

**Bakalářská práce**



**České  
vysoké  
učení technické  
v Praze**

**F3**

**Fakulta elektrotechnická  
Katedra počítačů**

## **Vyhledávání zaměstnanců dle kompetence**

**DOMINIK KOUBA**

**Vedoucí: Ing. Lukáš Zoubek  
Obor: Softwarové inženýrství a technologie  
Květen 2019**



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kouba** Jméno: **Dominik** Osobní číslo: **466040**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav: **Katedra počítačů**  
Studijní program: **Softwarové inženýrství a technologie**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Vyhledávání zaměstnanců dle kompetence**

Název bakalářské práce anglicky:

**Competence based employee search**

Pokyny pro vypracování:

Analýzujte zdroje osob, pracovišť a jejich znalostí na FEL ČVUT  
Analýzujte zdroje otevřených dat týkajících znalostí spojených s elektrotechnikou a vybranými obory na FEL (DBpedia apod.)  
Vytvořte ontologii osob, pracovišť a jejich znalostí na FEL ČVUT, využijte k tomu existujících ontologií např. UFO, SKOS; případně si existující ontologie vhodně rozšiřte  
Implementujte ontologické schéma pomocí vybrané triple-store databáze a naplňte jí vzorovými daty  
Implementujte webovou aplikaci popř. webovou službu využívající SPARQL rozhraní vybrané triple-store databáze pro vyhledávání pomocí klíčových slov  
Navrhněte testovací scénáře, proveďte a vyhodnoťte testování

Seznam doporučené literatury:

Studer, R.; Grimm, S. & Abecker, A., ed. (2007), Semantic Web Services: Concepts, Technologies, and Applications, Springer, Berlin  
BARBU MITITELU, Verginica a Eduard BARBU. From Dictionaries to Language Representation Formalisms. Revue Roumaine de Linguistique [online]. 2007(LII), (1-2), 19 ISSN 0035-3957. Dostupné z: <http://clic.cimec.unitn.it/eduard/publications/FromDictionaryToKR.pdf>

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Lukáš Zoubek, Centrum znalostního managementu FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **29.01.2019**

Termín odevzdání bakalářské práce: \_\_\_\_\_

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Lukáš Zoubek  
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.  
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

\_\_\_\_\_ Datum převzetí zadání

\_\_\_\_\_ Podpis studenta



## Poděkování

Při své cestě jsem měl dva hlavní průvodce vedoucího Lukáše Zoubka a konzultanta Denise Baručíče. Těmto dvěma patří velké díky za příjemnou spolupráci a rady, které mě vedly k cíli.

Další díky patří vyučujícím a spolužákům, kteří mi pomohli a byli ochotní věnovat mi svůj drahocenný čas. V neposlední řadě bych chtěl poděkovat mé rodině a kamarádům, kteří mě při psaní této práce podporovali. Obzvláště mé drahé Terezce, která mi byla oporou při všech chvílích.

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2019

.....

## Abstrakt

V této práci jsme se zabývali vyhledáváním osob a pracovišť dle kompetencí na FEL ČVUT.

Nejdříve jsme celý problém rozebrali obecně. V této části práce jsme rozhodli, že vhodnou reprezentací kompetencí budou ontologie. Hlavně proto, že jako jediný formální jazyk pro reprezentaci znalostí mají praktickou podporu technologií jako jsou RDF, OWL a SKOS. Dále jsme analyzovali možné datové zdroje, zejména zdroje znalostí osob a znalostní báze. Nakonec jsme se zabývali zpracováním dat, rozebrali jsme jejich transformaci do ontologií pomocí nástrojů jako je OntoRefine a následnou práci s nimi za pomoci jazyka SPARQL.

V druhé části práce jsme navrhli systém, který problematiku vyhledávání dle kompetencí řeší. Využili jsme k tomu hexagonální architekturu, která je pro náš případ užití dostatečně modulární. Dále jsme se v druhé části práce věnovali situaci na FEL ČVUT. Nejdříve jsme rozebrali datové zdroje znalostí, osob a pracovišť. Jako nejvýznamnější jsme vyhodnotili systémy Usermap a V3S.

Součástí této práce je také implementovaný prototyp, na kterém demonstrujeme funkčnost navrženého systému. Pro ukládání dat pomocí ontologií jsme využili existující *triple-store* GraphDB, jako schéma jsme použili existující ontologii SKOS rozšířenou o několik našich entit. Serverová část prototypu je implementovaná v jazyce JAVA za použití frameworku Spring Boot. Součástí prototypu je také klientská aplikace, která demonstruje průběh komunikace se serverem. V závěru této práce jsme rozebrali testování hexagonální architektury a některé z představených technik jsme aplikovali i na prototyp.

**Klíčová slova:** kompetence, vyhledávání, osoba, pracoviště, znalost, FEL, ČVUT, ontologie, SKOS

**Vedoucí:** Ing. Lukáš Zoubek  
Katedra ekonomiky, manažerství a humanitních věd

## Abstract

In this work, we dealt with competency-based people and workplace search at the CTU FEE.

First, we discussed the problem in general. In this part of the thesis, we decided that ontology would be a suitable representation of competencies. Mainly because, as the only formal language for knowledge representation, it has practical support for technologies such as RDF, OWL and SKOS. Furthermore, we analyzed possible data sources, especially the sources of people knowledge and the knowledge base. Finally, we dealt with data processing. We analyzed transformation into ontologies using tools such as OntoRefine. Then we described work with SPARQL.

In the second part of the thesis, we have designed a system that solves the competency-based search. We decided to use the hexagonal architecture, which is sufficiently modular for this use case. Moreover, in the second part of the thesis, we dealt with the situation at the CTU FEE. First, we discussed the data sources of knowledge, people and workplaces. We found Usermap and V3S the most important systems.

Last part of this work is a prototype, on which we demonstrate the suitable design of the system. We used existing *triple-store* GraphDB with SKOS ontology extended by several entities to store ontological data. The server part of the prototype is implemented in JAVA using the Spring Boot framework. The prototype also includes a client application that demonstrates the communication with the server side. At the end of this work, we discussed the testing of hexagonal architecture and applied some of the presented techniques to the prototype.

**Keywords:** competence, search, people, workplace, knowledge, FEE, CTU, ontology, SKOS

**Title translation:** Competency-based employee search

# Obsah

<b>1 Úvod</b>	<b>1</b>		
1.1 Předmluva	1		
1.2 Cíle a výstupy práce	1		
1.3 Motivace	1		
1.4 Struktura práce	2		
1.5 Rozsah práce	2		
<b>Část I</b>			
<b>Obecný problém</b>			
<b>2 Dekompozice problému</b>	<b>5</b>		
2.1 Seznam dílčích úkolů	5		
<b>3 Pojmy v kontextu této práce</b>	<b>7</b>		
3.1 Kompetence	7		
3.1.1 Složení kompetencí	7		
3.2 Znalosti a dovednosti	8		
3.3 Další pojmy	9		
Osoba	9		
Pracoviště	9		
Data	9		
Informace	9		
3.3.1 Jazyk	9		
3.4 Shrnutí	10		
<b>4 Konceptuální datový model</b>	<b>11</b>		
4.1 Osoby	11		
4.2 Pracoviště	11		
4.3 Kompetence	12		
4.4 Diagram konceptuálního modelu	12		
4.5 Shrnutí	12		
<b>5 Reprezentace dat</b>	<b>13</b>		
5.1 Kompetence	13		
5.1.1 Vymezení	13		
5.2 Znalosti	14		
5.3 Formální jazyky pro reprezentaci znalostí	15		
5.3.1 Sémantické sítě	15		
5.3.2 Rámce	16		
5.3.3 Pravidla	16		
5.3.4 Logika	17		
5.4 Ontologie	18		
5.4.1 Význam a Definice	18		
5.4.2 Základní charakteristiky	19		
5.4.3 Základní stavební kameny	19		
5.4.4 Zápis ontologií	19		
5.4.5 Použití ontologií	20		
5.4.6 Typy ontologií	20		
5.4.7 Přidružené technologie, standardy a další	21		
5.4.8 Reprezentace a datová struktura	23		
5.5 Shrnutí	23		
<b>6 Zdroj dat</b>	<b>25</b>		
6.1 Forma dat a přístup k nim	25		
6.2 Osoby a pracoviště	25		
6.3 Znalosti	26		
6.4 Znalosti osob	26		
6.5 Znalostní báze	27		
6.6 Shrnutí	27		
<b>7 Zpracování dat</b>	<b>29</b>		
7.1 Transformace dat	29		
7.1.1 Volba ontologie	30		
7.1.2 Transformace	30		
7.2 Operace nad daty	31		
7.2.1 Databáze	31		
7.3 CRUD operace	31		
7.4 Zpracování dat	32		
7.4.1 Síla znalosti	32		
7.5 Poskytování dat	32		
7.6 Shrnutí	33		
<b>8 Shrnutí první části</b>	<b>35</b>		
<b>Část II</b>			
<b>FEL ČVUT</b>			
<b>9 Úvod</b>	<b>39</b>		
9.1 Informační infrastruktura Fakulty elektrotechnické ČVUT	39		
<b>10 Analýza datových zdrojů</b>	<b>41</b>		
10.1 Kategorizace datových zdrojů	41		
10.2 Datové zdroje ČVUT	42		
10.2.1 Osoby a pracoviště	42		
10.2.2 Znalosti	43		
10.3 Externí (globální) datové zdroje	45		
10.3.1 Znalosti	45		
10.4 Shrnutí	47		
<b>11 Návrh aplikace</b>	<b>49</b>		
11.1 Podnět vzniku aplikace	49		
11.2 Vymezení této práce	50		
11.2.1 Rozsah řešené problematiky	50		
11.2.2 Rozsah navrženého systému	50		
11.2.3 Rozsah implementovaného prototypu	50		



11.3 Požadavky . . . . .	50	<b>13 Testování</b>	<b>71</b>
11.3.1 Základní business požadavek . . . . .	50	13.1 Testování hexagonální	
11.3.2 FURPS+ analýza . . . . .	51	architektury . . . . .	71
11.3.3 Případy užití . . . . .	52	13.1.1 Testování domény . . . . .	71
11.3.4 Wireframy . . . . .	52	13.1.2 Testování SPI . . . . .	72
11.4 Architektura aplikace . . . . .	52	13.1.3 Testování API . . . . .	72
11.4.1 Klient, Server . . . . .	53	13.2 Testování implementované	
11.4.2 Architektura . . . . .	53	aplikace . . . . .	72
11.4.3 Výběr architektury . . . . .	54	13.3 Průběh testu . . . . .	72
11.5 Hexagonální architektura - Porty		13.3.1 Testování persistentní vrstvy	73
a adaptéry . . . . .	56	13.3.2 Testování logiky . . . . .	74
11.6 Hexagonální architektura pro náš		13.3.3 Testování REST API . . . . .	75
případ . . . . .	58	13.4 Shrnutí . . . . .	75
11.6.1 Komponenty . . . . .	59	<b>14 Budoucí práce</b>	<b>77</b>
11.6.2 Komunikační sekvence . . . . .	60	14.1 Neprobraná témata . . . . .	77
11.7 Shrnutí . . . . .	60	14.1.1 Osobnost a dovednosti . . . . .	77
<b>12 Implementace</b>	<b>61</b>	14.1.2 Vlastní datový zdroj . . . . .	77
12.1 Rozsah implementovaného		14.1.3 Nestrukturovaná data . . . . .	77
prototypu . . . . .	61	14.1.4 Transformace dat . . . . .	78
12.1.1 Server . . . . .	61	14.2 Navazující práce . . . . .	78
12.1.2 Klient . . . . .	61	14.3 Výkon . . . . .	78
12.2 Diagram komponent . . . . .	61	14.4 Zvolený potup . . . . .	78
12.3 Hlavní použité technologie . . . . .	61	<b>15 Závěr</b>	<b>79</b>
12.3.1 Programovací jazyk pro		<b>Literatura</b>	<b>81</b>
serverovou část . . . . .	62	<b>Slovník</b>	<b>85</b>
12.3.2 Programovací jazyk pro část			
klient . . . . .	62		
12.3.3 Databáze . . . . .	62		
12.4 Struktura projektu . . . . .	62		
12.5 Doména . . . . .	63		
12.6 Spouštěcí modul . . . . .	64		
12.7 API adaptér . . . . .	64		
12.8 SPI adaptér . . . . .	65		
12.8.1 Přístup k datům . . . . .	65		
12.9 Klientská část prototypu . . . . .	66		
12.10 Databáze . . . . .	67		
12.10.1 Databázové schéma . . . . .	67		
12.10.2 OOM . . . . .	67		
12.11 Popis implementovaného řešení	68		
12.11.1 Jednotlivé kroky			
vyhledávání . . . . .	68		
12.11.2 SPARQL dotaz . . . . .	69		
12.12 Rozšiřitelnost . . . . .	69		
12.13 Dokumentace projektu . . . . .	69		
12.14 Shrnutí . . . . .	70		
		<b>Přílohy</b>	
		<b>A Konceptuální datový model</b>	<b>89</b>
		<b>B SKOS vizualizace</b>	<b>91</b>
		<b>C Diagram komponent</b>	<b>93</b>
		<b>D Sekvenční diagram</b>	<b>95</b>
		<b>E Doménový model</b>	<b>99</b>
		<b>F Diagram komponent pro naše</b>	
		<b>řešení FEL ČVUT</b>	<b>101</b>
		<b>G Ontologické schéma</b>	<b>103</b>
		<b>H SPARQL dotaz - vyhledávání</b>	
		<b>osob dle kompetencí</b>	<b>105</b>
		<b>I Výstup systému po vyhledání dle</b>	
		<b>kompetencí</b>	<b>107</b>
		<b>J Seznam použitých technologií,</b>	
		<b>nástrojů a knihoven</b>	<b>109</b>

**K Obsah přiloženého CD**

**111**



## Obrázky

3.1 Složení kompetencí znázorněná pomocí ledovce (zdroj [29]) . . . . .	8
5.1 Příklad sémantické sítě popisující některé rodinné vztahy (zdroj [51], přeložil autor) . . . . .	15
5.2 Typy ontologií (zdroj [51], přeložil autor) . . . . .	21
9.1 Schematické zobrazení informační infrastruktury ČVUT a komunikace systémů (zdroj autor) . . . . .	40
10.1 Webové rozhraní systému usermap (Dostupné z: <a href="https://usermap.cvut.cz/search">https://usermap.cvut.cz/search</a> ) . . . . .	42
10.2 Webové stránky pracovníka katedry matematiky, docenta Habaly (Dostupné z: <a href="https://math.feld.cvut.cz/habala/indexc.htm">https://math.feld.cvut.cz/habala/indexc.htm</a> ) . . . . .	44
10.3 Část ze schématu DBpedia v programu Protégé ( <a href="https://protege.stanford.edu/">https://protege.stanford.edu/</a> ) (zdroj autor) . . . . .	46
11.1 Obrázek k W-UC1 (zdroj autor) . . . . .	53
11.2 Cibulová architektura (zdroj <a href="https://dzone.com/">https://dzone.com/</a> ) . . . . .	56
11.3 Hexagonální architektura (zdroj [13]) . . . . .	57
11.4 Znázornění vzájemných závislostí v hexagonální architektuře (zdroj [7]) . . . . .	58
12.1 Snímek klientské části prototypu s načteným výsledkem hledání (zdroj autor) . . . . .	67
12.2 Mapování objektů zobrazené schématicky, příklad na objektu Koncept (zdroj autor) . . . . .	68
A.1 Diagram konceptuálního modelu (zdroj autor) . . . . .	89
B.1 Ontologie SKOS (zdroj: <a href="https://www.w3.org/2009/08/skos-reference/skos.rdf">https://www.w3.org/2009/08/skos-reference/skos.rdf</a> použitý nástroj: <a href="http://www.visualdataweb.de/webvow">http://www.visualdataweb.de/webvow</a> ) . . . . .	92
C.1 Diagram komponent (zdroj autor) . . . . .	94
D.1 Synchronní verze sekvenčního diagramu (zdroj autor) . . . . .	96
D.2 Asynchronní verze sekvenčního diagramu (zdroj autor) . . . . .	97
E.1 Doménový model (zdroj autor) . . . . .	99
F.1 Diagram komponent pro naše řešení (zdroj autor) . . . . .	102
G.1 Použité ontologické schéma (zdroj autor, použitý nástroj <a href="http://www.visualdataweb.de/webvowl/">http://www.visualdataweb.de/webvowl/</a> ) . . . . .	104

## Tabulky

10.1 Tabulka dostupných zdrojů osob a pracovišť FEL (zdroj autor) . . . . .	42
10.2 Tabulka dostupných zdrojů osobních znalostí na ČVUT (zdroj autor) . . . . .	43
10.3 Tabulka některých veřejně dostupných zdrojů osobních znalostí (zdroj autor) . . . . .	45
11.1 Tabulka vrstev architektury dle Browna (zdroj [23], přeložil autor)	55
13.1 Tabulka testovacích případů pro REST API . . . . .	75

# Kapitola 1

## Úvod

### 1.1 Předmluva

V této práci se budeme zabývat problematikou vyhledávání osob a pracovišť dle jejich kompetencí. Problém budeme nejdříve řešit naprosto obecně - vyhledávání osob dle kompetencí v obecné organizaci (firma, vědecký ústav, škola). Následně rozebereme konkrétní případ takového vyhledávání na Fakultě Elektrotechnické ČVUT v Praze (FEL ČVUT).

### 1.2 Cíle a výstupy práce

- Zanalyzovat problematiku vyhledávání osob a pracovišť dle kompetencí (výstup: dokument analýzy)
- Navrhnout systém pro vyhledávání osob a pracovišť dle kompetencí a provést rešerši dostupných dat (výstup: dokument návrhu systému a rešerše zdrojů)
- Implementovat prototyp systému pro vyhledávání osob a pracovišť dle kompetencí pro FEL ČVUT (výstup: zdokumentovaný a otestovaný prototyp navržené aplikace)

V pokynech k vypracování této práce jsou zmíněny další dílčí cíle, které navazují na výše zmíněné, hlavně potom na poslední dva. První cíl je přítomen kvůli systematickému přístupu k problému, který je netriviální a je třeba zhodnotit všechny okolnosti.

V jednotlivých kapitolách této práce budeme cíle dále rozpracovávat.

### 1.3 Motivace

Problematika reprezentace znalostí, jejich získávání a interpretace je v dnešní době velmi rozvinutá, hlavně v oblasti umělé inteligence. Je důležité říct, že tato práce se nezabývá znalostmi přímo v kontextu umělé inteligence, použité principy však staví na stejných základech. Zejména se potom jedná o ontologie a sémantický web [51], jakožto rychle se rozvíjející oblast datových věd.

Bez pochyby nelze popřít fakt, že vyhledávání osob dle jejich kompetencí (znalostí) je snem každého pracovníka HR oddělení. Tato problematika má velké komerční

využití. Jedním příkladem za všechny může být síť LinkedIn (dostupná z: <https://www.linkedin.com/>), pomocí které je možné nalézt odborníky z mnoha oblastí, čehož HR oddělení firem hojně využívají.

Tato síť staví kompetenční podklad na profilech vytvořených samotnými vlastníky a částečně na doporučení ostatních lidí. V této práci jsme se na celý problém zaměřili s větším nadhledem. Zkoumali jsme mimo jiné i způsob, jak informace o znalostech čerpat strojově z díla dané osoby (vědecké články, patenty aj.), jelikož takové zdroje můžeme považovat za věrohodnější než ručně vyplněné profily.

Na druhé straně se naskýtá i nekomerční využití, například v univerzitním prostředí je velmi typické hledat kompetentní osoby k vybraným činnostem - vedoucí závěrečné práce, konzultant k výzkumu a jiné.

## 1.4 Struktura práce

Práci jsme rozdělili do dvou celků, v každém z nich rozpracováváme některé z výše uvedených cílů. Tyto cíle případně dílčí úkoly jsou definovány vždy v úvodu každé části.

V první části (I) rozebíráme obecný problém, který máme v rámci práce řešit. Tento problém v úvodu nejdříve rozdělíme na dílčí úkoly a ty následně v jednotlivých kapitolách rozebíráme podrobněji. Cílem této části je rozebrat každý z dílčích úkolů a navrhnout způsob řešení, přidružené technologie a další podrobnosti.

Ve druhé části (II) navazujeme na první a celý problém již konkretizujeme. Převážně se věnujeme návrhu systému, kterým chceme náš problém řešit. Dále se věnujeme popisu implementovaného prototypu pro konkrétní případ užití na FEL ČVUT.

## 1.5 Rozsah práce

Rozsah této práce je větší než je u bakalářské práce obvyklé. Důvodem je, že očekáváme další navazující bakalářské, diplomové, disertační či jiné práce. V rámci této práce jsme se rozhodli rozebrat teoretický podklad a implementovat prototyp, na kterém jsme ověřili, že navazující práce mohou téma dále rozvíjet.



**Část I**

**Obecný problém**





## Kapitola 2

### Dekompozice problému

Konečným cílem naší práce je navrhnout na FEL ČVUT řešení pro vyhledávání osob a pracovišť dle jejich kompetencí. Předtím než přistoupíme k tomuto konkrétnímu problému, je třeba se na celou záležitost podívat obecněji. Zcela obecným problémem, který je třeba vyřešit, je vyhledávání osob a pracovišť dle jejich kompetencí ze specifikované množiny lidí (např. firma, stát, univerzita). Na dekompozici tohoto obecného problému se zaměříme v následující sekci. Situací na FEL ČVUT se budeme zabývat v další části práce (II).

#### 2.1 Seznam dílčích úkolů

1. Datový model a datová struktura používaná pro uložení dat
  - Která data potřebujeme?
  - Jak budeme data ukládat?
2. Zdroj potřebných dat
  - Kde data získáme?
  - Jak je získáme?
3. Zpracování získaných dat
  - Jak data transformujeme do příslušné datové struktury?
  - Jak data dále zpracujeme, abychom získali naší přidanou hodnotu (všechny potřebné metriky a přídatné informace)?
  - Jakým způsobem budeme provádět operace s daty (vyhledávání, editace, vytváření, mazání)?
  - Jakým způsobem budeme data poskytovat?

V následujících kapitolách rozebereme jednotlivé problémy podrobněji.



## Kapitola 3

### Pojmy v kontextu této práce

V této kapitole definujeme základní pojmy v kontextu této práce. Pojmy, jako kompetence, často nemají jen jednu definici, proto je nutné vybrat právě tu, v jejíž souvislosti budeme daný pojem používat. Ve většině případů se jedná o stručné definice, někdy je však potřeba pojem definovat podrobně a zmínit i další okolnosti.

#### 3.1 Kompetence

Pojem kompetence se vyskytuje v literatuře hned v několika významech. Dva nejdůležitější jsou: kompetence ve smyslu oprávnění a kompetence ve smyslu schopnost. Oba významy jsou si velmi podobné.

První význam je dostupný např. v českém slovníku cizích slov. Tento slovník mimo jiné uvádí, že kompetence je definována jako pravomoc nebo rozsah pravomocí.[33]

Druhý význam, který uvádí Cambridge dictionary je následující: Kompetence je schopnost osoby vykonat danou činnost správně (tj. uspokojivě popř. efektivně) [8] (překlad autor).

První případ, tedy kompetence ve smyslu oprávnění, se v našem případě nehodí. Zkoumáme totiž více kompetence ve smyslu způsobilosti v rámci nabitých dovedností a znalostí, nikoliv získaných oprávnění. Na druhou stranu oprávnění člověk často nabývá na základě svých znalostí a dovedností, takže jsou si významy opravdu velmi blízké. Přesto v této práci vezmeme za svou druhou definici.

Z upřednostněné definice plyne, že kompetence vždy souvisí s danou činností - tu nazveme předmětem kompetence. Dále si ještě definujeme slovní spojení *být kompetentní*. Tím navážeme na samotnou definici kompetence. Být kompetentní znamená: Mít schopnost vykonat danou činnost správně (tj. uspokojivě, efektivně).

Poslední pojem spojený s kompetencemi je míra kompetence. Míra kompetence je metrika vyjadřující, jak moc je osoba kompetentní.

##### 3.1.1 Složení kompetencí

Každá osoba má sadu svých kompetencí podmíněnou mnoha faktory. Menší část těchto faktorů lze poměrně jednoznačně (i když složitě) popsat - obr. č. 3.1, část nad hladinou ledovce, tedy znalosti a dovednosti. Naproti tomu větší část, bližší k osobnosti člověka a jeho vnitřnímu vnímání, je velmi těžké popsat a získat - obr. č. 3.1, část pod hladinou ledovce.



**Obrázek 3.1:** Složení kompetencí znázorněná pomocí ledovce (zdroj [29])

## 3.2 Znalosti a dovednosti

Znalosti a dovednosti jsou ze všech složek kompetencí nejlépe uchopitelné. To však není, vzhledem k jejich komplexnosti, nijak uspokojivé. Důvodem je, že u nich lze předpokládat, že je buď sám vlastník umí popsat, nebo je možné je poměrně jednoznačně vyčíst z jeho činů (např. z jeho vědeckých prací, z potvrzených referencí zaměstnavatelem). Na rozdíl od rysů, motivů a sociálních rolí (obr. č. 3.1), které ani samotný vlastník často nedokáže přesně definovat.

Je tedy velmi pravděpodobné, že optimální cesta za vyhledáváním dle kompetencí povede okolo znalostí a dovedností. Pro začátek si je definujeme.

### Znalost

Pochopení určité informace, které osoba získá zkušeností nebo studiem. [8] (přeložil autor)

#### Typy znalostí:

- Deklarativní
  - Popisují, jaké věci známe a co o nich víme; např. pes je zvíře, pes má ocas
  - Mohou být popsány pomocí grafů a podobných struktur
- Procedurální
  - Popisují, jak věci fungují, např. jestliže je pes hladový, najde si jídlo a sní ho
  - Mohou být popsány pomocí pravidel [47]

Znalosti ještě v kontextu jejich vyvozování dělíme na *explicitní* a *implicitní*. *Explicitní* znalosti jsou znalosti, které byly objektu (počítači, osobě) přímo sděleny, libovolnou formou. *Implicitní* znalosti je poté objekt schopen sám vyvodit na základě znalostí explicitních (např. pomocí logiky).[51]

Často používaným pojmem je *znalostní báze*, která se nejčastěji pojí k nějakému seskupení lidí, komunitě, popř. k jinému jednotlicímu prvku. V podstatě se jedná o množinu znalostí a jejich propojení, které se vztahují ke zvolené jednotlicí entitě např. firmě, vědeckému výzkumu, nebo i k určitému vědnímu oboru. Například součástí znalostní báze oboru informatika by mohly být všechny známé pojmy s jejich definicemi, propojené různými sémantickými vazbami (nadřazenost, podřazenost, souvislost a jiné).

### ■ Dovednost

Dovednost je učením a praxí získaná dispozice ke správnému, kvalitnímu, rychlému a úspornému vykonávání určité činnosti vhodnou metodou [27]. Jinými slovy je to schopnost využít znalosti efektivně k vykonávání určité činnosti [12].

## ■ 3.3 Další pojmy

### ■ Osoba

V kontextu této práce je osoba jakýkoliv člověk.

### ■ Pracoviště

Pracoviště je v kontextu této práce seskupení osob. Obvykle má název, místo působení a vedoucí osobou. Dále se dělí na týmy, které pracují společně na cílech definovaných vedoucím.

V této práci pojednáváme o vyhledávání osob a pracovišť dle kompetencí. Proto je nutné definovat, co znamená pojem kompetence pracoviště. Pracoviště je kompetentní k určitému předmětu kompetence v případě, že je kompetentní osoba, která je jeho součástí. Váha dané kompetence pracoviště je potom určena součtem váh dané kompetence přes všechny osoby na daném pracovišti.

### ■ Data

Fakta či fikce zapsané v čitelné podobě, čitelné pro stroje či lidi.

### ■ Informace

Informace jsou data o něčem nebo o někom. Jinými slovy jsou to data, která mají daný význam.

### ■ 3.3.1 Jazyk

Obecný jazyk definujeme jako prostředek komunikace mezi dvěma a více objekty, nebo jejich částmi. Jazyk je používán k reprezentaci informací.



## Kapitola 4

### Konceptuální datový model

V následující kapitole rozebereme zkoumanou doménu. Tím mimo jiného odpovíme i na otázku *Která data potřebujeme?* (definovanou v kapitole č. 2). Ujasníme si, které entity se v naší doméně vyskytují, jaké mají vlastnosti a jaké jsou mezi nimi vztahy. Ze zadání vyplývají základní řešené entity – osoby, pracoviště, kompetence.

#### 4.1 Osoby

##### Vlastnosti:

- Identifikátor v rámci organizace
- Jméno
- Příjmení
- Pracoviště
- *Abstraktní hodnocení* - vědecké (např. h-index), pracovní (roky zkušeností) a jiné
- Znalosti a dovednosti
- Osobnost - společenské vnímání, vnímání sebe sama (viz obr. č. 3.1 pod hladinou)

#### 4.2 Pracoviště

##### Vlastnosti:

- Identifikátor v rámci organizace
- Název
- Skupina osob
- Vedoucí





# Kapitola 5

## Reprezentace dat

V této kapitole odpovíme na otázku *Jak budeme data ukládat?*. Rozebereme tedy možnosti reprezentace jednotlivých entit z konceptuálního modelu (obr. A.1). V závěru této sekce bude jasné, která reprezentace je pro náš případ nejvhodnější a proč.

Některé entity z konceptuálního modelu je možné reprezentovat poměrně jednoduše - osoby a pracoviště. Stačí nám na to objektové paradigma [54], které se obvykle při modelování systémů používá.

Naproti tomu pracujeme s kompetencemi, znalostmi, dovednostmi a osobností. Modularita těchto entit je větší než u osob a pracovišť. Reprezentovat je klasickým objektovým způsobem může být velmi komplikované. U těchto entit je třeba jít do většího detailu. Reprezentaci osob a pracovišť můžeme později přizpůsobit dle zvolené reprezentace ostatních entit.

### 5.1 Kompetence

Kompetence osoby, jak je vidět z konceptuálního modelu (příloha A.1), podmiňují dva vlivy, osobnost a znalosti (resp. dovednosti). Tyto dvě oblasti určují jaké kompetence daná osoba má a v jaké míře.

#### 5.1.1 Vymezení

Reprezentace osobnosti tvoří velmi komplikovanou problematiku, na toto téma odkážeme v sekci budoucí práce (kapitola 14), jelikož stojí za další zkoumání. V této práci se tímto však nebudeme více zabývat, místo toho se zaměříme na znalosti.

Omezíme se na reprezentaci kompetencí pomocí znalostí a dovedností a připodobníme míru kompetence k síle znalosti popř. dovednosti, z diagramu (příloha A.1).

Dále problematiku zúžíme pouze na znalosti, jelikož z definice dovednosti plyne, že je závislá na znalosti. Není správné dovednosti úplně vyloučit, i na ně odkážeme v kapitole budoucí práce (kapitola 14). Společně se zkušenostmi by měly být dalším důležitým okruhem, který staví na znalostech a mimo jiné popisuje, jak je osoba umí využívat.

Otázku jsme tedy zúžili z reprezentace kompetencí na reprezentaci znalostí, na tu se zaměříme v další sekci.

## 5.2 Znalosti

Reprezentace znalostí (angl. *knowledge representation*) je společně s uvožováním (angl. *reasoning*) prudce se rozvíjejícím oborem umělé inteligence. V této sekci popíšeme, co vůbec reprezentace znalostí je a představíme si vybrané formální jazyky, které pro tuto reprezentaci lze využít (v některých zdrojích se mluví o formalismech, v jiných o formách, my jim budeme říkat formální jazyky, protože, to je zdaleka nejjednoznačnější pojmenování).

### Co je reprezentace znalostí

Dříve než přistoupíme ke konkrétním formálním jazykům, musíme si ujasnit termín reprezentace znalostí. Reprezentace znalostí se dá shrnout do čtyř základních charakteristik:

- Náhrada reality
- Množina ontologických závazků
- Fragmentární teorie inteligentního uvažování
- Prostředek pro počítač i člověka [21]

### Náhrada reality

Pod touto charakteristikou leží základní filozofický předpoklad, že realita existuje nezávisle na jakémkoliv pozorovateli - počítači, člověku aj.

Náhrada reality, tedy reprezentace znalostí, nám umožňuje uvažovat a modelovat situace. Základem uvažování je vyvozování závěrů bez samotných činů. Příkladem může být plán stavby domu. Anž bychom do ruky vzali jediný nástroj či materiál, jsme schopni dům zkonstruovat v rozsahu svých znalostí pouze ve vlastní hlavě.

Kvalita a rozsah naší reprezentace se vždy odvíjí od přesného účelu, pro který znalosti používáme/uchováváme. Je velmi nepravděpodobné, že by se nám podařilo reprezentovat naprosto všechno bez rozdílu od reality, je proto důležité vědět, jak moc a čím se reprezentace od reality liší.[40]

### Ontologické závazky

Předpokládáme, že nelze reprezentovat vždy všechno, každá reprezentace je v podstatě jen aproximací reality. Reprezentaci volí sám pozorovatel, tím dělá rozhodnutí, které znalosti bude reprezentovat a jak. Volba nejpříhodnější reprezentace je tedy množinou takových rozhodnutí - přesněji ontologických závazků.[21]

### Fragmentární teorie inteligentního uvažování

Reprezentace znalostí je známka inteligentního chování. Existují nejméně dva pohledy na to, co je inteligentní chování. První z nich odkazuje na logiku a říká, že cokoli,

co se chová podle logických pravidel, je inteligentní. Druhý pohled nahrazuje logická pravidla množinou mechanismů inspirovaných procesem poznáváním.[40]

Oběma těmito přístupy se existující formální jazyky inspiřují, některé využívají logiku, některé proces poznávání. [40]

### ■ Prostředek pro počítač i člověka

Počítačové zpracování znalostí reprezentovaných formálními jazyky pro to určenými musí být efektivní. Často platí, že čím je jazyk složitější, tím je složitější jej zpracovat.[40]

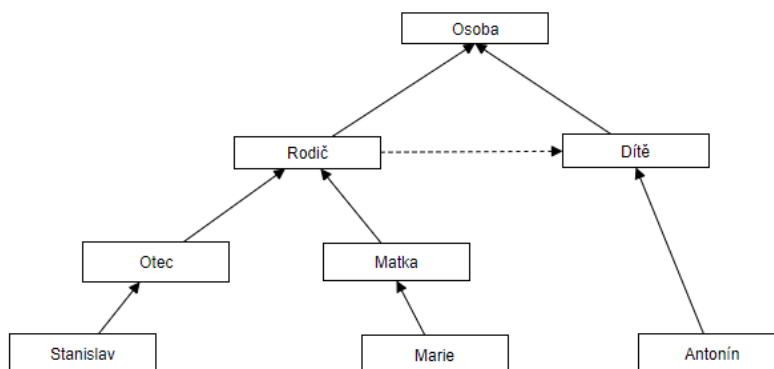
Naproti tomu musí být jazyk dobře pochopitelný pro člověka. Zde se nabízí přirozený jazyk, ten však díky svým vlastnostem (kapitola č. 3 sekce č. 3.3.1) není vhodný. Bude-li jazyk špatně pochopitelný pro člověka, nebude možné rychle ověřit pravdivost/důvěryhodnost sdělení vyjádřeného pomocí tohoto jazyka. Takový jazyk vědecká ani komerční komunita nepřijme. [40]

## ■ 5.3 Formální jazyky pro reprezentaci znalostí

V této sekci shrneme existující formální jazyky pro reprezentaci znalostí. Každý z nich zevrubně popíšeme, abychom v závěru této kapitoly mohli rozhodnout, který se pro naše účely hodí nejvíce.

### ■ 5.3.1 Sémantické sítě

Sémantická síť je graf, jehož uzly reprezentují koncepty a individuality, hrany potom reprezentují vztahy mezi nimi.[40]



**Obrázek 5.1:** Příklad sémantické sítě popisující některé rodinné vztahy (zdroj [51], přeložil autor)

**Individuality** reprezentují konkrétní objekty ze zkoumané domény, na obrázku č. 5.1 jsou to např. otec Stanislav nebo dítě Antonín.

**Koncepty** jsou množiny individualit. Na obrázku č. 5.1 jsou to např. rodič a osoba, které reprezentují všechny rodiče a všechny lidi ve sledované doméně.[40]

Každý uzel má své vlastnosti, jsou jimi atributy a relace. Atributem může být např. věk u osoby. Základními relacemi jsou IS-A a *instance-of*. Relace IS-A definuje hierarchii konceptů (tzv. taxonomii), v obrázku č.5.1 platí, že matka IS-A rodič. Vztah IS-A indikuje dědění všech vlastností nadřazeného konceptu konceptem podřízeným (např. matka dědí všechny vlastnosti rodiče). Vztah *instance-of* je vždy mezi konceptem a individualitou, v obr. č. 5.1 platí Stanislav *instance-of* otec.[40] Nejedná se samozřejmě o všechny druhy vazeb, jen o ty klíčové.

Sémantické sítě se nejčastěji používají pro modelování deklarativních znalostí, lze je však použít i pro procedurální znalosti.[47]

Původní význam slova *sémantická síť* je *fyzický model lidské paměti* [44]. Sémantické sítě se člověku snadněji chápou a práce s nimi je přirozenější než u dalších modelů, hlavně u těch logických.

Nevýhodou sémantických sítí je, že neexistuje jednotný standard pro interpretaci a jednotný standard pro hodnoty uzlů a hran - tedy neexistuje jednotná sémantika. Další nevýhodou je dědičnost, která nepodporuje žádné výjimečné stavy (např. tučňák je pták, tedy měl by mít všechny vlastnosti ptáků, ale nelétá). [44]

### ■ 5.3.2 Rámce

Sítí rámců (angl. frames) lze považovat za jeden z typů sémantických sítí. Vedle deklarativních znalostí, které reprezentuje obecná sémantická síť, reprezentují rámce i procedurální znalosti.

Základním stavebním kamenem této sítě jsou rámce. Ty v sobě uchovávají proceduru a skupinu atributů, které popisují danou situaci. Rámce se inspiřují v lidské paměti, která si uchovává situace - kombinace procedurálních a deklarativních znalostí. Tak může lidský mozek například na dvě, na první pohled rozdílné, situace reagovat stejným způsobem, protože nalezne podobnost.[47]

Rámce řeší některé nedostatky sémantických sítí jako je: řešení výjimečných stavů, kontrola vnitřní konzistence a zamezení konfliktům atributů při dědičnosti. Hlavní nevýhodou je složitost použití.[44]

### ■ 5.3.3 Pravidla

Deklarativní i procedurální znalosti lze reprezentovat jako množinu pravidel. Typický tvar těchto pravidel je tzv. JESTLIŽE-PAK (angl. IF-THEN), například *jestliže je nalezena odpověď, pak jí přestaň hledat, jinak pokračuj v hledání*[47]. Takovou reprezentaci využívají např. logické programovací jazyky.[44]

Výhodou pravidel je, že jsou modulární a zároveň jsou poměrně prostorově nenáročná. Jednotlivá pravidla mohou být přidávána a odebírána bez ovlivnění zbylých. Mezi nevýhody patří hlavně složitost odhalení rozporu mezi pravidly a složitost reprezentace složitějších strukturovaných znalostí (svazuje nás přesná forma). V neposlední řadě je velmi složité předpovědět akce, které na základě pravidel nastanou, a tak se pravidla mohou nekonečně řetězit.[44]

Pravidla se samozřejmě (např. v programovacích jazycích) aktivně využívají, ale nehodí se pro reprezentaci komplexních znalostí. V programovacích jazycích se používají velmi hojně pro řízení toku programu, ale oproti reprezentaci celé zkoumané domény znalostí je to stále poměrně málo.

### ■ 5.3.4 Logika

Logika mezi ostatními formálními jazyky pro reprezentaci znalostí převládá [51]. V této sekci projdeme klíčové koncepty logiky, nejdříve se zaměříme na obecnou rovinu, poté popíšeme více do detailu deskripční logiku.

Logika má mnoho forem, některé z nich se používají, nebo alespoň byly použity, pro reprezentaci znalostí. [44] Jen některé z těchto vyzkoušených forem se osvědčily.

V předchozích způsobech reprezentace, zejména potom u sémantických sítí, chyběla jednotně definovaná sémantika, proto je velmi komplikované tyto způsoby přímo aplikovat. [51] Znalosti vyjádřené pomocí sémantických sítí mohou být jednoduše vyjádřeny formou logiky, která sítím přidává potřebnou sémantiku. [44]

Následující dvě zobrazení reprezentují totéž:

$$\boxed{\text{Zaměstnanec}} \xrightarrow{\text{je druhem}} \boxed{\text{Osoba}} \quad \forall x : (\text{Zaměstnanec}(x) \rightarrow \text{Člověk}(x))$$

Stejně jako je tomu u sémantických sítí, tak i u logiky platí, že jsme pomocí ní schopni vyjádřit naprostou většinu přirozeného jazyka. Je důležité si uvědomit, že přestože má logika v sémantice navrch nad sítěmi, sítě jsou intuitivnější pro lidské chápání.

Formální jazyky založené na logice obvykle zakládají na logice prvního řádu (predikátové logice), zkratka FOL.[44]

### ■ Deskripční logika

Deskripční logika je natolik expresivním jazykem, že se v dnešní době stala nejčastějším logickým paradigmatem pro reprezentaci znalostí, zejména potom v sémantickém webu. [51].

Deskripční logika je téměř výhradně podmnožinou logiky prvního řádu (FOL), tedy logiky predikátové [51]. FOL popisuje doménu pomocí objektů (entity mající vlastní identitu), okolo těchto entit jsou konstruovány, pomocí funkcí, proměnných a logických spojek, logické vzorce (predikáty) [49]. Deskripční logika je typicky omezena pouze na unární a binární predikáty z FOL. [51]

#### Základními konstrukty jsou:

- Atomické koncepty
  - reprezentují pojmenované unární predikáty
  - např. *Rodič*
- Atomické role
  - reprezentují pojmenované binární predikáty
  - např. *má dítě, je dítětem*
- Individuality
  - reprezentují prvky konceptů, tzn. jednotlivce
  - např. *Jiří* [35]



### ■ 5.4.2 Základní charakteristiky

#### 1. Formální

- Ontologie musí být vyjádřena formálním jazykem (výhody těchto jazyků jsou popsány v kapitole č. 3 sekce č. 3.3.1).

#### 2. Explicitní

- Znalost musí být vyjádřena maximálně explicitně, pro optimální strojové zpracování.

#### 3. Sdílené

- Ontologie vždy reflektuje dohodu skupiny lidí v dané doméně, v podstatě se jedná o kolektivní ontologické závazky (viz sekce č. 5.2).

#### 4. Konceptuální

- Ontologie popisuje znalosti pomocí symbolů, které reprezentují koncepty a relace mezi nimi. Konceptualita spočívá v tom, že jsou ontologie intuitivně uchopitelné pro člověka, protože odpovídají lidskému mentálnímu modelu znalostí. Jako protipříklad lze uvést neuronové sítě, které též reprezentují určité znalosti, ale přirozenému lidskému chápání je to již vzdáleno.

#### 5. Doménově specifická

- Ontologie je vždy omezena na určitou oblast - doménu.[51]

### ■ 5.4.3 Základní stavební kameny

#### ■ Koncepty

- V sémantických sítích i v deskripční logice se tento konstrukt nazývá stejně
- Reprezentují ontologické kategorie

#### ■ Relace

- Analogicky se jedná o hrany sémantické sítě nebo role v deskripční logice
- Propojují koncepty a instance - reprezentují vazby mezi nimi

#### ■ Instance

- Individuality v sémantické síti nebo v deskripční logice
- Reprezentují konkrétní objekty v doméně [51]

### ■ 5.4.4 Zápis ontologií

Ontologie lze zapisovat několika způsoby. Například pomocí sémantických sítí, které typicky neobsahují všechny důležité informace, jelikož nejsou tak expresivní (viz sekce č. 5.1). Toto zobrazení je příznivé hlavně pro pochopení lidmi. Dále je zapisujeme pomocí serializovaného formátu (např. XML), vhodného pro strojové zpracování. Dále pomocí logických vzorců, to je vhodné pro další vyvozování (angl. reasoning). [51].

### 5.4.5 Použití ontologií

- **Propojení znalostí** - jednotný znalostní slovník napříč aplikacemi, datový zdroj znalostí
- **Abstrakce znalostí** - zdroj schématu znalostí (hierarchie a vztahy konceptů) pro budoucí využití
- **Automatizace zpracování znalostí** - automatické vyvozování závěrů z platných axiomů (angl. reasoning)
- **Integrace informací** - integrace na úrovni různých datových schémat a interpretace získaných dat z jednoho zdroje v rozdílném schématu
- **Získávání informací** - zdokonalování modelu znalostí s důrazem na jeho prohlédávání
- **Sémantická správa obsahu** - přidávání metadat k datům, která později slouží k jejich lepší identifikaci a lepšímu přístupu k nim
- **Organizace znalostí** - sdílený znalostní model daného seskupení lidí (např. firma, škola)
- **Expertní systémy** - systémy schopné řešit komplikované otázky (úlohy) - např. i v medicíně [51]

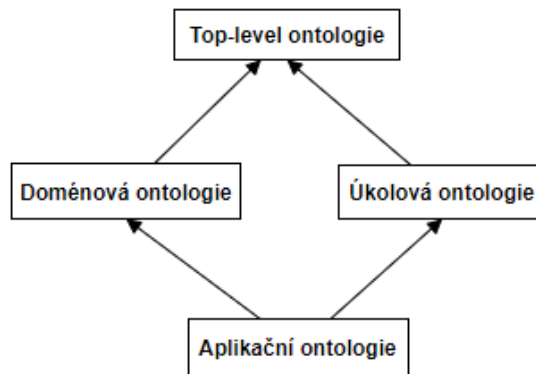
### 5.4.6 Typy ontologií

Existují čtyři základní typy ontologií. Jednotlivé typy se od sebe liší úrovní abstrakce reprezentovaných znalostí. [51]

- **Top-level**
  - Abstraktní a obecné koncepty (např. *fyzický objekt*, *abstraktní objekt*) s velkým potenciálem znovupoužití v mnoha doménách
  - Typicky se používají jako předloha pro tvorbu ontologií zmíněných níže
  - Příklady UFO [26], SUMO [42], DOLCE [24]
- **Doménové a úkolové ontologie**
  - Zachytávají znalost ve specifické doméně - např. medicíně, geografii; nebo znalost o specifickém úkolu - např. diagnostika, konfigurace
- **Aplikační ontologie**
  - Propojují doménové a úkolové ontologie pro účely konkrétní aplikace

Jedná se o inkluzivní hierarchii, nižší ontologie přejímají a konkretizují obecné koncepty a relace vyšších ontologií. [51]





Obrázek 5.2: Typy ontologií (zdroj [51], přeložil autor)

### ■ 5.4.7 Přidružené technologie, standardy a další

#### ■ RDF

V doslovném překladu se jedná o framework pro popis zdrojů. RDF je standardizovaný jazyk pro reprezentaci znalostí (informací) na webu. Tento jazyk lze však použít pro reprezentaci libovolných znalostí.

**Základními elementy jsou:**

- URI (Uniformní identifikátor zdroje) - slouží k pojmenování a identifikaci individualit, konceptů a jejich vlastností (příklad je *urn:isbn:0451450523*, což je URI knihy)
- Věty ve tvaru:  $\boxed{\text{Podmět}} \xrightarrow{\text{Predikát}} \boxed{\text{Předmět}}$  (jednotlivé členy jsou ve tvaru URI)

RDF je typicky zpracováváno v serializované podobě v XML formátu (viz příklad níže) [51]. Poměrně populární variantou pro serializaci RDF je též JSON-LD [37] nebo Turtle [20].

```

<rdf:Description rdf:about="http://ubiqbiz.com/web/MrX.html">
  <btr:hasAuthor rdf:resource="btr:PanX"/>
</rdf:Description>
<rdf:Description rdf:about="btr:PanX">
  <btr:employedAt rdf:resource="btr:UbiqBiz"/>
</rdf:Description>
  
```

Listing 5.1: Příklad syntaxe RDF XML

RDF je framework pro prostý popis zdrojů, pokud chceme datům dávat význam a s tímto významem pracovat, musíme využít další rozšíření. Typicky datům dodáváme soubor axiomů pro další vyvozování (angl. inference) popř. uvažování (angl. reasoning), které nám umožní je dále rozvíjet a efektivně s nimi pracovat. V následujícím seznamu uvádíme příklady takových rozšíření.



### ■ 5.4.8 Repräsentace a datová struktura

Každá reprezentace je nakonec realizována pomocí některé známé datové struktury. Například u sémantické sítě je to typicky graf. Je důležité striktně dodržovat sémantiku dané reprezentace, která ne vždy platí obecně pro použitou datovou strukturu. Například sémantická síť popisující rodinné vztahy v sobě netoleruje cykly, to ale pro obecný graf neplatí. [21]

## ■ 5.5 Shrnutí

Cílem této kapitoly bylo odpovědět na další z dílčích otázek našeho problému: *Jak budeme data ukládat?* Rozebrali jsme tedy možnosti reprezentace jednotlivých entit a nyní můžeme rozhodnout, která je pro naše účely nejvhodnější.

Je důležité zmínit, že námi zvolený postup jistě není jediný možný. Přestože v této kapitole vybereme jednu z představených možností, nadále budeme pracovat tak, aby naše řešení bylo modulární. Budeme jej navrhovat tak, aby jeho části bylo možné nahrazovat vylepšenými případně změněnými (hlavně při návrhu systému - kapitola č. 11).

Klíčovou částí této kapitoly byly formální jazyky pro reprezentaci znalostí. Jazyky byly představeny s vzestupnou tendencí, co se týká praktického použití. První představené, hlavně sémantické sítě, byly lépe pochopitelné pro člověka, ale postrádaly dostatečně definovanou sémantiku pro praktické aplikace. Naproti tomu později představená deskripční logika je kvalitnějším jazykem, jelikož předchozím formálním jazykům dodává zmíněnou potřebnou sémantiku.

Nakonec jsme se v této kapitole zabývali ontologiemi. Ontologie jsou více realizací zmíněných formálních jazyků, než že by do této problematiky přinášeli něco úplně nového. Ontologie staví na deskripční logice a přebírají expresivitu tohoto jazyka, zároveň jsou vizualizovány pomocí sémantických sítí, které jsou pro práci přehlednější pro člověka. V kontextu ontologií existují konkrétní jazyky (RDF, OWL) a praktické aplikace (SKOS), které jsou již ověřené.

Vzhledem k vlastnostem, které jsme zmínili, jsou ontologie a množina přílehlých technologií v našem případě vhodnou volbou pro reprezentaci znalostí.



# Kapitola 6

## Zdroj dat

V této kapitole rozebereme další dílčí problematiku, která je součástí našeho komplexního problému - zdroje dat.

Nejdříve se zaměříme na zdroje dat o pracovištích a osobách. Poté rozebereme zdroje znalostí s důrazem na strojovou zpracovatelnost.

### 6.1 Forma dat a přístup k nim

Přístup k datům je obvykle realizován pomocí webových rozhraní (tzv. API). Pro komunikaci s tímto API se obvykle využívá protokol HTTP. Velmi často používaným konceptem pro návrh rozhraní je REST [1]. Rozvíjejícím se dotazovacím jazykem se díky své modularitě stává GraphQL [4], který operuje také nad zmíněným HTTP protokolem.

Přístup k datům je typicky zabezpečen a není možné, aby jej získal kdokoliv. Často je nutné zažádat o přístup a poskytnout přesný účel, pro který data chceme získat. Každý datový zdroj má obvykle svou autentizační proceduru, tu je třeba vyjednat s poskytovatelem dat.

Po překonání autentizační procedury jsou data přístupná, obvykle v serializovaném formátu. Nejčastější serializované formáty jsou JSON [18] a XML [52]. V těchto formátech lze serializovat libovolná strukturovaná data například i RDF (viz kapitola č. 5.2).

Získaná data jsou samozřejmě strukturována dle datového schématu poskytovatele. Pokud se naše schéma liší od schématu poskytovatele, musíme data příslušným způsobem transformovat, tento problém vyřešíme v pozdějších kapitolách. Možností, jak data získat, je nespočet, zmínili jsme jen neobvyklejší metody.

### 6.2 Osoby a pracoviště

Data o těchto entitách jsou typicky dostupná v databázi/evidenci pracovníků resp. členů daného seskupení (firma, škola aj.).

S daty o osobách je nutno pracovat dle směrnice GDPR [6].

## 6.3 Znalosti

Znalosti mají v databázi kompetencí velmi důležitou roli. Komplexita této entity ve své podstatě určuje i práci s ostatními entitami a fungování celého systému.

V kontextu znalostí musíme vyřešit následující problémy:

- Znalosti osob
- Znalostní báze

Při vyhledávání osob dle znalostí můžeme zvolit různé přístupy v závislosti na získávaných znalostech. Identifikovali jsme dvě kategorie řešení:

- Se sémantikou - pracuje s významem vyhledávaných konceptů
- Bez sémantiky - pracuje na jiném principu, nebere v úvahu sémantiku

Na základě tohoto rozdělení můžeme konstatovat, že pokud se jedná o řešení bez sémantiky, z pravidla toto řešení nebude potřebovat znalostní bázi. Jedná se například o fulltextový vyhledávač přes danou sadu dokumentů a jednoduchá interpretace výsledku.

V případě řešení se sémantikou je třeba znalostní bázi vytvořit a udržovat. Příkladem takového řešení může být znalostní báze uložená pomocí ontologií a následné vyhledávání konceptů v tomto ontologickém slovníku.

## 6.4 Znalosti osob

Pro vyhledávání osob dle jejich znalostí je třeba tyto znalosti získat. Datový zdroj takových znalostí musí poskytovat minimálně následující atributy:

- Osobu
- Znalost
- Sílu znalosti

Znalosti osob je možné získat například z podpůrných systémů dané organizace (systémy pro organizaci znalostí, osobní profily zaměstnanců a jiné). Další možné zdroje jsou: knihovní systémy, vědecké registry, databáze vědeckých článků a výstupů vědecké činnosti (v organizaci, popřípadě světové).

Co se týká formy dat, tak jsou to například: vědecké publikace, výsledky experimentů, patenty, vyučované předměty, vedené práce, zaměření studia, pracovní zkušenosti, specifikace výzkumné činnosti, osobní stránky (sekce dovedností a znalostí) a životopisy.

Důležitá je vždy informace, o jakou množinu osob se jedná. U akademických organizací (vysoké školy apod.) budou nejspíš lépe dostupná data vědeckého a vzdělávacího charakteru (články, studované předměty). V neakademických organizacích s vyhovující infrastrukturou lze předpokládat existenci systému pro organizaci znalostí.

## 6.5 Znalostní báze

Znalostní bázi myslíme množinu znalostí, kterou bude systém disponovat. V minulých kapitolách jsme shledali jako nejlepší jazyk pro reprezentaci znalostí ontologie. V kontextu ontologií je znalostní báze v podstatě množina konceptů, individualit a vazeb mezi nimi. Jedná se zkrátka o slovník, se kterým při vyhledávání znalostí pracujeme. Datový zdroj znalostní báze musí poskytovat minimálně následující:

- Koncept s jeho atributy (název, id...)
- Vazby na jiné koncepty (potažmo význam těchto vazeb)

Konceptem je v tomto kontextu myšleno ve své podstatě cokoliv, co lze pojmenovat a definovat.

Zdrojem znalostí pro účely robustní znalostní báze by se mohla stát jakákoliv volně přístupná data ve vyhovujícím formátu (nejlépe RDF).

### Příklady takových zdrojů:

- CoceptNet - <http://conceptnet.io/>
- DBpedia - <https://wiki.dbpedia.org/>
- Yago - <https://datahub.io/collections/yago>
- WordNet - <https://wordnet.princeton.edu/>
- Popř. Systémy pro organizaci znalostí přímo v organizaci

Je důležité si uvědomit, že jeden fyzický datový zdroj může sloužit jako zdroj znalostí osob a zároveň znalostní báze.

## 6.6 Shrnutí

V této kapitole jsme rozeptali datové zdroje, které je možné použít při řešení problému vyhledávání osob dle kompetencí. Klíčovými jsou datové zdroje znalostí, které jsme rozdělili do dvou základních kategorií - znalosti osob a znalostní báze.

Četnost datových zdrojů může být v jednotlivých případech různá (různé organizace, různé infrastruktury). Kvalita zdrojů a jejich četnost rozhoduje o tom, v jaké míře bude třeba získaná data dále zpracovat. Problematiku zpracování dat rozebereme v následující kapitole.





# Kapitola 7

## Zpracování dat

Jak jsme již předestřeli v závěru minulé kapitoly, dalším velkým úkolem, který jsme si vytyčili, je rozhodnout o zpracování dat.

V kapitole č. 2 jsme zmínili následující celky:

- Transformace dat do vybrané datové struktury
- Další zpracování získaných dat
- Operace nad daty (CRUD)
- Poskytování dat

Každé z těchto témat má jistě další dílčí části, ty budeme rozebírat v následujících sekcích této kapitoly. Je důležité zmínit, že problematika zpracování dat naší povahy je velmi složitá. Cílem této kapitoly je, více než nalezení odpovědí na všechny otázky, kompletace maximálního množství otevřených bodů, které musí konkrétní řešení brát v úvahu. Ke každé dílčí problematice doplníme příklady technologií, které tuto problematiku řeší, nebo alespoň nastíníme možnosti řešení.

V této části je naše zkoumání dosti technologicky závislé. Jak a proč se rozhodneme konkrétní postup využít, záleží vždy na zvoleném řešení. Neprovedeme tedy volbu jednoho z možných přístupů, ale zhodnotíme spíše kladné a záporné vlastnosti každého z nich.

V další části práce, převážně v kapitole implementace (kapitola 12), na tuto kapitolu navážeme.

### 7.1 Transformace dat

V kapitole č. 5 jsme zvolili jako nejvhodnější reprezentaci znalostí pro naše účely ontologie. Do této reprezentace budeme data transformovat. S tímto problémem souvisí následující:

- Volba ontologie
- Transformace dat do ontologií (resp. do přidružených formátů)

### 7.1.1 Volba ontologie

Existují dva základní přístupy. Prvním přístupem je vybrat existující ontologii a tu použít. Druhý přístup je vytvořit ontologii novou. Oba přístupy lze kombinovat, tedy vybrat existující ontologii a v případě nutnosti jí vhodným způsobem rozšířit.

#### Existující ontologie

Ontologie se mezi sebou liší a každá má případ užití, pro který se hodí. Například ontologie SKOS je určená pro tvorbu slovníků, thesauri a podobně. Důležitým kritériem při výběru je poměr expresivity a formálnosti. SKOS dle specifikace [39] zcela postrádá formálnost, přesto se však hojně používá.

#### Návrh nové ontologie

Druhý možný přístup je navrhnout ontologii vlastní. Čistě pro potřeby databáze kompetencí.

V takovém případě je nutno brát v úvahu, že existují tzv. top-level ontologie (viz kapitola 5.2), dle kterých je možné navrhovat vlastní ontologie nižšího řádu. Takovou ontologii může být např. ontologie UFO [26].

### 7.1.2 Transformace

V kapitole č. 5.2 jsme již konstatovali, že používaným formátem pro ontologická data jsou tzv. RDF triply. Dále jsme v kapitole č. 10 konstatovali, že nejčastějšími formáty získaných dat jsou formáty XML a JSON. Je nutné dodat, že velké množství dat je stále nestrukturovaných (například prostý text).

V rámci transformace je tedy třeba zajistit převod získaných dat do RDF (RDF XML, JSON-LD, turtle).

RDF triply jsou však jen začátek, je důležité zmínit, že další rozšíření (OWL, RDFS) vyžadují ještě vyšší kvalitu dat a jejich dodatečná anotace je často nevyhnutelná.

#### Strukturovaná data

##### Existující řešení:

- Ontorefine - převod strukturovaných dat (.CSV, .JSON, .XML) do formátu RDF, podpora filtrování dat a dalších funkcí (zdroj: <http://graphdb.ontotext.com/documentation/free/loading-data-using-ontorefine.html>)
- URIBurner (zdroj: <http://linkeddata.uriburner.com/>)
- Tarql (zdroj: <http://tarql.github.io/>)
- Implementace RML (zdroj: <http://rml.io>)

#### Nestrukturovaná data

Jsou-li data, která máme k dispozici nestrukturovaná, je potřeba přistoupit ke složitějším technikám - anotace dat, zpracování přirozeného jazyka a podobně.

## 7.2 Operace nad daty

S tímto problémem souvisí následující:

- Databáze
- CRUD operace

### 7.2.1 Databáze

Ukládání dat v podobě RDF je realizováno speciálním druhem grafových databází tzv. *triple-store*. Lze samozřejmě použít libovolnou grafovou databázi, avšak *triple-store* je přímo určen pro práci s RDF triply.

Implementací těchto databází je samozřejmě nespočet, uvedeme nejznámější z nich:

- GraphDB - <http://graphdb.ontotext.com/>
- Allegrograph - <https://allegrograph.com/>
- JENA APACHE - <https://jena.apache.org/index.html/>
- Virtuoso - <https://virtuoso.openlinksw.com/>

## 7.3 CRUD operace

Každá databáze má k dispozici data a zároveň nástroj, který uživateli umožňuje k datům přistupovat a pracovat s nimi. Typicky jsou tímto nástrojem dotazovací jazyky. U relačních databází je to jazyk SQL, u grafových jazyk CYPHER a u *triple-store* databází je to obvykle jazyk SPARQL [45].

Dotazovací jazyk SPARQL má šest základních dotazů - SELECT, ASK, CONSTRUCT, DESCRIBE, INSERT a DELETE. Pomocí těchto šesti dotazů jsme schopni v *triple-storu* provádět základní CRUD operace.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
    card:i foaf:knows ?known .
    ?known foaf:homepage ?homepage .
}
```

**Listing 7.1:** Příklad SELECT dotazu v jazyce SPARQL (zdroj: <https://www.w3.org/2009/Talks/0615-qbe/>)

Vyhledávání, které je pro nás důležité, je zajištěno převážně dotazem SELECT.

Při vyhledávání pracovišť a lidí dle kompetencí je jednou z variant použití klíčových slov, případně i fulltextové vyhledávání. Toto ponecháváme otevřeným bodem, který závisí na konkrétní realizaci.

## 7.4 Zpracování dat

V případě, že se nám podaří data transformovat a uložit, máme za sebou nejnáročnější část tvorby databáze znalostí. V tu chvíli budeme mít, ať už jakýmkoliv způsobem, k dispozici databázi osob, pracovišť a jejich znalostí.

Dle konceptuálního modelu (příloha A.1) se na vazbu mezi osobou a znalostí váže atribut *síla znalosti*.

Existence této vazby již sama reprezentuje, že daná osoba znalost vlastní. Sama tato existence nám již dává přidanou hodnotu, kterou lze využít. Není tedy nutné dále vazbu rozvíjet.

Pokud však půjdeme více do hloubky, zanedbali jsme několik informací, které pro nás mohou být užitečné. Zanedbali jsme například četnost výskytů této vazby v externích zdrojích (je-li zdrojů více), nebo důvěryhodnost těchto zdrojů - jinými slovy všechny vazby mezi osobou a znalostí jsou rovnocenné. To ale ve skutečném světě tak není, osoby mají různé úrovně znalostí. V modelu, kde jsou všechny vazby rovnocenné, nelze porovnávat znalosti více osob, v případě, že se váží ke stejnému konceptu.

Zaměříme se tedy ještě na tuto problematiku. Uvedeme jeden z možných způsobů, jak přistoupit k výpočtu síly znalosti.

### 7.4.1 Síla znalosti

Síla znalosti  $I_K$  je metrika vyjádřená pomocí desetinného čísla, která se váže ke každé vazbě mezi osobou a znalostí. Předpokládáme, že máme  $k$  datových zdrojů, které nám poskytují parametry: síla znalosti v rámci datového zdroje  $w$  a identifikátor vlastníka pro danou znalost  $K$ . Tutéž vazbu nám datový zdroj může vrátit několikrát, počet těchto výskytů jedné vazby v konkrétním zdroji značíme  $q$ . Výpočet  $I_K$  je poté realizován pomocí následujícího vzorce:

$$I_K = \sum_{i=1}^k \sum_{j=1}^q r_i w$$

Konstanta  $r_i$  určuje spolehlivost popř. důvěryhodnost datového zdroje. Pokud jsou zdroje kategorizovány, lze jej vypočítat jako:  $r_i = R_{kat} R_{zdroj}$ , kde  $R_{kat}$  je důvěryhodnost kategorie zdrojů a  $R_{zdroj}$  důvěryhodnost zdroje, obě jsou to konstanty.

Síla znalosti v rámci datového zdroje  $w$  se pro každý datový zdroj může určovat jiným způsobem. Například v případě vědeckých článků se může odvíjet od některých uznávaných vědeckých indexů a podobně.

V oblasti zpracování dat mohou jistě vznikat další úskalí, která bude nutné během konkrétní realizace řešit. V této sekci jsme se pouze pokusili obecně nastínit postup při odvozování síly znalosti.

## 7.5 Poskytování dat

Data z *triple-store* databází jsou obvykle k dispozici jako tzv. SPARQL endpoint. Jak vyplývá z názvu *SPARQL Protocol and RDF Query Language*, SPARQL je i komunikační protokol. Pomocí tohoto protokolu je možné provádět dotazy v jazyce

SPARQL, typicky na vzdálené endpointy. Data je možné pomocí tohoto rozhraní poskytovat dalším stranám ve zvoleném formátu pro RDF triply. Protokol SPARQL staví na HTTP protokolu, endpoint je tedy identifikován pomocí URL (Uniform Resource Locator).

## ■ 7.6 Shrnutí

V této kapitole jsme odpověděli na otázky ohledně zpracování získaných dat. Provedli jsme rešerši existujících postupů a technologií. Dále jsme v některých případech upozornili na otevřené body, které je nutno vyřešit při návrhu konkrétního řešení. Na toto téma navážeme v druhé části práce a k některým otevřeným bodům se vrátíme.





## Kapitola 8

### Shrnutí první části

V rámci první části této práce jsme naplnili první z cílů, který jsme stanovili - *Zanalyzovat problematiku vyhledávání osob a pracovišť dle kompetencí.*

V úvodu této části jsme celý problém uchopili. V kapitole 4 jsme potom popsali jednotlivé zkoumané entity pomocí konceptuálního modelu (A.1). Stěžejním bodem této části byla kapitola 5, ve které jsme rozhodli o reprezentaci kompetencí v našem systému. Rozhodli jsme se kompetence (resp. znalosti) reprezentovat pomocí ontologií, vzhledem k jejich vyhovujícím vlastnostem. V druhé polovině této části jsme provedli rozbor dalších dílčích úkolů a řešili existující technologie.

Každá kapitola měla svůj cíl a má daný závěr. V druhé části práce na tyto závěry navážeme a rozvedeme problematiku jednotlivých dílčích úkolů do konkrétního případu užití.







**Část II**

**FEL ČVUT**



# Kapitola 9

## Úvod

V následující části práce navážeme na předchozí obecnou část a zaměříme se na dva zbylé cíle, které jsme definovali v úvodu.

**Jsou jimi:**

- Navrhnout systém pro vyhledávání osob a pracovišť dle kompetencí a provést rešerši dostupných dat
- Implementovat prototyp systému pro vyhledávání osob a pracovišť dle kompetencí pro FEL ČVUT

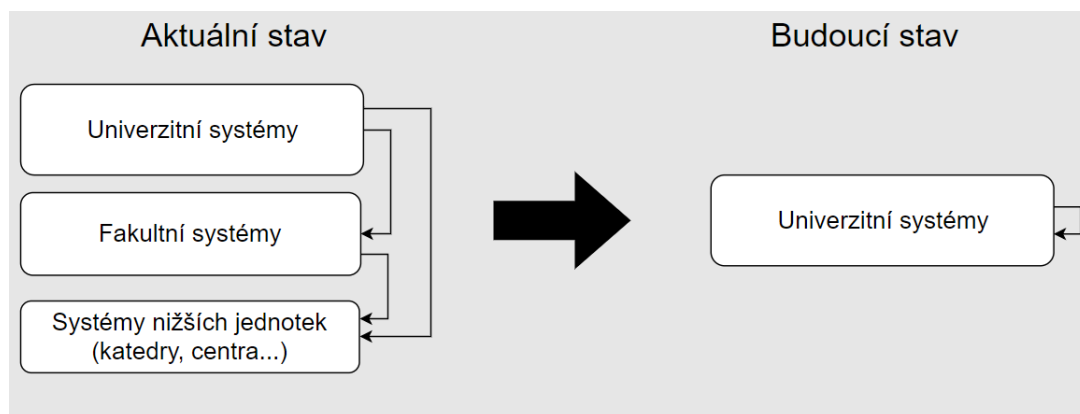
Nejdříve shrneme aktuální situaci týkající se strukturalizace a kategorizace znalostí a informací na FEL ČVUT. Na to navážeme analýzou datových zdrojů. Dále rozebereme návrh systému a v poslední fázi rozebereme implementaci prototypu za použití zvolené technologie, testování tohoto prototypu a další implementační detaily.

### 9.1 Informační infrastruktura Fakulty elektrotechnické ČVUT

České vysoké učení technické v Praze se skládá z osmi fakult a dalších přidružených pracovišť. Jednotlivé fakulty se skládají z dalších podřazených jednotek - kateder, center a podobně.

Informační infrastruktura ČVUT je tvořena od katederních systémů přes fakultní až k těm univerzitním. Komunikace mezi systémy probíhá typicky jen směrem dolů (tedy od systémů výše putují data k systémům níže). Takový způsob integrace je pro toto uspořádání poměrně vyhovující.

Z dlouhodobého hlediska je tendencí strukturu zjednodušovat, tedy přesouvat kompetenci správy systémů na vyšší úroveň v hierarchii. Například tedy zamezit vzniku katederním systémům a místo toho zakládat centrální fakultní systémy. Toto řešení má výhody při změnách (ať už metodických, tak implementačních). Na druhou stranu může být toto řešení dosti personálně náročné, jelikož by se univerzita musela v budoucnu starat o všechny systémy. Zároveň by se tímto způsobem však měla spousta systémů eliminovat. Přejít na takové vnitřní uspořádání nelze provést skokově. Vhodnější jsou pozvolné změny, které však trvají velmi dlouho.



**Obrázek 9.1:** Schematické zobrazení informační infrastruktury ČVUT a komunikace systémů (zdroj autor)

O jednotlivých systémech, které budou relevantní v kontextu této práce, se zmíníme v následujících kapitolách.

# Kapitola 10

## Analýza datových zdrojů

V následující kapitole shrneme výstupy analýzy datových zdrojů pro FEL ČVUT, která je jedním z cílů této části práce. Obecně jsme datové zdroje řešili v kapitole č. 6, nyní na tuto kapitolu navážeme a budeme řešit konkrétní příklady. Zdroje budeme dále kategorizovat do různých skupin. Důležité je zmínit, že v této kapitole nepopíšeme přesný způsob využití jednotlivých zdrojů. Spíše popíšeme, v jaké formě jsou data dostupná a jak by mohla být užitečná.

### 10.1 Kategorizace datových zdrojů

Datové zdroje můžeme rozdělit dle různých kritérií. V této sekci shrneme kategorie, které budeme používat v naší práci.

#### Dle typu obsahu na:

- Osoby a pracoviště
- Znalosti

#### Dle typu znalosti (viz kapitola č. 6) na:

- Znalostní báze
- Osobní znalosti
- Obojí

#### Dle příslušnosti datového zdroje na:

- Externí zdroj (případně globální)
- Interní zdroj pro danou organizaci - v našem případě pro ČVUT FEL

## 10.2 Datové zdroje ČVUT

### 10.2.1 Osoby a pracoviště

	Forma dat	Dostupnost dat	Správce
Usermap	REST API (JSON)	veřejná data o zaměstnancích, po domluvě dostupná	ČVUT spravuje systém, FIT spravuje API
UDB	LDAP (LDIF)	veřejná data o zaměstnancích, po domluvě dostupná	FEL

**Tabulka 10.1:** Tabulka dostupných zdrojů osob a pracovišť FEL (zdroj autor)

#### Usermap

Usermap je centrální systém ČVUT pro správu identit. Jsou zde uloženy informace o osobách, organizační struktura, pracovištích a jiné. Databáze Usermap má dvě části - LDAP databáze a přidružená relační databáze [31]. Databáze Usermap je spravována jednotně na úrovni celé univerzity.

The screenshot shows the Usermap web interface for a user profile. The header includes the Usermap logo and navigation links for 'Lidé na ČVUT' and 'Pracoviště'. The profile is for 'Ing. Lukáš Zoubek'. It features a profile picture placeholder, a QR code, and several sections of information:

- Profil uživatele:** Role na ČVUT: zaměstnanec / centrum znalostního managementu / Fakulta elektrotechnická; zaměstnanec / sekretariát děkana a tajemníka / Fakulta elektrotechnická; student / Fakulta elektrotechnická.
- Kontaktní údaje:** Telefon: +420-22435-3986; E-mail: zoubeluk@fel.cvut.cz, zoubeluk@fit.cvut.cz, lukas.zoubek@fel.cvut.cz, zoubek.lukas@fel.cvut.cz; Domovská stránka: http://czm.fel.cvut.cz/cs/o-nas/lide/zoubek.
- Údaje o identitě:** Identifikátor ORCID: https://orcid.org/0000-0002-8964-8529.

**Obrázek 10.1:** Webové rozhraní systému usermap (Dostupné z: <https://usermap.cvut.cz/search>)

REST rozhraní (*Usermap API*) k této databázi je oproti tomu spravováno FIT ČVUT, konkrétně Ing. Jakubem Jirůtkou.

*Usermap API (pracovní název) je agregátor údajů o identitách lidí na ČVUT, který vyvíjíme na FIT. Poskytuje základní informace o osobě jako je jméno, uživatelské jméno, emailové adresy, telefony, ... a tzv. byznys role (z IDM).*

Dále obsahuje informace o organizačních jednotkách ČVUT a místnostech (vč. adresy). [30]

Identifikace v rámci tohoto zdroje probíhá pomocí uid, uživatelského jména v rámci ČVUT. Ve webovém rozhraní aplikace usermap lze též získat ORCID dané osoby - jednoznačný identifikátor vědeckých a dalších akademických autorů[3].

## ■ UDB

Narozdíl od systému usermap systém UDB je spravován fakultou FEL. Typy obsahu jsou ve své podstatě totožné, nicméně v UDB se nacházejí některá data navíc (fakultního charakteru). K systému neexistuje žádná veřejně dostupná dokumentace.

## ■ 10.2.2 Znalosti

Již v kapitole č. 6 jsme zmínili, že jeden fyzický zdroj může být jak zdrojem osobních znalostí, tak zdrojem znalostní báze. V této kapitole jsme se pokusili tyto dvě kategorie od sebe oddělit, nicméně se dá předpokládat, že zdroj nemusí mít vždy jedno diskrétní využití.

### ■ Osobní znalosti

	Forma dat	Dostupnost dat	Správce
<b>V3S</b>	REST API (XML)	veřejná část	ČVUT spravuje systém, FIT spravuje API
<b>KOS</b>	REST API (XML)	veřejná část	ČVUT spravuje systém, FIT spravuje API

**Tabulka 10.2:** Tabulka dostupných zdrojů osobních znalostí na ČVUT (zdroj autor)

**Systém V3S:.** V naší práci, jak jsme zmínili již v úvodu, se zabýváme více formálními zdroji znalostí (např. vědecké články, patenty). Předpokládáme, že systém V3S je pro data tohoto typu klíčový. Pokud bychom řešení opřeli o tento systém, množinu vyhledávaných bychom omezili na ty, kteří jsou vědecky aktivní. Má tedy smysl se zabývat i ostatními zdroji.

Aplikace V3S obsahuje záznamy o veškeré vědecké činnosti na univerzitě. Záznamy obsahují metadata o publikovaných člancích, autorech a dalších souvisejících entitách. Plné texty některých článků jsou k dispozici v digitální knihovně ČVUT (Dspace - <https://dspace.cvut.cz/https://dspace.cvut.cz/>). Nejedná se však o všechny články, ne všechny může ČVUT takto ukládat a poskytovat.

REST rozhraní *V3S API* k této databázi je spravováno FIT ČVUT, konkrétně Ing. Jakubem Jirůtkou.

**KOS (Komponenta studia):** KOS je centrálním informačním systémem pro podporu výuky na ČVUT. Studenti ČVUT přistupují do tohoto systému pomocí webového rozhraní, které obsahuje pouze zlomek dat, které jsou ve skutečnosti v systému k dispozici (v jeho databázi). REST rozhraní *KOS API* k této databázi je spravováno FIT ČVUT, konkrétně Ing. Jakubem Jirůtkou.

KOS by mohl být potenciálně zdrojem obsahu jako: studované a vyučované předměty, vedené a nabízené závěrečné práce.

**Ostatní:** Pracovníci FEL ČVUT mohou sami sloužit jako zdroj jejich znalostí - jejich životopisy, osobní stránky a jiné. Tyto zdroje typicky postrádají jakoukoliv jednotnou strukturu či rozhraní, jsou tedy špatně strojově zpracovatelné. Při rešerši možných

## Petr HABALA



**Pozice:** Docent

**Telefon:** katedra: 420+22435 1111  
kanclík: 420+22435 3365 (skoro tam nejsem, raději zkuste e-mail)

**E-mail:** [habala@fel.cvut.cz](mailto:habala@fel.cvut.cz)

**Adresa:** [Katedra matematiky \(K301\)](#)  
Fakulta elektrotechnická  
ČVUT  
Technická 2  
Praha, 166 27

I may be aging but I refuse to grow up. (anonymous)  
(Možná stárnu, ale odmítám dospět.)

Moje [CV](#).

---

**Vyučování:** Konzultační hodiny pro jaro 2019: úterky 13:30—14:20 nebo dle dohody.  
Konzultace během zkouškového dle dohody.

Informace o kursech, které učím:

- [Diskrétní matematika pro OI](#)
- [DRN \(Diferenciální rovnice a numerická matematika\)](#).
- [DEN \(Differential equations and numerical mathematics\)](#).
- [DRN \(Diferenciální rovnice a numerická matematika\) \(verze pro dálkaře\)](#).
- [DEN \(Diferenciální rovnice a numerická matematika\) \(verze 4+2 pro OES\)](#).

**Obrázek 10.2:** Webové stránky pracovníka katedry matematiky, docenta Habaly (Dostupné z: <https://math.feld.cvut.cz/habala/indexc.htm>)

zdrojů jsme narazili ještě na systém DNEP (databáze nabídek, expertů a přístrojů), který nejspíše spravuje ČVUT. Systém obsahuje expertů pouze 93 (zjištěno z aplikace), na základě tohoto faktu jsme konstatovali, že není dále aktualizován a používán.

### ■ Znalostní báze

Potenciálně využitelným zdrojem znalostní báze přímo na ČVUT by mohl být systém V3S potažmo digitální knihovna DSpace. Data v těchto zdrojích (hlavně texty) by se



po anotaci [32] mohla použít při plnění ontologií, slovníků a jiných.

Sémantická anotace textu je jednou z možných technik pro obohacování textu o význam, což je při tvorbě znalostní báze dosti klíčové. Sémantická anotace textu není triviální činnost, ruční provedení může trvat velmi dlouho a semi-automatické či automatické necháme na budoucí práci (zabývají se jí například nástroje jako je Tagtog - dostupné z: <https://www.tagtog.net/>, nebo LightTag dostupné z: <https://www.lighttag.io/>).

## 10.3 Externí (globální) datové zdroje

Při získávání informací o osobách a pracovištích jsme odkázáni na interní zdroje, externí zdroje můžeme použít pro znalosti.

### 10.3.1 Znalosti

#### Osobní znalosti

System V3S považujeme za nejdůležitější a nejobektivnější zdroj znalostí vědeckých pracovníků FEL. Další podobné zdroje jsou zmíněny v této sekci, jedná se však již o systémy, které nejsou ve správě fakulty/univerzity.

	Forma dat	Správce
Scholar, google patenty	oficiální API chybí	Google
Scopus	REST API (různé formáty)	Elsevier B.V.
Science Direct	REST API (různé formáty, též fulltext search)	Elsevier B.V.
Springer	REST API (různé formáty)	Springer Nature
IEEE explore	REST API (různé formáty)	IEEE
Web of Science	REST API (různé formáty)	Clarivate Analytics
IS výzkumu, experimentálního vývoje a inovací	API [46]	Úřad vlády České republiky

**Tabulka 10.3:** Tabulka některých veřejně dostupných zdrojů osobních znalostní (zdroj autor)

V rámci veřejných rozhraní je třeba vyřešit identifikaci osob z FEL ČVUT. Vzhledem ke zdrojům které máme k dispozici, se zaměřujeme nejvíce na vědecky aktivní část FEL ČVUT. Existují různé způsoby, jak identifikovat vědce či vědkyně.

#### Příklady identifikátorů:

- ORCID - Open Research and Contributor ID (neproprietární)
- RID - ResearcherID (proprietary Web of Science)
- Scopus Author ID (proprietary Scopus)

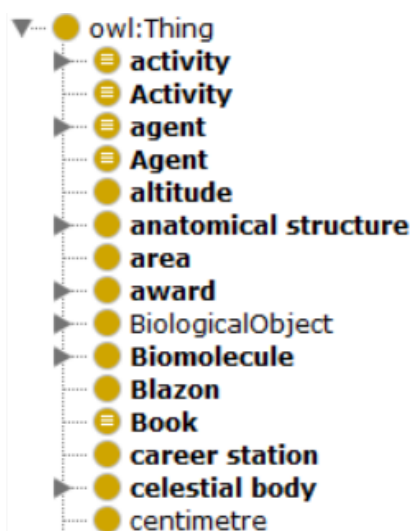
## Znalostní báze

Je důležité zmínit, že množství dostupných strukturovaných dat, které lze považovat za znalosti, je omezené. Existující datové množiny jsou mezi sebou provázány. Nejčastějšími primárními zdroji jsou Wordnet a Wikipedia (případně její strojově lépe čitelná verze DBpedia). Navazující projekty s těmito zdroji pracují a přidávají další, často užitečné, informace (aktivní přispívání znalostí však často neprobíhá).

Na základě tohoto kritéria datové zdroje rozdělíme do dvou skupin:

- Primární zdroje - DBpedia, Wordnet
- Agregátory a procesory znalostí - ConceptNet, Yago, Datamuse, Twinword API, WordsAPI

**DBpedia:** DBpedia extrahuje informace z Wikipedie a následně data propojuje s ostatními získanými daty. Tímto způsobem vzniká jednotná znalostní báze [41]. Tato znalostní báze je dostupná buď ve formě RDF s definovaným OWL schématem ke stažení, přes SPARQL rozhraní nebo též přes webové REST rozhraní [16]. V roce 2014 měla DBpedia 3 miliardy RDF triplů. [16]



**Obrázek 10.3:** Část ze schématu DBpedia v programu Protégé <https://protege.stanford.edu/> (zdroj autor)

**WordNet:** Dalším významným datovým setem je Wordnet. Jedná se o rozsáhlou lexikální databázi anglického jazyka. Slova jsou seskupována do množin synonym. Tyto množiny jsou později propojeny do sítě na základě významových a lexikálních vztahů. Wordnet neřeší slova pouze jako množinu znaků, orientuje se na jejich význam a pracuje například s podobností významu různých slov. [2]

Data jsou dostupná volně ke stažení, případně přes REST API [2].

**Agregátory a procesory:** ConceptNet je volně dostupná sémantická síť čerpající mimo jiné z projektů DBpedia a Wordnet. Jedním ze zdrojů dat byla též komunita

lidí, projekt Open Mind Common Sense. ConceptNet má API s výstupním formátem JSON-LD. [50]

Yago je rozsáhlá sémantická znalostní báze, která čerpá mimo jiné z Wikipedie a Wordnetu. Yago má více než 10 milionů entit. [48] Yago ontologie je k dispozici ke stažení, například ve formátu TURTLE. Software Yago je k dispozici ve formě zdrojového kódu též volně ke stažení.[11]

Dalšími datovými zdroji spadajícími do této kategorie mohou být například Data-muse [14], Twinword API [15] nebo Words API [5]. Funkce, kterými disponují jsou typicky hledání slov s podobným významem, hledání slov souvisejících, hledání definic, lematizace (postup převádějící slova na základní gramatický tvar).

## 10.4 Shrnutí

V této kapitole jsme se zabývali rešerší datových zdrojů na FEL ČVUT, tím jsme navázali na kapitolu 6, kde jsme tuto problematiku řešili obecně.

Datové zdroje jsme rozdělili do kategorií a u každého jsme zmínili důležité parametry. Tím jsme naplnili jeden z cílů této práce.



# Kapitola 11

## Návrh aplikace

V následující kapitole rozebereme zevrubně návrh systému pro vyhledávání osob a pracovišť dle kompetencí. Budeme postupovat od nejobecnějšího (cíle a požadavky) ke konkrétnímu (případy užití, volba architektury). Stále se budeme pohybovat po technologicky nezávislé rovině, tu rozebereme až v následující kapitole při popisu prototypu. Stejně tak otázky reprezentace jsou na této rovině irelevantní, přestože ty jsme již rozhodli.

### 11.1 Podnět vzniku aplikace

Idea této aplikace pro vyhledávání osob dle kompetencí vznikla na FEL ČVUT, konkrétně u pana Ing. Martina Klímy, Ph.D.

Pan Klíma měl představu, že by měla na webových stránkách FEL vzniknout sekce, kde by třetí strany, konkrétně průmysloví partneři, mohli vyhledávat osoby dle jejich kompetencí. Partneři by později mohli kontaktovat daného člověka případně jeho vedoucího pracovníka, pokud by měli zájem s ním více spolupracovat. Tímto způsobem by tedy fakulta nabízela pracovníky a jejich kompetence širší veřejnosti, samozřejmě s jejich souhlasem.

První práce, která vznikla na toto téma je bakalářská práce Romana Kuchára z roku 2016. Pan Kuchár problematiku řešil poměrně přímočaře, způsobem, který pan Klíma popsal. [34] Po analýze kódu lze postup shrnout do následujících kroků:

1. **Uživatel** zadá klíčové slovo tzv. expertise
2. **System** nalezne výskyt tohoto slova v článcích v systému V3S (v titulu, klíčových slovech či abstraktu)
3. **System** na základě hodnocení nalezených publikací (tzv. rivs) ohodnotí nalezené autory, přitom vezme v úvahu i míru jejich participace
4. **System** vrátí seznam nalezených autorů seřazený dle jejich vypočteného hodnocení

Řešení pana Kuchára v kontextu naší práce spadá pod varianty bez znalostní báze a bez práce se znalostmi vůbec (což nemusí být nutně špatně). V kapitole č. 14 se zmíníme o řešeních tohoto typu.

Cílem této práce je celou problematiku prověřit zevrubněji (jak je vidět z první části této práce). Při návrhu aplikace budeme vycházet z požadavků/cílů, které definoval v úvodu pan Klíma a z požadavků, které jsme při této práci dále identifikovali.

Existuje mnoho dalších možností, jak vzniklou aplikaci obohatit o další užitečné funkce. V této práci však budeme pracovat se základními požadavky, pomocí kterých budeme tvořit jádro aplikace.

## ■ 11.2 Vymezení této práce

### ■ 11.2.1 Rozsah řešené problematiky

Na úvod upřesníme, čím se budeme v následujících sekcích zabývat. Celou problematiku vyhledávání osob a pracovišť dle jejich kompetencí, která vyplývá z pokynů k vypracování této práce, jsme v předchozí části práce zúžili na vyhledávání osob a pracovišť dle znalostí (viz kapitola 5). Pro účely následujících kapitol zúžíme celý problém pouze na vyhledávání osob dle znalostí.

Primárním důvodem je, že z definice kompetence pracoviště (kapitola 3) vyplývá, že je složena z kompetencí osob. Vyřešíme-li tedy problém vyhledávání osob dle kompetencí, stěžejní část problému vyhledávání pracovišť dle kompetencí vyřešíme též. Se znalostmi je to v tomto kontextu stejné jako s kompetencemi.

### ■ 11.2.2 Rozsah navrženého systému

Jak jsme zmínili výše, v následující kapitole rozebereme návrh aplikace pro vyhledávání dle kompetencí. Důležité je zmínit, že navrhne celý systém až na část, která se zabývá transformací dat. Transformací dat, kterou jsme popsali v předchozí kapitole (č. 10), se vzhledem k rozsahu problematiky zabývat nebudeme. Zmíníme jí však v kapitole budoucí práce (č. 14), jelikož je to problematika, která musí být před nasazením navrženého systému vyřešena - bez dat jej nelze využívat.

### ■ 11.2.3 Rozsah implementovaného prototypu

V následující kapitole (č. 12) se budeme zabývat prototypem, který jsme v rámci naší práce implementovali. Účelem implementační části této práce je dokázat, že řešení navržené v této kapitole je možné použít a že skutečně může po dalších rozšířeních fungovat v praxi. Nejedná se tedy o kompletní implementovaný systém. V úvodu příští kapitoly zmíníme konkrétně rozsah obou částí prototypu.

## ■ 11.3 Požadavky

### ■ 11.3.1 Základní business požadavek

Základní business požadavek, který naše aplikace musí splnit je: *Umožnit třetím stranám fakulty (firmám, výzkumným centrům a jiným) vyhledávat pracovníky FEL ČVUT dle jejich kompetencí, s jejich svolením.*

## ■ 11.3.2 FURPS+ analýza

### ■ Funkce

- Vyhledat seznam pracovníků FEL seřazených dle jejich vypočtené síly znalosti na základě zadaného klíčového slova (libovolného nebo zvoleného z definované množiny slov)

### ■ Použití

- Část aplikace, do které bude přistupovat koncový uživatel, musí být dohledatelná pomocí běžných vyhledávačů dle klíčových slov (minimálně google a bing)
- Část aplikace, do které bude přistupovat koncový uživatel, musí být jednoduchá k použití - očekávají se uživatelé, kteří jsou gramotní při běžné práci s počítačem (textové procesory, práce s prohlížečem a webovými formuláři)
- Koncový uživatel aplikace musí mít k dispozici příručku (nejlépe interaktivní formou přímo v rozhraní)
- Aplikace musí být dostupná bez přihlášení, login se toleruje jen pro případná speciální administrátorská práva
- API musí být kompletně zdokumentováno pro budoucí použití jinými programátory

### ■ Spolehlivost a výkon

Aplikace nemá žádná zvláštní omezení na spolehlivost a výkon. Předpokládá se spolehlivost na úrovni ostatních nekritických aplikací univerzity - 98%. Aplikace nemá speciální požadavky na výkon.

### ■ Údržba

- Použitá technologie musí být kompatibilní s budoucím správcem aplikace (FEL)
- Aplikace bude koncipována jako webová, ne nativní

### ■ Bezpečnost

Data, která budou dostupná z aplikace, mohou být pouze data veřejně dostupná.

### ■ Implementace

Architektura aplikace musí být modulární. Celý problém vyhledávání osob dle kompetencí, reprezentace znalostí a vše okolo má velmi mnoho možných přístupů k řešení. Architektura aplikace musí umožňovat jednoduchou výměnu jednotlivých modulů za jiné implementace.

### 11.3.3 Případy užití

#### Aktéři

Aplikace bude ze své podstaty otevřena široké veřejnosti. Hlavním uživatelem tedy bude *nepřihlášený uživatel*.

#### UC1 - Vyhledat osoby na FEL dle kompetencí

##### Scénář:

1. **Uživatel** zadá klíčové slovo (libovolné nebo vybrané z definované množiny slov)
2. **System** nalezne na základě zadaného klíčového slova kompetenci (reprezentovanou libovolným způsobem) a osoby, které se k ní vztahují. Následně **system** vypočítá/vyhledá sílu kompetence pro každou nalezenou osobu. Nakonec tento seřazený seznam osob vrátí uživateli. Neexistuje-li žádná osoba s touto kompetencí, system vrátí uživateli prázdný seznam.

Seznam vrácený uživateli může být parametrizován pomocí offset a limit. **Limit** určuje počet výsledků, který má být vrácen. **Offset** potom určuje pořadové číslo prvního řádku z výsledku, který má být uživateli vrácen.

### 11.3.4 Wireframy

#### W-UC1 - Vyhledat osoby na FEL dle kompetencí

**Vyhledávací pole** - zajišťuje uživatelský vstup, může být implementováno jako prosté textové pole s omezením na délku, či jako výběr s možností vyhledání

##### Tabulka s výsledkem obsahuje sloupce:

- Jméno - celé jméno osoby včetně titulů
- Username - username v rámci FEL ČVUT (slouží též jako email, po přidání domény)
- Pracoviště - název a kód pracoviště
- Info - libovolné informace, relevantní pro podložení vyhledaných informací a podobně

## 11.4 Architektura aplikace

Zkoumaný problém není složitý počtem funkcí, ale náročností jednotlivých celků. Tato náročnost se dosti odráží na architektuře celé aplikace. Aplikace může být integrována s různými typy úložišť a modularita musí zajistit jednoduché nahrazování jednotlivých částí (vyplývá z FURPS+ analýzy (11.3.2)).



Vyhledávání osob na FEL dle kompetenci

Umělá |

Umělá inteligence  
Umělá hmota  
Umělá léčba

Výsledek hledání pojmu "Sensory"

#	jméno	username	pracoviště	info
1	Jan Matyáš	matyasja	katedra počítačů	Metrika: 0,5, zdroj: vědecké články
2	Marek Tulin	tulinma	katedra kybernetiky	Metrika: 0,2, zdroj: životopis
3	Karel Musil	musinka	katedry měření	Metrika: 0,7, zdroj: patenty, vědecké články

Obrázek 11.1: Obrázek k W-UC1 (zdroj autor)

### 11.4.1 Klient, Server

Infrastruktura FEL ČVUT sestává primárně z webových aplikací. Tato platforma bude dodržena i v našem případě - viz FURPS+ (11.3.2).

Hlavní část aplikace je koncipována jako server s definovaným rozhraním, skrz které budou dostupné výše zmíněné případy užití. S rozhraním bude komunikováno pomocí HTTP protokolu. V této době jsou nejpoužívanějšími komunikačními přístupy: REST [1] a GraphQL [4], volba záleží na konkrétní realizaci.

Součástí této práce bude i klientská aplikace, na které demonstrováme průběh komunikace se serverovou částí aplikace.

### 11.4.2 Architektura

Slovo architektura se v dnešní době používá v nesčetně významech. V praxi to často jde tak daleko, že již není ani zjevné, co má řečník opravdu na mysli.

Jak píše Martin Fowler, i když cynicky, ve své knize *Patterns of Enterprise Application Architecture*, slovo architektura se používá, když lidé chtějí upozornit na něco, co je opravdu důležité. [23]

Fowler poukazuje na to, že architektura je ve své podstatě rozpad systému na části a následné nastavení pravidel komunikace mezi těmito částmi. Možných architektonických vzorů je nespočet a klíčovým parametrem je nejčastěji dispozice ke změnám. Jinými slovy je klíčové, jestli se daná část aplikace bude v průběhu času měnit a jestli je tomu vybraná architektura dobře přizpůsobená. [23]

V jedné aplikaci může být použito více architektonických vzorů [23]. Dá se říct, že v tomto případě funguje přísloví *účel svěťí prostředky*, pokud tedy daný vzor v dané situaci funguje správně, byl použit správně. Neexistuje samozřejmě jen jedno možné řešení, obzvláště vezmeme-li v úvahu, jak rychle se technologie vyvíjí.

Architektura je subjektivní a je vždy závislá na systému i vývojovém týmu, který daný systém vyvíjí. [23] Motivací, proč řešit architekturu, je nejčastěji potřeba vyhnout se pozdějším změnám, které jsou často časově náročné až likvidační.

### 11.4.3 Výběr architektury

Než přistoupíme k architektuře, která se nejvíce hodí pro náš systém, musíme shrnout úkoly, které aplikace bude plnit a data, která bude zpracovávat.

#### Data:

- Aplikační data - uživatelé aplikace, jejich práva, další data týkající se přidavných funkcí vyhledávače (historie hledání a podobně)
- Zpracovávaná data - znalosti, osoby, pracoviště

#### Úkoly:

- Autorizace, autentizace, účty, cachování, další možnosti pro uživatele případně správce vyhledávače
- Poskytování všech druhů dat (CRUD operace)
- Uchovávání dat
- Zpracování znalostí, osob, pracovišť - agregace, reprezentace, operace při vyhledávání a jiné
- Získávání znalostí, osob, pracovišť

Ne všechny výše zmíněné úkony musí plnit první verze aplikace, dokonce nemusí existovat jen jediná aplikace, která celý problém bude řešit. Nicméně, jak jsme výše zmínili, je třeba počítat s budoucím nahrazováním jednotlivých částí a hledáním nových způsobů. Při volbě architektury tedy musíme počítat se všemi funkcemi, které nyní jsme schopni odhadnout.

Z FURPS+ analýzy (11.3.2) plynou zároveň požadavky na architekturu týkající se hlavně technologie a modularity vybraného řešení.

Systém není příliš složitý, co se týká počtu funkcí. Naproti tomu se však staví jeho komplexita vzhledem k integracím s jinými systémy (různé druhy databází, získávání dat z jiných API), která bude při volbě architektury klíčová. Z hlediska komplexity systému bychom tedy naši aplikaci nepovažovali za enterprise aplikaci v plném rozsahu,

jak jí definuje Fowler ve své knize. Fowler píše, že enterprise aplikace obvykle zahrnují větší množství dat a paralelní přístup mnoha uživatelů [23], což u naší aplikace nepředpokládáme. Naopak v ohledu integrace s jinými systémy se jedná o komplexní (enterprise) aplikaci a je třeba jí tímto způsobem navrhnout a spravovat.

Při tvorbě komplexnějších aplikací je jedním ze základních přístupů tzv. vrstvení (angl. layering) [23], rozhodli jsme se jej použít i v našem případě. Nejedná se o architektonický vzor, spíše o styl nebo přístup k tvorbě aplikací. Hlavním úkolem při tvoření vrstev je samozřejmě definice toho, za co každá vrstva zodpovídá [23] a jak jsou nastavené závislosti mezi jednotlivými vrstvami.

Z počátku jsme zamýšleli použít vrstevnatou architekturu dle Browna (tabulka č. 11.1). Jak píše Fowler, v případě *Brownovy architektury* můžeme vzít v úvahu

Prezentace
Kontroler, mediator
Doména
Mapování
Zdroj

**Tabulka 11.1:** Tabulka vrstev architektury dle Browna (zdroj [23], přeložil autor)

další varianty. Můžeme vzít hlavní tři vrstvy a mezivrstvy dodávat v případě přílišné komplexity systému. [23]

V takto vrstevnaté architektuře je komunikace mezi vrstvami typicky nastavena jedním směrem, tzn. že vrstvy mohou volat buď všechny vrstvy níže, nebo jen jednu o patro níž.

*Brownovu architekturu* můžeme klasifikovat jako klasickou vrstevnatou. Architektury tohoto typu mají jednu značnou nevýhodu, kterou je přílišné provázání celého systému (angl. coupling) s datovými zdroji. Datové zdroje (databáze, soubory, webové služby) jsou totiž typicky svázány přímo s aplikační logikou, přes definované mapování (viz předposlední vrstva v tabulce č. 11.1).

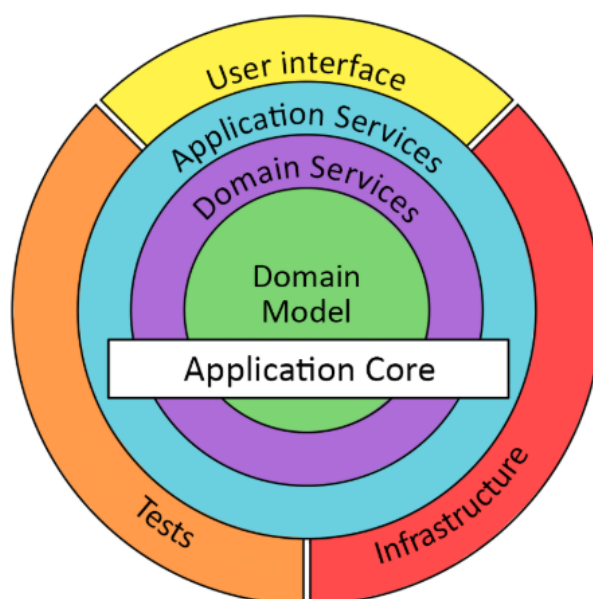
Na tento nedostatek poukázal ve své práci Jeffrey Palermo, který konkrétně zmiňuje:

Kámen úrazu je v tomto případě fakt, že aplikace je postavená okolo dat a ostatní infrastruktury. Jelikož je aplikace takto provázána, tak pokud se změni přístup k datům, externí služba a podobně, musí se změnit i business logika aplikace. [43] (přeložil autor)

Odstínění částí jako je databáze, které Palermo nazval infrastrukturou, je i pro nás velmi klíčové.

Palermo staví na místo toho jiný typ architektury. Architekturu, kterou nazval cibulová architektura (angl. Onion architecture). Tato architektura podle něj lépe pracuje s provázaností programu [43] a tedy se i více hodí pro náš případ. Palermo ve své práci dodává, že se nejedná o zcela nový přístup, on je ale ten, kdo jej pojmenoval a propagoval.

Cibulová architektura (obr. č. 11.2) se vyznačuje především tím, že infrastruktura je vždy považována za vnější vrstvu (na obrázku *Tests, User interface a Infrastructure*). Ve středu je tzv. doménový model (na obrázku *Domain Model*), který v podstatě reprezentuje entity reálného světa, které v systému používáme [43] ve formě datových



**Obrázek 11.2:** Cibulová architektura (zdroj <https://dzone.com/>)

struktur (objektů). Doménový model je jediná část aplikace, která je závislá pouze sama na sobě. Na tuto vrstvu navazuje libovolný počet dalších vrstev. Obvykle zde bývá vrstva, která se zabývá manipulací s doménovým modelem - ukládání a získávání (na obrázku *Domain Services*). Tato vrstva se zde vždy nachází ve formě rozhraní, implementace tohoto rozhraní je totiž až součástí vyšší vrstvy infrastruktury. Nad těmito vrstvami se může nacházet další vrstva aplikační logiky, kde se již řeší business logika aplikace (na obrázku *Application Services*). Okolo potom stojí infrastruktura (DB, soubory, webové služby), testy, uživatelské rozhraní. Závislosti mezi vrstvami jsou vždy koncipovány, tak aby jedna vrstva závisela pouze na vrstvách, které jsou v obrázku č. 11.2 blíže ke středu. [43]

Při použití této architektury bychom v našem případě mohli vyhovět požadavku na jednoduché provádění změn v infrastruktuře. To je totiž právě důvod, proč je cibulová architektura navržena tímto způsobem. Dle Palerma se totiž nejčastěji mění právě infrastruktura a tato architektura jí proto staví do vnější vrstvy, jejíž změny neovlivní doménu ani business logiku. [43]

Cibulová architektura je tedy pro náš případ nejvhodnější, proto jsme se rozhodli pro jednu z variant této architektury - tzv. hexagonální architekturu.

## 11.5 Hexagonální architektura - Porty a adaptéry

Robert C. Martin (Uncle Bob) se ve své práci zmiňuje o několika architekturách, mezi nimi i cibulová a hexagonální, a shrnuje jejich společné znaky. Zmiňuje, že obě architektury mají stejný cíl, a to je oddělení zájmů (angl. separation of concerns), o tom se mimochodem zmiňuje i Jeffrey Palermo [43]. Obě architektury tohoto dosahují rozdělením systému do vrstev a obě architektury oddělují business část aplikace od infrastruktury. [38] Systém, který staví na těchto principech je obvykle nezávislý na

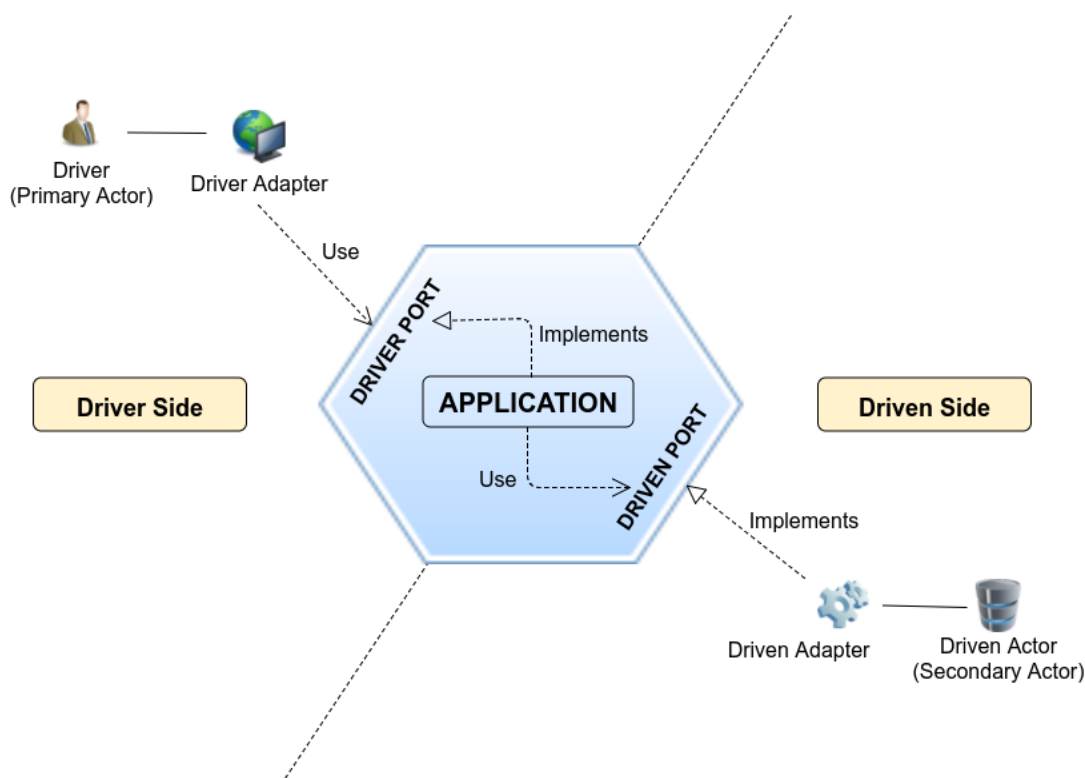
knihovných, dobře testovatelný, architektonicky nezávislý na typu UI, typu databáze i na jiných externích systémech. [38].

Hlavní myšlenku hexagonální architektury zformuloval Alistair Cockburn. Ve svém článku se odkazuje na stejné principy, které jsou zmíněny výše u Palerma. Nejvíce se opírá o myšlenku, že problematika poskytování dat a získávání dat je ve své podstatě totožná a není třeba jí řešit různými způsoby. Cockburn si všiml, že v jiných architekturách (například v Brownově zmíněné výše) je častým jevem, že při návrhu systému se s těmito dvěma problematikami zachází různým způsobem. Architektura se jinak nazývá též porty a adaptéry, protože ty se staly nástrojem, jak se vypořádat s poskytováním a získáváním dat. [19]

- **Porty** jsou vstupními body, které jádro aplikace poskytuje.[53]
- **Adaptéry** slouží jako most mezi aplikací a službami, které jsou potřeba pro její chod. Vždy patří ke specifickému portu. [53]

Přestože se architektura snaží řešit porty a adaptéry univerzálně, vzhledem k existenci různých aktérů v rámci systému, i porty a adaptéry se dále dělí. [19]

Aktéři jsou buď primární či sekundární. Primární aktér řídí aplikaci (typicky uživatel, ale například i test) - na obr. č. 11.3 *Driver Side*. Sekundární aktér je řízen aplikací (typicky databáze, resp. databázový repositář) - na obr. č. 11.3 *Driven Side*. [19] [13]



Obrázek 11.3: Hexagonální architektura (zdroj [13])

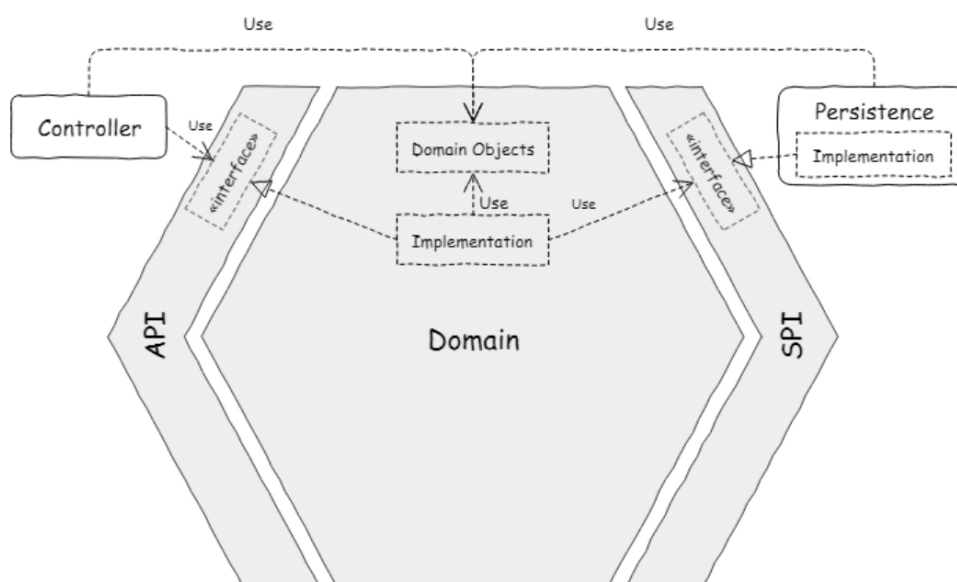
### ■ Dělení portů

- Primární - tvoří aplikační rozhraní nabízené aplikací (na obrázku č. 11.4 API), jsou volány primárními adaptéry [53] [13]
- Sekundární - tvoří rozhraní vyžadované aplikací (na obrázku č. 11.4 SPI), jsou volány jádrem aplikace [53] [13]

### ■ Dělení adaptérů

- Primární - volají primární porty, například testy aplikační logiky, kontroler pro komunikaci s UI (na obrázku č. 11.4 Controller jako příklad) [53]
- Sekundární - implementují sekundární porty, například databázové repozitáře, nebo mockovací knihovny (na obrázku č. 11.4 Persistence jako příklad)[53]

Na obrázku č. 11.4 je navíc dobře vidět, že celá aplikace je svázána doménovým modelem (na obrázku Domain Objects).



**Obrázek 11.4:** Znázornění vzájemných závislostí v hexagonální architektuře (zdroj [7])

Na závěr této sekce je důležité zdůraznit, že přes zmíněné kladné vlastnosti hexagonální architektury existují i záporné. Mezi ně patří především komplexita celého řešení, počet modulů zkrátka roste s počtem portů a adaptérů [13]. Následně můžeme narazit díky komplexitě projektu (závislosti, moduly a podobně) na problémy s výkonem [13] a s komplikovanou správou celého projektu. Jak jsme již zmínili záporny kompenzujeme klady, především dobrou připraveností na změny a testy.

## ■ 11.6 Hexagonální architektura pro náš případ

V následující sekci shrneme, jak bude v našem případě použita hexagonální architektura. Představíme diagram komponent, doménový model a sekvenční diagramy. Důležité

je, že se stále jedná o technologicky nezávislé modely, které popisují obecný návrh systému pro vyhledávání dle kompetencí.

### 11.6.1 Komponenty

Na diagramu komponent, který je součástí přílohy č. C.1, vidíme jednotlivé komponenty systému. Závislosti mezi nimi jsou navrženy tak, jak je tomu v hexagonální architektuře.

#### Doména

Jedná se o centrální část systému, která obsahuje doménový model, business logiku aplikace a porty. Na doméně závisí všechny ostatní komponenty aplikace.

**Doménový model:** Tvoří jádro celého systému, v podstatě se v něm nachází základní entity reálného světa, které jsou v systému namodelovány. Součástí doménového modelu musí být pouze entity, jejich metody a vlastnosti, které nejsou závislé na konkrétním řešení. Doménový model je součástí přílohy č. E.1 a má následující části:

- Osoba - entita reprezentující osobu v dané organizaci (v našem případě FEL ČVUT)
- Pracoviště - entita reprezentující pracoviště v dané organizaci (v našem případě se jedná o katedry)
- Znalost - reprezentuje vazbu mezi osobou a konceptem, přidává jí navíc sílu a poznámku (například jak daná znalost vznikla, resp. odkud jsme jí odvodili)
- Koncept - reprezentuje cokoliv, co může být poznáno osobou, jedná se o informace či atomické pojmenované skutečnosti (konkrétní reprezentace je podmíněna konkrétním řešením, domníváme se však, že bez konceptů se v nějaké podobě neobejde žádné řešení); je důležité nezaměňovat tento pojem s ontologickým konceptem, který reprezentuje ontologickou kategorii (viz 5.4)

V diagramu jsou dále k vidění vazby mezi jednotlivými objekty i s popisem.

**Business logika:** V této části aplikace jsou typicky k dispozici operace s entitami domény (CRUD) a implementované případy užití (hlavní aplikační logika). Business logika využívá SPI porty a její součástí je implementace API portů.

V našem případě se jedná převážně o logiku vyhledávání osob dle kompetencí a operace s tím spojené. V této části se používají SPI adaptéry, které poskytují další služby (například vyhledávání v DB).

**Porty a adaptéry:** Součástí domény jsou API a SPI porty ve formě rozhraní. Dle idey hexagonální architektury je součástí domény zároveň implementace rozhraní API portů.

## ■ API

V levé části diagramu komponent (C.1) je tzv. řídicí část hexagonální architektury ve formě API adaptérů. Tyto adaptéry závisí na API portech implementovaných v doméně a volají jejich funkce. V naší aplikaci tyto adaptéry budou představovat typicky REST API případně GraphQL API. Předpokládáme, že tímto způsobem bude aplikace komunikovat s případnými příjemci (v diagramu komponent je takovýto klient také zobrazen). V obrázku lze též vidět, že předpokládáme logické oddělení Knowledge API, umožňující vyhledávání osob a Base API, které umožňuje základní aplikační případy užití (autentizace, případně další týkající se více správy aplikace, než samotného vyhledávání).

## ■ SPI

Na pravé straně diagramu (C.1) je vidět tzv. řízená část hexagonální architektury ve formě SPI adaptérů. Tato část je klíčová z pohledu odolnosti naší architektury vůči změnám použitého řešení. Bez ohledu na to, jaké konkrétní řešení pro vyhledávání dle kompetencí použijeme, domény se to dotkne jen minimálně. Naopak v této části SPI adaptérů, konkrétně u adaptérů zajišťujících samotné vyhledávání, předpokládáme změny. U uživatelských adaptérů (jak jsme je nazvali na obrázku C.1), které poskytují spíše aplikační logiku (například přihlašování), opět neočekáváme velké změny, přesto by to bylo možné. V diagramu jsme navrhli několik možných adaptérů, jedná se však pouze o příklady, v konkrétním případě záleží na zvoleném řešení. SPI Adaptéry zajišťují další komunikaci s případnými externími systémy (například databáze, API jiných systémů).

### ■ 11.6.2 Komunikační sekvence

Součástí přílohy D je diagram sekvencí, na kterém jsme se pokusili demonstrovat, jak bude probíhat komunikace mezi cílovým uživatelem naší aplikace a aplikací samotnou při volání UC1 (11.3.3). Zároveň je na něm k vidění, jak spolu komunikují jednotlivé komponenty aplikace.

Důležité je, že komunikace s aplikací může probíhat synchronně (D.1) či asynchronně (D.2). To jinými slovy znamená, že výsledek hledání může koncový uživatel dostat ihned po dotazu, nebo mu bude navrácen identifikátor hledání a na jeho výsledek se zeptá později. Která varianta bude použita, záleží na zvoleném řešení.

## ■ 11.7 Shrnutí

V této sekci jsme se zabývali návrhem systému pro vyhledávání osob dle jejich kompetencí. Začali jsme od cílů, požadavků a případů užití a pokračovali jsme až k detailnímu popisu zvolené architektury. V následující sekci na náš návrh navážeme popisem implementace prototypu.



# Kapitola 12

## Implementace

V následující kapitole přistoupíme k popisu prototypu, který jsme v rámci této práce implementovali. Tím navážeme na předchozí kapitolu, ve které jsme řešili návrh systému.

### 12.1 Rozsah implementovaného prototypu

#### 12.1.1 Server

V serverové části prototypu je implementován základní případ užití, tedy UC-1 (11.3.3). Prototyp obsahuje funkční zpracování hexagonální architektury, kterou jsme popsali v předchozí kapitole. Vertikální průřez aplikací je funkční, probíhá synchronní dotaz na funkční datové úložiště s testovacími daty, stejně jako je vidět na sekvenčním digramu (příloha D.1).

#### 12.1.2 Klient

Klientská část prototypu je přítomna pro demonstraci toho, že komunikace se serverovou částí probíhá a klient je schopný data zpracovávat. V tomto klientu lze vidět implementovanou obdobu W-UC1 (obr. č. 11.1).

### 12.2 Diagram komponent

Diagram komponent konkrétního řešení je k vidění v příloze F.1. Jelikož vychází z návrhu zmíněného v minulé kapitole, je obdobou obecného digramu komponent z přílohy C.1. V následujících sekcích rozebereme jednotlivé komponenty, technologie, které jsme pro jejich vývoj použili a zmíníme případné ukázky.

### 12.3 Hlavní použité technologie

V této sekci shrneme základní použité technologie, konkrétnější zástupce poté rozebereme u jednotlivých modulů. Soupis všech použitých technologií je poté připojen v příloze J.



```

knowledge_search/
|-- application/
    |-- config/
    |-- KnowledgeSearchApplication
|-- domain/
    |-- api/
    |-- logic/
    |-- model/
    |-- spi/
|-- gdb_solution/
    |-- adapters/
    |-- model/
    |-- dao/
|-- rest
    |-- controllers/
    |-- model/

```

**Listing 12.1:** Schématicky zobrazená struktura projektu

## 12.5 Doména

V diagramu komponent (F.1) je doména ve střední části, ve struktuře projektu se nazývá `domain`. Doména má čtyři části `logic`, `model`, `spi` a `api`.

V části `logic` se nachází business logika celé aplikace. V nynější verzi se v této vrstvě volají `spi` repozitáře, složitější logika není přítomna.

Část `model` obsahuje doménový model aplikace ve formě rozhraní.

```

public interface Knowledge extends Identifiable{
    Double getMetric();
    String getInfo();
    <T extends Person> T getOwner();
    <T extends Concept> T getConcept();
}

```

**Listing 12.2:** Ukázka rozhraní jedné z entit v doménovém modelu

Modul `spi` obsahuje repozitáře ve formě rozhraní, které později implementuje modul `gdb_solution`.

```

public interface PersonRepository {
    Collection<? extends Knowledge>
    searchPeopleByKnowledge(String conceptId, int offset, int limit);
    Collection<? extends Person> getAllPersons();
}

```

**Listing 12.3:** Ukázka rozhraní SPI repozitáře

Poslední modul domény je `api`, třídy v tomto modulu obsahují rozhraní a implementaci api portů. Tyto porty jsou později používány api adaptéry při volání aplikační logiky.

## 12.6 Spouštěcí modul

Modul `application` má v systému zásadní roli - spouštění a konfigurace aplikace. Spouštění aplikace zajišťuje vstupní bod, třída `KnowledgeSearchApplication`. Konfigurace obsahuje adresář `config/`. V tomto adresáři jsou definovány tzv. *Spring Bean* třídy, které zajišťují předávání námi zvolené instance určitého objektu Spring IoC kontejneru. Kontejner s instancí dále v běhu aplikace nakládá a vrací jí tam, kde jí potřebujeme. V tomto modulu tedy konfigurujeme *Spring Bean* třídy pro všechny ostatní moduly aplikace. Díky tomu máme plnou moc nad tím, kterou implementaci konkrétní části architektury, použijeme. To podporuje náš požadavek na jednoduché změny (viz FURPS+ analýza - 11.3.2).

Mimo výše zmíněného se v této části aplikace nachází další nastavení např. bezpečnostní.

```
@Configuration
public class RestConfiguration {
    @Bean
    public PersonController
        personController(PersonService personService){
        return new PersonController(personService);
    }

    @Bean
    public ConceptController
        conceptController(ConceptService conceptService){
        return new ConceptController(conceptService);
    }
}
```

Listing 12.4: Příklad konfigurace REST kontrolerů

## 12.7 API adaptér

V našem případě se tento modul nazývá `rest`, jelikož se jedná o implementaci REST rozhraní. Na diagramu komponent (F.1) je tento adaptér k vidění v levé části. Tento modul zprostředkovává komunikaci s klientskou částí prototypu.

Ve frameworku *Spring boot* je implementace REST velmi přímočará, je k tomu zapotřebí pouze několik anotací (viz příklad níže). Jedná se o API adaptér, používá tedy API porty implementované v části `domain`, v příkladu níže je to například implementace třídy `ConceptService`.

```
@RestController
@RequestMapping("/concepts")
```

```

public class ConceptController {

    private final ConceptService conceptService;

    public ConceptController(ConceptService conceptService) {
        this.conceptService = conceptService;
    }

    @GetMapping("")
    List<ConceptDTO> getAllConcepts() {
        return conceptService.getAllConcepts().stream()
            .map(ConceptDTO::new)
            .collect(Collectors.toList());
    }
}

```

**Listing 12.5:** Příklad REST kontroleru pro operace s koncepty

V příkladu je navíc vidět, jak jsou koncepty poskytované doménou mapovány na entity vytvořené pro účely klientského použití (anglicky tzv. Data transfer object, zkratka DTO). Toto mapování je zde z toho důvodu, aby nebyla zbytečně přenášena data, která klient nutně nepotřebuje. V našem případě například nepotřebujeme z konceptu nic více než jen název a id. Mapování je v tomto případě jednoduché, je tedy přímo v této třídě. Druhou možností je mít oddělené mapovací třídy.

## 12.8 SPI adaptér

Klíčovou částí našeho řešení, zajišťující vyhledávání osob dle kompetencí je modul `gdb_solution`. Tento modul zajišťuje komunikaci se zvolených datovým uložištěm, v našem případě graphDB. Na diagramu komponent (F.1) je tento adaptér k vidění v pravé části.

Modul je rozdělen do tří vrstev - `adapters`, `model` a `dao`. Vrstva `adapters` obsahuje implementaci SPI repozitářů, které jsou pomocí rozhraní definovány v doméně. Tyto repozitáře dále využívají třídy vrstvy `dao`, která zajišťuje komunikaci s databází.

Vrstva `dao` obsahuje třídy pro přístup k datům (anglicky tzv. data access object). V následující sekci popíšeme, zevrubněji, jak tato vrstva komunikaci s databází zajišťuje.

Poslední vrstvou SPI adaptéru je `model`, který obsahuje entity. Tyto entity slouží k mapování výsledků databázových dotazů na JAVA objekty. Více se o tomto mapování zmíníme v následující sekci.

### 12.8.1 Přístup k datům

Jak už jsme výše zmínili, vrstva `dao` zodpovídá za komunikaci s databází. Komunikace s GraphDB probíhá pomocí HTTP protokolu a při programovém zpracování této komunikace lze využít dříve zmíněnou knihovnu RDF4J. Mapování výsledků dotazů na vlastní JAVA objekty je poměrně pracné, ostatně proto jsou u relačních databází populární knihovny zajišťující ORM (angl. object-relation mapping) neboli objektově-

relační mapování. V našem případě jsme udělali rešerši knihoven podporující OOM neboli ontologicky-relační mapování.

#### Nalezli jsme následující knihovny:

- Komma (dostupné na: <http://komma.enilink.net/docs/framework/objectmapping/index.html>)
- RDFBeans (dostupné na: <https://rdfbeans.github.io/quickstart.html>)
- JOPA (dostupné na: <https://github.com/kbss-cvut/jopa/wiki/>)

Knihovnou, kterou jsme zvolili, je JOPA [36]. Projekt JOPA je, dle repozitáře na github, stále aktivní a shodou okolností je vyvíjen na stejné fakultě, na které vzniká naše práce, FEL ČVUT. V případě problémů bychom tedy mohli jednoduše oslovit tvůrce. Zároveň JOPA umožňuje mapování výsledků dotazů definovaných v dao (pomocí jazyka SPARQL) na objekty z balíčku `gdb_solution/model`, což je přesně to, co potřebujeme.

```
@Id
private URI uri;

@ParticipationConstraints(nonEmpty = true)
@OWLDataProperty(iri =
"http://onto.fel.cvut.cz/koubadom/properties#name")
private String name;

@ParticipationConstraints(nonEmpty = true)
@OWLDataProperty(iri =
"http://onto.fel.cvut.cz/koubadom/properties#surname")
private String surname;

@ParticipationConstraints(nonEmpty = true)
@OWLDataProperty(iri =
"http://onto.fel.cvut.cz/koubadom/properties#username")
private String username;
```

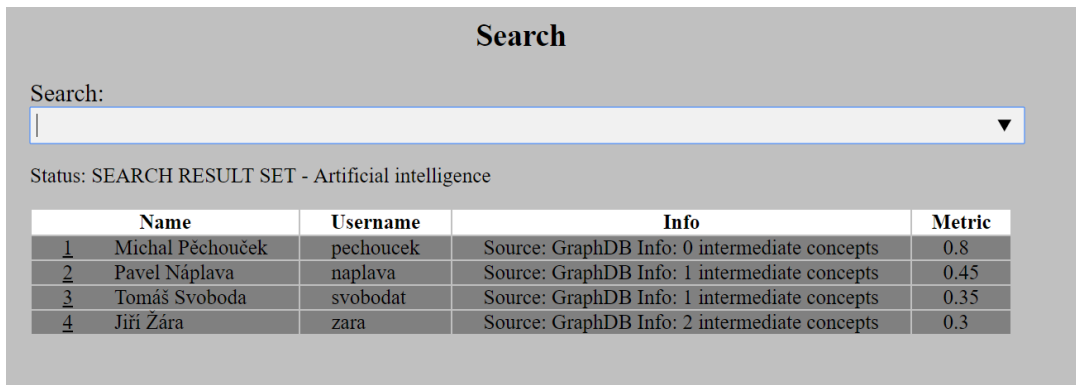
**Listing 12.6:** Příklad definice proměnných polí v entitě PersonGDB za pomoci anotací z knihovny JOPA

## 12.9 Klientská část prototypu

Klientská část našeho prototypu (na diagramu F.1 na levé straně) komunikuje se serverovou a demonstruje využitelnost API.

Funkční i obsahová stránka klienta není příliš složitá. Součástí klienta je HTML stránka s jednoduchým vyhledávacím polem a tabulkou s výsledkem vyhledávání. Tuto strukturu funkčně obohacuje několik javascriptových tříd. Hlavní funkcí javascriptu je pochopitelně načítání dat z API, to je realizováno rozhraním funkce *fetch* (dokumentace

dostupná na: [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)).



The screenshot shows a search interface with the title "Search". Below the title is a search input field with the text "Search:" and a dropdown arrow. Below the input field, the status is "Status: SEARCH RESULT SET - Artificial intelligence". Below the status is a table with four columns: "Name", "Username", "Info", and "Metric". The table contains four rows of search results.

	Name	Username	Info	Metric
1	Michal Pěchouček	pechoucek	Source: GraphDB Info: 0 intermediate concepts	0.8
2	Pavel Náplava	naplava	Source: GraphDB Info: 1 intermediate concepts	0.45
3	Tomáš Svoboda	svobodat	Source: GraphDB Info: 1 intermediate concepts	0.35
4	Jiří Žára	zara	Source: GraphDB Info: 2 intermediate concepts	0.3

**Obrázek 12.1:** Snímek klientské části prototypu s načteným výsledkem hledání (zdroj autor)

## 12.10 Databáze

Jak jsme zmínili výše, jako triplě-store databázi jsme zvolili GraphDB. V této sekci popíšeme ontologické schéma navržené pro naši databázi.

### 12.10.1 Databázové schéma

Již v kapitole č. 7 jsme zmínili, že volbu ontologického schematu lze realizovat buď vlastní ontologií, existující ontologií, nebo kombinací obojího.

Podrobně naše řešení rozebereme v pozdějších sekcích této kapitoly. Zde však upozorníme na to, k čemu jsme konkrétně použili ontologie. V našem konkrétním řešení používáme ontologii v podstatě jako slovník, ve kterém jsou shromážděny koncepty provázané vazbami. V tomto slovníku poté vyhledáváme koncepty a hlavně jejich souvislosti.

Jak jsme již zmínili, tvorba slovníku nebo tezauru je typickým případem užití pro existující ontologii SKOS. Rozhodli jsme se, že pro náš prototyp použijeme tu.

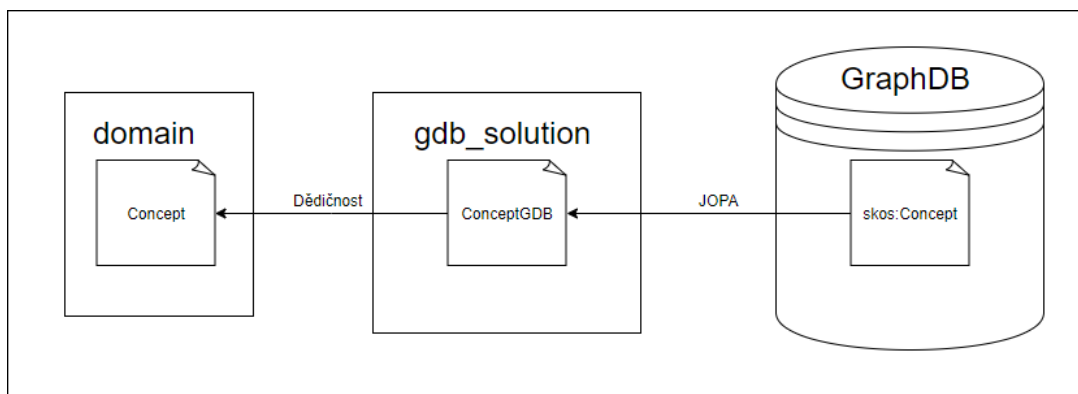
Slovník bude obsahovat koncepty, zbývají nám tedy ostatní entity z našeho doménového modelu - znalost, osoba a pracoviště. Tyto budeme ukládat též do ontologií, jelikož toto datové úložiště již máme k dispozici a není třeba zvyšovat komplexnost systému další databází.

V příloze G.1 je k vidění použité schéma, tzn. původní SKOS schéma (zdroj: <https://www.w3.org/2009/08/skos-reference/skos.rdf>) obohacené o další entity (osoba, pracoviště a znalost).

### 12.10.2 OOM

Ontologicky-relační mapování probíhá z databáze na entity v balíčku `gdb_solution/model`, které jsou implementací rozhraní entit domény z balíčku `domain/model`. Tento programový vzor jsme zvolili, jelikož implementace SPI adaptérů se může v čase měnit.

Model příslušného řešení tedy musí implementovat rozhraní modelu domény, nicméně může obsahovat další atributy, specifické pro konkrétní řešení. Na obrázku 12.2 je celá situace schématicky zobrazena na entitě Koncept. Vidíme, že Koncept z databáze se se mapuje na třídu `ConceptGDB` z `gdb_solution/model` a s tou dále v `domain` pracujeme jako s implementací rozhraní `Concept`.



**Obrázek 12.2:** Mapování objektů zobrazené schématicky, příklad na objektu Koncept (zdroj autor)

## 12.11 Popis implementovaného řešení

Výše jsme rozebrali části, ze kterých se navrhovaný systém skládá. V této sekci rozebereme, v jakých krocích vyhledávání lidí dle kompetencí probíhá a jednotlivé kroky popíšeme.

### 12.11.1 Jednotlivé kroky vyhledávání

1. **Seznam konceptů** - klientská část vyšle požadavek na seznam konceptů, na které je možno se dále dotazovat
2. **Příjem požadavku a odpověď** - rest modul požadavek přijme a předá jej `domain`, ta poté předá modulu `gdb_solution`, který vrací zpět skrz předchozí moduly seznam konceptů s názvem a identifikátorem
3. **Vyhledání dle kompetence** - klientská část pošle identifikátor konceptu, pro který je třeba nalézt seznam kompetentních osob
4. **Příjem požadavku** - rest modul požadavek přijme a předá jej `domain`, ta poté předá modulu `gdb_solution`
5. **Zpracování požadavku** - hledaný koncept se využije jako parametr pro použitý databázový dotaz (vysvětleno níže), výsledek dotazu je ještě v komponentě `gdb_solution` dále zpracován (vysvětleno níže) a následně vrácen do `domain`
6. **Návrat dat** - výsledek je vrácen jako seznam osob s vypočtenou silou znalosti, příklad kompletního výsledku je možno vidět v příloze I



## ■ 12.11.2 SPARQL dotaz

### ■ Dotaz

Součástí přílohy H je použitý databázový dotaz v jazyce SPARQL. Nejpřímočařejší je popsat dotaz pomocí jeho výsledku. Dotaz vrací výčet konceptů ležících mezi vyhledávaným konceptem a danou osobou a váhu znalosti, kterou osoba k poslednímu z konceptů má. Jako hrana se v tomto průchodu používá relace *semanticRelation*. Tato relace je součástí ontologie SKOS a je nejobecnější vazbou mezi dvěma koncepty, značí, že spolu dva koncepty významově souvisí.

Dotaz ve své podstatě prohledává po kružnicích okolí konceptu, ke kterému hledáme znalost, a vyhledává osoby, které se nachází maximálně čtyři *semanticRelation* relace daleko od tohoto konceptu. Číslo čtyři v tomto nemá žádný význam, vzniklo empiricky, jelikož provázanost konceptů přes vazbu *semanticRelation* je tak rozsáhlá, že u pěti již dotaz vrací příliš velký výsledek.

### ■ Zpracování výsledku dotazu

Výsledek dotazu je třeba uživateli interpretovat, dotaz vrací sílu znalosti k poslednímu konceptu, který leží mezi osobou a vyhledávaným konceptem, a výčet všech mezilehlých konceptů, neexistuje tedy jednotná metrika, dle které by bylo možné výsledek seřadit. Navíc ve výsledku dotazu se osoby opakují a ve finálním výsledku by se měla každá vyskytovat maximálně jednou.

Výslednou sílu znalosti  $W$  jsme vypočítali následujícím vzorcem:

$$W = \frac{w}{n + 1}$$

Kde  $w$  je síla znalosti k poslednímu z konceptů a  $n$  je počet mezilehlých konceptů.

Výskyt duplicitních hodnot jsme vyřešili tak, že z celého výsledku bereme pro každou z osob vždy jen záznam s nejvyšším  $W$ .

## ■ 12.12 Rozšiřitelnost

Z FURPS analýzy (11.3.2) vyplývá požadavek na modularitu systému, aby jednotlivé moduly (hlavně `gdb_solution`) byly jednoduše nahrazovány jinými implementacemi a podobně.

Pokud k takové změně dojde (konkrétně u SPI adaptéru), je třeba implementovat rozhraní potřebných SPI portů. Následně je třeba modifikovat konfiguraci v `application` a poskytnout Spring kontejneru příslušné nové implementace SPI adaptéru.

## ■ 12.13 Dokumentace projektu

Dokumentace implementační části této práce vznikla na několika úrovních:

- Javadoc - součástí odevzdaného programu



# Kapitola 13

## Testování

V následující kapitole rozebereme testování. Kapitola bude mít dvě části. V první z nich rozebereme obecně, jakým způsobem lze testovat hexagonální architekturu. V druhé části demonstrujeme některé z představených druhů testů na našem prototypu.

### 13.1 Testování hexagonální architektury

Alistar Cockburn, který hexagonální architekturu nejvíce prosadil, zmiňuje jako jeden z hlavních přínosů této architektury právě izolované testování. Primárně mluví o automatizovaném testování domény, odstíněném od ostatní infrastruktury (databáze, API jiných systémů) [19]. Doména obsahuje hlavní případy užití, je velmi výhodné je testovat izolovaně od všeho ostatního a hlavně automatizovaně.

Testování hexagonální architektury může probíhat na všech částech systému, nejen na doméně.

#### Rozdělení testování:

- Testování domény
- Testování SPI
- Testování API

#### 13.1.1 Testování domény

Testy domény jsou architektonicky na stejné úrovni jako jsou API adaptéry, ve své podstatě se jedná o API adaptéry [19]. Tyto adaptéry volají implementaci API portů, které obsahují logiku, a tuto logiku testují. Logika (případy užití) je dále nabízena ostatním systémům, je tedy nutné jí otestovat.

V případě tohoto druhu testování je důležité doménu izolovat od ostatních vlivů, v podstatě se jedná o jednotkové testování. Ostatními vlivy jsou v tomto kontextu myšleny SPI adaptéry, které můžeme nahradit například mock objekty [19]. Jinými slovy můžeme SPI adaptéry, které slouží integraci s jinými systémy, nahradit dalšími adaptéry, které budou tuto integraci pouze předstírat a budou vracet námi definovaný správný výsledek.

### 13.1.2 Testování SPI

Poté co zajistíme doménu, je třeba zajistit, že SPI adaptéry budou poskytovat všechna data ve správné podobě (v podobě jako v mock objektech).

SPI adaptéry resp. moduley, které obsahují SPI adaptéry, mohou mít libovolnou vnitřní architekturu. Tuto architekturu je třeba testovat tak, jak je pro ní typické, a tak zajistit funkčnost SPI adaptérů.

Na klasické vrstevnaté architektuře (obdoba brownovy architektury 11.1) lze například provádět testování po jednotlivých vrstvách jako je tomu zvykem. Obvyklé je využití mock objektů, in-memory databází a obdobných technik.

### 13.1.3 Testování API

Posledním modulem, který zbývá jsou API adaptéry. Existují různé druhy těchto adaptérů, pro testování bereme v úvahu hlavně REST a GraphQL.

Tento druh testování se v případě malé aplikace může provést ručně. V komplikovanějších případech existují různé testovací knihovny a frameworky (například `HttpClient` - <http://hc.apache.org/httpcomponents-client-ga/index.html>). Tyto knihovny typicky volají API systému a kontrolují funkčnost, jako například návratové kódy.

## 13.2 Testování implementované aplikace

Prototyp aplikace, který jsme v rámci této práce vyvinuli se nejvíce opírá o modul `gdb_solution`, na ten je třeba se zaměřit při testování nejintenzivněji.

Oblasti naší aplikace k testování:

- Persistentní vrstva modulu `gdb_solution`
- Logika modulu `gdb_solution`
- REST API

Samozřejmě je možné zapojit další druhy testů, vzhledem k rozsahu prototypu jsou však tyto tři úrovně dostačující.

### 13.3 Průběh testu

Testy jsou v naší aplikaci vždy rozděleny na tři části:

- *arrange* - příprava akce (inicializace instancí, jiné integrační akce a podobně)
- *act* - testovaná akce
- *assert* - ověření výsledku

### 13.3.1 Testování persistentní vrstvy

Integrační testy persistentní vrstvy jsou v kontextu naší práce klíčové. Hlavně proto, že nepoužíváme jiné datové úložiště než graphDB. Navíc hlavní dotaz pro vyhledávání osob dle znalosti probíhá pomocí SPARQL na této úrovni.

Při testování integrace se používá testovací repositář GraphDB. Hlavním důvodem je, že integrační testy musí mít izolované prostředí, nezávislé na ostatních (produkčních a podobně).

#### Pokrytí testy

Jelikož je modul `gdb_solution/dao` naprogramován genericky, je poměrně přímočaré otestovat společné metody dao objektů (CRUD). Individuálně se testují jen specifické funkce jednotlivých dao objektů, v našem případě jen `PersonDao`.

#### Příklad integračního testu s popisem

Níže je k vidění příklad integračního testu, který testuje metodu `findAll`. Test probíhá nad entitou `PersonGDB`, nicméně se jedná o generickou metodu, testujeme tedy i ostatní entity. V části *arrange* připravujeme prostředí, tedy vytváříme tři instance entity osoba a zapisujeme je do testovacího repositáře graphDB. V části *act* čteme všechny záznamy osob z repositáře graphDB. V části *assert* kontrolujeme obsah výsledku s obsahem původního seznamu, používáme k tomu funkce `assertEquals` a `assertTrue` z knihovny JUnit 4 (<https://junit.org/junit4/>).

```
@Test
public void findAllReturnsAllExistingInstances() {
    //arrange
    final List<PersonGDB> persons = new ArrayList<>();
    persons.add(new PersonGDB("Dominik", "Kouda", "koudadom"));
    persons.add(new PersonGDB("Dominik2", "Kouda1", "koudadom1"));
    persons.add(new PersonGDB("Dominik1", "Kouda2", "koudadom2"));
    //act
    personDao.persist(persons);
    final List<PersonGDB> result = personDao.findAll();
    //assert
    assertEquals(persons.size(), result.size());
    boolean found = false;
    for (PersonGDB person : persons) {
        for (PersonGDB person2 : result) {
            if (person.equals(person2)) {
                found = true;
                break;
            }
        }
    }
    assertTrue(found);
}
```

```
}
}
```

**Listing 13.1:** Příklad integračního testu

### 13.3.2 Testování logiky

Logikou je v modulu `gdb_solution` myšlena hlavně třída `KnowledgeDTOList`. Tato třída reprezentuje množinu znalostí. Množina má navíc tu vlastnost, že v případě vložení duplicitního záznamu, zůstane v množině vždy prvek s větší silou znalosti. Tato třída se využívá při vyhledávání osob dle znalosti ke kompletaci výsledku SPARQL dotazu.

V testu níže lze vidět obdobný postup jako tomu bylo výše. V tomto konkrétním příkladu kontrolujeme, že při přidání jedné osoby vícekrát se aktualizuje její síla znalosti, byla-li přidávána vyšší.

```
@Test
public void addsOnlyHighest() {
    //arrange
    KnowledgeDTOList list = new KnowledgeDTOList();
    PersonGDB p1 = new PersonGDB("Dominik", "Kouda", "koudadom");
    PersonGDB p2 = new PersonGDB("Dominik2", "Kouda1", "koudadom1");
    PersonGDB p3 = new PersonGDB("Dominik1", "Kouda2", "koudadom2");
    //act
    list.add(new KnowledgeDTO(p1, 0.5, new HashSet<>()));
    list.add(new KnowledgeDTO(p2, 0.5, new HashSet<>()));
    list.add(new KnowledgeDTO(p3, 0.6, new HashSet<>()));
    list.add(new KnowledgeDTO(p1, 0.7, new HashSet<>()));
    //assert
    assertEquals(3, list.size());
    boolean success = false;
    for(KnowledgeDTO dto: list.getKnowledgeDTOs()){
        if(dto.getOwner().equals(p1) &&
            dto.getWeightOfKnowledge() == 0.7){
            success = true;
        }
    }
    assertTrue(success);
}
}
```

**Listing 13.2:** Jednotkový test aplikační logiky modulu `gdb_solution`

Tímto druhem testů je otestována správná funkce této třídy. Konkrétně je to kontrola duplicit v množině a maximalizace síly znalosti každé osoby v množině, při postupném přidávání.

### 13.3.3 Testování REST API

API má dva základní případy použití - vrácení seznamu všech konceptů a vyhledání osob dle zvoleného konceptu. Vzhledem k tomu, že tato část aplikace není příliš složitá, rozhodli jsme se pro ruční otestování.

#### Scénáře

Testovaný případ	Očekávaný výsledek	Výsledek testu
Dotaz na všechny koncepty	Seznam všech konceptů ve formátu JSON	OK
Vyhledání dle konceptu (existujícího)	Seznam osob ve formátu JSON	OK
Vyhledání dle konceptu (neexistujícího)	Status 404	Nalezena chyba

**Tabulka 13.1:** Tabulka testovacích případů pro REST API

Testování proběhlo bez větších problémů, byla nalezena chyba při dotazu na neexistující koncept. Chyba byla opravena.

## 13.4 Shrnutí

V této kapitole jsme shrnuli testování naší aplikace. V úvodu jsme popsali výhody testování hexagonální architektury a používané principy. V druhé části kapitoly jsme popsali testování serverové části našeho prototypu. Vzhledem k rozsahu prototypu jsme nepoužili všechny druhy testů, které je možné pro hexagonální architekturu implementovat.

Implementace testů proběhla bez problémů, nalezené chyby jsme opravili. Ruční testování proběhlo též bez komplikací, nalezené chyby jsme opravili.





# Kapitola 14

## Budoucí práce

### 14.1 Neprobraná témata

Během práce jsme narazili na několik témat, které jsme z rozsahu této práce vyloučili. Není však pravda, že tato témata nemají potenciál, proto je zmiňujeme zde.

#### 14.1.1 Osobnost a dovednosti

V sekci zabývající se reprezentací dat (kapitola č. 5), jsme se rozhodli zanedbat část z vlivů na kompetence osoby - osobnost (povaha, motivy). Toto téma je velmi komplexní z hlediska reprezentace těchto dat i z hlediska potenciálních datových zdrojů - vlastní popis sama sebe, reference zaměstnavatele a jiné.

Ve stejné sekci jsme vyloučili dovednosti, v této práci jsme je v podstatě sloučili se znalostmi. Znalosti však nejsou totéž co dovednosti. Společně se zkušenostmi by dovednosti měli být dalším důležitým okruhem, který staví na znalostech a popisuje, jak je osoba umí využívat.

#### 14.1.2 Vlastní datový zdroj

V sekci týkající se datových zdrojů (kapitola č. 10) jsme rozebrali primárně zdroje, které již existují. Je možné vytvořit vlastní zdroj - např. na základě vstupů od osob (formulář pro tvorbu profilu v databázi kompetencí). Případně je možné vlastní zdroj vytvořit z existujících - anotace existujících dat a podobně.

#### 14.1.3 Nestrukturovaná data

V sekci zpracování dat (kapitola č. 7), konkrétně jejich transformace do požadovaných formátů, jsme se zaměřili pouze na serializovaná/strukturovaná data. Data však nejsou vždy k dispozici v takto strojově dobře čitelném formátu. Často jsou nestrukturovaná a nevhodná pro strojové zpracování např. prostý text. Zde se jedná o problematiku sémantické anotace dat, zpracování přirozeného jazyka a podobně. Jedná se např. o zdroje jako jsou životopisy nebo osobní stránky.



# Kapitola 15

## Závěr

Tato práce měla tři základní cíle. Prvním cílem bylo *zanalyzovat problematiku vyhledávání osob a pracovišť dle kompetencí*. Tento cíl jsme splnili v první části, kdy jsme celý problém rozdělili na dílčí úkoly, a ty jsme postupně rozebrali. Začali jsme konceptuálním modelem celého problému, ze kterého jsme byli schopni lépe pochopit problematiku. Dále jsme zanalyzovali možnosti reprezentace kompetencí resp. znalostí, postupovali jsme od těch jednodušších (sémantické sítě) k těm složitějším (deskripční logika). V této části jsme jako nejvhodnější reprezentaci pro náš případ zvolili ontologie. Hlavně proto, že jako jediný formální jazyk pro reprezentaci znalostí mají praktickou podporu technologií jako jsou RDF, OWL a SKOS.

Dále jsme zanalyzovali možné datové zdroje, zejména zdroje znalostí osob a znalostní báze. V této části jsme zjistili, že znalosti osob lze v akademickém prostředí nejlépe získat prostřednictvím vědeckých publikací a jiných výstupů vědecké činnosti. V komerčním prostředí se pro naše účely mohou vyskytovat znalostní systémy. Nejrozsáhlejší zdroje znalostní báze jsou na globální úrovni DBpedia a WordNet. Zjistili jsme také, že zdroje strukturovaných dat jsou často velmi omezené a že práce s nestrukturovanými daty jako je prostý text vyžaduje další zpracování (sémantická anotace a podobně).

Nakonec jsme se zabývali zpracováním dat. Rozebrali jsme jejich transformaci do ontologií pomocí nástrojů jako je OntoRefine. Následně jsme rozebrali jazyk SPARQL, který je klíčový v oblasti práce s ontologickými daty (RDF triply) a umožňuje provádět CRUD operace. V závěru této sekce jsme ještě zmínili obecný způsob získání síly znalosti, která je klíčovou metrikou při porovnávání znalostí více osob.

Druhý cíl *navrhnout systém pro vyhledávání osob a pracovišť dle kompetencí a provést rešerši dostupných dat* jsme naplnili ve druhé části práce. Nejdříve jsme rozebrali datové zdroje dostupné na FEL ČVUT. Klíčové jsou v tomto ohledu systémy Usermap a V3S. Systém Usermap může díky svému API sloužit jako zdroj osob a pracovišť. Systém V3S obsahuje rozsáhlou databázi vědecké činnosti na ČVUT. Dalším zjištěním v této části práce bylo, že na ČVUT neexistuje žádné jednotné rozhraní pro získávání nevědeckých dat o zaměstnancích - životopisy, osobní stránky a podobně.

Na analýzu datových zdrojů jsme navázali návrhem systému pro vyhledávání dle kompetencí. Systém jsme rozebrali počínaje požadavky, hlavním z nich je funkční požadavek na samotné vyhledávání, další důležitý je implementační požadavek na modularitu celého systému. Důraz na modularitu klademe, protože vyhledávání dle kompetencí je komplikovaný problém a není zjevné, kolik možných řešení existuje a jak je lze kombinovat.

Dále jsme postupovali přes případy užití a wireframy k výběru architektury. Požadavek na modularitu systému byl hlavním důvodem, proč jsme se odklonili od klasické vrstevnaté architektury. Nejdříve jsme rozebrali cibulovou architekturu, která je vzhledem ke svému nízkému provázání (angl. decoupling) vhodným kandidátem pro systémy jako je ten náš. Nakonec jsme zvolili variantu této architektury, totiž architekturu hexagonální. Tato architektura navíc ještě definuje způsob komunikace pomocí portů a adaptérů, což se v našem případě hodí. Hexagonální architekturu jsme tedy zevrubně představili a následně jí aplikovali na náš případ.

V poslední části práce jsme rozebrali prototyp navržené aplikace, který jsme implementovali. Prototyp má dvě části, server a klient. Demonstrovali jsme na něm funkčnost navržené aplikace. Jako technologii pro serverovou část jsme zvolili programovací jazyk JAVA, který vyhovuje technologickému prostředí ČVUT FEL. Pro ukládání dat pomocí ontologií jsme využili existující *triple-store* GraphDB, jako schéma jsme použili ontologii SKOS rozšířenou o několik našich entit. Nakonec jsme v poslední kapitole rozebrali testování hexagonální architektury, nejdříve obecně a následně jsme některé z technik demonstrovali na našem prototypu.

Splnili jsme všechny stanovené cíle. Rozebrali jsme zevrubně celý problém vyhledávání dle kompetencí, navrhli systém, který tuto problematiku řeší a implementovali a otestovali prototyp pro FEL ČVUT. Nakonec jsme v kapitole budoucí práce shrnuli, jak na tuto práci navázat. Největší otázkou pro budoucí práce zůstává transformace existujících dat do zvolených struktur, do RDF triplů, a jejich následné začlenění do systému.



## Literatura

- [1] *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [2] Princeton University "About WordNet." WordNet. Princeton University. <http://wordnet.princeton.edu>, 2010. Navštíveno: 2019-04-20.
- [3] Orcid. <https://orcid.org/>, 2012. Navštíveno: 2019-04-18.
- [4] GraphQL. <https://facebook.github.io/graphql/>, 2015. Navštíveno: 2018-12-25.
- [5] Wordsapi. <https://www.wordsapi.com/>, 2015. Navštíveno: 2019-04-21.
- [6] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). *Official Journal of the European Union L 119/1* (2016).
- [7] Decoupling your technical code from your business logic with the hexagonal architecture (hexarch). <https://beyondxscratch.com>, 2017. Navštíveno: 2019-04-28.
- [8] Cambridge dictionary. <https://dictionary.cambridge.org/>, 2018. Navštíveno: 2018-11-25.
- [9] Collins dictionary. <https://www.collinsdictionary.com>, 2018. Navštíveno: 2018-12-01.
- [10] Macmillan dictionary, 2018.
- [11] max plack institute informatik. <https://www.mpi-inf.mpg.de>, 2018. Navštíveno: 2019-04-21.
- [12] Merriam-webster. <https://www.merriam-webster.com/dictionary>, 2018. Navštíveno: 2018-11-25.
- [13] Ports and adapters pattern (hexagonal architecture). <https://softwarecAMPment.wordpress.com/portsadapters/#tc8>, 2018. Navštíveno: 2019-04-28.

- [14] Datamuse. <http://www.datamuse.com>, 2019. Navštíveno: 2019-04-21.
- [15] Twinw-rd api. <https://www.twinword.com/api/>, 2019. Navštíveno: 2019-04-21.
- [16] Wiki dbpedia. <https://wiki.dbpedia.org/>, 2019. Navštíveno: 2019-04-20.
- [17] BORST, W. N. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente, Enschede, Netherlands, 1997. base-search.net (ftunivtwente:oai:doc.utwente.nl:17864).
- [18] BRAY, T. The javascript object notation (json) data interchange format. Tech. rep., 2017.
- [19] COCKBURN, A. The pattern: Ports and adapters (“object structural”). <http://alistair.cockburn.us/Hexagonal+architecture>, 2005. Navštíveno: 2019-04-27.
- [20] DAVID BECKETT, TIM BERNERS-LEE, E. P.-G. C. Terse rdf triple language. <https://www.w3.org/TR/turtle/>. Navštíveno: 2018-12-28.
- [21] DAVIS, R., SHROBE, H., AND SZOLOVITS, P. What is a knowledge representation? *AI Magazine 1993*, 17-33 Spring (1993), 33.
- [22] DOWNEY, A. *Think Python*, 2nd edition, updated for python 3 ed. O’Reilly Media, Sebastopol, CA, 2016.
- [23] FOWLER, M., AND RICE, D. *Patterns of Enterprise Application Architecture*. A Martin Fowler signature book. Addison-Wesley, 2003.
- [24] GANGEMI, A., GUARINO, N., MASOLO, C., OLTRAMARI, A., AND SCHNEIDER, L. Sweetening ontologies with dolce. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web* (Berlin, Heidelberg, 2002), A. Gómez-Pérez and V. R. Benjamins, Eds., Springer Berlin Heidelberg, pp. 166–181.
- [25] GRUBER, T. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5, 2 (1993), 199–220.
- [26] GUIZZARDI, G. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, 10 2005.
- [27] HARTL, P., AND HARTLOVÁ, H. *Psychologický slovník*, třetí, aktualizované vydání ed. Portál, Praha, 2000.
- [28] ISAAC, A., AND SUMMERS, E. Skos simple knowledge organization system. *Primer, World Wide Web Consortium (W3C)* (2009), 44.
- [29] JATERKOVÁ, L. Využití kompetenčního přístupu při řízení podniku, 2006.
- [30] JIRŮTKA, J. Usermap-api. <https://rozvoj.fit.cvut.cz/Main/usermap-api>, 2014. Navštíveno: 2019-04-18.

- [31] JIRŮTKA, J. Usermap-db. <https://rozvoj.fit.cvut.cz/Main/usermap-db>, 2014. Navštíveno: 2019-04-18.
- [32] KIRYAKOV, A., POPOV, B., TERZIEV, I., MANOV, D., AND OGNANOFF, D. Semantic annotation, indexing, and retrieval. *Journal of Web Semantics* 2, 1 (2004), 49 – 79.
- [33] KLIMEŠ, L. *Slovník cizích slov*, 7. vyd., v spn vyd. 2., rozš. a dopl ed. SPN - pedagogické nakladatelství, Praha, 2005.
- [34] KUCHAR, R. Vyhledávání zaměstnanců fel Čvut, 2016.
- [35] KŘEMEN, P. Deskripční logika. Dostupné z: [https://cw.fel.cvut.cz/old/\\_media/courses/a4m33rzn/p1-uvod-dl-print.pdf](https://cw.fel.cvut.cz/old/_media/courses/a4m33rzn/p1-uvod-dl-print.pdf), 2011.
- [36] LEDVINKA., M., AND KŘEMEN., P. Jopa: Accessing ontologies in an object-oriented way. In *Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 2: ICEIS*, (2015), INSTICC, SciTePress, pp. 212–221.
- [37] MANU SPORNY, DAVE LONGLEY GREGG KELLOGG, M. L. N. L. Json-ld 1.0. <https://www.w3.org/TR/json-ld>. Navštíveno: 2018-12-26.
- [38] MARTIN, R. C. The clean architecture. <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>, 2012. Navštíveno: 2019-04-27.
- [39] MILES, A., AND BECHHOFFER, S. Skos simple knowledge organization system. <https://www.w3.org/TR/skos-reference/>. Navštíveno: 2018-12-24.
- [40] MITITELU, V. B., AND BARBU, E. From dictionaries to language representation formalisms. *Revue Roumaine de Linguistique*, 1-2 (2007 (LII)), 19.
- [41] MORSEY, M., LEHMANN, J., AUER, S., STADLER, C., AND HELLMANN, S. Dbpedia and the live extraction of structured data from wikipedia. *Program: electronic library and information systems* 46 (04 2012), 157–181.
- [42] NILES, I., AND PEASE, A. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001* (New York, NY, USA, 2001), FOIS '01, ACM, pp. 2–9.
- [43] PALERMO, J. Onion architecture. <https://jeffreypalermo.com/tag/onion-architecture/>, 2008. Navštíveno: 2019-04-26.
- [44] PATEL, A., AND JAIN, S. Formalisms of representing knowledge. *Procedia Computer Science* 125 (2018), 542 – 549. The 6th International Conference on Smart Computing and Communications.
- [45] PRUD'HOMMEAUX, E., AND SEABORNE, A. Sparql query language for rdf. <https://www.w3.org/TR/rdf-sparql-query>. Navštíveno: 2018-12-23.

- [46] RADA PRO VÝZKUM, V. A. I. Zpráva o činnosti rady pro výzkum, vývoj a inovace za rok 2018. <https://www.vyzkum.cz/FrontClanek.aspx?idsekce=860250>, 2018. Navštíveno: 2019-04-19.
- [47] RAMIREZ, C., AND VALDES, B. A general knowledge representation model for the acquisition of skills and concepts. In *2009 8th IEEE International Conference on Cognitive Informatics* (June 2009), pp. 412–417.
- [48] REBELE, T., SUCHANEK, F. M., HOFFART, J., BIEGA, J., KUZEY, E., AND WEIKUM, G. YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II* (2016), pp. 177–185.
- [49] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.
- [50] SPEER, R., CHIN, J., AND HAVASI, C. Conceptnet 5.5: An open multilingual graph of general knowledge. In *AAAI* (2017).
- [51] STEPHAN, G. PASCAL, H. AND ANDREAS, A.  
*Semantic Web Services: Concepts, Technologies, and Applications*. Springer, Berlin, 2007.
- [52] TIM BRAY, JEAN PAOLI, C. M. S.-M. E. M. F. Y. Extensible markup language. <https://www.w3.org/TR/xml/>. Navštíveno: 2018-12-24.
- [53] VAN BERGEN, P. Ports-and-adapters / hexagonal architecture. [http://www.dossier-andreas.net/software\\_architecture/ports\\_and\\_adapters.html](http://www.dossier-andreas.net/software_architecture/ports_and_adapters.html), 2012. Navštíveno: 2019-04-27.
- [54] WEGNER, P. Concepts and paradigms of object-oriented programming. *SIGPLAN OOPS Mess.* 1, 1 (Aug. 1990), 7–87.





## Slovník

API	Rozhraní pro programování aplikací (angl. Application Programming Interface), v našem kontextu spíše webová služba nebo rozhraní pro poskytování dat
CRUD	Operace vytvoření, čtení, změna a vymazání (angl. create, read, update a delete)
CSS	Kaskádové styly (angl. Cascading Style Sheets)
FEL ČVUT	Fakulta elektrotechnická Českého vysokého učení technického v Praze
FIT ČVUT	Fakulta informačních technologií Českého vysokého učení technického v Praze
FOL	Logika prvního řádu (angl. First order logic)
GDPR	Obecné nařízení o ochraně osobních údajů (angl. General Data Protection Regulation)
HTML	Hypertextový značkový jazyk (angl. Hypertext Markup Language)
HTTP	Hypertext Transfer Protokol
IdM	Správa identit (angl. Identity Management)
JSON-LD	Javascriptová objektová notace pro propojená data (angl. JavaScript Object Notation for Linked Data)
OOM	Ontologicko-relační mapování (angl. Object-ontological mapping)
ORM	Objektově-relační mapování (angl. Object-relational mapping)
OWL	Webový ontologický jazyk (angl. Web Ontology Language)
RDF	Framework pro popis zdrojů (angl. Resource Description Framework)
REST	Representational State Transfer (bez překladu)
SKOS	Systém pro organizaci znalostí (angl. Simple Knowledge Organization System)

SPARQL	Protokol a RDF dotazovací jazyk (angl. Protocol and RDF Query Language)
SPI	Rozhraní poskytovatele služby (angl. Service Provider Interface)
UFO	Unifikovaná základní ontologie (angl. Unified Foundational Ontology)
UML	Unifikovaný modelovací jazyk (angl. Unified Modelling Language)
URI	Uniformní identifikátor zdroje (angl. Uniform Resource Identifier)
URL	Jednotná adresa zdroje (angl. Uniform Resource Locator)
XML	Rozšiřitelný značkovací jazyk (angl. Extensible Markup Language)

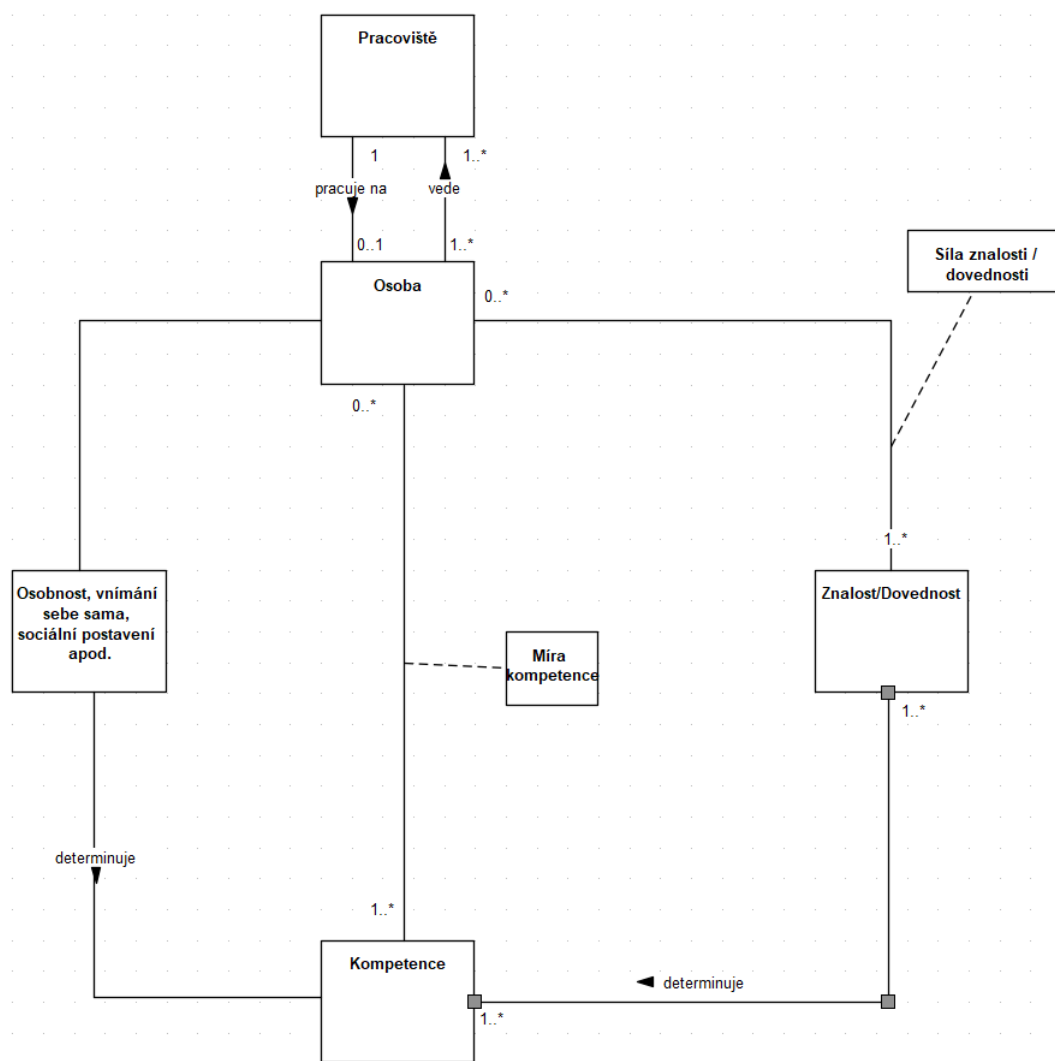


**Přílohy**



## Příloha A

### Konceptuální datový model



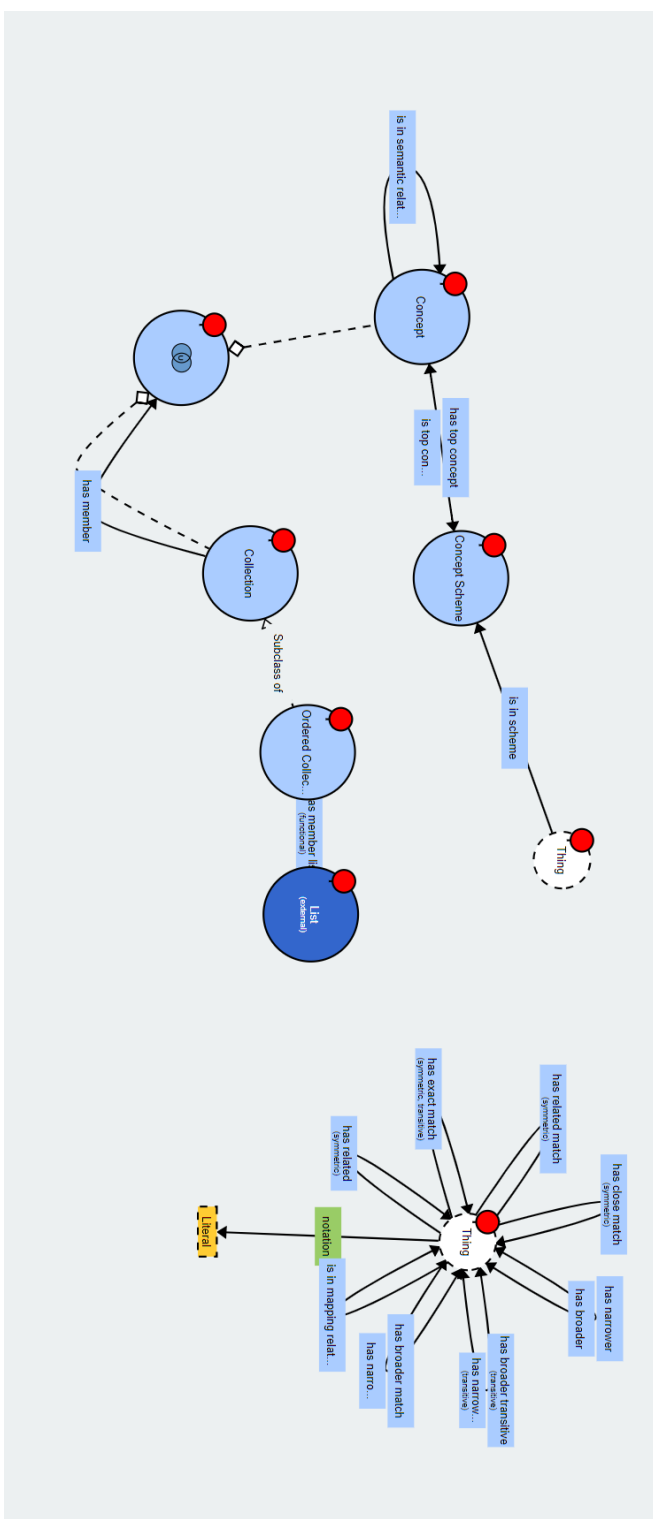
Obrázek A.1: Diagram konceptuálního modelu (zdroj autor)





## **Příloha B**

### **SKOS vizualizace**



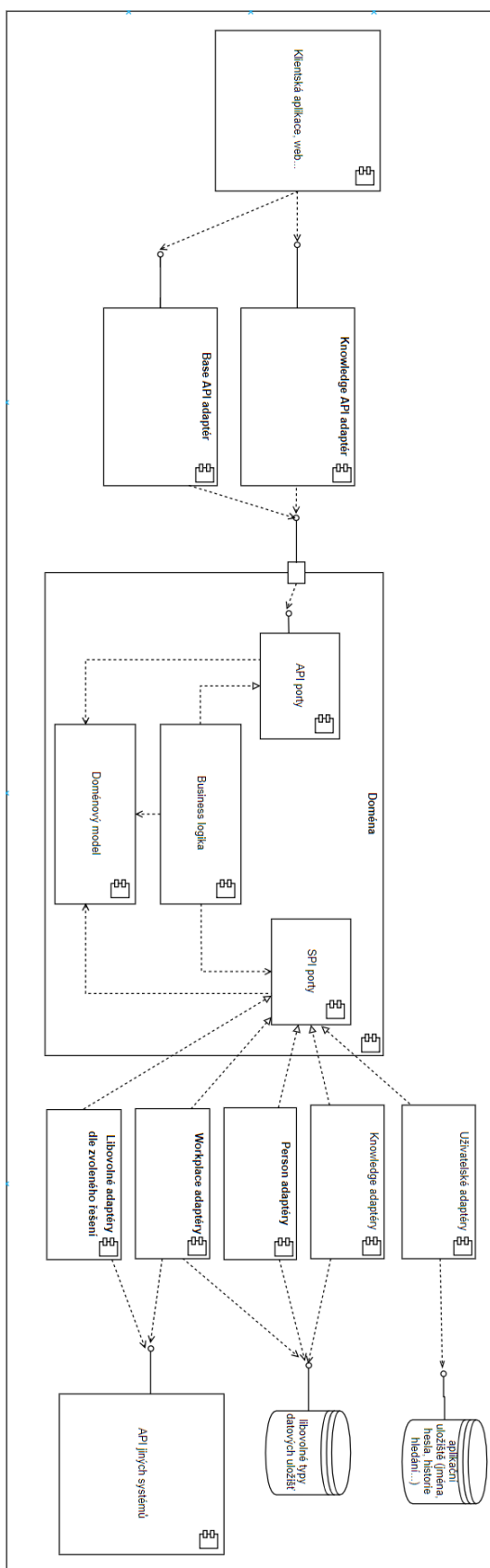
**Obrázek B.1:** Ontologie SKOS (zdroj: <https://www.w3.org/2009/08/skos-reference/skos.rdf> použitý nástroj: <http://www.visualdataweb.de/webvow/>)





**Příloha C**

**Diagram komponent**

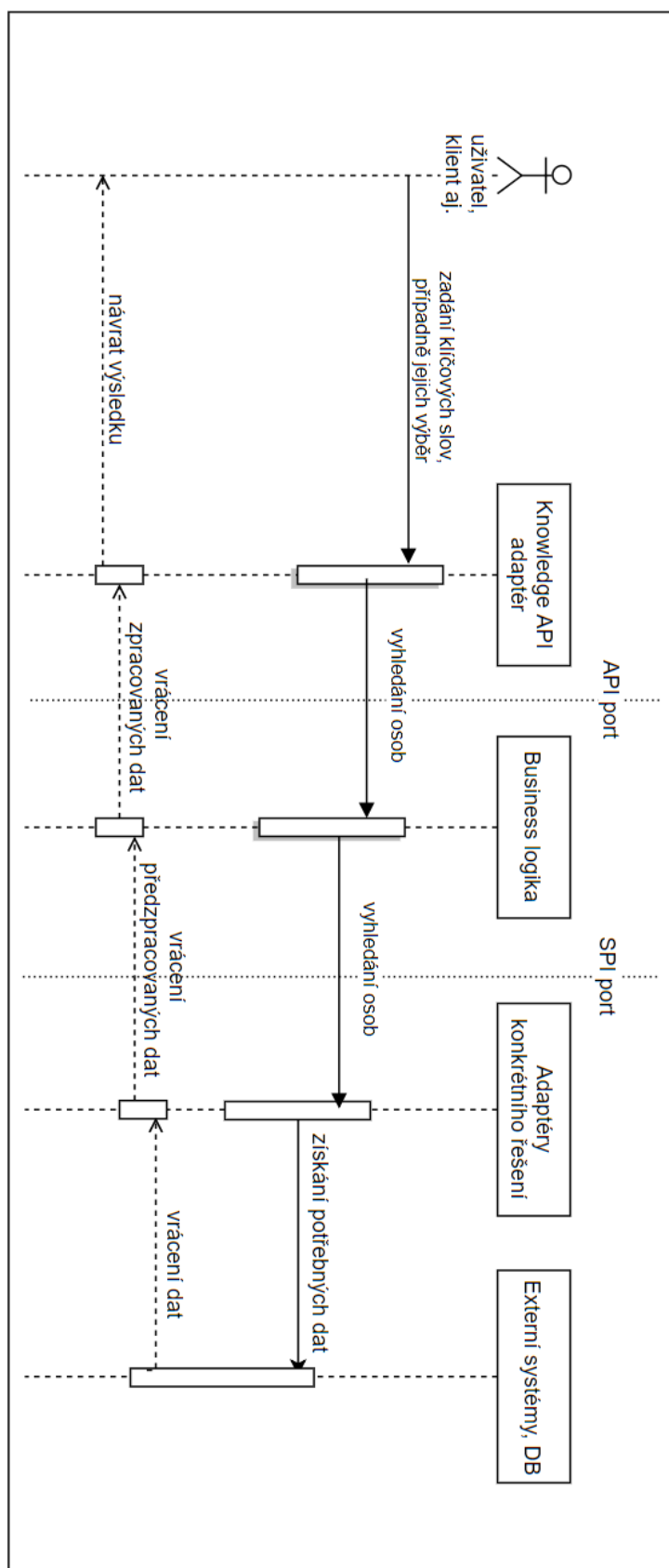


Obrázek C.1: Diagram komponent (zdroj autor)

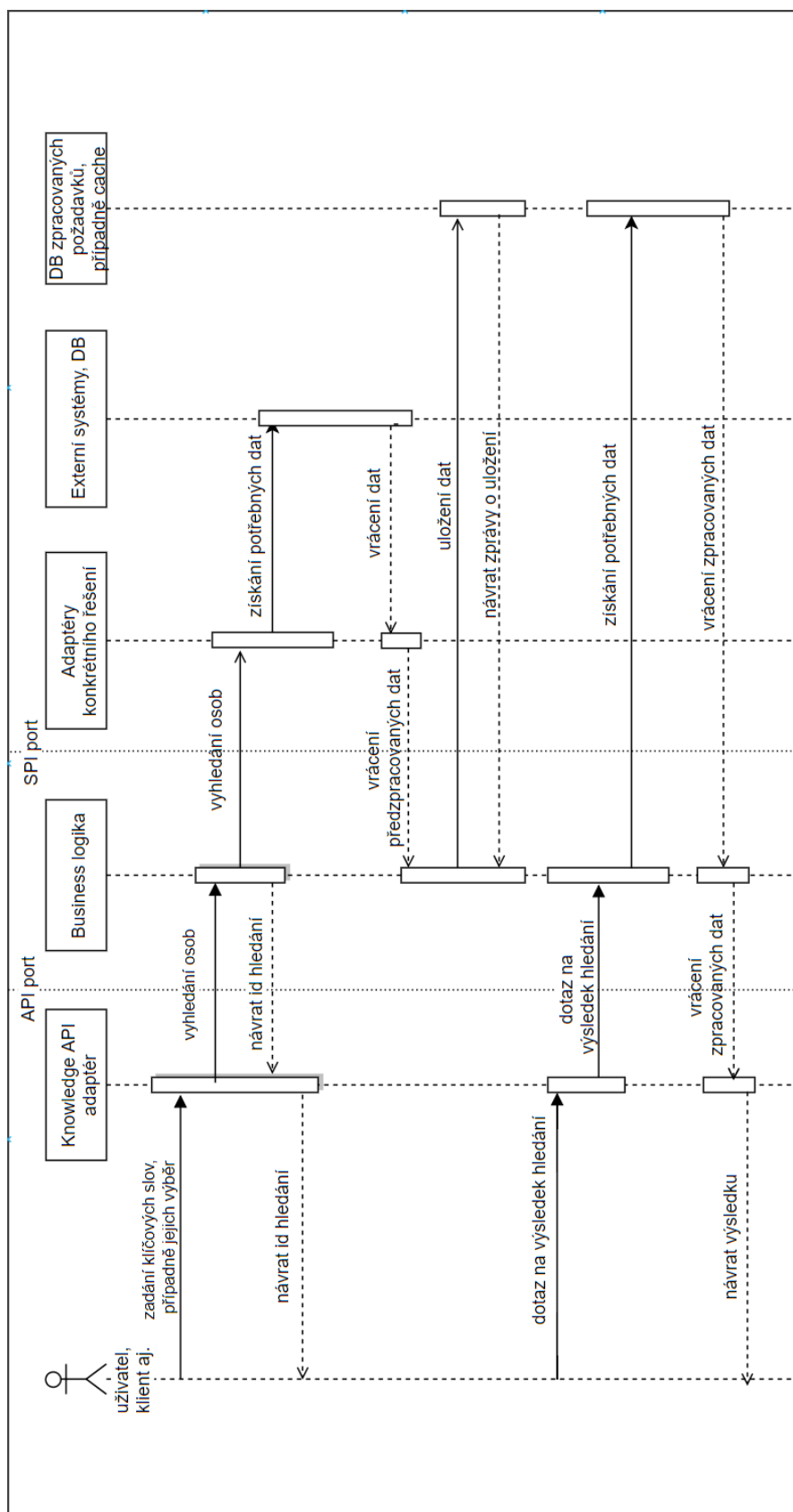


**Příloha D**

**Sekvenční diagram**



**Obrázek D.1:** Synchronní verze sekvenčního diagramu (zdroj autor)

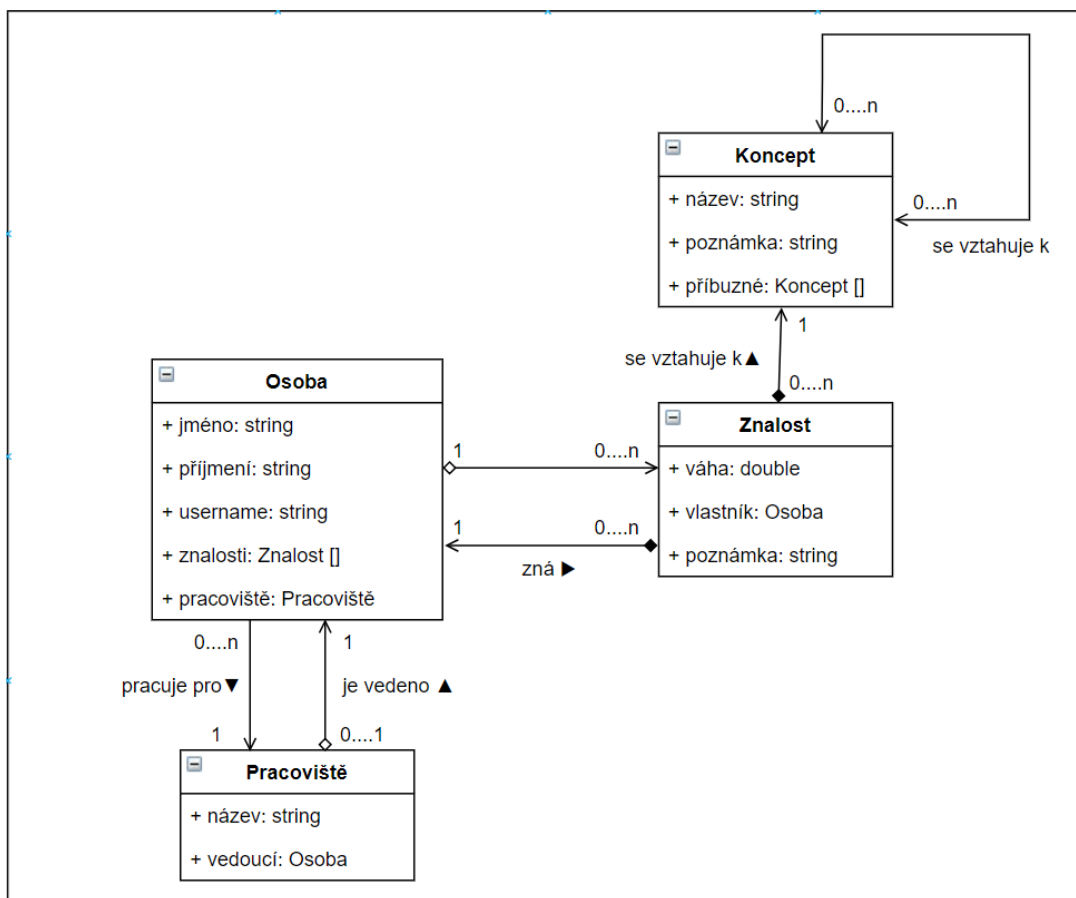


Obrázek D.2: Asynchronní verze sekvenčního diagramu (zdroj autor)



## Příloha E

### Doménový model



Obrázek E.1: Doménový model (zdroj autor)

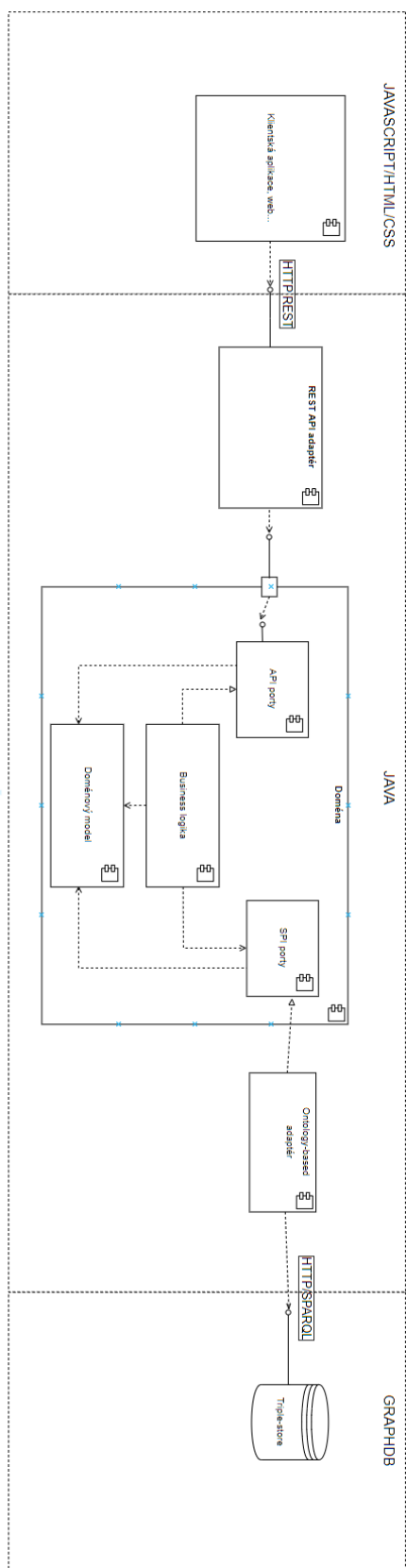






## **Příloha F**

### **Diagram komponent pro naše řešení FEL ČVUT**

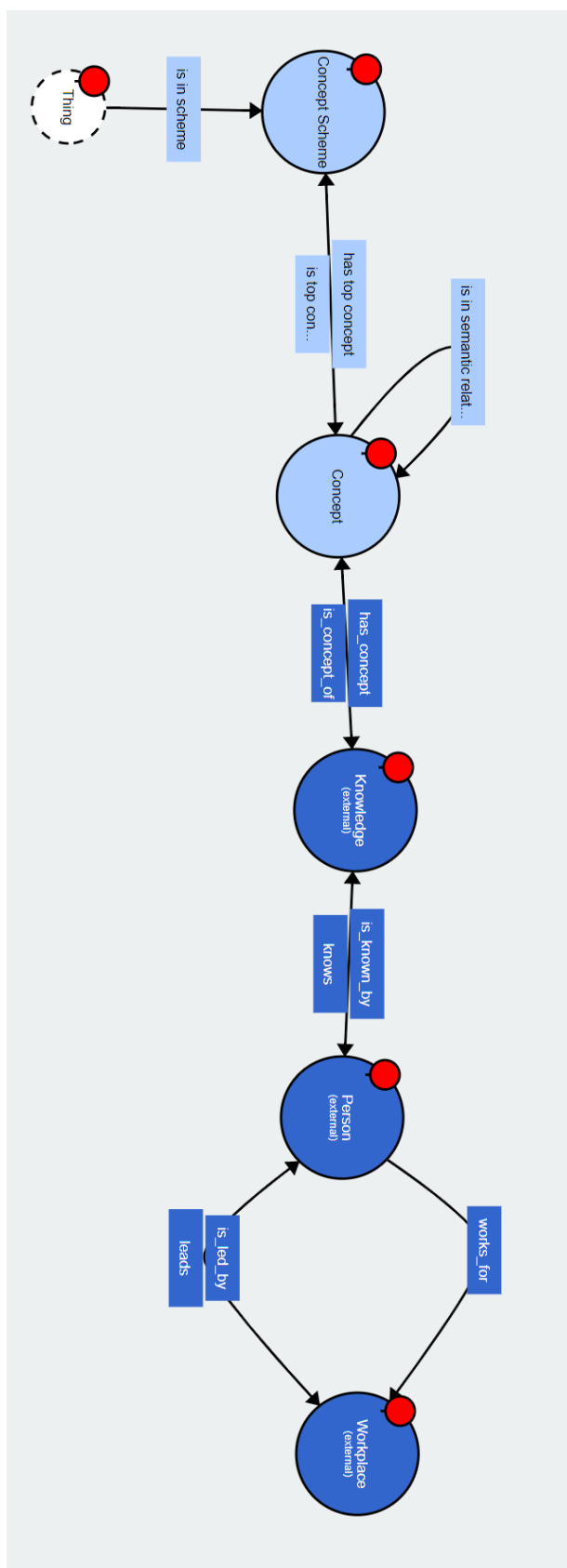


Obrázek F.1: Diagram komponent pro naše řešení (zdroj: autor)



## **Příloha G**

### **Ontologické schéma**



**Obrázek G.1:** Použité ontologické schéma (zdroj autor, použitý nástroj <http://www.visualdataweb.de/webvowl/>)

## Příloha H

### SPARQL dotaz - vyhledávání osob dle kompetencí

```
select ?person ?name ?surname ?username ?weight ?sub
?sub2 ?sub3 ?sub4 where {
  {
    ?person ?knows          ?k;
             ?hasName       ?name;
             ?hasUsername   ?username;
             ?hasSurname    ?surname .
    ?k      ?hasWeight      ?weight;
             ?hasConcept    ?concept .
  }
  UNION
  {
    ?person ?knows          ?k;
             ?hasName       ?name;
             ?hasUsername   ?username;
             ?hasSurname    ?surname .
    ?k      ?hasWeight      ?weight;
             ?hasConcept    ?sub1 .
    ?sub1   ?semanticRelation ?concept .
  }
  UNION
  {
    ?person ?knows          ?k;
             ?hasName       ?name;
             ?hasUsername   ?username;
             ?hasSurname    ?surname .
    ?k      ?hasWeight      ?weight;
             ?hasConcept    ?sub1 .
    ?sub1   ?semanticRelation ?sub2 .
    ?sub2   ?semanticRelation ?concept .
  }
}
```

```

        ?person ?knows      ?k;
                ?hasName    ?name;
                ?hasUsername ?username;
                ?hasSurname ?surname .
    ?k      ?hasWeight ?weight;
           ?hasConcept ?sub1 .
    ?sub1   ?semanticRelation ?sub2 .
    ?sub2   ?semanticRelation ?sub3 .
    ?sub3   ?semanticRelation ?concept .
    }
    UNION
    {
        ?person ?knows      ?k;
                ?hasName    ?name;
                ?hasUsername ?username;
                ?hasSurname ?surname .
    ?k      ?hasWeight ?weight;
           ?hasConcept ?sub1 .
    ?sub1   ?semanticRelation ?sub2 .
    ?sub2   ?semanticRelation ?sub3 .
    ?sub3   ?semanticRelation ?sub4 .
    ?sub4   ?semanticRelation ?concept .
    }
}

```

**Listing H.1:** SPARQL dotaz pro vyhledávání osob dle jejich kompetencí

## Příloha I

### Výstup systému po vyhledání dle kompetencí

```
[
  {
    "person":{
      "name":"Tom",
      "surname":"Sobotka",
      "username":"sobotat",
      "id":"http://onto.fel.cvut.cz/koubadom/individuals/persons
      #Tomas_Sobotka"
    },
    "metric":0.5,
    "info":"Source: GraphDB Info: 0 intermediate concepts"
  },
  {
    "person":{
      "name":"Ji",
      "surname":"k",
      "username":"zakji",
      "id":"http://onto.fel.cvut.cz/koubadom/individuals/
      persons#Jiri_Zak"
    },
    "metric":0.3,
    "info":"Source: GraphDB Info: 2 intermediate concepts"
  },
  {
    "person":{
      "name":"Michal",
      "surname":"Sblk",
      "username":"sablimi",
      "id":"http://onto.fel.cvut.cz/koubadom/individuals/persons
      #Michal_Sablik"
    },
    "metric":0.2,
    "info":"Source: GraphDB Info: 3 intermediate concepts"
  },
]
```

```
{
  "person":{
    "name":"Pavel",
    "surname":"Zpravda",
    "username":"zaprapa",
    "id":"http://onto.fel.cvut.cz/koubadom/individuals/persons
    #Pavel_Zpravda"
  },
  "metric":0.18,
  "info":"Source: GraphDB Info: 4 intermediate concepts"
}
]
```

**Listing I.1:** Výstup systému při dotazu na vyhledání dle kompetencí



## Příloha J

### Seznam použitých technologií, nástrojů a knihoven

#### Serverová část:

- **Java 8** a její prostředí, SDK v1.8.0\_92 (<https://www.java.com/en/>)
- **Spring Boot** a přidružené knihovny (<https://spring.io/projects/spring-boot/>)
- **JOPA** (<https://github.com/kbss-cvut/jopa>)
- **Maven** (<https://maven.apache.org/>)
- **GraphDB Free** (<http://graphdb.ontotext.com/>)

#### Klientská část:

- **Javascript** (implementaci následujícího <https://www.ecma-international.org/ecma-262/5.1/> )
- **npm** (<https://www.npmjs.com/>)
- **Webpack** a přidružené knihovny (<https://webpack.js.org/>)
- **HTML5** (<https://www.w3.org/TR/html52/>)
- **CSS3** (<https://www.w3.org/TR/css-2018/>)

#### Nástroje:

- **IntelliJ IDEA** (<https://www.jetbrains.com/idea/>)
- **Webstorm** (<https://www.jetbrains.com/webstorm/>)
- **Enterprise architect** (studentská verze) (<https://sparxsystems.com/>)
- **Draw.io** (<https://www.draw.io/>)
- **Prohlížeč Mozilla Firefox** (<https://www.mozilla.org/en-US/>)
- **Prohlížeč Google Chrome** (<https://www.google.com/chrome/>)
- **Protégé** (<https://protege.stanford.edu/>)

**Inspirace:**

- **Reporting tool** (<https://github.com/kbss-cvut/reporting-tool/>)
- **Spring-hexagonal-example**  
(<https://github.com/gshaw-pivotal/spring-hexagonal-example>)



## Příloha K

### Obsah přiloženého CD

- `prototype.zip` - zdrojové soubory prototypu aplikace
- `db-scheme.zip` - databázové schéma ve formátu turtle
- `test-dataset.zip` - testovací dataset ve formátu turtle
- `text-source.zip` - zdrojové soubory textu
- `thesis.pdf` - elektronická verze tohoto textu