy, opis algoritmov aplikačnej vrstvy a opis architektúry a jednotlivých modulov systému. Uvádzame tu tiež používané implementačné prostredie a podporné nástroje.

V kapitole 7 opisujeme spôsob overenia systému, jeho priebeh a dosiahnuté výsledky. Zaoberáme sa tu chybami, ktoré sme počas overenia odhalili.

Kapitola 8 obsahuje záverečné zhrnutie projektu. Hodnotíme v nej dosiahnutie cieľov projektu, možné použitie výsledku projektu v iných doménach, ako aj ďalšie smery rozvoja projektu. Konkrétne vymenúvame funkcie, ktoré sme nestihli implementovať, resp. ktorými je možné systém v budúcnosti doplniť.

Zadanie projektu

Obdobie štúdia na vysokej škole je zamerané na formovanie profesijného profilu, charakteru, zmyslu pre povinnosť a pre spoluprácu v tíme. Charakter a rozsah projektov, či už je to v komerčnej oblasti alebo vo vede a výskume, ukazuje, že schopnosť spolupráce je v mnohom rozhodujúca pre úspešnosť projektov, pracovných tímov v praxi. Spolupráca predpokladá určitú vyváženosť vedomostí, zručností a vzájomnej ústretovosti. Čas štúdia na vysokej škole dáva možnosť tieto schopnosti vysledovať a využiť pri výbere študentov do projektov, pri odporúčaní na študijné stáže a podobne. Tieto informácie sú dôležité a užitočné pre učiteľov, ale aj pre študentov, napríklad pri vytváraní tímov na študentské súťaže, projekty, stáže.

Úlohou tímov bude navrhnúť novú architektúru s využitím moderných technológií.

Požiadavky na výsledný produkt:

- Spolupráca so systémami používanými na FIIT (AIS, YONBAN).
- Získavanie a ukladanie znalostí o znalostiach študentov do bázy znalostí a informácií.
- Manipulácia zo znalosťami, ich indexovanie a vyhľadávanie.
- Používateľský príjemná prezentácia informácií o znalostiach. Personifikovaný prístup k informáciám (učiteľ, študent...).
- Akceptácia externých informačných entít.

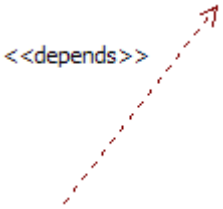
Skratky

Tab. 0.1 Použité skratky

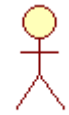
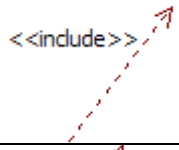
SKRATKA	VÝZNAM
AIS	Akademický informačný systém
API	Application Programming Interface
CRUD	Create, Read, Update, Delete (vytvorenie, načítanie, aktualizácia, vymazanie)
DAO	Data Access Object
DB	Databáza
FIIT	Fakulta informatiky a informačných technológií
HQL	Hibernate Query Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IoC	Inversion of Control
JDBC	Java Database Connectivity
JEE	Java Enterprise Edition
JNDI	Java Naming and Directory Interface
Obr.	Obrázok
ORM	Objektovo-relačné mapovanie
OWL	Web Ontology Language
SQL	Structured Query Language
Tab.	Tabuľka
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Použitá notácia

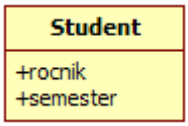
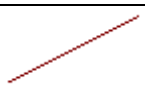
Tab. 0.2 Notácia pre diagram cieľov

	Biznis cieľ
	Asociácia vyjadrujúca závislosť

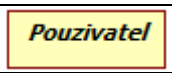
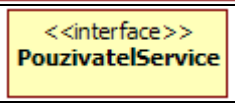
Tab. 0.3 Notácia pre diagram prípadov použitia

	Hráč
	Prípad použitia
	Neorientovaná asociácia
	Orientovaná asociácia
	Asociácia vyjadrujúca zahrnutie jedného prípadu použitia v druhom
	Asociácia vyjadrujúca zovšeobecnenie

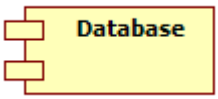
Tab. 0.4 Notácia pre diagram logického modelu údajov

	Entita logického modelu s niektorými atribútmi
	Neorientovaná asociácia medzi entitami
	Asociácia vyjadrujúca slabú agregáciu vrátane kardinalít

Tab. 0.5 Notácia pre diagram tried

	Abstraktná trieda
	Trieda
	Rozhranie
	Asociácia vyjadrujúca slabú agregáciu
	Asociácia vyjadrujúca vzťah dedenia

Tab. 0.6 Notácia pre diagram tried

	Komponent systému
	Rozhranie
	Asociácia vyjadrujúca závislosť
	Asociácia vyjadrujúca tok riadenia

1 Opis riešeného problému

Opis riešeného problému vychádza zo zadania uvedeného v úvode.

1.1 Prehľad problémovej oblasti

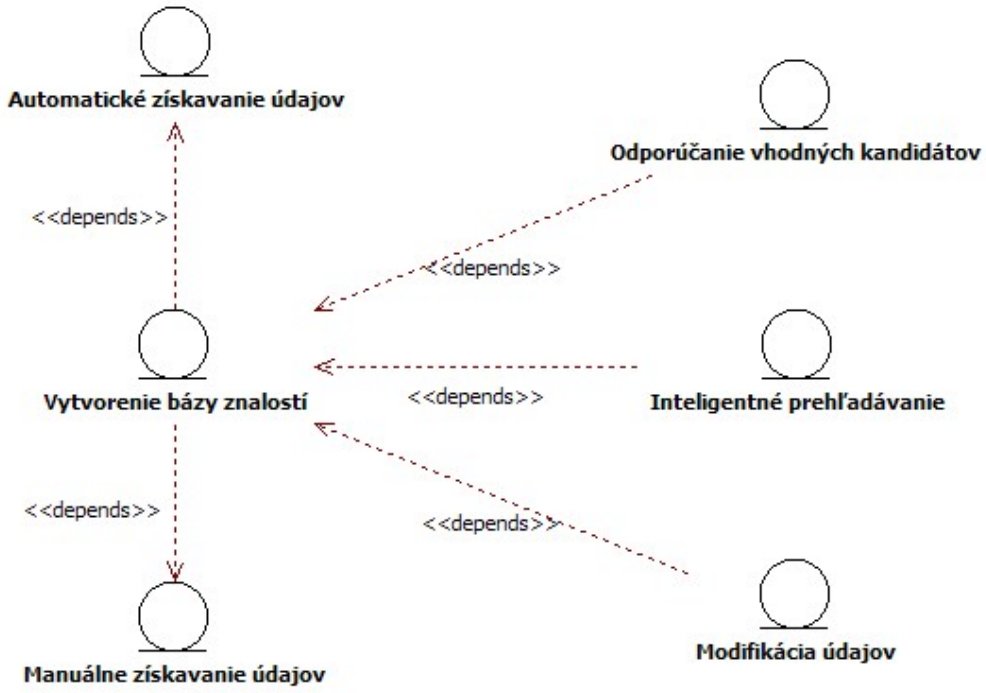
Fakulta informatiky a informačných technológií (FIIT) dostáva rozličné ponuky na účasť na stážach, konferenciách prípadne na spoluprácu na rozličných projektoch. Tieto ponuky sú často určené aj pre študentov. FIIT využíva na uchovávanie informácií o študentoch prevažne celouniverzitný systém AIS. V ňom sa nachádza množstvo údajov o jednotlivých študentoch, no medzi údajmi nie je zrejmá súvislosť. AIS nedokáže z jednotlivých údajov sformovať znalosti daného študenta. Ten si navyše nemôže upravovať svoj osobný profil pridávaním informácií o absolvovaných seminároch, školeniach, praxi, ktoré vylepšujú jeho znalosti v jednotlivých odboroch. Niektoré údaje o študentoch sú uložené aj v iných systémoch, ktoré fakulta používa, no je snaha previesť všetky tieto údaje do AISu.

Nakoľko na FIIT študuje veľké množstvo študentov, nie je v súčasnosti možné efektívne odporučiť vhodného študenta na odpovedanie na ponuku. Poverený pracovník tak musí prejsť množstvo údajov o študentoch a manuálne v nich hľadať tých, ktorí by mohli mať predpoklady a potenciálny záujem o danú ponuku. Pri tom stratí veľa času a môže sa stať, že niektorých vhodných kandidátov prehliadne.

1.2 Ciele projektu

Účelom nami vytváraného informačného systému je zjednodušenie a zefektívnenie procesu vyhľadania vhodných študentov, ktorí by mohli reagovať na ponuku. Z tohto dôvodu je nutné vytvoriť bázu znalostí a zručností študentov, a pravidelne ju obnovovať.

Pre analyzovaný informačný systém sme identifikovali tieto biznis ciele: vytvorenie bázy znalostí, manuálne a automatické získavanie údajov, odporúčanie vhodných kandidátov, inteligentné prehľadávanie, modifikácia údajov. Ich vzájomné vzťahy sú zobrazené na Obr. 1.1.



Obr. 1.1 Diagram biznis cieľov

1.3 Slovník pojmov problémovej oblasti

V tejto časti uvádzame vysvetlenie hlavných pojmov, ktoré používame v projekte.

Znalosť je súbor faktov a informácií, získaných skúsenosťami alebo štúdiom, poprepájaných vzťahmi, ktoré navzájom prinášajú známosť, vedomosť v oblasti poznávania [1] [2].

Zručnosť je preukázaná osobná vlastnosť alebo preukázaná schopnosť aplikovať nadobudnuté znalosti pri praktickom riešení úloh [3].

V našom systéme je potrebné dôsledne rozlišovať pojmy znalosť a zručnosť, hoci môžu byť vyjadrené rovnakými kľúčovými slovami, a teda sa na prvý pohľad javiť ako totožné. Znalosť chápeme ako zloženú vedomosť v oblasti poznania, ktorá sa skladá zo zručností. Ako príklad uvedieme znalosť jazyka Java. K tejto znalosti môžu prispieť aj zručnosti vyjadrené kľúčovými slovami Eclipse alebo Net Beans. Samotné kľúčové slovo Java však môže v inom kontexte predstavovať zručnosť. Tu sa jedná o elementárnu schopnosť. Zručnosť Java môže spolu so zručnosťou UML prispievať k znalosti objektovo-orientovaného programovania.

V systéme ďalej používame výraz fakt. Pod týmto pojmom rozumieme určitú vzdelávaciu jednotku, ktorá pomohla študentovi nadobudnúť zručnosti, ktoré sa následne premietnu do jeho znalostí. Faktom môže byť napr. absolvované školenie o programovaní v Jave. Študent na ňom mohol získať zručnosti Eclipse a Java, ktoré prispejú k jeho znalosti OOP.

Každý fakt má svoje atribúty. Pre uvedený príklad faktu školenie môžu byť atribúty názov školenia, dĺžka trvania, forma ukončenia.

Posledným dôležitým pojmom je hodnota atribútu faktu. Rozumieme pod ním konkrétne hodnoty jednotlivých atribútov faktov. V uvedenom príklade faktu školenie môže mať atribút dĺžka trvania hodnotu napr. 1 týždeň.

[1] http://www.thefreedictionary.com
[2] http://www.translationbureau.gc.ca/index.php
[3] http://www.ism3.com/index.php

2 Biznis analýza

V tejto kapitole analyzujeme ako projekty zaoberajúce sa podobnou problematikou, tak aj postupy, ktoré by sme mohli uplatniť pri vytváraní nášho riešenia. Najprv uvádzame analýzu výsledkov tímov zaoberajúcich sa tvorbou bázy znalostí a zručností z minulých rokov.

2.1 Tím Černé ofce

V školskom roku 2007/2008 riešil tému Báza znalostí a zručností študentov tím č. 10 s názvom Černé ofce. Tento tím vytvoril systém s nasledujúcimi vlastnosťami [3].

2.1.1 Identifikované roly v systéme

Tím Černé ofce identifikoval nasledujúce roly:

- **Študent** – Zadáva do systému informácie o vlastných znalostiach a zručnostiach.
- **Pedagogický pracovník** – Zadáva do systému informácie o študentoch, požaduje od systému znalosti o študentoch.
- **Administrátor** – Spravuje systém a má na starosti jeho správny chod počas bežnej prevádzky.
- **Externá osoba** – Využíva znalosti poskytované systémom po súhlase študenta.

2.1.2 Údaje evidované systémom a spôsob ich získania

Systém navrhnutý tímom Černé ofce eviduje nasledovné údaje:

- používateľ systému,
    - študent
    - pedagóg
    - administrátor
    - hosť
- predmety,
- kľúčové slová,
- váhy,
- hodnotenie študenta pedagógom a
- študentovo vlastné hodnotenie.

Informácie o študentoch sú importované z Akademického informačného systému, ostatné skupiny používateľov sú do systému pridané ručne. Predmety sú importované tiež z Akademického informačného systému. Sebahodnotenie vyplní prihlásený používateľ (študent). Obdobne, prihlásený pedagóg môže vyplniť hodnotenie jednotlivým študentom.

Samotný logický model databázy je jednoduchý a prehľadný. Je však otázne, do akej miery by ho bolo možné ďalej normalizovať. Bolo by napríklad možné zahrnúť do modelu databázy aj váhy sebahodnotenia a hodnotenia učiteľov. V dokumentácii sa nachádza zmienka o možnosti ďalšieho rozširovania hodnotení o započítavanie rôznych diplomov, súťaží, atď. Databázový model v existujúcej podobe je však málo flexibilný a aj algoritmy vyhodnocovania sú príliš viazané na jednotlivé entity.

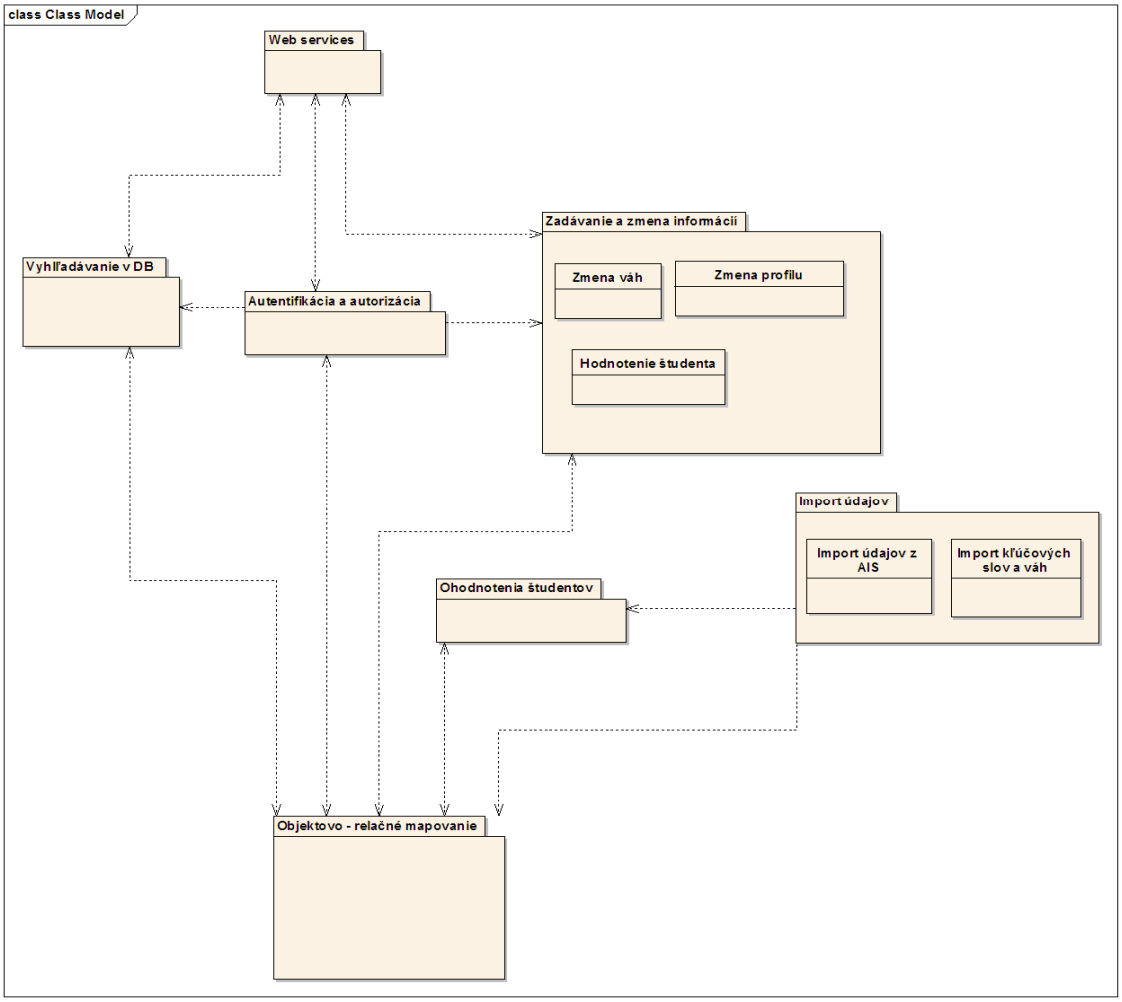
2.1.3 Funkcie systému

Systém, ktorý tím Černé ofce vytvoril, má nasledovné funkcie:

- Poskytnutie súhlasu pre tretie strany – rozhodnutia študenta zverejniť informácie o sebe aj tretej strane. Túto funkciu môže používať len študent.
- Zadávanie informácií o študentovi – vloženie informácií týkajúcich sa znalostí, zručností a študijných výsledkov študenta. Túto funkciu môže používať ako študent, tak aj pedagogický pracovník.
- Vyhľadanie študenta/študentov – systém nájde študenta/študentov na základe zadaných kritérií hľadania. Túto funkciu môže používať študent, pedagogický pracovník, ako aj externá osoba.
- Kontaktovanie študenta – spojenie sa so študentom využitím poskytnutého kontaktu (e-mail, tel. č., ICQ, ...). Túto funkciu môžu použiť všetci používatelia systému.
- Správa servera – zabezpečenie správneho chodu servera, rôzne nastavenia. Túto funkciu môže používať iba administrátor.
- Zmeny v databáze – uskutočnenie nevyhnutných zmien v databáze v prípade potreby. Ako predošlú funkciu, aj túto môže používať iba administrátor.

2.1.4 Architektúra systému

Na Obr. 2.1 je zobrazený diagram architektúry systému tímu Černé ofce.



Obr. 2.1 Diagram architektúry systému tímu Černé ofce [3]

Používateľ pristupuje do systému cez používateľské prostredie, ktoré je generované webovými službami. Bude vyzvaný, aby zadal svoje prístupové meno a heslo. Tieto údaje sa cez webové služby pošlú do modulu autentifikácie a autorizácie, kde sa zistí korektnosť zadaných údajov. Zistí sa tiež, aké práva daný používateľ má. Následne, podľa prístupových práv, bude môcť používateľ používať funkcie systému.

Na implementáciu a chod systému použil tím Černé ofce nasledovné nástroje a technológie:

- aplikačný rámec Apache,
- databázový server PostgreSQL,
- vývojové prostredie NetBeans,
- objektovo-relačný mapovačť Hibernate a
- Java Server Pages (JSP).

2.1.5 Zhodnotenie

Tím Černé ofce navrhol pomerne jednoduchý systém, ktorý však v dostatočnej miere umožňuje poskytovanie informácií o výsledkoch študentov. Tento systém následne úspešne implementovali. Nedostatok vidíme v nízkej flexibilite systému, čo môže spôsobovať problémy pri snahe o rozšírenie systému. Systém je tiež úzko zameraný a je v ňom málo možností nastavenia.

2.2 Tím _elf_

V školskom roku 2005/2006 riešil tému Báza znalostí a zručností študentov tím č. 11 s názvom _elf_. Tento tím vďaka skúsenostiam z praxe úspešne vyhodnotil existujúce systémy v danej oblasti. Na základe týchto poznatkov navrhol a implementoval vlastný systém s nasledujúcimi vlastnosťami [2].

2.2.1 Identifikované roly v systéme

Tím _elf_ identifikoval nasledovné roly:

- **Študent** – Zadáva do systému informácie o vlastných znalostiach a zručnostiach.
- **Vyučujúci** – Zadáva do systému informácie o študentoch, požaduje od systému znalosti o študentoch.
- **Externý systém** – Využíva znalosti poskytnuté navrhovaným systémom na ich ďalšie spracovanie alebo priamu prezentáciu, alebo poskytuje informácie o študentovi (ako sú známky, vážený študijný priemer, a pod.).
- **Administrátor** – Spravuje systém a má na starosti jeho správny chod počas rutinnej prevádzky.
- **Konfigurátor** – Nastavuje parametre systému, určuje typy informácií prístupné používateľom.

2.2.2 Údaje evidované systémom a spôsob ich získania

Systém navrhnutý tímom _elf_ eviduje nasledovné údaje:

- používatelia systému a ich roly,
- študenti a učitelia,
- predmety,
- kľúčové slová,
- znalosti – znalosti, zručnosti, certifikáty, známky z predmetov, hodnotenia od učiteľov,
- konfiguračné parametre systému a
- balíčky – preddefinované kritériá vyhľadávania.

Údaje o študentoch, učiteľoch a predmetoch získava systém importom dát zo systému ŠTUDENT, ako aj zo vstupov od používateľov. Ostatné údaje sa do systému dostávajú ako vstupy od rôznych typov používateľov, teda nie len od študentov, ale aj od učiteľov, administrátorov, konfigurátorov systému a od externých systémov.

Logická štruktúra uložených údajov je vhodná. Je navrhnutá tak, aby sa systém dal dostatočne konfigurovať a prispôsobovať, a aby sa s ním dalo efektívne pracovať. Umožňuje napríklad definovať tzv. balíčky, ktoré by sa dali charakterizovať ako vopred definované kritéria pre vyhľadávanie študentov. Existujú globálne balíčky, ktoré môžu znovu použiť učitelia, prípadne si ich môžu modifikovať a uložiť do svojich vlastných preddefinovaných balíčkov.

Údaje, ktoré sa týkajú znalostí, sú váhované, vďaka čomu sa dá lepšie opísať miera danej znalosti. Umožňuje to aj presnejšie vyhľadávanie.

2.2.3 Funkcie systému

Role **študent** poskytuje systém nasledovnú funkcionalitu:

- poskytovanie detailných informácií o svojom profile,
- zobrazenie informácií o predmetoch,
- zobrazenie znalostí, zručností, atď. uchovávaných v systéme,
- zmena osobného profilu a
- notifikácia študenta, aby si obnovil svoj profil.

Role **vyučujúci** poskytuje systém nasledovnú funkcionalitu:

- zobrazenie informácií o predmetoch,
- zobrazenie znalostí, zručností, atď. uchovávaných v systéme,
- prehliadanie informácií o študentoch,
- správa osobných balíčkov a
- vyhľadávanie študentov na základe balíčkov.

Role **administrátor** poskytuje systém nasledovnú funkcionalitu:

- import údajov zo systému ŠTUDENT,
- správa používateľov systému a
- poslanie notifikácie študentom, aby si obnovili profil.

Role **konfigurátor** poskytuje systém nasledovnú funkcionalitu:

- správa globálnych balíčkov,
- správa zoznamu zručností, znalostí, certifikátov a
- konfigurácia interných nastavení systému.

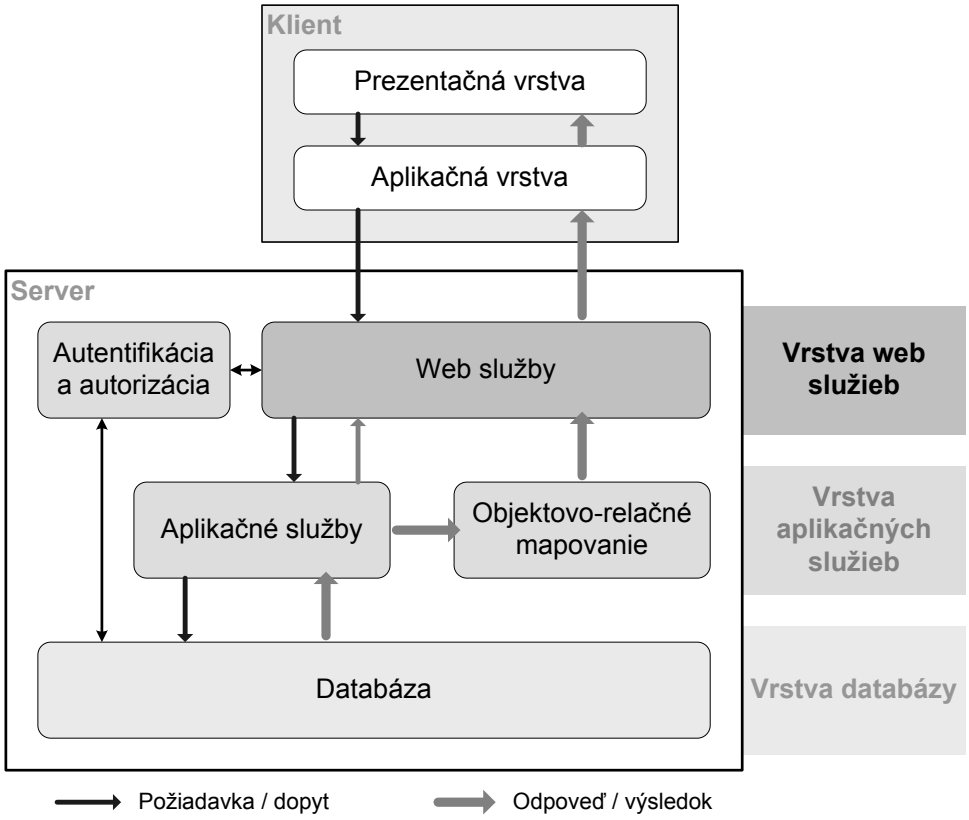
Role **externý systém** poskytuje systém nasledovnú funkcionalitu:

- správa balíčkov,
- poskytnutie známky a
- poskytnutie indexovaného zoznamu študentov.

2.2.4 Architektúra systému

Architektúra systému je postavená na báze klient – server. Vonkajšie rozhranie je vytvorené pomocou webových služieb. Ďalej je systém rozdelený na jednotlivé vrstvy. Klientskú časť tvoria prezentačná a aplikačná vrstva. Serverová časť systému má na starosti webové služby, autentifikáciu a autorizáciu,

aplikačné služby, objektovo-relačné mapovanie a prístup do databázy. Architektúra tohto systému je zobrazená na Obr. 2.2.



Obr. 2.2 Diagram architektúry systému tímu _elf_ [2]

Systém je implementovaný v jazyku Java. Na vývoj použil tím _elf_ integrované prostredie Eclipse. Ako databázový systém bol zvolený PostgreSQL. Na tvorbu webových služieb využili aplikačný rámec Apache AXIS. Prístup do databázy bol zjednodušený vlastnou implementáciou objektovo-relačného mapovania. Na vytvorenie prezentačnej vrstvy tím _elf_ použil rámec Struts, ktorého výstupom boli Java Server Pages (JSP) stránky. Aplikácia bola nasadená na aplikačný server Tomcat. Zostavovanie bolo zabezpečené pomocou nástroja Maven. Na logovanie dôležitých činností systému bol použitý nástroj Log4J. Systém bol testovaný pomocou knižnice JUnit. Serializácia XML dokumentov bola zabezpečená pomocou nástroja XStream.

2.2.5 Vlastnosti systému

Systém implementovaný tímom _elf_ má nasledovné vlastnosti:

- komunikácia s ostatnými systémami pomocou webových služieb (angl. *web services*),
- dôraz na bezpečnosť – autorizácia a autentifikácia,
- komunikácia pomocou zabezpečeného protokolu HTTPS,
- ochrana osobných údajov študentov a
- prepracovaný algoritmus na vyhľadávanie študentov na základe zadaných kritérií (balíčkov a skupín).

2.2.6 Zhodnotenie

Tímu _elf_ sa podarilo navrhnúť a implementovať fungujúci systém na evidenciu znalostí a zručností študentov. Vytvorený logický model má veľmi dobrú štruktúru, ktorá umožňuje systém dostatočne

konfigurovať. Niektoré údaje sa však evidujú zbytočne, napr. údaje o predmetoch alebo o učiteľoch neprinášajú nijakú špeciálnu hodnotu pre používateľa a sú už evidované v systéme AIS.

Vytvorený systém je dostatočne zabezpečený a chráni osobné údaje študentov v zmysle zákona. Ochrana osobných údajov je však na niektorých miestach až prehnaná, napr. zaraďovanie výsledného zoznamu študentov do tried na základe štatistických metód.

Navrhnutý algoritmus na vyhľadávanie študentov je dostatočne prepracovaný a umožňuje učiteľom pohodlne a efektívne vyhľadávať študentov podľa ich potreby.

Vytvorený systém spolupracuje so systémom ŠTUDENT, importuje z neho informácie o študentoch a predmetoch. Tento systém v súčasnosti nahradil na celej univerzite systém AIS.

2.3 Analýza modelu používateľa

Na reprezentáciu znalostí a zručností študenta potrebujeme model, ktorý nám umožní uchovávať a ohodnocovať jeho rozličné schopnosti. Najlepšie sa nám javí rozdeliť model na 3 časti:

- všeobecnú časť - charakteristiky používateľa ako napr. vek, pohlavie, atď.
- doménovo nezávislú časť – charakteristiky používateľa nezávisiace od domény, tieto charakteristiky sa menia len sporadicky. Patria sem dosiahnuté vzdelanie, odbor, atď.
- špecifickú časť

V [1] navrhli autori systém, v ktorom špecifická časť obsahuje schopnosti používateľa ohodnotené váhami v intervale <0;100>. Následne pri zadávaní požiadavky na vhodných používateľov zadávateľ určil požadované schopnosti, ktoré tiež ohodnotil váhami. Na základe toho systém vybral vhodných kandidátov.

V systéme boli vytvorené stereotypy. Stereotyp je množina ľudí s rovnakými charakteristikami, napr. návrhár, analytik, programátor, tester, atď. Každý stereotyp má určené schopnosti, ktoré človek musí na zaradenie do tohto stereotypu spĺňať, a tiež mieru tohto splnenia v percentách. Jeden stereotyp bol úplne bez určených schopností, aby bol model všeobecný (do tohto stereotypu boli zaradení používatelia bez akýchkoľvek znalostí).

Model používateľa bol uchovaný vo forme ontológie, v jazyku OWL. Systém bol vytvorený v jazyku Java pomocou rámca Sesame (www.openrdf.org).

2.4 Analýza informačných systémov používaných na FIIT

2.4.1 Akademický informačný systém

AIS je komplexný celouniverzitný informačný systém evidujúci informácie o študentoch, predmetoch, vyučujúcich a pod. Tento systém je postavený na databázovom systéme Oracle. V súčasnej dobe prebieha vytváranie pohľadu do tejto databázy, ktorý bude obsahovať:

- údaje o študentoch (ID, meno, adresa, št. program, ročník, semester, predmety, rozvrh a informácie o štúdiu) a
- údaje o pedagógoch (predmety, rozvrh, pracovisko, rola v danom predmete).

Náš systém bude čerpať všetky údaje práve z AISu.

2.4.2 Yonban

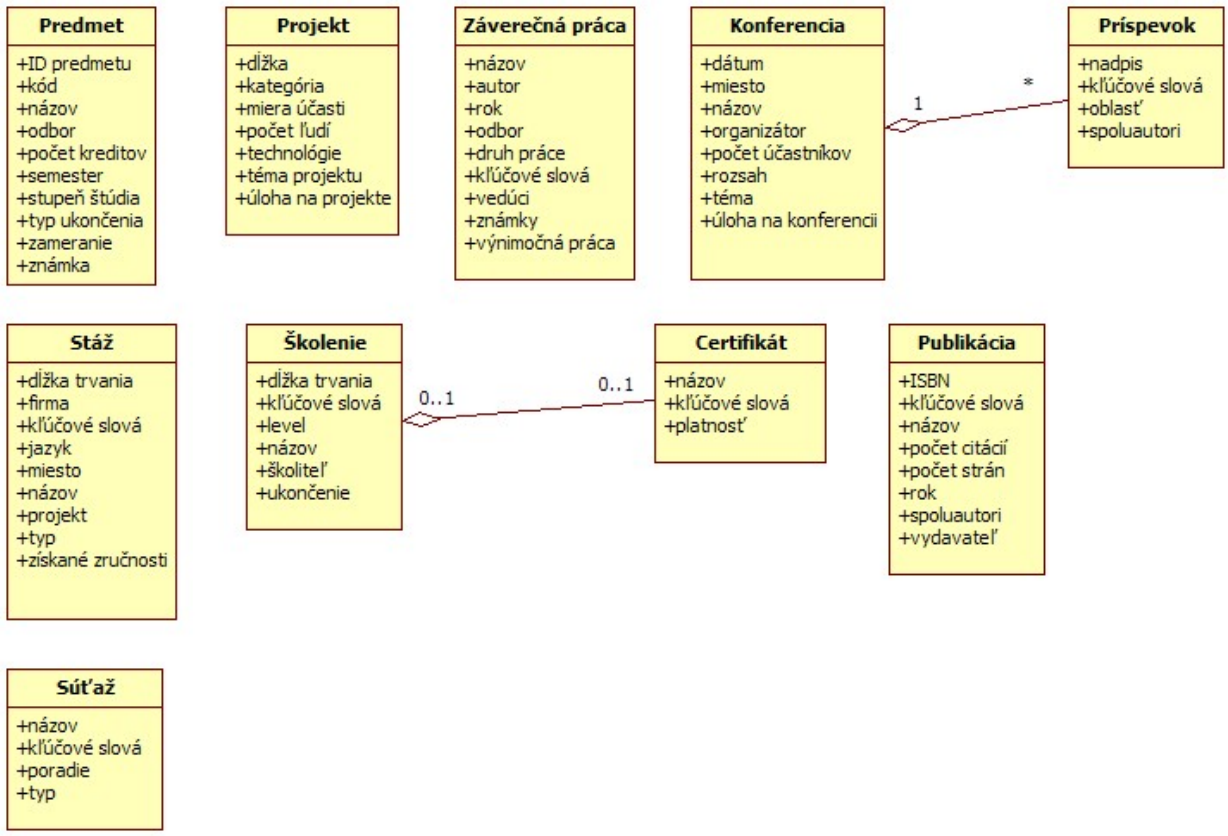
Yonban slúži na evidenciu bakalárskych, diplomových a iných záverečných prác. Všetky informácie v ňom evidované sa nachádzajú aj v systéme AIS, takže ho nebudeme v našom projekte využívať.

2.5 Zmena váh

V rámci procesu vzdelávania občas nastávajú zmeny v systéme výučby a vtedy môže dôjsť aj k zmenám váh, ktoré ovplyvňujú kľúčové slová uchované v databáze. Tieto kľúčové slová sú potrebné pre určenie znalostí študenta. Aby systém mohol reagovať na takéto zmeny a zakaždým určoval čo najpresnejšie znalosti študenta, musí obsahovať modul, pomocou ktorého budú tieto váhy upravené. Takýto modul navrhol v svojom projekte z roku 2007/2008 tím Černé ofce.

2.6 Analýza faktov

Identifikovali sme 8 druhov znalostí, z ktorých môže náš systém pri zostavovaní modelu študenta čerpať. Diagram zobrazujúci tieto fakty je na Obr. 2.3. Ku každému faktu sme identifikovali niekoľko jeho atribútov. Nie všetky atribúty sú pre náš systém dôležité. V budúcnosti môže vzniknúť potreba definovať ďalšie typy faktov. Nami navrhovaný systém je všeobecný a umožňuje jednoduché definovanie nových faktov spolu s ich atribútmi. Tým je zabezpečená jeho univerzálnosť.



Obr. 2.3 Diagram faktov a ich atribútov

3 Špecifikácia požiadaviek na informačný systém

V tejto kapitole uvádzame špecifikáciu požiadaviek na vytváraný informačný systém Znalec.

3.1 Funkcionálne požiadavky na systém

Na základe analýzy a komunikácie so zákazníkom sme identifikovali funkcionálne požiadavky. Tie sme roztriedili podľa priority nasledovne:

- vysoká priorita
  - evidencia študentov
  - pridávanie konkrétnych faktov študentmi a ich ohodnotenie váhami
  - vyhodnotenie znalosti študenta na základe faktov
  - vyhľadávanie študentov na základe znalostí
- stredná priorita
  - získavanie informácií o študentoch zo systému AIS
  - vysoká konfigurovateľnosť systému
  - vypĺňanie osobných profilov študentov
- nízka priorita
  - prezeranie študentských profilov osobami mimo fakulty
  - zaraďovanie študentov do skupín podľa znalostí
  - vytváranie preddefinovaných pravidiel vyhľadávania študentov

3.2 Ďalšie požiadavky na systém

Okrem funkcionálnych požiadaviek sme identifikovali aj ďalšie požiadavky, týkajúce sa najmä prevádzky a údržby systému:

- Bezpečnost – Prístup k databáze je možný len prostredníctvom nášho systému. Používateľom nebude umožnený priamy prístup k databáze. Každý používateľ systému sa musí prihlásiť pomocou svojho prihlasovacieho mena a hesla, ktoré sú uložené v šifrovanej podobe. Prihlasovacie heslo musí byť dostatočne robustné, čo systém overuje pri jeho vytváraní (musí obsahovať určitý minimálny počet alfanumerických a špeciálnych znakov).
- Spoľahlivosť – Systém monitoruje vykonané zmeny a pravidelne vykonáva zálohu databázy.
- Korektnosť údajov – Systém je zabezpečený zámkami, ktoré zaručujú, že jednu entitu (napr. profil študenta) môže naraz editovať len jeden používateľ a nenastal tak konflikt.

3.3 Prepojenie cieľov a požiadaviek

Jednotlivé funkcionálne požiadavky na systém, špecifikované v predchádzajúcej podkapitole, predstavujú biznis prípady použitia, ktoré realizujú ciele definované v kapitole 1.2. Nasleduje sumár prepojení biznis prípadov použitia na ciele, ktoré realizujú.

Automatické získavanie údajov: získavanie informácií o študentoch zo systému AIS

Manuálne získavanie údajov: pridávanie konkrétnych faktov študentmi a ich ohodnotenie váhami

Vytvorenie bázy znalostí: evidencia študentov, o vyhodnotenie znalosti študenta na základe faktov

Odporúčanie vhodných kandidátov: vyhľadávanie študentov na základe znalostí, zaraďovanie študentov do skupín podľa znalostí

Inteligentné prehľadávanie: prezeranie študentských profilov osobami mimo fakulty, vytváranie preddefinovaných pravidiel vyhľadávania študentov

Modifikácia údajov: vysoká konfigurovateľnosť systému, vypĺňanie osobných profilov študentov

3.4 Charakteristika rolí

V systéme sme identifikovali nasledujúce roly:

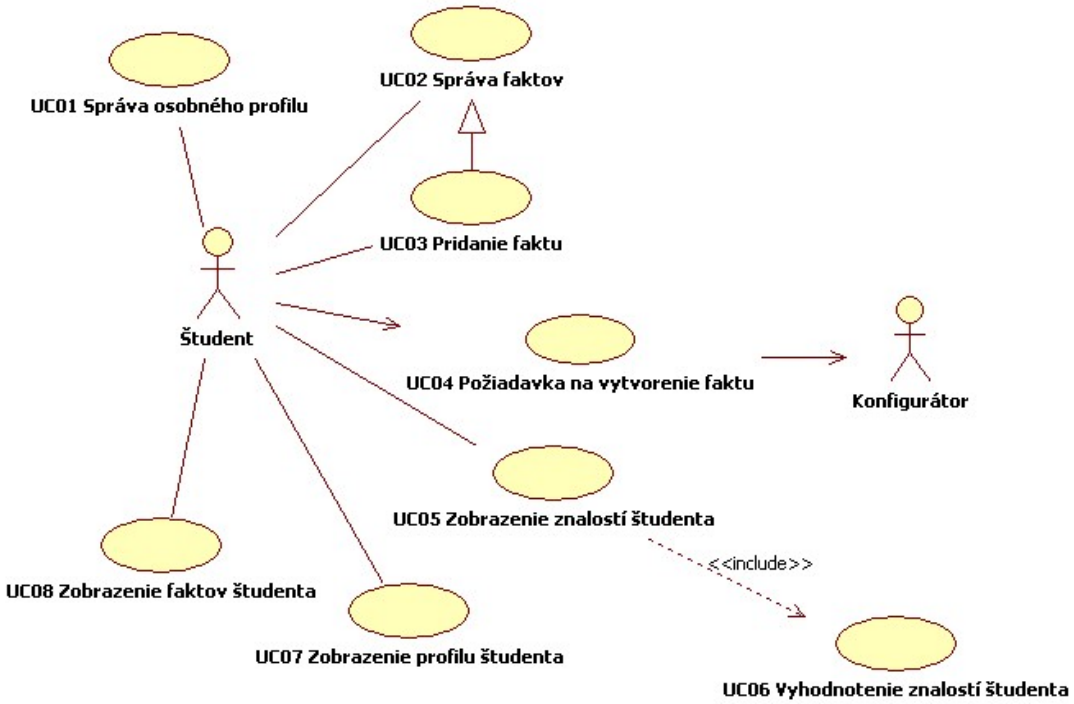
- **Študent** – riadne zapísaný študent
- **Pedagóg** – pedagogický zamestnanec školy
- **Administrátor** – osoba poverená správou systému
- **Externý používateľ** – osoba mimo fakulty, musí však mať v systéme účet
- **Konfigurátor** – osoba zodpovedná za konfiguráciu faktov, znalostí a kľúčových slov, a vzťahov medzi nimi

3.5 Diagramy prípadov použitia

Na základe uvedených funkcionálnych požiadaviek sme vytvorili diagramy prípadov použitia nášho systému. Každý diagram zobrazuje jednu rolu a prípady použitia, ktoré k nej prislúchajú. Následne je každý prípad použitia opísaný v samostatnej tabuľke.

3.5.1 Prípady použitia pre rolu študent

Pre rolu študent sme identifikovali prípady použitia zobrazeného na Obr. 3.1.



Obr. 3.1 Diagram prípadov použitia pre rolu študent

V nasledujúcich tabuľkách uvádzame podrobný opis prípadov použitia pre rolu študent.

Tab. 3.1 Prípad použitia UC01 – Správa osobného profilu

Správa osobného profilu	
Identifikátor	UC01
Názov	Správa osobného profilu
Opis	CRUD operácie nad osobným profilom študenta
Priorita	stredná
Vstupné podmienky	
Výstupné podmienky	Uložené zmeny sú perzistentné.
Používatelia	Študent

Tab. 3.2 Prípad použitia UC02 – Správa faktov

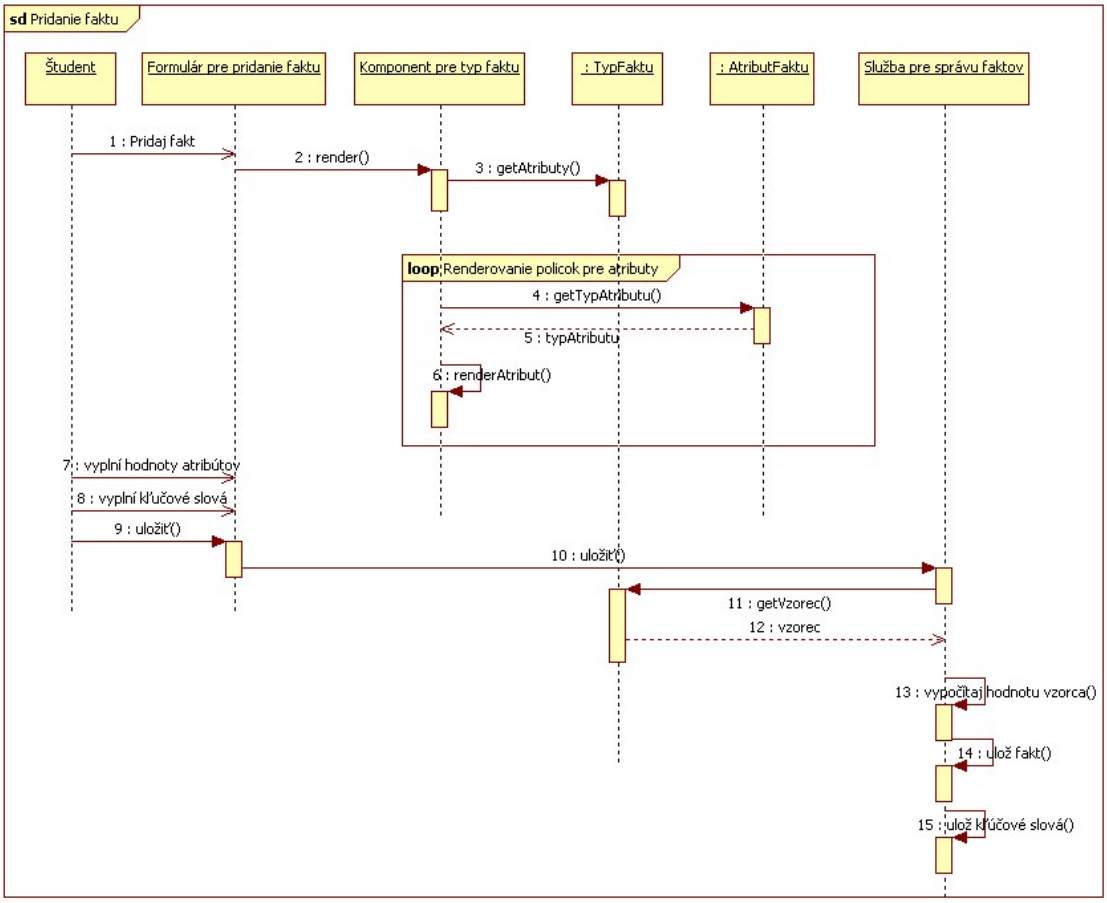
Správa faktov	
Identifikátor	UC02
Názov	Správa faktov
Opis	CRUD operácie nad faktami študenta
Priorita	vysoká
Vstupné podmienky	
Výstupné podmienky	Uložené zmeny sú perzistentné.
Používatelia	Študent

Tab. 3.3 Prípad použitia UC03 – Pridanie faktu

Pridanie faktu	
Identifikátor	UC03
Názov	Pridanie faktu
Opis	Umožňuje pridať fakt k študentovi. Fakt môže pridať prihlásený študent len sebe. Prihlásený pedagóg môže pridať fakt ľubovoľnému študentovi. Fakt môže pridať aj systém pri importe údajov z AISu.
Priorita	vysoká
Vstupné podmienky	Používateľ je prihlásený ako študent alebo používateľ je prihlásený ako pedagóg, alebo používateľ je prihlásený ako administrátor.
Výstupné podmienky	Pridaný fakt je perzistentný.
Používatelia	Študent, Pedagóg, Administrátor

Postupnosť pre študenta	**Krok**	**Činnosť**
	1	Študent zvolí typ faktu, ktorý chce pridať.
	2	Systém zobrazí formulár pre zadanie hodnôt atribútov daného faktu definovaných typom faktu.
	3	Študent vyplní položky formulára. Vyplní aj kľúčové slová, ktoré s faktom súvisia a určí ich váhy podľa dôležitosti v rámci daného faktu.
	4	Študent potvrdí pridanie faktu.
	5	Systém uloží zadané údaje do databázy. Ak sa zadané kľúčové slová ešte nenachádzajú v databáze, tak sa do nej pridajú. Ak sa nachádzajú, tak sa len prepoja s daným faktom a priradia sa im zadané váhy.
Postupnosť pre pedagóga	**Krok**	**Činnosť**
	1	Pedagóg zvolí študenta, ktorému chce pridať nový fakt.
	2	Ďalšie kroky sú rovnaké ako pri študentovi.
Postupnosť pri importe údajov z AISu	1	Systém pridáva relevantné fakty (absolvovanie predmetov, záverečné práce a pod.) k študentom na základe údajov z AISu. Váhy sa nastavia na implicitnú hodnotu.
Poznámky	Tento prípad použitia je špecializáciou prípadu použitia správa faktov. Dôvod jeho vyčlenenia je ten, že pri ňom vystupujú viacerí hráči a aj okolnosti jeho vykonania sú špeciálnejšie.	

Na Obr. 3.2 je zobrazený sekvenčný diagram pre prípad použitia UC03 – Pridanie faktu. Študent najskôr zvolí, že chce pridať fakt. Pritom zadá aj aký typ faktu chce pridať. Systém mu ponúkne stránku s formulárom pre zadanie hodnôt atribútov pridávaného faktu. Formulár sa kreslí genericky v závislosti od typu faktu. Grafický komponent požiada entitu typ faktu, aby mu vrátila zoznam atribútov. Tento komponent potom v závislosti od typu atribútu vykreslí príslušné vstupné pole formulára. Po vykreslení formulára študent vyplní jeho vstupné polia, zadá kľúčové slová a uloží ho. Služba pre správu faktov vypočíta hodnotu faktu na základe vzorca a vyplnených atribútov. Táto služba nakoniec uloží fakt s jeho vyplnenými atribútmi a kľúčové slová do databázy.



Obr. 3.2 Sekvenčný diagram pre prípad použitia UC03 – Pridanie faktu

Tab. 3.4 Prípad použitia UC04 – Požiadavka na vytvorenie faktu

Požiadavka na vytvorenie faktu		
Identifikátor	UC04	
Názov	Požiadavka na vytvorenie faktu	
Opis	Umožňuje upozorniť konfigurátora, aby pridal do systému nový typ faktu.	
Priorita	nízka	
Vstupné podmienky		
Výstupné podmienky	Konfigurátor bol notifikovaný e-mailom.	
Používatelia	Študent, Pedagóg	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ zvolí, že chce pridať nový fakt
	2	Systém zobrazí formulár pre zadanie opisu žiadaného faktu.
	3	Používateľ vyplní položky formulára.
	4	Systém pošle e-mail s požiadavkou na vytvorenie faktu s údajmi zadanými používateľom.

Tab. 3.5 Prípad použitia UC05 – Zobrazenie znalostí študenta

Zobrazenie znalostí študenta		
Identifikátor	UC05	
Názov	Zobrazenie znalostí študenta	
Opis	Zobrazí používateľovi zistené znalosti daného študenta.	
Priorita	vysoká	
Vstupy	Študent, ktorého znalosti sa budú vyhodnocovať a zobrazovať	
Výstupy	Zistené znalosti o danom študentovi	
Vstupné podmienky	Používateľ je prihlásený a má aspoň jednu z rolí Študent, Pedagóg. Ak je používateľ prihlásený ako Externý používateľ, tak daný študent musí mať povolené zobrazovanie informácií pre verejnosť.	
Výstupné podmienky		
Používatelia	Študent, Pedagóg, Externý používateľ	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ zvolí študenta, ktorého znalosti chce zobraziť.
	2	Spustí sa prípad použitia Vyhodnotenie znalostí študenta. Na jeho vstup pôjde meno daného študenta.
	3	Systém zobrazí výstup prípadu použitia Vyhodnotenie znalostí študenta

Tab. 3.6 Prípad použitia UC06 – Vyhodnotenie znalostí študenta

Vyhodnotenie znalostí študenta		
Identifikátor	UC06	
Názov	Vyhodnotenie znalostí študenta	
Opis	Vyhodnotí znalosti študenta na základe faktov, priradených tomuto študentovi.	
Priorita	vysoká	
Vstupy	Študent, ktorého znalosti sa budú vyhodnocovať	
Výstupy	Usporiadané dvojice (znalosť, úroveň znalosti)	
Vstupné podmienky	Študent má priradené nejaké fakty. Tieto fakty majú priradené kľúčové slová a ich váhy. V systéme sú evidované znalosti, ku ktorým sú priradené kľúčové slová a ich váhy.	
Výstupné podmienky		
Používatelia		
Postupnosť krokov	**Krok**	**Činnosť**
	1	Iterujeme cez všetky fakty daného študenta.
	2	Pre každý fakt nájdeme znalosti, s ktorými súvisí. To zistíme na základe prieniku kľúčových slov, ktoré definujú daný fakt a zároveň znalosť.
	3	K zatiaľ vypočítanej úrovni znalosti pripočítame hodnotu: znalosť.vahaZnalosti * fakt.vahaFaktu.
	4	Algoritmus vráti usporiadané dvojice (znalosť, úroveň znalosti)

Tab. 3.7 Prípad použitia UC07 – Zobrazenie profilu študenta

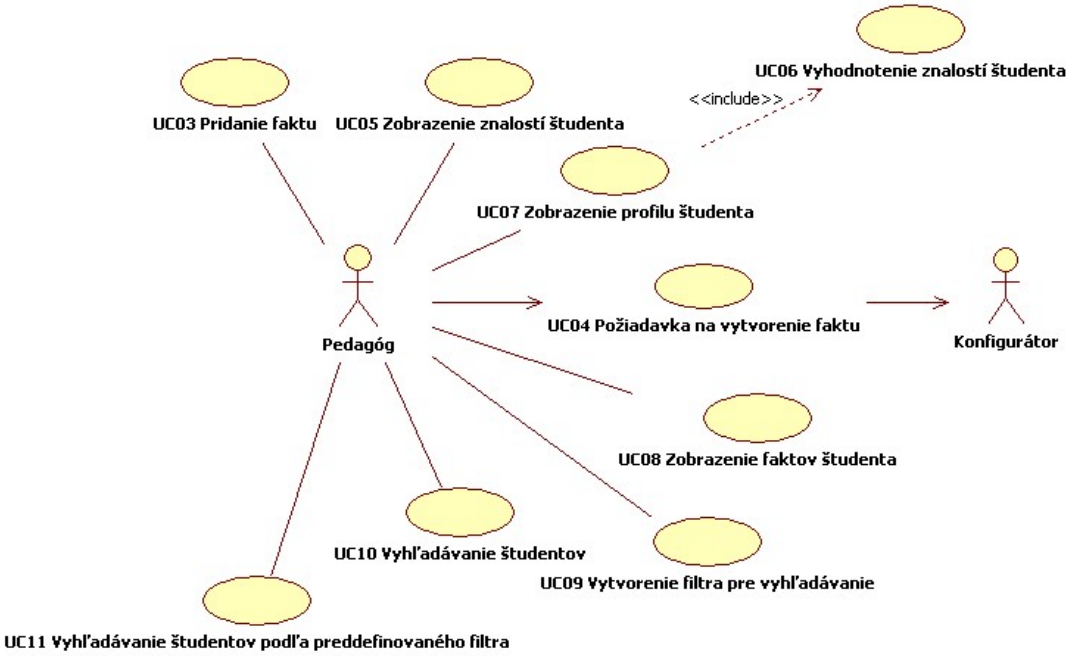
Zobrazenie profilu študenta		
Identifikátor	UC07	
Názov	Zobrazenie profilu študenta	
Opis	Zobrazí osobný profil daného študenta. Osobný profil obsahuje informácie ako meno, stupeň štúdia, odbor, fotka, informácie o sebe, e-mail, odkaz na svoj web a pod.	
Priorita	vysoká	
Vstupy	Identifikátor študenta, ktorého profil sa má zobraziť.	
Výstupy	Osobný profil študenta.	
Vstupné podmienky	Používateľ je prihlásený a má aspoň jednu z rolí Študent, Pedagóg. Ak má rolu Externý používateľ, tak daný študent musí mať povolené zobrazovanie informácií pre verejnosť.	
Výstupné podmienky		
Používatelia	Študent, Pedagóg, Externý používateľ	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ zvolí študenta, ktorého profil chce zobraziť.
	2	Systém zobrazí osobný profil študenta.

Tab. 3.8 Prípad použitia UC08 – Zobrazenie faktov študenta

Zobrazenie faktov študenta		
Identifikátor	UC08	
Názov	Zobrazenie faktov študenta	
Opis	Zobrazí fakty priradené danému študentovi.	
Priorita	vysoká	
Vstupy	Študent, ktorého fakty sa majú zobraziť.	
Výstupy	Fakty študenta.	
Vstupné podmienky	Používateľ je prihlásený a má aspoň jednu z rolí Študent, Pedagóg. Ak je používateľ prihlásený ako Externý používateľ, tak daný študent musí mať povolené zobrazovanie informácií pre verejnosť.	
Výstupné podmienky		
Používatelia	Študent, Pedagóg, Externý používateľ	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ zvolí študenta, ktorého fakty chce zobraziť.
	2	Systém zobrazí fakty študenta rozdelené do skupín podľa faktov. Pri jednotlivých faktoch zobrazí aj kľúčové slová a ich váhy.

3.5.2 Prípady použitia pre rolu pedagóg

Pre rolu pedagóg sme identifikovali prípady použitia zobrazené na Obr. 3.3.



Obr. 3.3 Diagram prípadov použitia pre rolu pedagóg

V nasledujúcich tabuľkách uvádzame podrobný opis prípadov použitia pre rolu pedagóg.

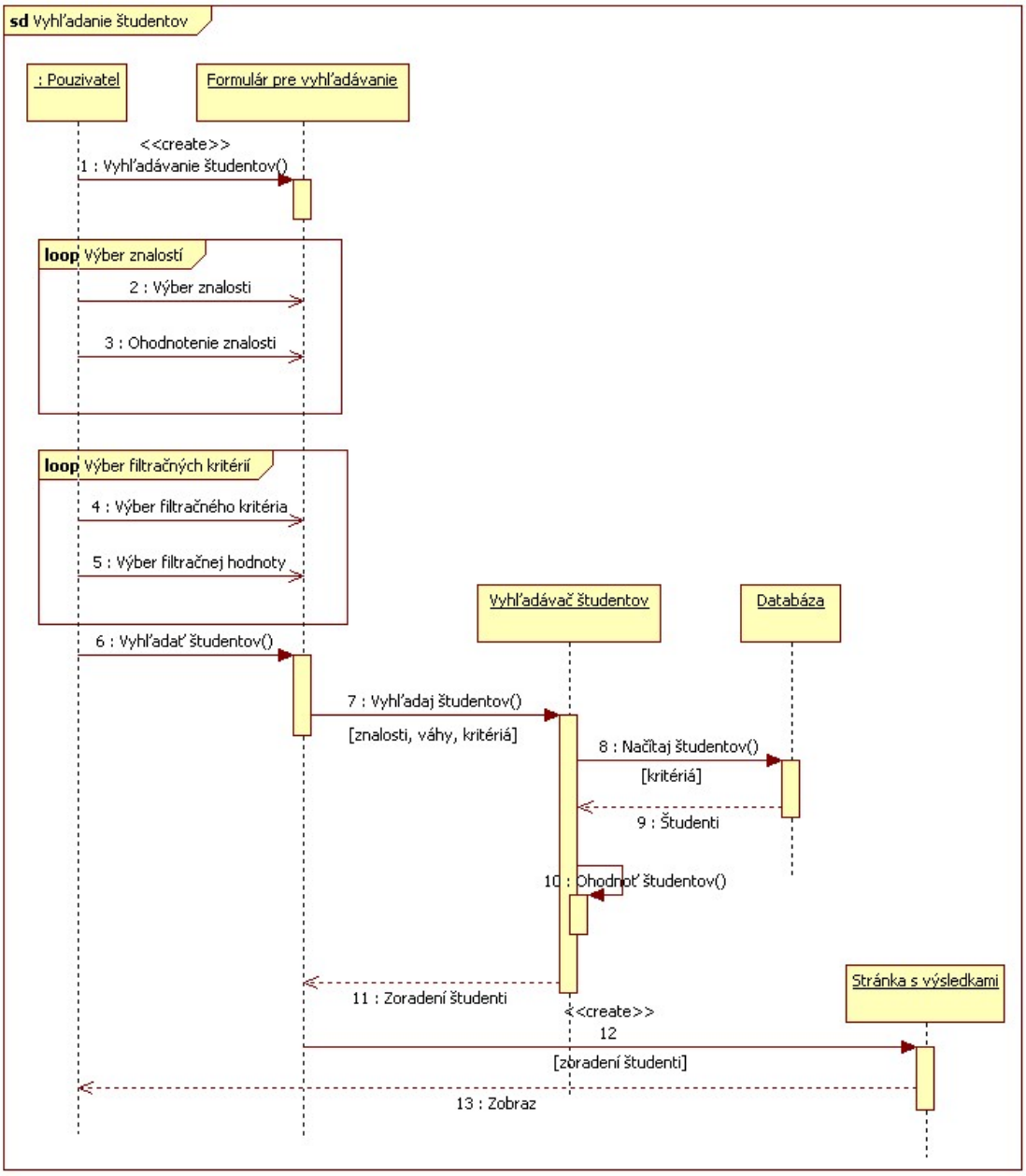
Tab. 3.9 Prípad použitia UC09 – Vytvorenie filtra pre vyhľadávanie

Vytvorenie filtra pre vyhľadávanie		
Identifikátor	UC09	
Názov	Vytvorenie filtra pre vyhľadávanie	
Opis	Umožňuje vytvoriť preddefinované vyhľadávacie pravidlá.	
Priorita	nízka	
Vstupy	Kritéria pre filtrovanie výsledkov. Kritéria, podľa ktorých sa majú zobrazené výsledky zotriediť.	
Výstupy		
Vstupné podmienky		
Výstupné podmienky	Filter je perzistentný.	
Používatelia	Pedagóg, Konfigurátor	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce vytvoriť nový filter.
	2	Systém mu zobrazí formulár s položkami pre filtrovanie a zoraďovanie výsledkov
	3	Používateľ vyberie položky, podľa ktorých chce filtrovať a triediť.
	4	Používateľ zvolí filtrovacie a triediace kritériá (<, >, =, má, nemá, …)
	5	Systém uloží tento filter do databázy.

Tab. 3.10 Prípad použitia UC10 – Vyhľadávanie študentov

Vyhľadávanie študentov		
Identifikátor	UC10	
Názov	Vyhľadávanie študentov	
Opis	Umožňuje vyhľadávať študentov podľa zadaných kritérií.	
Priorita	vysoká	
Vstupy	Kritéria pre filtrovanie výsledkov. Kritéria, podľa ktorých sa majú zobrazené výsledky zotriediť.	
Výstupy	Zoznam študentov, spĺňajúcich dané kritériá, zoradených podľa triediacich kritérií. V prípade, že používateľom je externý používateľ, výsledný zoznam neobsahuje študentov, ktorí nemajú povolené zobrazovanie informácií pre verejnosť.	
Vstupné podmienky	Prihlásený používateľ má rolu Pedagóg alebo Externý používateľ.	
Výstupné podmienky		
Používatelia	Pedagóg, Externý používateľ	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce vyhľadávať študentov.
	2	Systém mu zobrazí formulár s položkami pre filtrovanie a zoraďovanie výsledkov.
	3	Používateľ vyberie položky, podľa ktorých chce filtrovať a triediť výsledky.
	4	Používateľ zvolí filtrovacie a triediace kritériá (<, >, =, má, nemá, …)
	5	Systém zobrazí zoznam študentov, spĺňajúcich dané kritériá, zoradených podľa triediacich kritérií. V prípade, že používateľom je externý používateľ, výsledný zoznam neobsahuje študentov, ktorí nemajú povolené zobrazovanie informácií pre verejnosť

Na Obr. 3.4 je zobrazený sekvenčný diagram pre prípad použitia UC10 – Vyhľadávanie študentov.



Obr. 3.4 Sekvenčný diagram pre prípad použitia UC10 – Vyhľadávanie študentov

Tab. 3.11 Prípad použitia UC11 – Vyhľadávanie študentov podľa preddefinovaného filtra

Vyhľadávanie študentov podľa preddefinovaného filtra	
Identifikátor	UC11
Názov	Vyhľadávanie študentov podľa preddefinovaného filtra
Opis	Umožňuje vyhľadávať študentov podľa preddefinovaného filtra.
Priorita	nízka
Vstupy	Preddefinovaný filter.
Výstupy	Zoznam študentov, spĺňajúcich dané kritériá, zoradených podľa triediacich kritérií.
Používatelia	Pedagóg, Externý používateľ

Postupnosť krokov	Krok	Činnosť
	1	Používateľ vyberie, že chce vyhľadávať študentov podľa preddefinovaného filtra.
	2	Systém mu ponúkne preddefinované filtre v systéme.
	3	Používateľ vyberie filter ktorý chce použiť.
	4	Používateľ môže upraviť filtrovacie kritériá
	5	Systém zobrazí zoznam študentov, spĺňajúcich dané kritériá, zoradených podľa triediacich kritérií.

3.5.3 Prípady použitia pre rolu administrátor

Pre rolu administrátor sme identifikovali prípady použitia zobrazené na Obr. 3.5.

Obr. 3.5 Diagram prípadov použitia pre rolu administrátor

V nasledujúcich tabuľkách uvádzame podrobný opis prípadov použitia pre rolu administrátor.

Tab. 3.12 Prípad použitia UC12 – Správa používateľských účtov

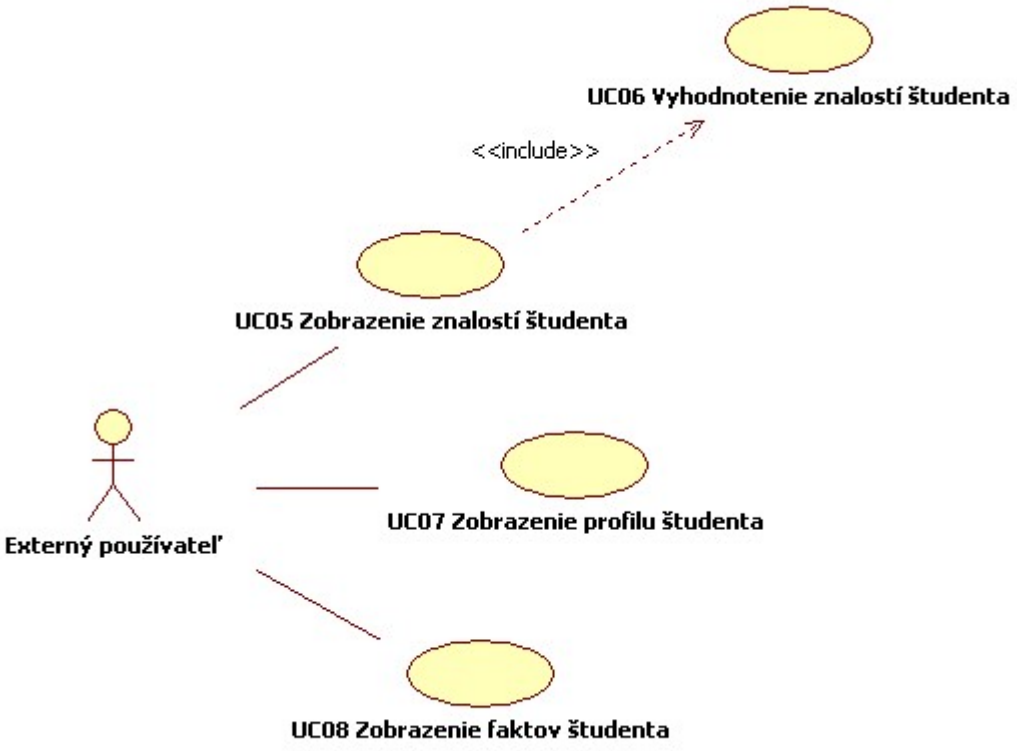
Správa používateľských účtov	
Identifikátor	UC12
Názov	Správa používateľských účtov
Opis	CRUD operácie nad účtami používateľov. Tento prípad použitia umožňuje odstraňovať nesprávne, nepravdivé alebo nevhodné fakty o študentoch. Tento prípad použitia tiež umožňuje vygenerovanie nového hesla k používateľskému účtu.
Priorita	stredná
Vstupné podmienky	
Výstupné podmienky	Uložené zmeny sú perzistentné.
Používatelia	Administrátor

Tab. 3.13 Prípad použitia UC13 – Import údajov z DB AISu

Import údajov z DB AISu		
Identifikátor	UC13	
Názov	Import údajov z DB AISu	
Opis	Importuje do systému údaje z akademického informačného systému.	
Priorita	stredná	
Vstupné podmienky	Dá sa pripojiť do databázy AIS.	
Výstupné podmienky	Údaje z AISu sú importované a transformované do databázy nášho systému vo forme faktov. Pôvodné fakty v systéme sú nezmenené (za predpokladu, že sa nezmenili v AISe).	
Používatelia	Administrátor	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ zvolí, že chce importovať údaje z AISu.
	2	Systém sa pripojí do databázy AISu.
	3	Systém zistí, ktoré údaje sa zmenili a ktoré pribudli od posledného importovania. Tieto údaje transformuje do faktov a pridá ich do systému použitím prípadu použitia Pridanie faktu.

3.5.4 Prípady použitia pre rolu externý používateľ

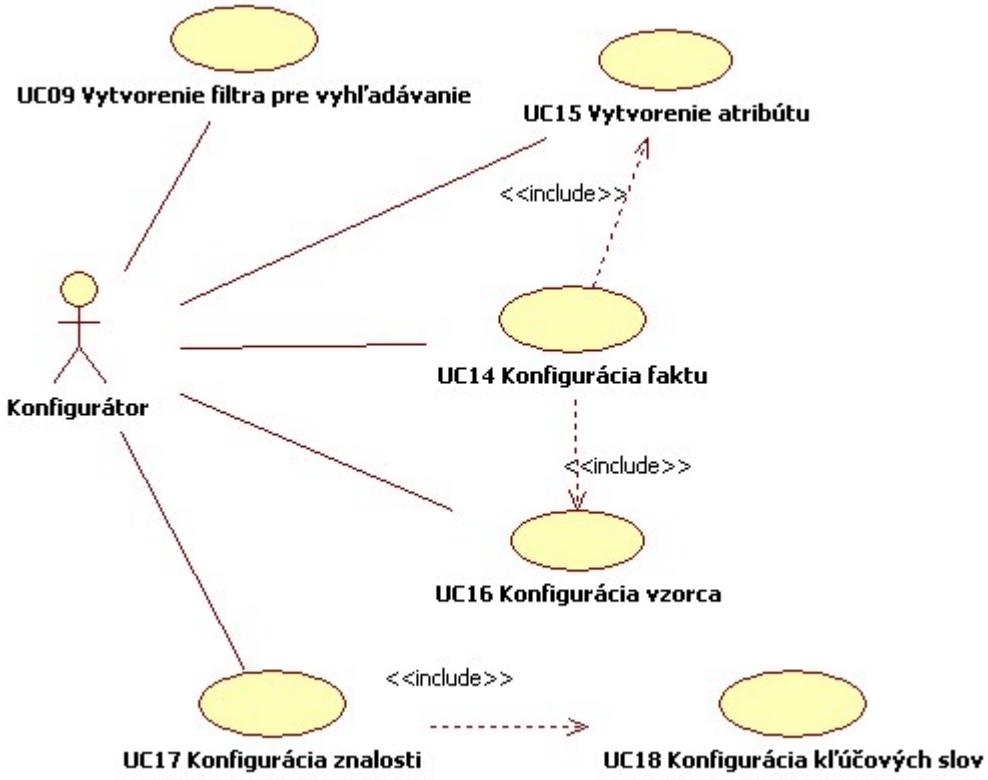
Pre rolu externý používateľ sme identifikovali prípady použitia zobrazené na Obr. 3.6.



Obr. 3.6 Diagram prípadov použitia pre rolu externý používateľ

3.5.5 Prípady použitia pre rolu konfigurátor

Pre rolu konfigurátor sme identifikovali prípady použitia zobrazené na Obr. 3.7.



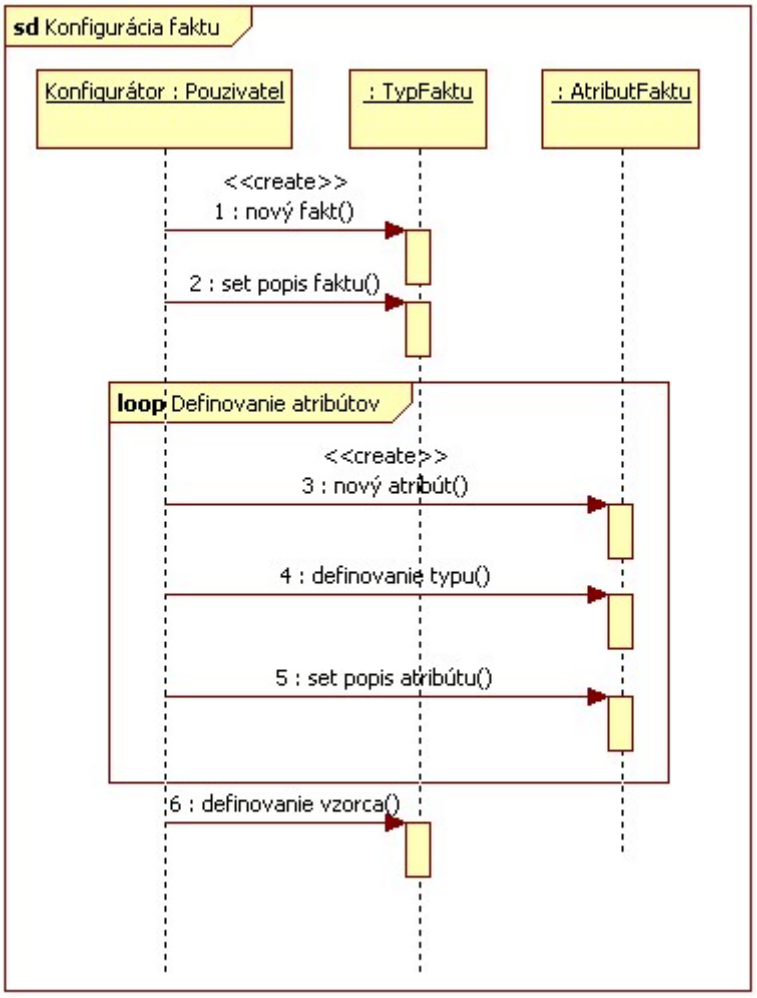
Obr. 3.7 Diagram prípadov použitia pre rolu konfigurátor

V nasledujúcich tabuľkách uvádzame podrobný opis prípadov použitia pre rolu konfigurátor.

Tab. 3.14 Prípad použitia UC14 – Konfigurácia faktu

Konfigurácia faktu		
Identifikátor	UC14	
Názov	Konfigurácia faktu	
Opis	Umožňuje pridávať a upravovať typy faktov.	
Priorita	stredná	
Vstupy	Názov a popis faktu.	
Výstupy	Nové fakty v databáze.	
Používatelia	Konfigurátor	
Postupnosť krokov pre pridanie nového faktu	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce pridať nový typ faktu.
	2	Používateľ pomenuje a popíše tento typ faktu. Zadefinuje atribúty faktu pomocou prípadu použitia UC15 Konfigurácia atribútov
	3	Používateľ vytvorí vzorec pre výpočet hodnotnosti faktu pomocou prípadu použitia UC16 Konfigurácia vzorca.
	4	Systém uloží nový typ faktu do databázy.

Na Obr. 3.8 je zobrazený sekvenčný diagram pre prípad použitia UC14 – Konfigurácia faktu.



Obr. 3.8 Sekvenčný diagram pre prípad použitia UC14 – Konfigurácia faktu

Tab. 3.15 Prípad použitia UC15 – Vytvorenie atribútu

Konfigurácia atribútu		
Identifikátor	UC15	
Názov	Vytvorenie atribútu	
Opis	Umožňuje pridávať a upravovať atribúty faktu.	
Priorita	stredná	
Vstupy	Názov a popis a dátový typ atribútu.	
Výstupy	Nové atribút v databáze.	
Používatelia	Konfigurátor	
Postupnosť krokov pre pridanie nového atribútu	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce pridať nový atribút.
	2	Používateľ pomenuje a popíše tento atribút. Zadefinuje dátový typ tohto atribútu
	3	Systém uloží nový atribút do databázy.

Tab. 3.16 Prípad použitia UC16 – Konfigurácia vzorca

Konfigurácia vzorca		
Identifikátor	UC16	
Názov	Konfigurácia vzorca	
Opis	Umožňuje vytvoriť vzorec pre výpočet hodnotnosti faktu.	
Priorita	nízka	
Vstupy	Atribúty faktu, pre ktorý sa vzorec vytvára.	
Výstupy	Vytvorený vzorec	
Používatelia	Konfigurátor	
Postupnosť krokov pre pridanie novej znalosti	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce pridať novú znalosť.
	2	Používateľ pomocou rozhrania vytvorí vzorec pre výpočet hodnotnosti faktu z jeho atribútov.
	3	Systém uloží daný vzorec k vstupnému faktu.

Tab. 3.17 Prípad použitia UC17 – Konfigurácia znalosti

Konfigurácia znalosti		
Identifikátor	UC17	
Názov	Konfigurácia znalosti	
Opis	Umožňuje pridávať a upravovať znalosti v systéme.	
Priorita	nízka	
Vstupy	Názov a popis znalosti. Kľúčové slová, viažuce sa na túto znalosť a ich váhy.	
Výstupy	Nová znalosť v databáze.	
Používatelia	Konfigurátor	
Postupnosť krokov pre pridanie novej znalosti	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce pridať novú znalosť.
	2	Používateľ pomenuje a popíše túto znalosť. Opíše túto znalosť pomocou kľúčových slov, ktorým pridelí váhy pomocou prípadu použitia Konfigurácia kľúčových slov.
	3	Systém uloží novú znalosť do databázy.

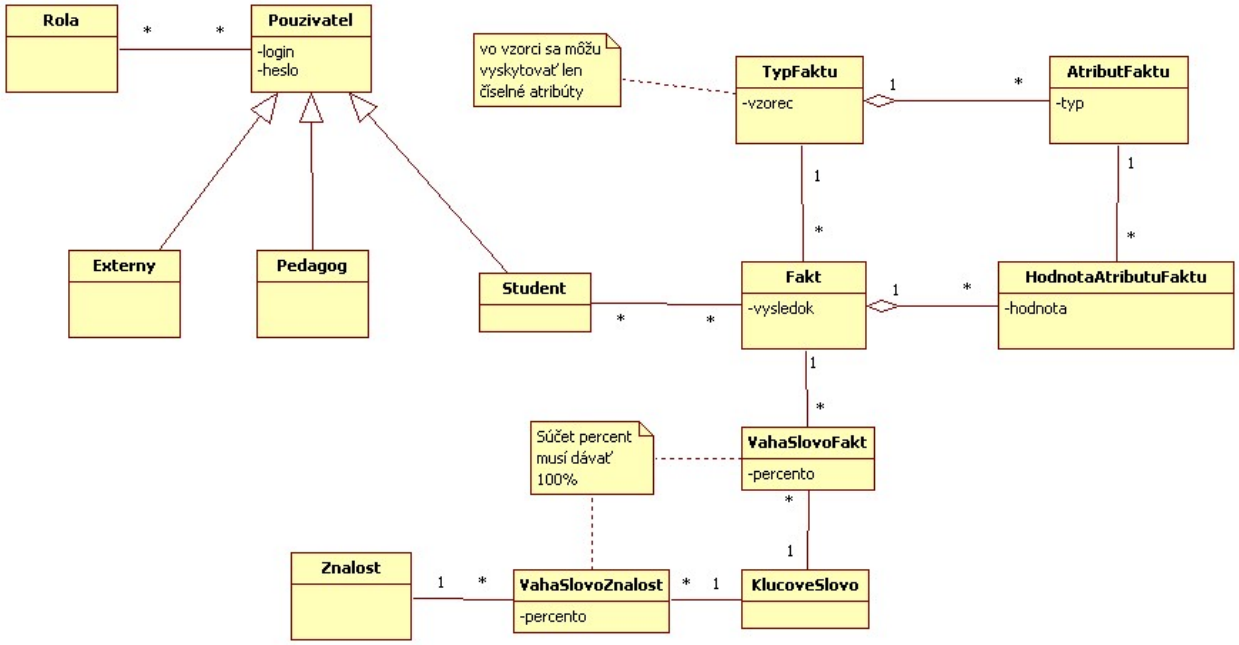
Tab. 3.18 Prípad použitia UC18 – Konfigurácia kľúčových slov

Konfigurácia kľúčových slov	
Identifikátor	UC18
Názov	Konfigurácia kľúčových slov
Opis	Umožňuje pridávať a upravovať kľúčové slová a ich váhy k danej znalosti.
Priorita	nízka
Vstupy	Identifikátor znalosti.
Výstupy	
Používatelia	Konfigurátor

Postupnosť krokov pre pridanie novej znalosti	Krok	Činnosť
	1	Používateľ vyberie zadá kľúčové slová a ich váhy smerom ku vstupnej znalosti.
	2	Systém uloží kľúčové slová a ich váhy do databázy.

3.6 Logický model údajov

Logický model nášho systému je možné rozdeliť do troch častí, z ktorých každá pozostáva z viacero navzájom súvisiacich entít. Prvá množina slúži na uchovávanie používateľov, druhá na evidenciu a pridávanie faktov o študentoch, a tretia na evidenciu znalostí. Diagram logického modelu údajov je zobrazený na Obr. 3.9.



Obr. 3.9 Logický model údajov systému Znalec

Základnou entitou prvej množiny je abstraktná entita ***Pouzivatel***, ktorej hlavnými atribútmi sú **prihlasovacie meno** a **heslo**, ktoré slúžia na autorizáciu vstupu používateľa do systému. Používatelia môžu mať priradené viaceré roly, opísané entitou **Rola**. Od entity ***Pouzivatel*** sú odvodené jednotlivé skupiny používateľov. Prvou je entita **Externy**, ktorá reprezentuje externého používateľa. Ním môže byť napríklad pracovník externej firmy hľadajúci budúcich zamestnancov. Druhou je **Pedagog**, ktorá slúži na uchovávanie pedagógov. Poslednou entitou odvodenou od ***Pouzivatel***-a je **Student**. Študent je špeciálne vyčlenený, aby sme mohli evidovať, s akými faktami je asociovaný.

Druhá množina entít opisuje fakty súvisiace so študentmi. Entita **TypFaktu** určuje druh faktu, ktorý môžeme o študentovi evidovať, ako napríklad účasť na projekte, konferencií alebo jeho záverečné práce. Každý takýto **TypFaktu** sa dá opísať viacerými atribútmi. Pomocou číselných atribútov je definovaný vzorec pre výpočet „kvality“ daného faktu. Atribút typu faktu je reprezentovaný entitou **AtributFaktu** (napr. pre typ faktu Projekt to môžu byť dĺžka projektu, počet zúčastnených ľudí, rola v tíme, programovací jazyk). Entita AtributFaktu má atribút typ na určenie, aký dátový typ reprezentuje. Entita **Fakt** predstavuje inštanciu typu faktu (napr. účasť na projekte XYZ). Táto inštancia je tvorená hodnotami jej atribútov. Hodnoty sú uložené pomocou entity **HodnotaAtributuFaktu**, ktorá reprezentuje hodnotu typu atribútu (napr. dĺžka projektu – 5 mesiacov). Pomocou vzorca pre príslušný typ faktu sa z nich počíta miera vplyvu daného faktu

na znalosti študenta. Táto miera sa ukladá v atribúte vysledok. Takáto abstrakcia typov faktov a ich atribútov nám umožňuje konfigurovať systém bez potreby programovania.

Entita **Fakt** je prepojená s entitou **Student** s kardinalitou M ku N, pretože k študentovi sa môže viazať viacero faktov a jeden fakt sa môže viazať k viacerým študentom (napr. účasť na spoločnom školskom projekte). Entita **KlucoveSlovo** predstavuje evidované kľúčové slovo. Pomocou väzobnej entity **VahaSlovoFakt** sú namapované kľúčové slová na jednotlivé fakty pomocou percent. Súčet percent kľúčových slov daného faktu musí dávať 100 %.

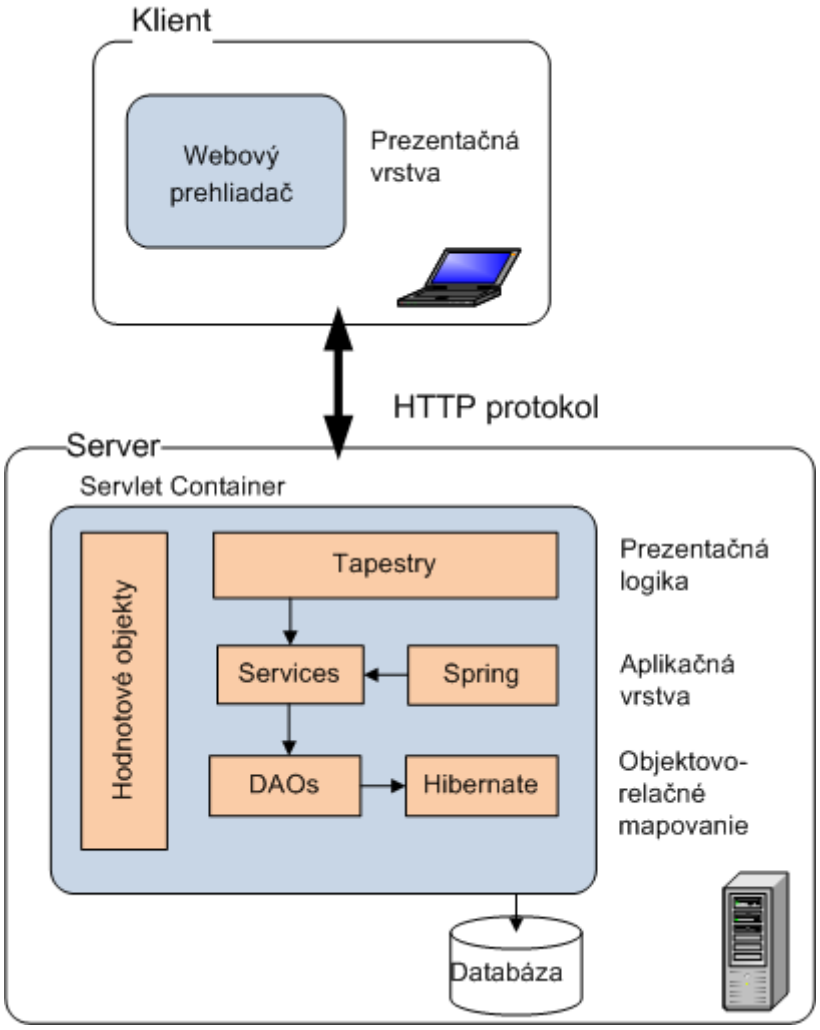
Posledné skupina pozostáva z entít **KlucoveSlovo, Znalost** a **VahaSlovoZnalost**. **VahaSlovoZnalost** slúži na prepojenie kľúčových slov so znalosťami a každému takémuto prepojeniu ešte pridáva aj percento, akým sa to-ktoré kľúčové slovo podieľa na jednotlivých znalostiach. Opäť platí, že súčet musí dávať 100 %.

4 Hrubý návrh riešenia

V tejto kapitole opisujeme návrh architektúry navrhovaného systému pre bázu znalostí a zručností študentov.

4.1 Aplikácia na báze JEE

Navrhovaný systém sme sa rozhodli implementovať ako webovú aplikáciu na platforme Java. Systém je založený na trojvrstvovej architektúre JEE. Na implementáciu tejto architektúry použijeme voľne dostupné aplikačné rámce a knižnice s otvoreným zdrojovým kódom. Návrh architektúry je uvedený na Obr. 4.1.



Obr. 4.1 Návrh architektúry systému Znalec

Základnou architektúrou je klient-server. Používateľovi je „viditeľný“ len tzv. tenký klient, ktorý sa skladá z prezentačnej vrstvy čo predstavuje webový prehliadač. Oveľa zaujímavejšou je serverová časť, ktorá sa skladá z prezentačnej logiky, aplikačnej vrstvy a vrstvy pre objektovo-relačné mapovanie. V nasledujúcich kapitolách opíšeme jednotlivé vrstvy.

4.2 Prezentačná vrstva

Úlohou prezentačnej vrstvy je poskytnúť používateľovi rozhranie na prácu s aplikáciou. Ako prezentačnú vrstvu sme si zvolili webové rozhranie. Na implementáciu tejto vrstvy sme sa rozhodli použiť voľne

dostupný javovský rámec s otvoreným zdrojovým kódom Tapestry[4]. Je to rámec na tvorbu dynamických webových aplikácií založený na jazyku Java. Webová aplikácia je rozdelená na stránky a tie sa skladajú z komponentov. To umožňuje vysokú znovupoužiteľnosť. Tapestry disponuje sadou často používaných komponentov, je však rozšíriteľný, takže každý si môže prispôsobiť existujúce komponenty alebo vytvoriť vlastné. Tento rámec je plne objektovo orientovaný, takže každá stránka a každý komponent je reprezentovaný Java triedou. To umožňuje programátorom sústrediť na funkcionalitu a nemusia sa starať o nízkoúrovňové procesy ako parsovanie parametrov z URL alebo vytváranie HTTP požiadaviek a odpovedí. Stránky a komponenty sú založené na HTML. HTML slúži však iba ako šablóna, ktorá sa za behu aplikácie môže meniť (a spravidla mení). Previazanie medzi HTML šablónou a Java kódom je zabezpečené pomocou metadát. Tieto metadáta môžu mať viacero foriem: XML dokument, špeciálne značky a atribúty v HTML alebo Java anotácie v kóde.

Tapestry zabraňuje „znovuvynachádzaniu“ kolesa tým, že zabezpečuje často sa opakujúce činnosti ako validáciu vstupu od používateľa alebo internacionalizáciu aplikácie. Trochu nezvyčajnou črtou Tapestry je princíp inverznej kontroly (angl. *Inversion of Control*). Je to opačný princíp ako pri bežných knižniciach, kde programátor volá služby knižnice. Pri rámcoch je to naopak – rámec „volá“ služby (objekty, metódy), ktoré napísal programátor. Preto väčšina Tapestry tried, ktoré budeme písať bude abstraktná, pretože Tapestry od nich za behu dedí a dopĺňa ďalšie vlastnosti.

Prezentačná vrstva len sprostredkuje interakciu s používateľom, aplikačná logika sa však skrýva v aplikačnej vrstve, ktorej služby používa prezentačná vrstva.

4.3 Aplikačná vrstva

Aplikačná vrstva predstavuje aplikačnú logiku, voľne povedané, v nej sa nachádza to dôležité, o čom vlastne aplikácia je. Táto vrstva sa dá reprezentovať ako množina služieb, ktoré poskytuje prezentačnej vrstve. Ide teda o použitie vzoru Facade. Služby sú prezentované ako metódy, ktoré sú logicky zoskupené do celku, ktorý je opísaný pomocou Java rozhrania. Ako príklad môže slúžiť bankový účet. Služby bankového účtu môžu byť opísané Java rozhraním *AccountService*, ktoré má metódy ako *createAccount*, *deposit*, *withdraw* a pod. Prezentačná vrstva „vidí“ len tieto rozhrania, ich implementácia je však vnútornou záležitosťou aplikačnej vrstvy. Tento princíp sa nazýva programovanie voči rozhraniam a umožňuje vymieňanie implementácie bez toho, aby si to klient (používateľ rozhraní) „všimol“. To napríklad zjednodušuje testovanie, kedy sa aplikácia môže javiť ako plne funkčná a pri tom napr. vôbec nemusí byť pripojená do databázy.

Na túto činnosť slúži aplikačný rámec Spring[5]. Pomocou neho môžeme konfigurovať celú aplikáciu a spájať jednotlivé moduly. Spring je veľmi silný nástroj a poskytuje veľa užitočných funkcií. My ho však v našom systéme budeme najviac využívať na konfiguráciu servisných služieb, ich prepojenie na objektovo-relačný mapovač a na testovanie modulov (angl. *unit testing*). Na konfiguráciu sa používa, ako je to pri JEE zvykom, formát XML.

4.4 Objektovo-relačné mapovanie

Pod týmto pojmom sa skrýva pomerne jednoduchá vec. Ide o mapovanie objektov do relačnej databázy. Ako nástroj na toto mapovanie sme si zvolili Hibernate[6], ktorý sa v tejto oblasti stal už de facto štandardom na platforme Java. Zjednodušene sa dá povedať, že každá Java trieda je reprezentovaná tabuľkou v relačnej databáze, každý stĺpec tabuľky reprezentuje atribút triedy a jeden riadok predstavuje konkrétny objekt, teda inštanciu Java triedy. Opäť, na takéto mapovanie, sa používajú metadáta. Tie sú najčastejšie vo forme XML, ale môžeme použiť aj Java anotácie.

Výhodou tohto prístupu je, že programátor pracuje s Java triedami a nemusí sa zaoberať písaním SQL selectov na bežné CRUD operácie. Používatelia Hibernate síce nepoužívajú jazyk SQL, ale jazyk HQL , ktorý je SQL veľmi podobný. Výhodou je, že sa používa tzv. bodková notácia namiesto SQL príkazov join.

[4] http://tapestry.apache.org
[5] http://www.springframework.org
[6] http://www.hibernate.org

Tieto dopyty podporujú aj polymorfizmus, čo sa dá niekedy s výhodou využiť. Ďalšou výhodou je, že Hibernate ukrýva databázový systém pred programátorom, takže ten vlastne ani nemusí vedieť, na akom databázovom systéme je aplikácia nasadená. Rovnako to umožňuje (teoreticky) vymeniť databázový systém bez toho, aby to nejako ovplyvnilo aplikáciu. Opäť ide o akúsi transparentnosť keď jedna vrstva skrýva svoje implementačné detaily a poskytuje svojmu klientovi len služby.

Ďalšou abstraktnou vrstvou je použitie Java EE vzoru DAO. DAO je trieda, ktorá predstavuje abstrakciu CRUD operácií, typické metódy DAO sú *save*, *update*, *delete*, *executeQuery* a pod.

4.5 Hodnotové objekty

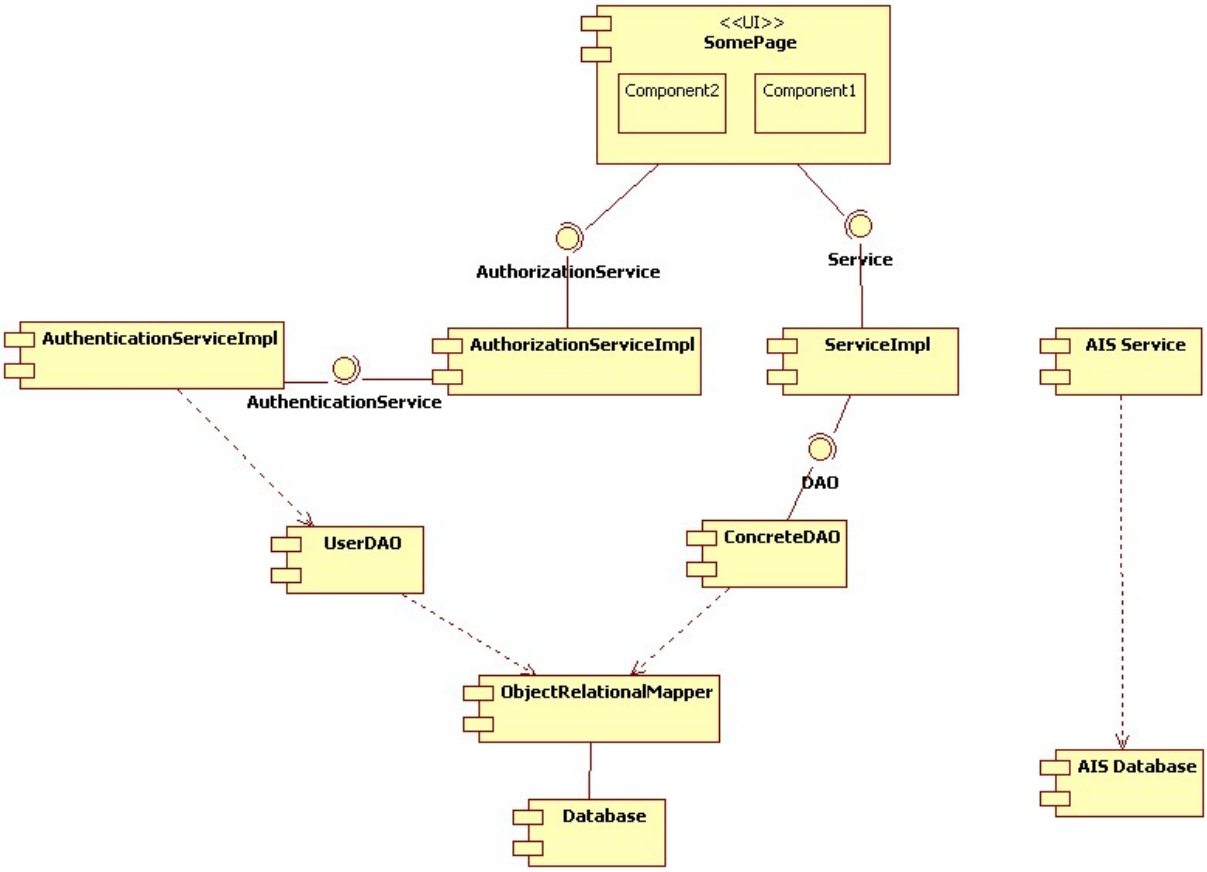
Hodnotové objekty (angl. *value objects*) tvoria doménový model aplikačnej oblasti. Sú to teda objekty aplikačnej domény, ktorú modelujeme. Ide napr. o objekty ako zmluva, faktúra, ale aj abstraktnejšie ako proces, udalosť a pod. Tieto objekty sa používajú naprieč celou aplikáciou, t.j. vo všetkých vrstvách. Tieto objekty sa ukladajú do databázy.

4.6 Servlet kontajner

Webové aplikácie na platforme Java sú väčšinou postavené na Servlet API, čo je zjednodušene povedané obsluha HTTP požiadaviek a vytváranie HTTP odpovedí. Každý servlet, musí bežať v nejakom „prostredí“. Toto prostredie sa nazýva servlet kontajner (angl. *servlet container*). Takýmto servlet kontajnerom je napríklad voľne dostupný aplikačný server Tomcat, ten poskytuje aj ďalšie služby.

4.7 Moduly systému

Na Obr. 4.2 obrázku opisujeme moduly systému a ich závislosti pomocou diagramu komponentov.



Obr. 4.2 Diagram komponentov opisujúci moduly systému Znalec

Na najvyššej úrovni je webová stránka prezentovaná Java triedou SomePage. Stránky však poskytujú len používateľské rozhranie, ale ich činnosť závisí od služieb. Služby modelujeme pomocou rozhrania Service. V aplikačnej vrstve sa nachádza konkrétna implementácia tejto služby ServiceImpl. Služba väčšinou potrebuje pristupovať do databázy, preto závisí od rozhrania DAO. DAO poskytuje službe databázové služby. Konkrétne DAO poskytuje implementáciu týchto služieb. Na implementáciu týchto služieb používa objektovo-relačný mapovač, ktorý pristupuje do databázy. Na zistenie, či má nejaký používateľ práva vidieť nejakú stránku ho potrebujeme autorizovať. Na tu slúži modul AuthorizationService, implentuje ho AuthorizationServiceImpl. Na to, aby sme mohli používateľa autorizovať, potrebujeme ho najsôr autentifikovať, teda zistiť, kto to vlastne je. Preto Autorizačná služba závisí od Autentifikačnej. Tá zisťuje používateľove role z databázy. Na to využíva UserDAO, ktorý do nej pristupuje pomocou ORM.

Na import údajov z AIS-u slúži modul AIS Service. Ten pristupuje priamo do databázy AIS-u.

5 Prototyp

V tejto časti opisujeme prototyp vytvorený v zimnom semestri.

5.1 Ciele prototypovania

Pri tvorbe prototypu sme si stanovili nasledovné ciele:

- vyskúšať tvorbu dynamických stránok pomocou Tapestry,
- overiť možnosť implementovania entít navrhnutých v logickom modeli,
- vytvoriť služby aplikačnej vrstvy zabezpečujúce biznis logiku a
- implementovať základný scenár použitia systému.

Výsledkom prototypovania by mali byť nájdené nedostatky v špecifikácii a návrhu systému. Tiež by sme chceli získať praktické poznatky o rozsahu systému a množstve práce potrebnej na dokončenie implementácie finálnej verzie systému v letnom semestri.

5.2 Vybrané časti systému na prototypovanie

Rozhodli sme sa vytvoriť evolučný prototyp, t. j. taký, ktorý v letnom semestri rozšírime do výsledného systému. V tejto časti sme sa rozhodli prototypovať aplikačnú vrstvu zabezpečujúcu biznis logiku a prezentačnú vrstvu umožňujúcu používateľovi prístup do systému. Prototyp nespolupracuje s databázou, všetky údaje sú vytvárané a uchovávané počas vykonávania v pamäti. Taktiež zatiaľ nespolupracuje so systémom AIS, nakoľko prístup k tomuto systému sme dostali až tesne pred koncom semestra. Prototyp funguje len lokálne na našich vývojových počítačoch. Nie je zatiaľ nasadený na aplikačnom serveri. Výsledný systém bude pracovať na serveri v rámci fakulty.

5.2.1 Scenár použitia systému realizovaný prototypom

Vytvorili sme základný scenár použitia výsledného systému, ktorého jednotlivé kroky sme sa rozhodli implementovať v prototype. Tento scenár nám poskytol užitočný pohľad na systém ako celok. Scenár pozostáva z nasledujúcich krokov:

1. Používateľ si zobrazí úvodnú stránku systému a prihlási sa.
2. Jeho domovská stránka mu poskytne menu pozostávajúce z nasledovných položiek: pridanie nového používateľa, editovanie profilu, správa faktov, zobrazenie znalostí, odhlásenie.
3. Prvá položka umožní pridať do systému nového používateľa vyplnením základných údajov.
4. V časti editovanie profilu má prihlásený používateľ možnosť upraviť svoje údaje.
5. V časti správa faktov si môže používateľ vytvárať, editovať a mazať svoje fakty.
6. Dôležitou položkou je tiež zobrazenie znalostí, ktorá používateľovi ukáže zoznam znalostí, ktoré mu systém na základe vyplnených faktov pridelil.

V tomto scenári ešte nerozlišujeme medzi používateľmi, t. j. neberieme do úvahy ich roly. Prototyp zatiaľ nie je konfigurovateľný, typy faktov a znalosti sú zadané v zdrojovom kóde.

5.2.2 Prípady použitia vybrané na prototypovanie

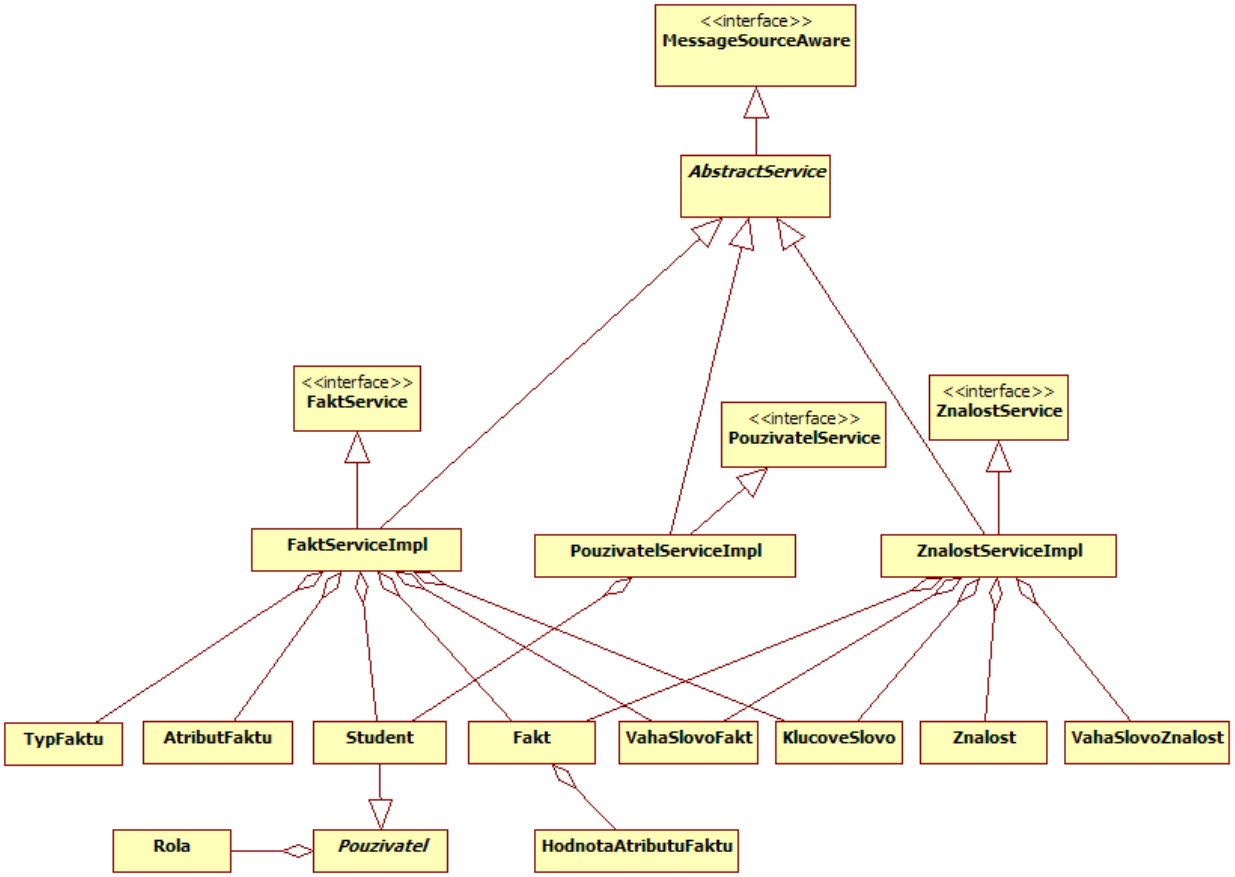
Pri vyberaní častí systému do prototypu sme postupovali podľa definovaných prípadov použitia. Vybrali sme tie, ktoré najviac demonštrujú zaujímavé vlastnosti systému. Niektoré z týchto prípadov použitia sme neimplementovali úplne, napr. z dôvodu, že obsahujú aj prístup do databázy alebo prístup k údajom

z externého fakultného systému. Tieto funkcie sme nahradili dočasnými údajmi, ktoré sme zapísali priamo v zdrojovom kóde prototypu. Na prototypovanie sme vybrali nasledujúce prípady použitia:

- UC01 Správa osobného profilu
- UC02 Správa faktov
- UC03 Pridanie faktu
- UC05 Zobrazenie znalostí študenta
- UC06 Vyhodnotenie znalostí študenta
- UC07 Zobrazenie profilu študenta
- UC08 Zobrazenie faktov študenta

5.3 Aplikačná vrstva

Ťažiskom prototypu je aplikačná vrstva. V nej sme sa snažili implementovať hlavné služby, ktoré táto vrstva poskytuje prezentačnej vrstve. Služby sme implementovali ako rozhrania a od nich odvodené triedy, ktoré zabezpečujú potrebnú funkcionalitu. Na implementáciu služieb sme použili rámec Spring, ktorý umožňuje jednoducho spájať vytvorené moduly do výsledného systému. Služby sme navrhli tak, aby bolo v ďalšej fáze možné nahradiť statické údaje údajmi z databázy. Na to bude stačiť implementovať príslušné metódy, ktoré vytiahnu údaje z databázy. Spracovanie údajov ani ich poskytnutie prezentačnej vrstve sa však nezmení. Na **Error! Reference source not found.** je zobrazený diagram tried aplikačnej vrstvy a ich závislosti. Hlavné časti tvoria triedy implementujúce rozhrania služieb: FaktServiceImpl, PouzivatelServiceImpl a ZnalostiServiceImpl. Tieto služby v sebe agregujú ďalšie triedy (TypFaktu, AtributFaktu, Student, Fakt, VahaSlovoFakt, KlucoveSlovo, Znalost, VahaSlovoZnalost), ktoré predstavujú implementáciu logického modelu údajov.



Obr. 5.1 Diagram tried aplikačnej vrstvy

Tieto služby sú cez svoje rozhrania sprístupnené prezentačnej vrstve. Nasleduje stručný opis jednotlivých rozhraní.

5.3.1 FaktService

Toto rozhranie slúži na manipuláciu s typmi faktov v systéme, najmä na ich pridávanie. Umožňuje tiež pridávať a editovať fakty jednotlivým študentom. Prezentačnej vrstve poskytuje zoznam všetkých typov faktov, ktoré sú v systéme, ako aj zoznam faktov zadaného študenta. Fakty sú v tomto prípade zoskupené podľa typu.

5.3.2 PouzivatelService

Rozhranie PouzivatelService slúži na manipuláciu s používateľmi. Umožňuje pridať nového používateľa do systému a upraviť profil existujúceho používateľa. Taktiež poskytuje funkcionalitu autentifikácie používateľa, ktorá sa využíva pri procese prihlasovania sa do systému.

5.3.3 ZnalostService

Toto rozhranie slúži na získanie znalostí zadaného študenta. Porovnáva pri tom všetky kľúčové slová jednotlivých faktov študenta s kľúčovými slovami definovanými v znalostiach. Pri zhode priradí túto znalosť študentovi. V ďalšej časti implementácie dopracujeme výpočet miery získania danej znalosti, kedy budú do úvahy brané aj typy jednotlivých faktov a ich príspevky k znalostiam. Študent potom nebude mať pridelenú iba konkrétnu znalosť, ale aj percentuálnu mieru jej splnenia. To umožní usporadúvať študentov.

5.4 Prezentačná vrstva

Prezentačnú vrstvu tvoria dynamické HTML stránky, ktoré zobrazujú používateľovi informácie a prijímajú od neho príkazy a zadané údaje. Prezentačná vrstva využíva verejné metódy služieb z aplikačnej vrstvy, aby jej tak odovzdala údaje na spracovanie a dostala od nej výsledky. Využitím rámca Tapestry je z týchto výsledkov vygenerovaná dynamická stránka, ktorá je prostredníctvom aplikačného servera Tomcat zobrazená používateľovi.

5.5 Testovanie

V rámci testovania prototypu sme uplatnili dva prístupy:

- testovanie typu čierna skrinka a
- regresné testovanie.

V rámci testovania typu čierna skrinka sme postupne vykonávali jednotlivé kroky špecifikované v prototypovaných prípadoch použitia (s ohľadom na scenár prototypu). Takto sme odhalili chyby v dizajne, ako aj v aplikačnej logike, ktoré sme následne odstraňovali.

V rámci regresného testovania sme využili možnosti nástroja JUnit. Vytvorili sme niekoľko testov, ktoré testujú služby aplikačnej vrstvy. JUnit test vykonáva metódy vybranej služby s predpripravenými testovacími údajmi. Výsledné údaje, ktoré mu služba poskytne, porovnáva s očakávanými údajmi. Ak sa nezhodujú, skončí test s chybou. Nástroj JUnit umožňuje vytvoriť ľubovoľný počet testov a následne ich spúšťať buď jednotlivo alebo viacero naraz. Výhoda týchto testov spočíva v tom, že ich môžeme bezo zmien použiť pri implementovaní prístupu systému k databáze. Pri jeho implementovaní doplníme metódy služieb o získavanie údajov z databázy, vstupné a výstupné údaje sa však nezmenia. Toto považujeme za veľkú výhodu JUnit testov, pre ktorú sme sa ich rozhodli využiť.

5.6 Dosiahnuté výsledky

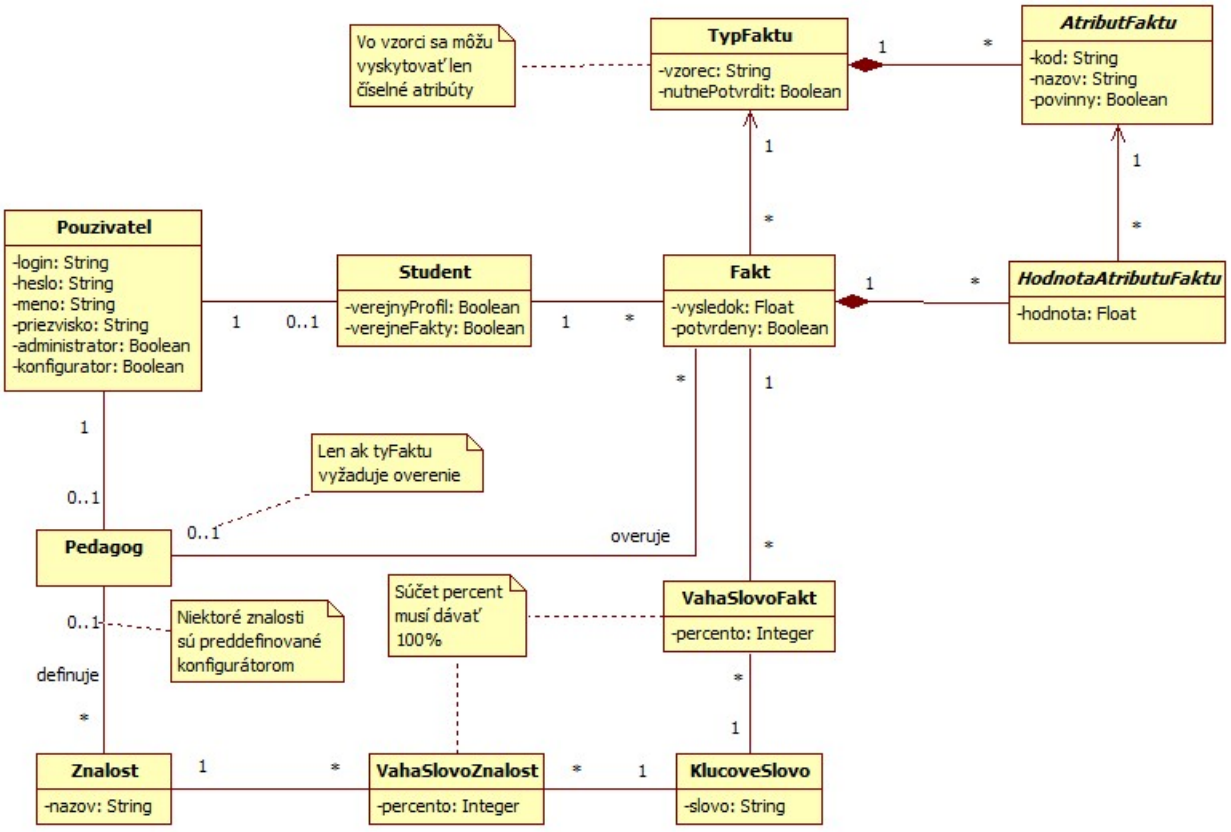
Môžeme skonštatovať, že sa nám podarilo splniť všetky ciele, ktoré sme si v úvode prototypovania stanovili. Podarilo sa nám vytvoriť funkčný prototyp demonštrujúci základné funkcie budúceho systému. Úspešne sme implementovali dynamické stránky prezentačnej vrstvy pomocou rámca Tapestry. Vytvorili sme hlavné služby aplikačnej vrstvy, ktoré v ďalšej fáze vývoja doplníme o prístup k údajom v databáze. Úspešne sme tiež implementovali jednotlivé entity logického modelu údajov. Jediný problém, ktorý sme v tejto časti identifikovali, spočíva v návrhu previazania znalostí a faktov na kľúčové slová prostredníctvom váh. V ďalšej časti budeme musieť rozhodnúť, ako má byť toto previazanie realizované (v prototype sme ho zatiaľ vyriešili staticky pri zadávaní znalostí). Taktiež sme narazili na problém identifikovania kľúčových slov z jednotlivých faktov. Oba problémy však pokladáme za ľahko vyriešiteľné v letnom semestri.

6 Implementácia

V tejto kapitole podrobne opisujeme implementáciu systému spolu s diagramami, ktoré lepšie znázorňujú hlavné komponenty systému.

6.1 Zmeny v logickom modeli

Oproti pôvodnému návrhu (opísaného v kapitole 3.6), ktorý sme vytvorili ešte v zimnom semestri, nastali v našom logickom modeli niektoré menšie zmeny. K týmto zmenám sme boli prinútení pristúpiť po niekoľkých problémoch, ktoré sa vyskytli pri implementácií, a týkali sa predovšetkým používateľa – doktoranda ako kombinácie pedagóga a študenta. Z pôvodnej štruktúry dedičnosti, kde boli entity *Student*, *Pedagog* a *Externy* zdedené od abstraktnej triedy *Pouzivatel* a ich role boli určené entitou *Rola* sme prešli na nový model zobrazený na Obr. 6.1. V ňom vznikla nová trieda *Pouzivatel*, ktorá informáciu o role používateľa uchováva v jeho boolovských atribútoch administrator a konfigurator. Dedičnosť sme nahradili asociáciami s triedami *Student* a *Pedagog***.** Ak je používateľ študentom, tak tento atribút bude mať priradeného študenta, inak ostane ako null, teda študentom nie je. Pedagóg je riešený rovnakým spôsobom.

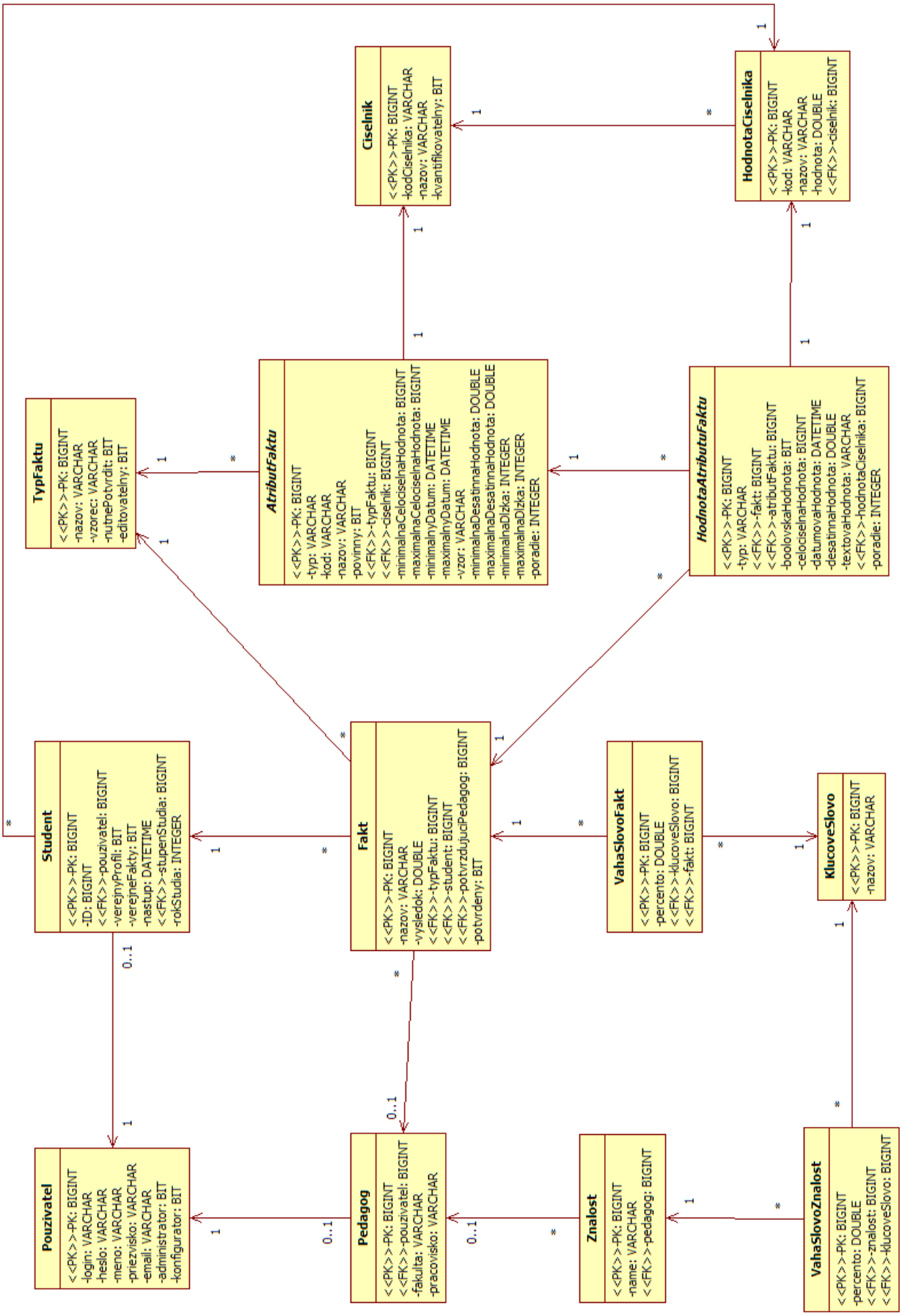


Obr. 6.1 Zmenený logický model.

K triede *Pedagog* pribudli asociácie s triedou *Znalost*, pretože každý pedagóg má možnosť definovania si vlastných znalostí, ktoré potom hľadá u študentov. Pedagóg je ešte spojený aj s triedou *Fakt*. Je to dané tým, že školské fakty, respektíve tie, ktoré sa nejako týkajú školy a študent ich získal pod pedagogickým vedením, je potrebné zo strany zodpovedajúceho pedagóga potvrdiť. Týmto sme dodatočne zvýšili hodnovernosť uvádzaných informácií. Zvyšok modelu ostal totožný s pôvodným návrhom.

6.2 Fyzický model

Fyzický model, zobrazený na Obr. 6.2, vznikol transformáciou logického modelu na Obr. 6.1. Je doplnený o entity *Ciselnik* a *HodnotaCiselniku* a všetky atribúty tak, aby zodpovedal štruktúre našej databázy, v ktorej máme uložené všetky naše dáta. Za zmienku stojí spôsob ukladania atribútov a ich hodnôt do databázy. Inštancie tried *AtributFaktu* sa mapujú na databázové tabuľky pomocou techniky Table per Hierarchy, t.j. pre celú hierarchiu tried sa použije jedna databázová tabuľka, ktorej stĺpce sú zjednotením všetkých atribútov tried danej hierarchie. Táto technika bola zvolená ako kompromis medzi redundanciou a efektivitou. Rovnakým spôsobom sa mapuje aj hierarchia tried *HodnotaAtributuFaktu*, t.j. pre každý dátový typ (string, int, double, atď.) existuje vlastný stĺpec. Štruktúra dedenia pre tieto triedy je opísaná na Obr. 6.3.



Obr. 6.2 Fyzický model údajov.

6.3 Zmeny v prípadoch použitia

Zmeny nastali aj v niektorých prípadoch použitia. Jednu podstatnú zmenu spôsobil fakt, že sme v našom dátovom modeli zrušili rolu Externý používateľ. Preto v každom prípade použitia, ktorý mohol takýto používateľ vykonať, už takáto možnosť nefiguruje.

Zmena nastala aj v UC03 Pridanie faktu. Pôvodný návrh počítal s možnosťou používania okrem študentmi aj pedagógmi. Túto možnosť sme zrušili, aby sme zbytočne nezaťažovali pedagógov. Zmena nastala aj pri zadávaní kľúčových slov, keď systém povolí zadať iba také, ktoré sú už v ňom definované. Ak si študent pridal fakt, ktorého typ je potrebné potvrdiť nejakým pedagógom, tak sa síce do databázy k študentovi uloží, ale do počítania znalostí sa zarátava až po odsúhlasení.

V UC05 Zobrazenie znalostí študenta došlo ku zmene privilégií. Znalosti si môže pozerať každý študent iba sám svoje a ani pedagóg si nemôže pozrieť všetky znalosti zvoleného študenta.

Rozsiahle zmeny podstúpil aj UC10 Vyhľadávanie študentov. Prvým rozdielom oproti návrhu je možnosť vyhľadávať jednej znalosti, teda nie ich nejaký prienik. To sa ale na druhú stranu kompenzuje prístupom pedagóga ku prípadu použitia UC17 Konfigurácia znalosti, v ktorom si môže pedagóg vytvoriť na základe kľúčových slov a ich ohodnotení ľubovoľnú znalosť a potom ju použiť na vyhľadávanie. Druhým rozdielom je absencia filtrov, ktoré sa teda nedajú použiť.

6.3.1 Nový prípad použitia

Pridali sme nový prípad použitia s názvom Konfigurácia číselníkov. Ten umožňuje konfigurátorovi vytvoriť nový číselník počas prevádzky systému, ktorý je následne možné použiť ako hodnotu atribútu zvoleného typu faktu.

Tab. 6.1 Prípad použitia UC19 – Konfigurácia číselníkov

Konfigurácia číselníkov		
Identifikátor	UC19	
Názov	Konfigurácia číselníkov	
Opis	Umožňuje pridávať a upravovať číselníky v systéme.	
Priorita	stredná	
Vstupy	Názov a kód číselníka. Názvy a kódy hodnôt, ktoré predstavujú možné hodnoty číselníka.	
Výstupy	Nový číselník v databáze.	
Používatelia	Konfigurátor	
Postupnosť krokov	**Krok**	**Činnosť**
	1	Používateľ vyberie, že chce pridať nový číselník.
	2	Používateľ pomenuje a zvolí si kód nového číselníka. Opíše hodnoty, ktoré bude môcť tento číselník nadobúdať.
	3	Systém uloží nový číselník do databázy.

6.4 Atribúty faktov a ich hodnoty

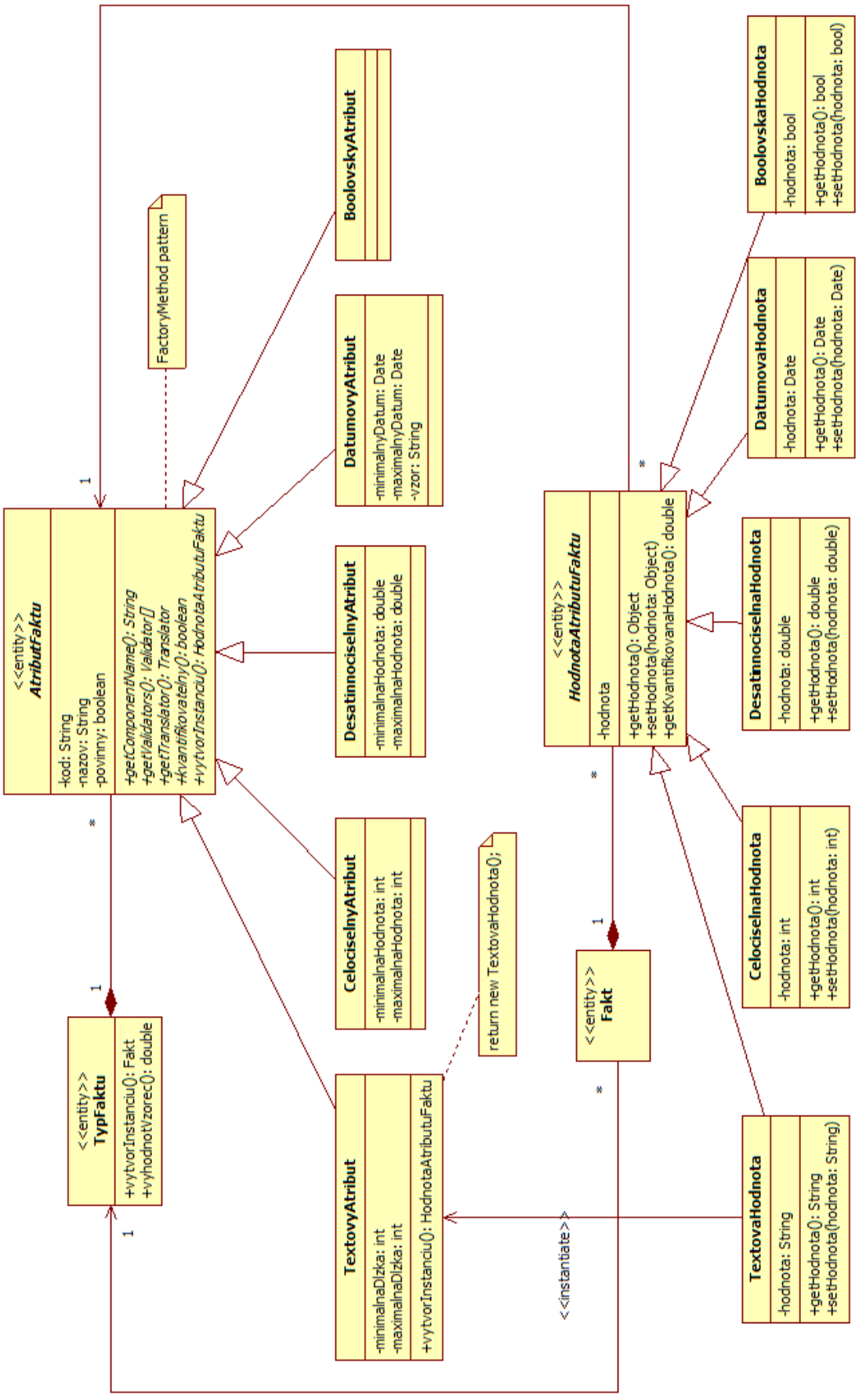
Každý typ faktu sa skladá z viacerých atribútov. Tieto atribúty môžu byť rôzneho typu – textové, číselné, dátumové, a pod. a platia pre ne rôzne obmedzenia – napr. minimálna, maximálna hodnota. Tejto metaúrovni musí vyhovovať reálna úroveň. To znamená, že fakt sa skladá z hodnôt týchto atribútov. Treba zaručiť, aby ku zvolenému typu atribútu prislúchala hodnota toho istého dátového typu (textovému atribútu textová hodnota, atď.) a aby platili obmedzenia. Riešenie je zobrazené na Obr. 6.3. Odlíšenie typov atribútov a ich správania je riešené pomocou dedičnosti a polymorfizmu.

Entita *AtributFaktu* je abstraktná trieda, ktorá slúži ako nadtrieda pre konkrétne atribúty faktov. Na jeho jednoznačnú identifikáciu slúži atribút *kód*. Reálny názov atribútu, ktorý sa zobrazuje vo formulároch, je uložený v atribúte *názov*. To, či je atribút potrebné vyplniť pri vypĺňaní formulára, je uložené v atribúte *povinný*. Táto trieda obsahuje len abstraktné metódy, ktoré potomkovia musia implementovať. Metóda

getComponentName() má vrátiť názov komponentu prezentačnej vrstvy, ktorý je zodpovedný za vyrenderovanie tohto atribútu. Metóda *getValidators()* vráti pole Validatorov, čo je Tapestry rozhranie používané na validáciu formulárových hodnôt. Takto môžu jednotlivé atribúty definovať svoje obmedzenia (minimálna, maximálna hodnota). Ďalšou metódou, ktorá súvisí s Tapestry, je *getTranslator()*. Tá vráti Tapestry rozhranie Translator, ktoré slúži na prekladanie reťazcových hodnôt na Java typy (keďže v HTML je všetko len reťazec). Má význam len pri niektorých typoch atribútov. Niektoré atribúty môžu kvantifikovať kvalitu faktu (dĺžka projektu, známka z predmetu) a teda sa môžu nachádzať vo vzorci pre daný fakt. Či je daný atribút kvantifikovateľný určuje metóda *kvanitifikovatelny()*. Aby bolo zaručené, že k danému typu atribútu sa vytvorí správna hodnota, bol použitý návrhový vzor Factory Method. Touto Factory Method je metóda *vytvorInstanciu()*, ktorá pre daný *AtributFaktu* vytvorí príslušnú triedu *HodnotaAtributuFaktu*, ktorá je akoby inštanciou daného atribútu faktu. Naznačené to je na triede *TextovyAtribut*, ktorého metóda vráti inštanciu triedy *TextovaHodnota*. Pri ostatných triedach je pre prehľadnosť diagramu táto metóda skrytá.

Od triedy *AtributFaktu* dedia triedy pre jednotlivé typy atribútov a implementujú jej abstraktné metódy. Rovnakú metódu *vytvorInstanciu()* má aj trieda *TypFaktu,* a tá v cykle vytvorí prázdne hodnoty všetkých atribútov a nakoniec vráti inštanciu triedy *Fakt* obsahujúcu tieto hodnoty.

Konkrétnu hodnotu atribútu faktu predstavuje entita *HodnotaAtributuFaktu*. Táto trieda je abstraktná a poskytuje abstraktné metódy, ktoré musia jej potomkovia implementovať. Takýmito metódami sú *getHodnota()* a *setHodnota()*, ktoré sú zodpovedné za správu samotnej hodnoty atribútu. Používajú pri tom triedu *Object* ako nadtriedu pre všetky typy v jazyku Java. Tu by bolo možno vhodné použitie generických typov tohto jazyka. V prípade, že je atribút kvanitifikovateľný, jeho hodnotu vráti metóda *getKvantifikovanaHodnota()*, ktorá už vráti hodnotu ako typ double.



Obr. 6.3 Diagram atribútov faktu.

6.5 Číselníky

Číselník (angl. *enumeration*) je dátový typ, ktorý má vymenované hodnoty. Číselníkom je napríklad známka z predmetu alebo spôsob ukončenia predmetu. Číselníky sme implementovali pomocou tried *Ciselnik* a *HodnotaCiselnika*. Znázornené sú v diagrame na Obr. 6.4.

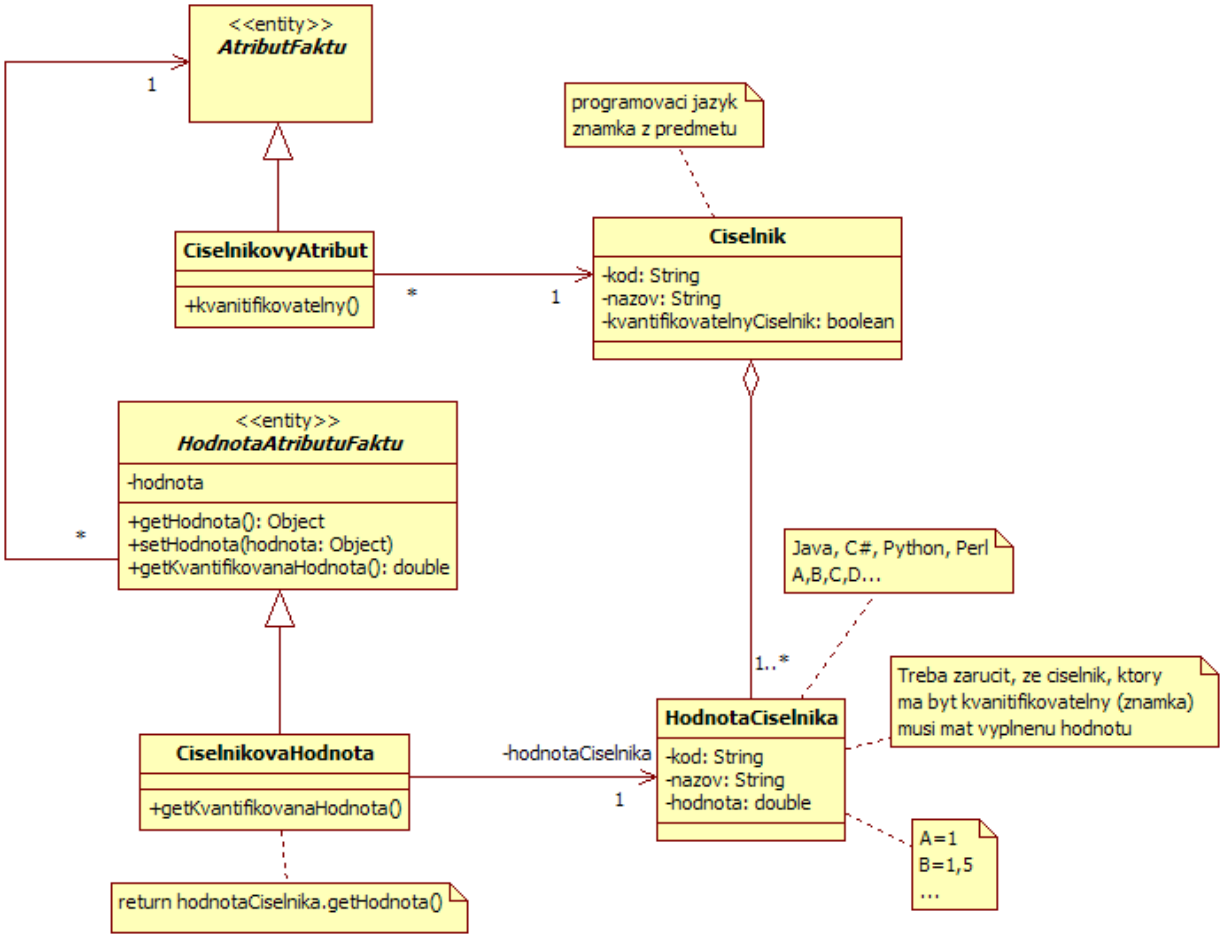
Trieda *Ciselnik* teda predstavuje číselník. Atribút *kód* ho jednoznačne identifikuje. Atribút názov slúži ako slovný názov číselníka a používa sa aj vo formulároch. Niektoré číselníky môžu byť kvantifikovateľné, na rozlíšenie takýchto číselníkov slúži atribút *kvanatifikovatelnyCiselnik*.

Trieda *Ciselnik* sa skladá z hodnôt číselníkov, čo predstavuje trieda *HodnotaCiselnika*. Táto trieda má opäť štandardné atribúty *kód* a *názov*. Špeciálnym atribútom je *hodnota*, ktorá nesie číselnú hodnotu tejto hodnoty číselníka, má význam len pre kvanitifikovateľné číselníky. Napríklad pre známku A je to 1, pre známku B 1,5 a pod.

Číselníky úzko súvisia s atribútmi faktu. Preto existuje ďalšia podtrieda *AtributuFaktu* a tou je *CiselnikovyAtribut*. Ten teda predstavuje taký atribút, ktorého hodnoty sú vymenované, napr. už spomínaná známka z premetu alebo úroveň certifikátu. Preto má takýto atribút jednosmernú asociáciu na triedu *Ciselnik* s multiplicitou 1. Abstraktnú metódu *kvantifikovatelny()* implementuje *CiselnikovyAtribut* tak, že vráti to, či je daný asociovaný číselník kvanitifikovateľný.

Keďže existuje špeciálny atribút faktu pre číselníky, musí existovať aj špeciálna hodnota atribútu faktu pre číselníky. Touto triedou je *CiselnikovaHodnota*. Tá má ako svoju hodnotu asociáciu na konkrétnu *HodnotuCiselnika*. Abstraktnú metódu *getKvantifikovanaHodnota()* implementuje tak, že vráti atribút *hodnota* asociovanej *HodnotyCiselnika*.

Pomocou takto definovaných číselníkov môže konfigurátor vytvárať za behu aplikácie nové číselníky a aj číselníkové atribúty typu faktu.



Obr. 6.4 Diagram číselníkov.

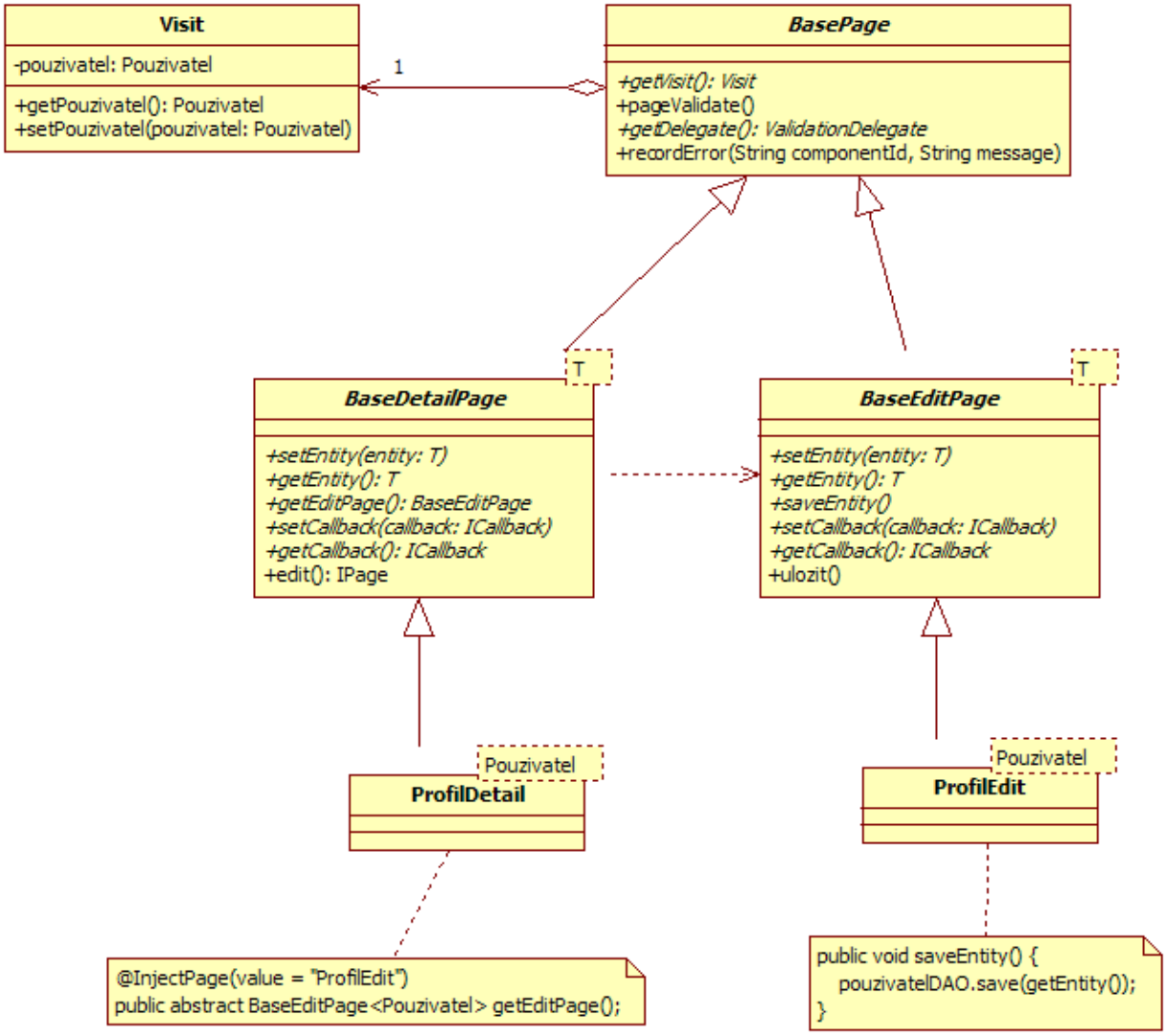
6.6 Základné triedy prezentačnej vrstvy

Za každou stránkou prezentačnej vrstvy je aj Java trieda. Táto trieda dedí od Tapestry triedy *BasePage*. V našom systéme Znalec je vytvorená spoločná nadtrieda *BasePage*, ktorá dedí od Tapestry *BasePage*. Táto nadtrieda poskytuje funkcionalitu spoločnú všetkým triedam, ktoré stoja za stránkami. Prvou takouto službou je poskytnutie triedy *Visit* pomocou metódy *getVisit()*. Visit je trieda, ktorá reprezentuje reláciu medzi používateľom a aplikáciou. V tejto relácii si môžeme ukladať rôzne informácie týkajúce sa komunikácie používateľa s aplikáciou. V našom prípade si v tejto triede pamätáme len to, ktorý používateľ je práve v systéme prihlásený. Túto triedu je treba špeciálne zaregistrovať v Tapestry konfiguračnom súbore. Ďalšou spoločnou metódou *BasePage* je *pageValidate()*. Táto metóda pochádza z Tapestry rozhrania, ktoré trieda *BasePage* implementuje, a slúži na účely zabezpečenia stránok. V prípade systému Znalec slúži na kontrolu, či je používateľ prihlásený, ak nie je, pošle ho na prihlasovaciu stránku. Metóda *getDelegate()* slúži na vrátenie Tapestry rozhrania *ValidationDelegate*, ktoré sa používa na validáciu vstupu od používateľa. Poslednou metódou nadtriedy *BasePage* je *recordError()*, ktorá slúži na zapísanie validačnej chyby do stránky.

Keďže prezeranie a editovanie entít doménového modelu je často sa opakujúci scenár, vytvorili sme dve generické nadtriedy pre tieto činnosti – *BaseDetailPage* a *BaseEditPage*. Na prezeranie entít je navrhnutá trieda *BaseDetailPage*. Táto trieda je generická a jej typový parameter je typ entity, ktorú zobrazuje. Táto trieda má metódy *get/setEntity()* na uchovávanie entity. Pred zobrazením takejto stránky musí volajúca stránky nastaviť entitu, ktorá sa má zobraziť pomocou volania *setEntity()*. Táto stránka umožňuje otvoriť stránku na editovanie zobrazenej entity. Aby vedela, akú stránku má zobraziť, používa abstraktnú metódu *getEditPage()*, ktorá je tiež generická a musí byť implementovaná podtriedami. Na

samotné otvorenie editačnej stránky slúži metóda *edit()*. Tá si vypýta editačnú stránku pomocou *getEditPage()*, nastaví jej zobrazenú entitu a callback a zobrazí túto editačnú stránku. Na pohyb po stránkach sa používajú tzv. callback-y. *ICallback* je Tapestry rozhranie a používa sa na vyvolanie stránky, na ktorú má byť používateľ po nejakej akcii presmerovaný.

Na editáciu entít sa používa stránka *BaseEditPage*, ktorá je tiež generická a funguje podobne ako *BaseDetailPage*. Okrem štyroch metód, ktoré má aj *BaseDetailPage* má táto stránka ďalšie dve. Prvou je *saveEntity()*. Táto metóda je abstraktná a podtriedy ju musia implementovať. Jej úlohou je uložiť zmeny vykonané na danej entite. Druhou je metóda *ulozit()*. Tá najskôr skontroluje, či v stránke nie sú validačné chyby, ak nie sú, tak zavolá *saveEntity()*, a nakoniec sa vráti na stránku zadanú cez callback. Diagram týchto tried s ukážkou pre entitu *Pouzivatel* sa nachádza na Obr. 6.5.

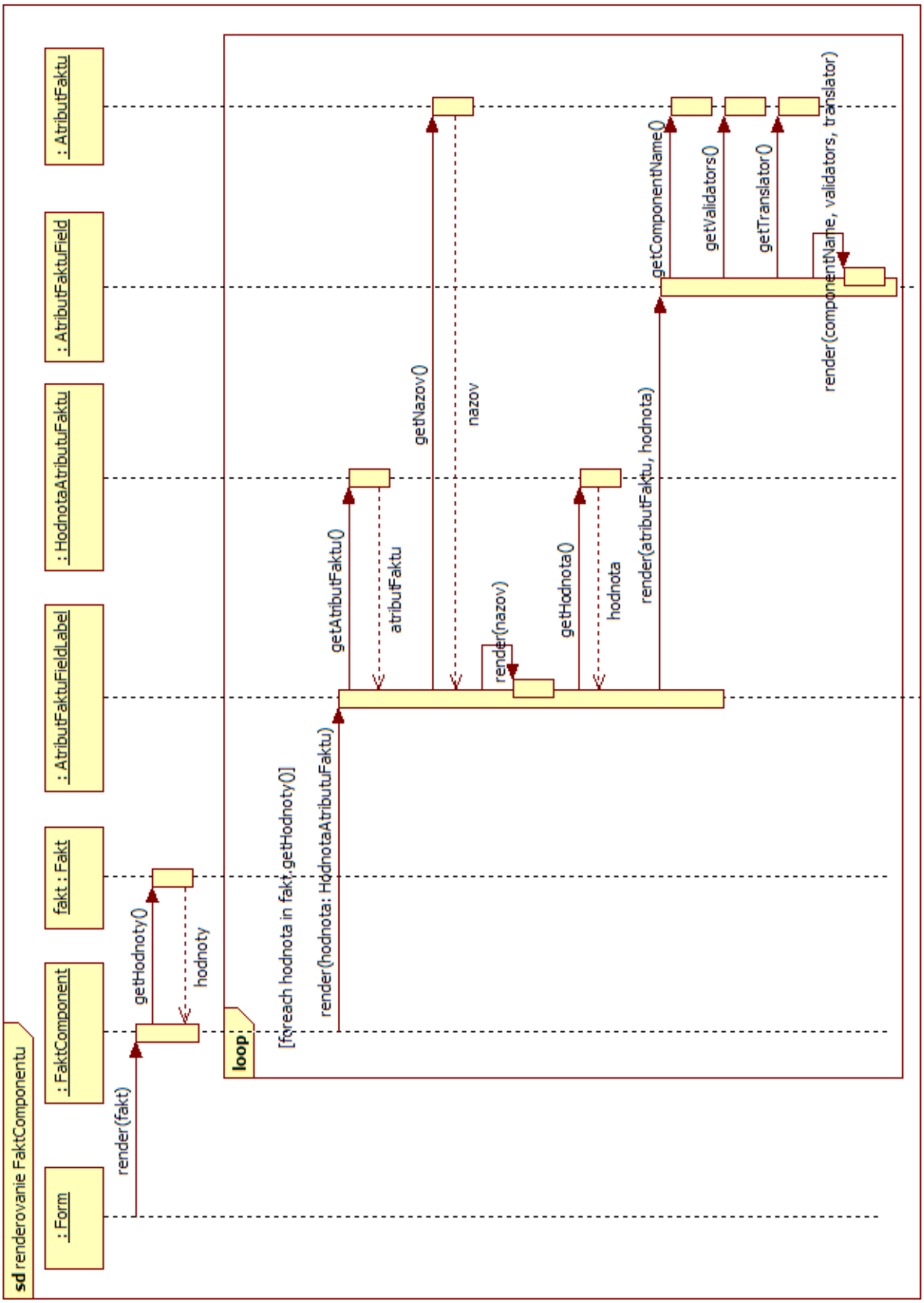


Obr. 6.5 Diagram nadtried pre stránky zobrazujúce entity.

6.7 Dynamické generovanie formulárov na základe metadát

Jedným z cieľov vyvíjaného systému bola vysoká konfigurovateľnosť. To sa podarilo pomocou princípu abstrakcie atribútov. Tento princíp je však užitočný aj v prezentačnej vrstve. Umožňuje totiž vykresľovanie (renderovanie) formulárov pre vyplnenie faktu dynamicky, takže programátori nemusia vytvárať nový formulár pre každý typ faktu ručne. Túto funkcionalitu zabezpečuje nami vytvorený komponent pre rámec Tapestry, ktorý sa nazýva *FaktComponent*. Tento komponent dostane ako parameter inštanciu triedy *Fakt*. V cykle prechádza cez všetky jeho hodnoty, teda inštancie triedy *HodnotaAtributuFaktu*. Tieto hodnoty

posúva ďalej na renderovanie komponentu *AtributFaktuFieldLabel*. Ten je zodpovedný za vykreslenie popisu (label) atribútu a editovacieho políčka atribútu. Popis atribútu je jeho názov, teda *HodnotaAtributuFaktu.AtributFaktu.nazov*. Na vykreslenie editovacieho políčka sa používa komponent *AtributFaktuField*. Ako parameter dostane typ atribútu a samotnú hodnotu tohto atribútu. Tento komponent je schopný na základe typu atribútu vykresliť príslušné editačné políčko, teda pre textový atribút obyčajný TextField, pre boolovský atribút Checkbox, pre číselník Combobox a pod. spolu s ich validačnými podmienkami. Tejto funkcionality je schopný vďaka abstraktným metódam definovaných v triede *AtributFaktu*: *getComponentName()* vráti názov komponentu, ktorý sa má vyrenderovať (Textfield, Checkbox), *getValidators()* vráti pole validátorov a *getTranslator()* vráti prekladač reťazcov na Java triedy. Do takto vykresleného políčka vyplní danú hodnotu atribútu. Sekvenčný diagram tohto renderovania je znázornený na Obr. 6.6. Komponent *FaktComponent* má aj režim len na čítanie, ktorý sa dá zadať ako jeho parameter. V takom prípade namiesto editačných políčok len vloží hodnotu atribútu ako text.



Obr. 6.6 Sekvenčný diagram pre vykreslenie faktu pomocou FaktComponent.

6.8 Opis algoritmov

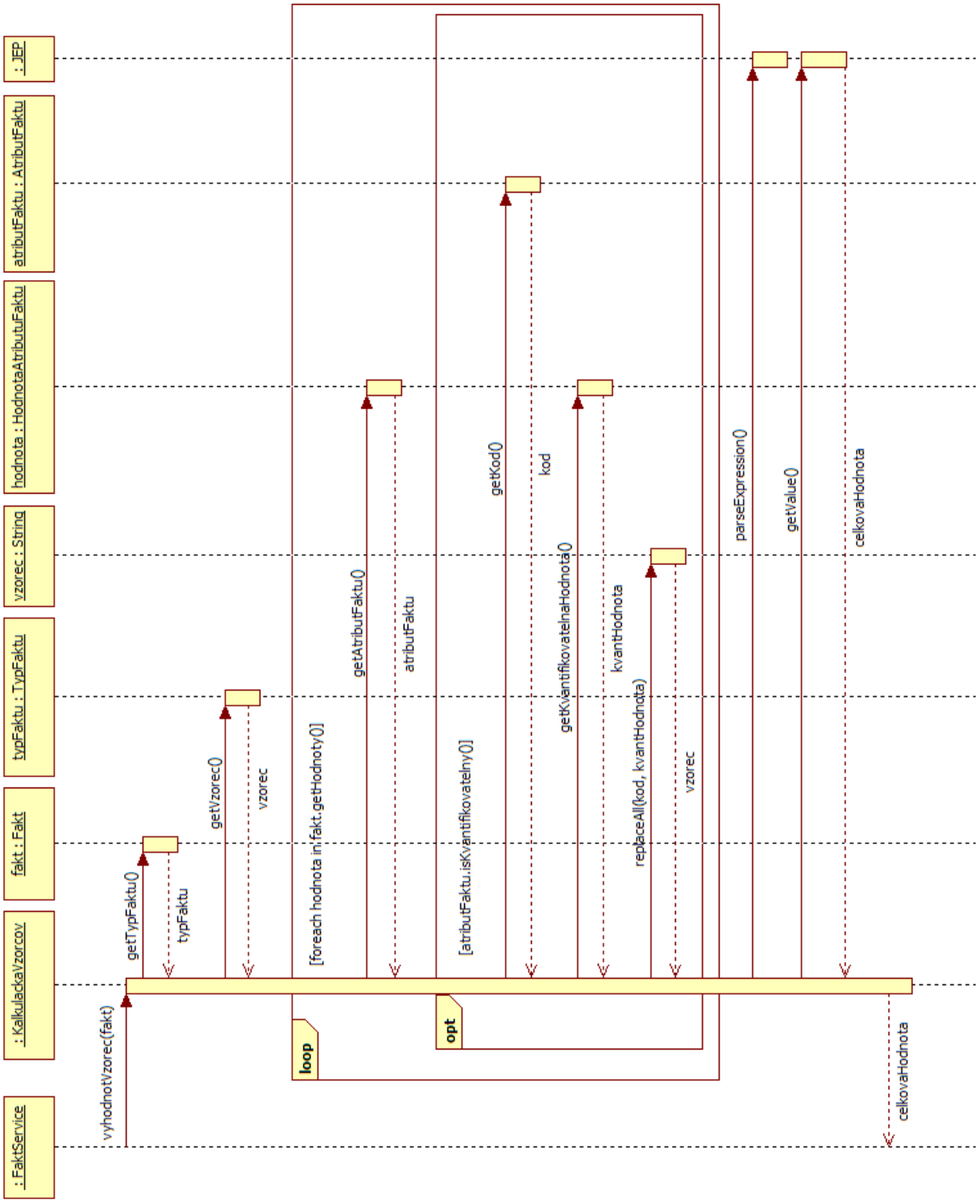
V tejto časti opisujeme tri nosné algoritmy, ktoré vedú k vyhľadaniu a výpočtu miery znalosti pre jednotlivých študentov.

6.8.1 Výpočet vzorca faktu

Hlavnou úlohou algoritmu je číselne vyjadriť hodnotu faktu podľa vzorca, ktorý je definovaný konfigurátorom pre daný typ faktu. Spravidla sa vykonáva pri pridávaní faktu alebo zmene hodnoty niektorého z jeho atribútov.

Na začiatku získa algoritmus predpis vzorca pre daný typ faktu vo forme textového reťazca. Následne pre každý kvantifikovateľný atribút hľadá jeho kód vo vzorci. Ak ho nájde, nahradí ho konkrétnou číselnou hodnotou atribútu faktu. Kód atribútu je jeho jedinečný textový identifikátor. Definuje ho konfigurátor a slúži práve pre potreby tohto výpočtu. Komponent na zadávanie vzorca pritom zabezpečuje, že do vzorca sa dajú pridávať len kvantifikovateľné atribúty (nie je tak možné zadať vzorec obsahujúci napr. názov faktu). Takto získaný výraz už neobsahuje žiadne premenné a môže byť vyhodnotený. Na to používame softvér JEP[7] (Java Math Expression Parser), ktorý sa nám osvedčil už v predchádzajúcich projektoch. Sekvenčný diagram s postupnosťou krokov vykonávaných pri výpočte je zobrazený na Obr. 6.7.

[7] http://www.singularsys.com/jep



Obr. 6.7 Výpočet hodnoty skóre faktu podľa zadaného vzorca.

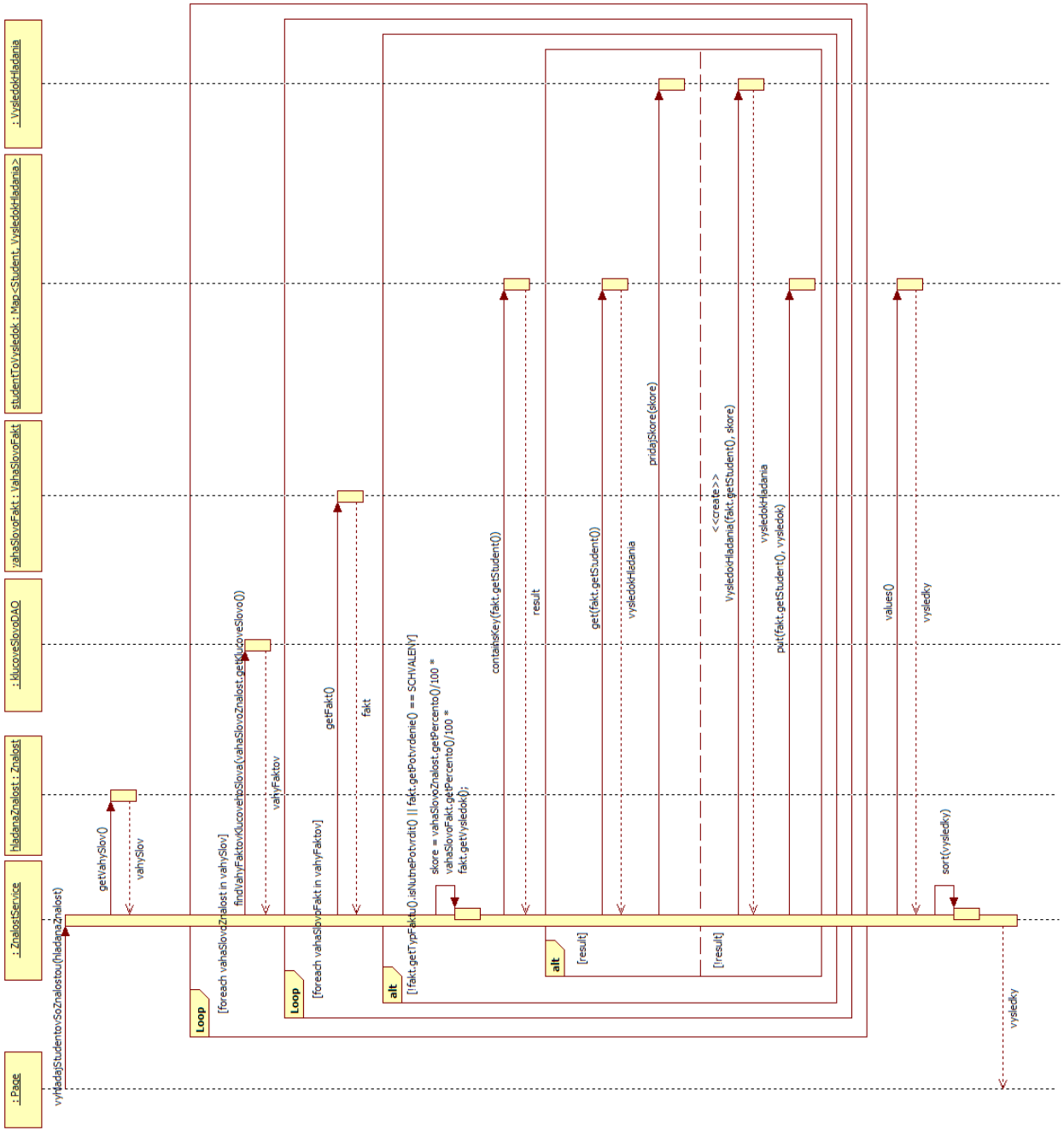
6.8.2 Vyhľadanie študentov so znalosťou

Algoritmus sa používa na vyhľadanie študentov podľa definovanej znalosti. Výsledkom je zoznam študentov, pričom každý z nich ma skóre, ktorá charakterizuje mieru ovládania znalosti študentom. Pre každé kľúčové slovo špecifikovanej znalosti nájde algoritmus všetky fakty, ktoré ho obsahujú. Potom pre každý z faktov vypočíta skóre a pripočíta ho k študentovi, ktorému ten fakt patrí. Skóre vypočíta pomocou vzorca:

Rovnica 1 Vzorec na výpočet skóre faktu.

$$skóre = \frac{VSZ}{100} * \frac{VSF}{100} * VF$$

kde VSZ (vahaSlovoZnalost) predstavuje percentuálny podiel kľúčového slova na znalosti a VSF (vahaSlovoFakt) percentuálny podiel kľúčového slova na fakte. VF je výsledok faktu, ktorý sme získali v predchádzajúcom algoritme. Sekvenčný diagram opisujúci tento algoritmus je zobrazený na Obr. 6.8.

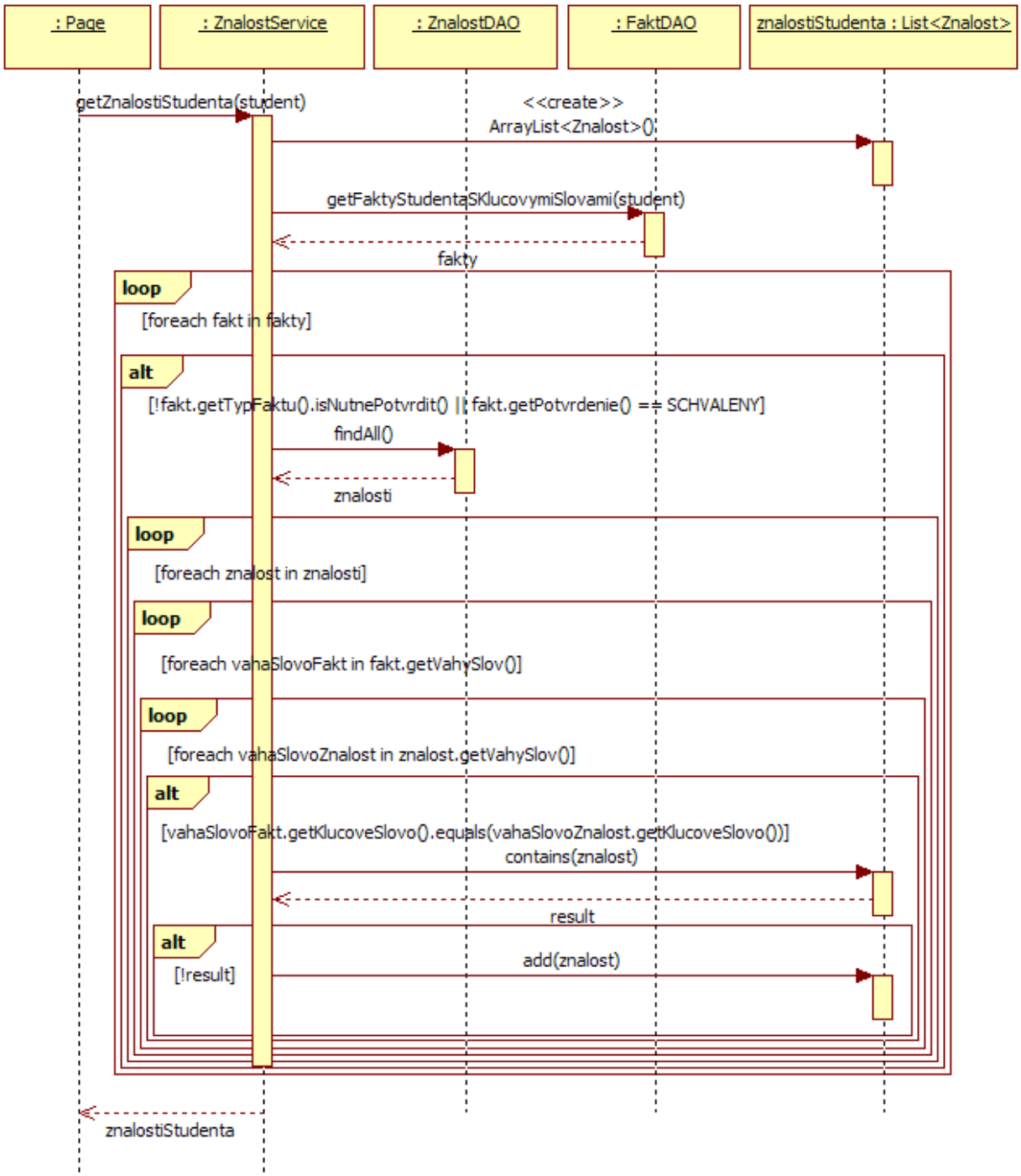


Obr. 6.8 Vyhľadanie študentov s definovanou znalosťou.

6.8.3 Vyhľadanie znalostí študenta

Algoritmus slúži na vyhľadanie všetkých znalosti daného študenta. Nevypočítava pritom skóre pre danú znalosť, iba zobrazuje získané znalosti. Pre všetky fakty študenta a všetky definované znalostí hľadá ich

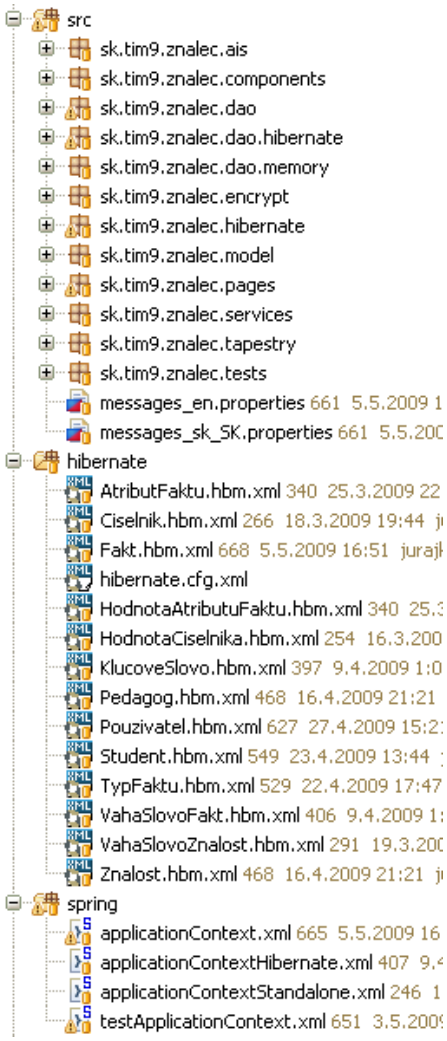
spoločné kľúčové slová. Ak nájde zhodu, príslušnú znalosť uloží do zoznamu nájdených znalostí. Sekvenčný diagram pre tento algoritmus je zobrazený naObr. 6.9.



Obr. 6.9 Vyhľadanie všetkých znalostí jedného študenta.

6.9 Opis architektúry a modulov systému

Celková architektúra systému bola navrhnutá už vo fáze prototypovania a odvtedy sa nezmenila. Diagram tejto architektúry je zobrazený na Obr. 6.10. V tejto kapitole bližšie opíšeme jednotlivé moduly tejto architektúry.



Obr. 6.10 Jednotlivé balíky projektu v prostredí Eclipse.

Tieto moduly opíšeme z pohľadu vrstiev systému a namapujeme ich na balíky v zdrojovom kóde.

6.9.1 Databáza a prístup k nej

Ako databázový systému sme si zvolili voľne dostupný systém MySQL. V ňom je spustená databáza *znalec*, ktorú náš systému používa. Prístup k databáze je vyriešený pomocou JDBC ovládača a dátového zdroja (angl. *data source*), ktorý poskytuje aplikačný server Tomcat. Uvedený dátový zdroj je pomerne efektívny a zároveň poskytuje odkladanie databázových konekcií (angl. *connection pooling*). Dátový zdroj je vystavený ako JNDI (*Java Naming and Directory Interface*) zdroj, čo umožňuje konfiguráciu pripojenia k databáze mimo aplikačného kódu.

6.9.2 Dátová vrstva

Účelom dátovej vrstvy je prístup k databáze a poskytovanie dátových služieb vyšším vrstvám. V našom systéme sme použili objektovo-relačné mapovanie pomocou rámca Hibernate. Na poskytovanie dátových služieb sme použili JEE vzor DAO. Dátovej vrstve zodpovedajú balíky systému: Zdrojový adresár *hibernate* obsahuje konfiguračné súbory vo formáte XML, ktoré mapujú Java objekty na relačné tabuľky. V zdrojovom adresári *src* zastupujú dátovú vrstvu balíky *sk.tim9.znalec.dao*, *sk.tim9.znalec.dao.hibernate* a *sk.tim9.znalec.dao.memory*.

Vzor DAO je abstrakcia prístupu k dátovej vrstve. Jednotlivé DAO objekty majú jedinú úlohu, a to poskytovať dátové služby, ako napríklad „nájdi všetky fakty študenta XY“. Služby sú definované pomocou rozhraní, ktoré sa nachádzajú v balíku *sk.tim9.znalec.dao*. Tieto rozhrania zastupuje generické rozhranie

GenericDAO, ktoré poskytuje základné CRUD služby. Od neho dedia špeciálne rozhrania pre každú významnú entitu doménového modelu a poskytujú špeciálne služby pre danú entitu. Implementácie týchto rozhraní sa nachádzajú v ostatných dvoch balíkoch. V balíku s koncovkou *memory* sa nachádzajú implementácie, ktoré pracujú s objektmi len v pamäti pomocou kolekcií – používajú sa len na testovacie účely. V balíku s koncovkou *hibernate* sú implementácie využívajúce nástroj Hibernate, tieto sa používajú v reálnej prevádzke.

V balíku *sk.tim9.znalec.hibernate* sa nachádzajú triedy a rozhrania pre vlastné dátové typy pre nástroj Hibernate. Tie používame na mapovanie Java enumerácií na databázové stĺpce.

Na konfiguráciu dátovej vrstvy používame rámec *Spring* (podrobne opísaný nižšie). Slúži na to XML *applicationContextHibernate.xml*. V ňom si vypýtame dátový zdroj pomocou spomínaného JNDI a vyberieme, ktoré implementácie DAO rozhraní budeme používať. V tomto súbore definujeme aj transakčný manažér, ktorý sa stará o korektnú prácu s transakciami. Transakciami sa budeme zaoberať neskôr v tejto kapitole.

6.9.3 Aplikačná vrstva

Aplikačná vrstva zabezpečuje biznis logiku aplikácie a poskytuje služby prezentačnej vrstve. Táto vrstva je rozdelená na viacero modulov: základné služby, modul pre prístup k AIS a modul pre šifrovanie. V nasledujúcich kapitolách opíšeme jednotlivé moduly.

Základné služby

Základné služby predstavujú jadro biznis logiky. Tieto služby poskytujú logiku pre prácu s používateľmi systému, faktami systému, konfiguráciu systému a pod. Ich rozhrania a implementácie sa nachádzajú v balíku *sk.tim9.znalec.services*.

Každá služba teda tvorí samostatný modul (Java trieda), ktorý je zodpovedný za jednu konkrétnu činnosť a od ostatných modulov je čo najmenej závislý. Zároveň každý modul potrebuje na svoju prácu využívať služby dátovej vrstvy (DAO), na ktorých je tiež závislý. Tieto závislosti spravuje integračná vrstva.

Modul pre prístup k AIS

Modul pre prístup k AIS má jedinú funkcionalitu, a tou je import dát o študentoch a ich predmetoch z AIS. Tento modul naimportuje dáta o študentoch, ak ešte v systéme nie sú evidovaní, vytvorí im konto a zriadi prihlasovacie meno a heslo. Informácie o absolvovaných predmetoch sú pretransformované do formy faktov a sú previazané na kľúčové slová.

Tento modul sa nachádza v balíku *sk.tim9.znalec.ais*. Na import dát je definované rozhranie. Keďže sa nám ku koncu semestra nepodarilo zriadiť reálny prístup k databáze AIS, implementovali sme toto rozhranie tak, že číta dáta z CSV súboru, ktorý sme dostali ako ukážku dát. Tento súbor obsahuje záznam zo zimného semestra 2008 a nachádzajú sa v ňom informácie o viac ako 1000 študentoch, čo je pre otestovanie funkčnosti aplikácie dostatočné. Import prebieha inkrementálne, takže študenti a ich predmety sú importovaní len raz a do systému sa pridajú len nové informácie.

Modul pre šifrovanie

Pre zvýšenie zabezpečenia sú používateľské heslá v databáze šifrované. O šifrovanie pri ukladaní a dešifrovanie pri čítaní entity Používateľ z databázy sa stará modul pre šifrovanie. Tento modul sa nachádza v balíku *sk.tim9.znalec.encrypt*.

6.9.4 Prezentačná vrstva

Ako bolo spomínané v kapitole 4.2, prezentačná vrstva je vytvorená pomocou rámca Tapestry, pomocou ktorého sú vytvorené dynamické webové stránky.

V balíku *sk.tim9.znalec.pages* sa nachádzajú triedy pre webové stránky. Komponenty prezentačnej vrstvy sa nachádzajú v balíku *sk.tim9.znalec.components*. Ďalšie podporné triedy pre rámec Tapestry sa nachádzajú v balíku *sk.tim9.znalec.tapestry.*

6.9.5 Objekty domény

Triedy, ktoré predstavujú objekty problémovej oblasti, nazývané aj model domény, sa nachádzajú v balíku *sk.tim9.znalec.model*. V tomto balíku sú triedy ako *Student*, *Fakt*, ale aj abstraktnejšie triedy, ako napríklad *VysledokHladania* alebo *KalkukalckaVzorcov*.

6.9.6 Unit testy

Triedy, ktoré predstavujú JUnit testy, sú umiestnené v balíku *sk.tim9.znalec.tests*. Testovaniu systému sa podrobne venujeme v kapitole 7.

6.9.7 Integrácia vrstiev

Ako bolo spomínané vyššie, aplikácia je rozdelená na viacero modulov a vrstiev. Tieto vrstvy a moduly majú medzi sebou rôzne závislosti – napríklad vyššie vrstvy závisia od vrstiev pod nimi.

Na previazanie týchto závislostí používame rámec Spring. Ten používa ako hlavnú myšlienku vzor *Dependency Injection* (vkladanie závislostí)[8]. Tento vzor podporuje princíp programovania voči rozhraniam a jednoduchú zameniteľnosť implementácií, čo podporuje voľnú previazanosť modulov (angl. *loose coupling*). Konfiguračné súbory pre rámec Spring sa nachádzajú v adresári *spring*.

Opis a konfigurácia služieb biznis vrstvy sa nachádza v súbore *applicationContext.xml*. Tu je definované, aké implementácie biznis rozhraní sa použijú, a ktoré DAO triedy majú byť do týchto implementácií vložené (angl. *injected*). V tomto súbore sú definované aj biznis transakcie. Biznis transakcia je skupina biznis metód a databázových volaní, ktoré plnia jednu ucelenú biznis funkciu a musia byť uzavreté v jednej databázovej transakcii. Ak priebeh transakcie zlyhá (napr. nastane výnimka), celá transakcia je zrušená (angl. *rollback*), aby nedošlo ku nekonzistencii dát – princíp ACID (Atomicity, Consistency, Isolation, Durability). Na správu týchto transakcii využívame opäť rámec Spring. Ten umožňuje definovať transakcie deklaratívne pomocou aspektovo-orientovaného programovania. Takto je dosiahnuté oddeľovanie záujmov (angl. *separation of concerns*).

Integrácia medzi prezentačnou a aplikačnou vrstvou je vyriešená cez anotáciu *InjectSpring*. Táto anotácia sa používa v triedach zastupujúcich prezentačnú vrstvu. Ako parameter tejto anotácie sa dáva názov služby, ktorú stránka vyžaduje.

Typická spolupráca vrstiev prebieha nasledovne. Používateľ zvolí nejakú akciu, prezentačná vrstva požiada aplikačnú vrstvu o vykonanie tejto akcie, prebehne aplikačná logika, ktorá prípadne vykoná akciu v databáze. Prezentačná vrstva ide zobraziť stránku ako reakciu na akciu, požiada aplikačnú vrstvu o dáta, aplikačná vrstva si vypýta všetky potrebné dáta z dátovej vrstvy a odovzdá ich prezentačnej vrstve. Dôležité je, aby dátová vrstva vrátila všetky dáta, ktoré prezentačná vrstva potrebuje, v opačnom prípade by došlo k výnimke, keďže Hibernate načítava len to, čo si explicitne vypýtame. Tento princíp spolupráce sa nazýva aj transakčný skript[9].

6.10 Opis implementačného prostredia

Systém sme vyvíjali na technológiách, ktoré sme spomínali v kapitole 4. V tomto smere sa nič nezmenilo. Dôraz pri výbere nástrojov sme kládli na ich voľnú dostupnosť. Taktiež sme preferovali nástroje, s ktorými mal aspoň jeden člen tímu predchádzajúce skúsenosti, aby sme nestratili veľa času ich štúdiom.

Ako vývojové prostredie sme si zvolili Eclipse[10], ktorý je momentálne asi najlepší voľne dostupný nástroj pre vývoj na platforme Java. Pre svoju prispôsobiteľnosť a rozšíriteľnosť bol aj pre nás správnou voľbou.

Na správu zdrojového kódu sme používali SVN službu na serveri http://xp-dev.com, kde bezplatné konto ponúka priestor až 500 MB. Umožnilo nám to zdieľať zdrojové kódy, sledovať zmeny, a tak

[8] http://martinfowler.com/articles/injection.html
[9] http://martinfowler.com/eaaCatalog/transactionScript.html
[10] http://www.eclipse.org

efektívnejšie vyvíjať aplikáciu. Na prístup k tomuto úložisku sme používali zásuvný modul do vývojového prostredia s názvom Subclipse[11]. Ten sa integroval do vývojového prostredia Eclipse a umožňoval pohodlnú prácu so zdieľanými zdrojovými textami systému.

Pre automatické zostavovanie projektu sme použili nástroj Apache Ant[12] a manažment tohto procesu sme zabezpečovali nástrojom LuntBuild[13]. Ten umožňoval automatizovať získanie najnovších zdrojových kódov systému z SVN úložiska a vykonanie zostavenia systému do podoby nasaditeľnej na aplikačnom serveri.

[11] http://subclipse.tigris.org
[12] http://ant.apache.org
[13] http://luntbuild.javaforge.com

7 Overenie výsledku

V tejto kapitole uvádzame spôsob overenia výsledku a jeho priebeh, zistené nedostatky a závery z tejto fázy riešenia projektu.

7.1 Spôsob overenia

Už počas predošlých fáz riešenia projektu sme sa dohodli na troch spôsoboch overenia výsledku, ktorými sú:

- automatické testy aplikačnej logiky,
- akceptačné testy po skončení implementácie,
- voľné testovanie systému pri prevádzke.

7.1.1 Automatické testy aplikačnej logiky

Vo fáze implementácie sme súbežne s vytváraním aplikačnej vrstvy implementovali testy pre hlavné algoritmy a služby tejto vrstvy. Túto vrstvu sme testovali z toho dôvodu, že v nej bola realizovaná implementácia hlavnej myšlienky fungovania systému. Na tvorbu automatických testov sme využili nástroj jUnit na platforme Java. Vytvorili sme niekoľko testovacích scenárov, ktoré obsahovali viacero testov. Tie sme priebežne spúšťali počas vývoja, aby sme sa uistili, že služby aplikačnej vrstvy fungujú korektne.

7.1.2 Akceptačné testy po skončení implementácie

Vo fáze špecifikácie požiadaviek na výsledné riešenie sme uviedli prípady použitia, ktoré by náš systém mal spĺňať. Na základe týchto prípadov použitia sme následne vytvorili sedem akceptačných testov. Tieto testy sme následne dali dvom študentom, ktorí nie sú členmi nášho tímu. Tester dostal ku každému testovaciemu scenáru tabuľku, ktorá obsahovala okrem iného účel testu, mieru jeho dôležitosti, postupnosť krokov a očakávaných reakcií. Tester vedľa očakávaných reakcií dopisoval skutočné reakcie systému. Takto sme overili splnenie cieľov vzhľadom na špecifikáciu požiadaviek z fázy analýzy.

7.1.3 Voľné testovanie systému pri prevádzke

Systém sme nasadili na verejne dostupný server, umiestnený v rámci fakulty. Chceli sme tak vyskúšať jeho fungovanie v reálnej prevádzke. Požiadali sme niekoľkých študentov, aby v ňom skúšali voľne pracovať. Systém sme skúšali používať aj sami. Pri vývoji systému sme pracovali s malou vzorkou údajov, boli sme teda zvedaví, ako sa bude systém správať pri väčšom množstve dát. Importovali sme doň približne tisíc používateľov (študentov fakulty), každému študentovi sme importovali v priemere päť predmetov. Tieto údaje nám poskytli správcovia systému AIS a mali predstavovať simulované prepojenie nášho systému s tým akademickým.

7.2 Výsledky overenia

Výsledok akceptačných testov hodnotíme úspešne. Väčšina krokov prebehla podľa očakávaní. Vyskytli sa niektoré nedostatky, o ktorých sme vedeli, no neboli pre hlavnú prácu systému kritické. Jedná sa napr. o odstraňovanie entít (používateľov, znalostí, atď.) bez nutnosti potvrdzovania takejto akcie, nezobrazenie číselného ohodnotenia v zozname znalostí, nemožnosť dodatočného filtrovania výsledkov hľadania. Tieto nedostatky považujeme za menej dôležité.

Počas voľného testovania systému sme objavili viacero menších chýb. Najzávažnejšou bola nefungujúca diakritika v používateľskom rozhraní. Tento problém sme nakoniec vyriešili, navyše sme pridali podporu viacjazyčnosti. Používateľské rozhranie tak môže byť preložené do iných jazykov. Problém s diakritikou je tiež na strane databázy. Nesprávne sa zobrazujú niektoré znaky (č, ť) v údajov uložených

v databáze (meno študenta, názov faktu, atď.). Tento problém sa nám nepodarilo odstrániť, príčina môže byť v nastaveniach databázy alebo v nastaveniach servera. Pri testovaní na lokálnych vývojových počítačoch sa tento problém neobjavil.

Ďalšia chyba nastáva pri pokuse o odstránenie entity, ktorá sa používa. Príkladom je pokus o odstránenie kľúčového slova, ktoré je spojené s nejakým faktom. Táto chyba ostáva takisto neodstránená. Bolo by potrebné analyzovať a definovať žiaduce správanie pri takejto akcii (má sa vypísať chybová správa alebo sa má kľúčové slovo odstrániť? Ako sa následne majú upraviť váhy ostatných kľúčových slov?).

V systéme sme nekontrolovali odstraňovanie používateľov. Mohlo sa tak stať, že administrátor odstránil zo systému sám seba a následne sa systém nedal spravovať. Túto chybu sme včas odhalili a opravili. Teraz musí v systéme ostať vždy minimálne jeden administrátor.

7.3 Prevádzka s reálnymi dátami

Do systému sme importovali približne tisíc používateľov – študentov našej fakulty. Takisto sme ku každému z nich importovali v priemere päť predmetov. Chceli sme vyskúšať rýchlosť odozvy systému. Pri vyhľadávaní študentov s určitou znalosťou totiž systém prechádza všetky fakty všetkých študentov, čo sa nám javilo ako časovo náročná operácia. Takisto načítavanie väčšieho počtu kľúčových slov by mohlo byť pomalé. Našťastie sa ukázalo, že sme algoritmy aplikačnej vrstvy implementovali efektívne. Na systéme nie sú zrejmé žiadne známky oneskorenia, pracuje rovnako rýchlo ako pri malej vzorke údajov. Jediným problémom môže byť objem prenášaných údajov. Keď si administrátor chce zobraziť všetkých používateľov systému (približne tisíc), musí sa k nemu preniesť približne 6 MB údajov. To však považujeme vzhľadom na povahu takejto akcie za prijateľné.

8 Zhrnutie

V kapitole 1.2 sme stanovili ciele projektu, ktorými boli vytvorenie bázy znalostí, automatické a manuálne získavanie údajov, odporúčanie vhodných kandidátov, inteligentné prehľadávanie a modifikácia údajov. Tieto ciele sa nám podarilo z veľkej časti naplniť. Úspešne sme navrhli metódu tvorby bázy znalostí študentov, ktorú sme overili implementáciou reálneho systému. Údaje sa pre bázu znalostí získavajú automaticky (importom údajov vygenerovaných systémom AIS), ako aj manuálne (priamym zadávaním faktov študentmi). Systém umožňuje vytvoriť požadovanú znalosť a na jej základe hľadať vhodných študentov, čím spĺňa cieľ odporúčania vhodných kandidátov. Študenti môžu pridávať nové údaje a modifikovať už zadané. Jediným cieľom, ktorý sa nám nepodarilo splniť, je inteligentné prehľadávanie. Tu sme na začiatku navrhli možnosť použitia filtrov na výsledky vyhľadávania, ktorými by sa dali tieto výsledky ďalej obmedziť (napr. zobraziť len študentov inžinierskeho štúdia). Implementáciu takejto funkcionality sa nám nepodarilo stihnúť. Nepovažujeme ju však za kľúčovú.

Systém navyše umožňuje dodatočnú kontrolu zadávaných údajov zo strany študenta. Implementovali sme podporu pre potvrdzovanie faktov pedagógmi. Pedagóg má tak možnosť zadaný fakt potvrdiť alebo zamietnuť, čo údajom pridáva na hodnovernosti. Výsledný systém je tiež plne modifikovateľný, konfigurátor môže pridávať nové typy faktov bez nutnosti zásahu do zdrojového kódu. V rámci tejto funkcionality môže upravovať atribúty, ako aj vzorec použitý na výpočet výsledku faktu. Systém sa tak dá prispôsobiť konkrétnym potrebám alebo doméne. Má potenciál byť využitý aj v inej doméne. Príkladom môže byť báza znalostí zamestnancov firmy. Konfigurátor pomocou nášho systému zadefinuje typy faktov relevantné pre túto oblasť. Zamestnanci si následne môžu pridávať konkrétne fakty, pričom je možné nastaviť nutnosť ich potvrdzovania. Potvrdzovať fakty by mohol napr. šéf daného zamestnanca. Takýto systém by mohol slúžiť na podporu výberu vhodných zamestnancov do tímov riešiacich konkrétne firemné projekty.

Systém Znalec nie je dokončený do tej miery, že na ňom nie je čo robiť. Dalo by sa na ňom vylepšovať množstvo vecí. Medzi ne určite patrí bezpečnejšia autentifikácia používateľov, implementácia bezpečnostných mechanizmov (napr. nutnosť potvrdzovať vymazanie entity zo systému), pridanie možnosti filtrovania výsledkov vyhľadávania, sociálne aspekty, akými sú komunikácia používateľov systému, možnosť vytvárania celých tímov s definovanými rolami, pokročilejšia možnosť spätnej väzby od pedagóga voči študentovi (jeho subjektívne hodnotenie znalostí študenta), a iné. Takisto je možné pracovať na reálnej integrácii so systémom AIS a importe údajov z neho. Náš systém však predstavuje pevný základ, ktorý implementuje nosné myšlienky tvorby bázy znalostí, a je možné na tomto základe ďalej stavať. Nakoľko sme v tomto projekte začínali od znova návrhom úplne nového systému, sme s dosiahnutým výsledkom spokojní.

Pri realizácii tohto projektu sme sa všetci veľa naučili. Niektorí z nás sa naučili pracovať s novými technológiami, s ktorými doteraz nemali skúsenosti. Všetci sme si tiež mali možnosť vyskúšať tímovú prácu, samostatné rozhodovanie sa a riadenie tímu. Hoci sme si to na začiatku projektu nemysleli, tento netechnický aspekt projektu je aspoň tak dôležitý, ako technická realizácia.

Použité zdroje

1. Šimún, M., Andrejko, A., Bieliková, M.: Inicializácia a aktualizácia modelu používateľa pri hľadaní pracovných ponúk na webe. In Znalosti 2007, P. Mikulecký, J. Dvorský, M. Krátký (Eds.), Ostrava, Czech Republic (2007) 109-120.
2. Tím _elf_: Báza znalostí a zručností študentov – Projektová dokumentácia. FIIT STU, máj 2006 [cit. 2008-10-20]. Dostupný z WWW: <http://labss2.fiit.stuba.sk/TeamProject/2005/team11/documents/_elf_LS_Projekt_v1.0.doc>.
3. Tím Černé ofce: Báza znalostí a zručností študentov – Dokumentácia k projektu. FIIT STU, máj 2008 [cit. 2008-10-20]. Dostupný z WWW: < http://labss2.fiit.stuba.sk/TeamProject/2007/team10is-si/files/dok-leto.pdf>.

Príloha A – Obsah CD nosiča

Súčasťou tejto práce je CD nosič, na ktorom sa nachádza:

- elektronická verzia dokumentácie,
- výsledný implementovaný systém v podobe nasaditeľnej na aplikačný server,
- zdrojové texty systému,
- skript pre vytvorenie tabuliek v databáze a
- statická verzia tímovej stránky.

Príloha B – Formuláre z testovania

V tejto prílohe sa nachádzajú vyplnené formuláre z akceptačného testovania a formuláre s nájdenými chybami počas voľného používania systému.

Príloha C – Príspevok na konferenciu IIT.SRC

V tejto prílohe uvádzame článok, ktorý bol prijatý na študentskú konferenciu IIT.SRC 2009, ktorá sa konala 29.4.2009 na Fakulte informatiky a informačných technológií STU v Bratislave. Článok bol vo forme rozšíreného abstraktu a je uverejnený v zborníku z tejto konferencie.

Príloha D – Plagát na konferenciu IIT.SRC

V tejto prílohe uvádzame plagát, ktorým sme prezentovali náš systém na študentskej konferencii IIT.SRC 2009, ktorá sa konala 29.4.2009 na Fakulte informatiky a informačných technológií STU v Bratislave.

Príloha E – CD nosič