VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DETEKCE VIZUÁLNÍCH VZORŮ VE WEBOVÝCH STRÁNKÁCH

VISUAL PATTERN DETECTION IN WEB PAGES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN KOTRAŠ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Kotraš Martin, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze
Název: **Detekce vizuálních vzorů ve webových stránkách**
Visual Pattern Detection in Web Pages
Kategorie: Web
Zadání:

1. Seznamte se s experimentálním nástrojem FitLayout, jeho aplikačním rozhraním a způsobem reprezentace dokumentů.
2. Prostudujte současné přístupy pro extrakci dat z webových dokumentů na základě vyhledávání vizuálních vzorů.
3. Zvolte vhodnou metodu extrakce a navrhněte architekturu aplikace pro extrakci dat z webových stránek s využitím detekce vizuálních vzorů. Svá rozhodnutí konzultujte s vedoucím.
4. Implementujte navrženou aplikaci pomocí vhodných technologií.
5. Proveďte vyhodnocení funkčnosti vytvořené aplikace na vhodné sadě webových dokumentů.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Alarte, J.; Insa, D.; Silva, J.; et al.: Main Content Extraction from Heterogeneous Webpages. In Web Information Systems Engineering - WISE 2018. Cham: Springer International Publishing. 2018. ISBN 978-3-030-02922-7. pp. 393-407.
- Burget, R.: Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: *15th International Conference on Web Information Systems and Technologies*. Vienna: SciTePress - Science and Technology Publications, 2019, s. 326-333. ISBN 978-989-758-386-5
- Rámec FitLayout https://github.com/FitLayout

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 11. října 2021

Abstrakt

Práce řeší extrakci informací z webových stránek pomocí techniky vyhledávání vizuálních vzorů – prostorových vztahů mezi oblastmi na webové stránce a stejných vizuálních stylů těchto oblastí – s rozšířením o nové techniky zlepšení výsledků. Využívá přitom uživatelem zadaného ontologického modelu dat, který popisuje, které datové položky se budou ze zadané webové stránky extrahovat a jak jednotlivé položky na stránce vypadají zejména z textového pohledu.

V rámci práce vznikla konzolová aplikace VizGet v jazyce Java využívající aplikační rámec FitLayout pro získání vizuálního modelu webové stránky. Testování aplikace na 7 různých doménách zahrnujících mj. žebříček nejlepších filmů, produktů v elektronickém obchodě nebo předpovědi počasí ukázalo, že se úspěšnost aplikace pohybuje ve zhruba 75 % dílčích testů nad 85 % F-skóre a ve více než 90 % testů nad 60 % F-skóre, kde 45 % testů dosahuje F-skóre 100 %. Aplikace VizGet tak může být nasazena pro praktické využití v nekritických aplikacích, přičemž je otevřena dalším rozšířením a možnostem zlepšení.

Abstract

The work solves the extraction of information from websites using the technique of searching for visual patterns – spatial relations between areas on the website and the same visual styles of these areas – with the extension of new techniques to improve results. It uses a user-specified ontological data model, which describes which data items will be extracted from the specified web page and how the individual items on the page look, mainly from a text point of view.

As part of the work, a console application VizGet in Java was created using the FitLayout framework to obtain a visual model of the website. Testing the application on 7 different domains, including a list of the best movies, e-shop products, or weather forecasts, showed that the success rate of the application ranges in about 75 % of subtests above 85 % F-score and in more than 90 % of subtests above 60 % F-score, where 45 % of subtests achieve an F-score of 100 %. The VizGet application can thus be deployed for practical use in non-critical applications, while it is open to further extensions and possibilities for improvement.

Klíčová slova

extrakce informací, extraktor, vizuální vzory, webové stránky, VizGet, FitLayout

Keywords

information extraction, extractor, visual patterns, web pages, VizGet, FitLayout

Citace

KOTRAŠ, Martin. *Detekce vizuálních vzorů ve webových stránkách.* Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Detekce vizuálních vzorů ve webových stránkách

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Kotraš
3. května 2022

Poděkování

Rád bych poděkoval svému vedoucímu práce doc. Ing. Radku Burgetovi Ph.D. za pravidelné konzultace, pomoc a cenné rady, které mi poskytl.

Obsah

Kapitola 1

Úvod

Webové stránky zpřístupněné prostřednictvím sítě internet jsou v 21. století neustále se rozšiřující zdroj informací, ke kterému má celosvětově přístup čím dál tím více lidí. Ať už se jedná o různé zpravodajské servery, internetové encyklopedie pro všeobecná i specializovaná témata, internetové blogy, diskuze, databáze hudby, filmů, seriálů, her, videí a spousty dalšího jistě užitečného obsahu, jedno má většina takového obsahu společné – je vytvářena pro zpracování lidským čtenářem. Získat z takového množství obsahu automatizovaně a pokud možno i relativně jednoduše relevantní informace, které uživatel požaduje, je nelehký, ale přesto potřebný či jinak užitečný úkol.

Metod, jak tyto relevantní informace z webových stránek získat, dnes existuje celá řada, ale doposud jen málokterá využívá pro svou činnost vizuální zpracování webové stránky, tedy způsob, jakým se na webovou stránku dívá člověk. Přitom právě tak bývá většina webových stránek navržena. Práce si tak klade za cíl pomocí vhodně zvolené metody založené právě na vizuálním zpracování webové stránky vytvořit v praxi použitelnou aplikaci. Ta musí umět získat informace z webové stránky na základě předem specifikovaného modelu dat, který je určitým popisem, jak data na webové stránce vypadají.

Kapitola 2 shrnuje současné přístupy k extrakci informací z webu, kategorizuje je do několika skupin a představuje využitý vizuální model. Přístup k extrakci založený na vizuálních vzorech, na kterém zakládá tato práce, představuje kapitola 3. Základní rozbor aplikačního rámce FitLayout, který je specializován na vizuální extrakci a který je následně použit v práci, představuje kapitola 4.

Z teoretických poznatků předcházejících kapitol je navržena aplikace VizGet, přičemž tento vysokoúrovňový návrh je popsán v kapitole 5. Z návrhu vychází implementace, která začíná popisem zpracování vstupu v kapitole 6 , následuje kapitolou 7 popisující získání, segmentaci a vůbec celkově přípravu před hledáním vizuálních vzorů. Hlavní jádro algoritmů použitých pro vyhledávání vizuálních vzorů poté popisuje kapitola 8 a implementační část uzavírá kapitola 9 věnující se generování výstupu aplikace na základě vzorů. Nakonec je aplikace otestována na sadě webových stránek v kapitole 10 a dosažené výsledky a výstupy této práce jsou shrnuty v kapitole 11.

Kapitola 2

Současné přístupy k extrakci informací z webu

Způsobů, jak v současné době k extrakci informací z webových stránek přistupovat, existuje dnes již celá řada. Podle různých úhlů pohledu, jak jednotlivé metody extrakce pracují, lze současné přístupy zhruba rozdělit do následujících kategorií:

- **podle specializace extraktoru** zpracovávaných webových stránek,
- **podle použitého modelu**, jakým je na vstupní webovou stránku nahlíženo,
- **podle směru zpracování** zdrojového kódu webových stránek.

Následující sekce uvádějí k daným rozdělením již konkrétnější metody extrakce informací z webových stránek a tyto metody dále rozebírají a přibližují pro pozdější analýzu řešeného problému této práce.

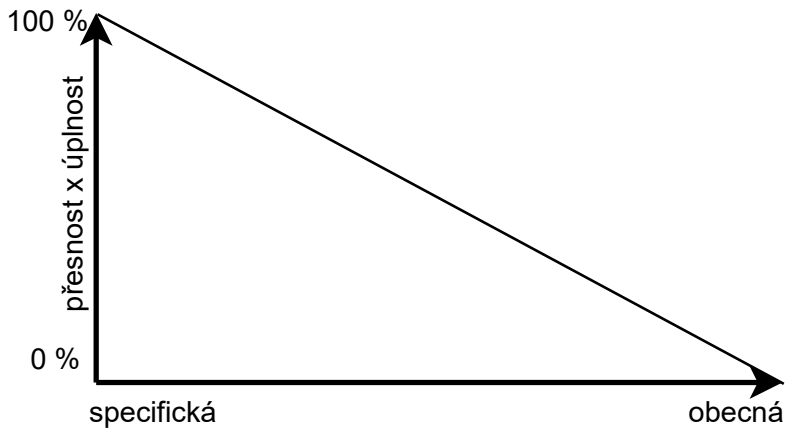
2.1 Rozdělení podle specializace extraktoru

Podle specializace extraktoru informací, a tedy i množství informací, které má extraktor o zdrojových webových stránkách předem k dispozici, můžeme současné přístupy k extrakci informací z webových stránek dále rozdělit na: [14]

- **obecnou extrakci** zpracovávající webové stránky s předem neznámou strukturou a
- **specifickou extrakci** zpracovávající webové stránky s předem známou strukturou.

2.1.1 Obecná extrakce informací

Obecná extrakce informací spočívá ve zpracování často velkého množství rozdílných webových stránek, [14] přičemž je předem známé, co za informace je potřeba z webových stránek extrahovat, avšak struktura webových stránek známa předem není nebo není do extraktoru nijak zanesena. Už ze své podstaty tak tento způsob často dosahuje méně přesných či úplných výsledků než extrakce specifická pro daný typ webové stránky, na druhou stranu ale dokáže zpracovávat pestrou škálu rozdílných webových stránek, kterých je web plný. Provázanost a rozdílnost mezi obecnou a specifickou extrakcí ilustruje obrázek 2.1.



Obrázek 2.1: Ilustrace závislosti specifické a obecné extrakce informací na přesnosti a úplnosti extrahovaných informací: čím více obecný extraktor je, tím méně přesné a úplné výsledky bude pravděpodobně podávat.

Příkladem využití obecné extrakce může být získání a možná agregace informací o předem známé doméně, například o produktech prodávaných v rozdílných elektronických obchodech. Přestože bude struktura webových stránek jednotlivých elektronických obchodů pravděpodobně rozdílná, lze se díky známé řešené doméně snažit na jednotlivých stránkách hledat požadované informace. Takový způsob řešení lze pojmenovat jako **extrakci řízenou modelem dat** i přes fakt, že nemusí být použití modelu často na první pohled zřejmé a model může být v takovém případě už přímou součástí kódu.

2.1.2 Specifická extrakce informací

Specifická extrakce informací je na rozdíl od obecné přizpůsobena konkrétní zpracovávané webové stránce a zpracovává tedy webové stránky s předem známou strukturou. Díky znalosti struktury je tak možné tento způsob využít pro zacílení přesných míst, kde se požadované informace nacházejí. To má za následek velmi vysokou až naprosto dokonalou přesnost a úplnost získávaných informací.

Cenou za takto přesné a úplné výsledky je poměrně značná dodatečné práce, která musí být vynaložena na analýzu dané webové stránky a zaměření míst, ze kterých se mají informace extrahovat. Navíc je tato metoda náchylná na změny v designu webové stránky a v případě jeho kompletní nebo i dostatečně velké částečné změny typicky přestane fungovat. To jen dále zvyšuje nákladnost této metody, zejména co se údržby týče. [14]

V praxi se jedná zřejmě o nejrozšířenější metodu extrakce informací z webových stránek díky své jednoduchosti na pochopení, ve výsledku i jednoduché implementaci a zejména již zmíněné přesnosti a úplnosti získávaných informací. Protože je ale nutné pro každou rozdílnou webovou stránku potřeba vytvořit zvláštní strategii zpracování, lze tento způsob efektivně provozovat pouze na omezeném množství vzájemně rozdílných webových stránek.

2.2 Rozdělení podle použitého modelu

V závislosti na použitém modelu zpracovávané webové stránky a tedy způsobu, jakým je na ni nahlíženo, lze podle [4] modely využívané při extrakci rozdělit na:

- **řetězec znaků** zpracovávající vstupní webovou stránku po jednotlivých znacích,
- **řetězec symbolů**[1] využívající lexikální analýzu k rozpoznání jednotlivých symbolů,
- **hierarchické modely**, z nichž je nejčastěji využíván **Document Object Model** (zkráceně DOM) a
- **vizuální modely** zejména pro potřeby vizuálních zpracování např. pomocí **Box Modelu**.

Výše jmenované modely prezentují dnes nejčastější způsoby náhledu na webové stránky, přičemž vizuální modely představují nejnovější způsob, ve kterém probíhá aktivní výzkum. Jednotlivé modely jsou pak dále podrobněji popsány v následujících sekcích.

2.2.1 Řetězec znaků

Model ve formě řetězce znaků je jedním z nejjednodušších modelů, který lze pro extrakci informací z webových stránek použít. Nesnaží se vstupní webovou stránku zpracovávat řádnou syntaktickou analýzou, která by mohla zajistit lepší porozumění a následné zpracování webové stránky, ale zpracovává ji po jednotlivých znacích. Pro takové zpracování je pak možné využít algoritmy pro vyhledávání podřetězců, ad hoc vytvořené řetězcové algoritmy, regulární výrazy nebo HLRT wrappery fungující na principu detekce prefixu a sufixu datového bloku a levého a pravého oddělovače jednotlivých datových polích [4].

Výhodou tohoto modelu je jeho velmi jednoduchá implementace, kdy stačí zvoleným algoritmem zpracovávat proud znaků putující od použitého HTTP klienta, případně znaky zpracovávat po stažení do operační paměti. S tím souvisí také výborná rychlost a škálovatelnost takového řešení. Cenou za tento jednoduchý přístup pak může být složité odstraňování nepotřebných HTML značek nebo využití hierarchické struktury zpracovávané webové stránky.

Příklad jednoduché v praxi použitelné extrakce z webové stránky s využitím modelu řetězce znaků ukazuje obrázek 2.2.

```
1 wget -qO - https://www.fit.vut.cz/ |
2   grep -Po "(?<=c-media__link-title\">)[^<]+"
```

Obrázek 2.2: Ukázka extrakce titulků aktuálních akcí na Fakultě informačních technologií VUT v Brně v linuxovém interpretu příkazů. Příkaz vypíše titulky oddělené novým řádkem.

2.2.2 Řetězec symbolů

Na vyšší úrovni než model ve formě řetězce znaků, pracuje model založený na zpracovávání řetězce či posloupnosti symbolů. Ten využívá pro zpracování vstupní webové stránky lexikální analyzátor nebo v případě mírně pokročilejšího zpracování analyzátor syntaktický pracující s proudem dat. Znaky vstupní webové stránky už v takovém případě nejsou zpracovávány jednotlivě, ale shlukují se do posloupností symbolů, které mají ve vstupní webové stránce příslušný význam dle HTML specifikace. Příkladem symbolů může být začátek a konec značky, název značky, data, resp. textový řetězec, název atributu a podobně.

[1] Symbol je také znám z angličtiny pod názvem *token*.

Při použití vhodného lexikálního či syntaktického analyzátoru může i tato metoda pracovat s proudem dat, nezatěžovat tak operační paměť a být stále podstatně rychlejší než složitější modely. Nevýhodou může být složitější práce pro programátora, kdy je potřeba si pamatovat informace o kontextu pro správné určení místa, ze kterého je informaci potřeba extrahovat.

Příklad použití lexikálního analyzátoru, který slouží jako základ pro zpracování tímto způsobem, je možné vidět na obrázku 2.3.

```
1  import Lexer from "html-lexer";
2
3  const lexer = new Lexer
4  ({
5    write: ([type, text]) => console.log([type, text]),
6    end: () => null
7  });
8
9  lexer.write("<p>Test</p>");
10 lexer.end();
```

Obrázek 2.3: Ukázka použití lexikálního analyzátoru v jazyce JavaScript. V tomto konkrétním případě je funkce `write` zavolána celkem 8x – jednou pro začátek zpracování, poté `<`, `p`, `>`, `Test`, `</`, `p` a nakonec `>`.

2.2.3 Hierarchické modely

Hierarchické modely už nereprezentují vstupní webovou stránku jako nějaký řetězec, seznam či posloupnost neboli lineární strukturu, ale jednotlivé části či značky vstupní webové stránky jsou uspořádány do stromové struktury formující hierarchii. Nejčastějším využívaným hierarchickým modelem je pro účely extrakce z webových stránek tzv. Document Object Model (zkráceně DOM), případně nějaké jeho zjednodušené varianty. V takové struktuře je pak možné se libovolně procházet od rodičovských uzlů na potomky a naopak nebo vyhledávat například skrze CSS selektory.

Stromová povaha hierarchických modelů a zejména standard pro DOM, který obsahuje spoustu funkcí či metod pro práci s webovou stránkou, zajišťuje programátorovi vysoký komfort při programování extraktoru a nabízí pokročilý způsob práce v podstatě na stejné úrovni, na jaké pracuje i webový prohlížeč. Problémem takového přístupu může být podstatně větší náročnost na výpočetní prostředky, kdy je nutné celý model udržovat v operační paměti a také ho do požadované struktury předzpracovat. V praxi se ale nejedná o zásadní omezení, neboť velikost webových stránek nebývá příliš velká.

Příklad, jak by mohla vypadat jednoduchá extrakce z webové stránky, pokud by byl použit DOM jako model dokumentu, ukazuje obrázek 2.4.

2.2.4 Vizuální modely

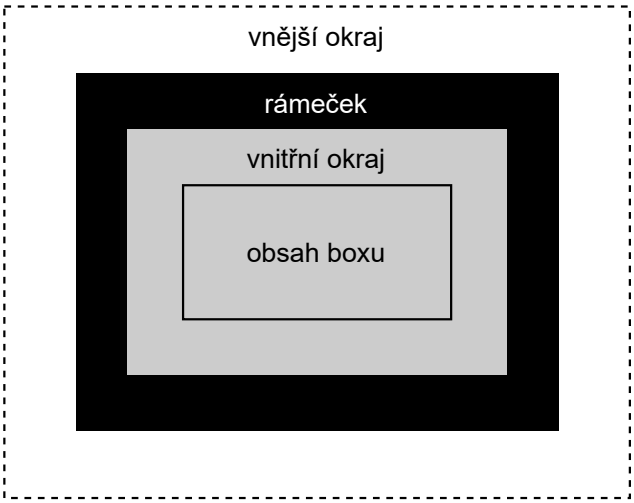
Z jiného úhlu pohledu se na stránku dívají vizuální modely. Ty už nepracují přímo na úrovni zdrojového kódu webové stránky, ale využívají reprezentace vhodnější pro vizuální

```
import cheerio from "cheerio";
import fetch from "node-fetch";

const response = await fetch("https://www.fit.vut.cz/");
const html = await response.text();
const dom = cheerio.load(html);

const news = dom(".c-media__link-title")
  .map((_, el) => dom(el).text().trim())
  .toArray()
  .join("\n");

console.log(news);
```

Obrázek 2.4: Ukázka použití DOM v jazyce JavaScript pro extrakci aktuálních akcí na FIT VUT v Brně podobně jako v sekci 2.2.1.

zpracování. Jednou z takových vhodných reprezentací může být tzv. CSS[2] box model, který pro vizuální vykreslování používají webové prohlížeče.

Reprezentace webové stránky ve formě CSS box modelu, jak je definován v [7], vznikne transformaci stromové struktury vstupní webové stránky na množinu boxů neboli obdélníkových oblastí webové stránky. Boxy v tomto modelu se pak skládají povinně z oblasti obsahu a volitelně z vnitřního okraje (anglicky *padding*), rámečku či ohraničení (anglicky *border*) a vnějšího okraje (anglicky *margin*). Velikost těchto oblastí boxu a celkově velikost boxu včetně jeho pozice potom určují příslušné CSS vlastnosti. Ukázku, jak vypadají jednotlivé oblasti takového boxu, je možné vidět na obrázku 2.5.



Obrázek 2.5: Ilustrace jednotlivých oblastí CSS boxu tak jak je definováno ve standardu. Inspirováno [7].

[2]CSS je zkratka pro kaskádové styly (anglicky *Cascading Style Sheets*).

Výhodou použití CSS box modelu pro vizuální zpracování je jeho blízkost finálnímu vykreslení stránky, a tedy i tomu, co člověk uvidí na obrazovce. Navíc je tento model vstupního dokumentu možné vytvořit a použít i pro jiné typy dokumentů než klasické webové stránky v jazyce HTML. V zásadě jedinou podmínkou je, aby takový dokument podporoval jistou úroveň stylování, jako tomu je například u PDF dokumentů.

Nevýhodou tohoto přístupu je pak z toho vyplývající potřeba použít méně či více pokročilé vykreslovací jádro, které provede transformaci ze vstupní webové stránky či jiného vstupního dokumentu na onu množinu boxů. V ideálním případě může být vykreslovacím jádrem přímo vykreslovací jádro webového prohlížeče, což s sebou přináší zvýšené nároky na systémové prostředky.

2.3 Rozdělení podle směru zpracování

V případě použití hierarchického modelu vstupní webové stránky ve formě stromové struktury, jakou je například DOM nebo i jiné vlastní, potenciálně za běhu budované stromové reprezentace, lze webové stránky zpracovávat podle [2] buď:

- **shora dolů**, což v závislosti na použitém způsobu extrakce přibližně znamená od kořene stromu nebo uzlů na vyšších úrovních postupně až k listům, resp. uzlům na nižších úrovních, nebo naopak
- **zdola nahoru**, což opět v závislosti na použitém způsobu extrakce přibližně znamená od listů stromu nebo uzlů na nižších úrovních postupně až ke kořeni nebo uzlům na vyšší úrovni.

Rozdělení podle směru zpracování není zcela striktní a v závislosti na zpracovávané webové stránce a obecnosti extraktoru mohou být oba přístupy kombinovány.

2.3.1 Zpracování shora dolů

Extrakce shora dolů vychází z kořene stromu v použitém hierarchickém modelu zpracovávané webové stránky a snaží se tak například detekovat větší oblasti, které jsou z hlediska extrakce zajímavé, jako například hlavní obsah článku. Zároveň jsou tak eliminovány oblasti, které tvoří pouze šum a nepřispívají extrakci žádnou zajímavou informací, jako je tomu například u reklamních bloků, pokud přímo ty nejsou cílem extrakce.

Zpracovávání webové stránky shora dolů je také typické při použití DOM, kdy je například nejprve nalezena tabulka, následně její řádky a konečně jednotlivé buňky v řádcích, ze kterých už mohou být extrahovány požadované informace. Zároveň je tento přístup podle [2] jedním z nejčastějších a může tak být úspěšně zkombinován i s přístupem zdola nahoru například pro určitou formu předzpracování.

2.3.2 Zpracování zdola nahoru

Extrakce zdola nahoru vychází z nejmenších nebo téměř nejmenších částí v použitém hierarchickém modelu vstupní webové stránky, tedy často z listů stromové struktury nebo z uzlů, které jsou listům velmi blízko. Z nich je postupně budována často nová stromová struktura, která reprezentuje logické extrahované celky, což mohou být ony celé extrahované entity s jejich atributy.

Zpracování webové stránky zdola nahoru není tak časté jako v případě zpracování shora dolů, ale může přijít vhod právě u metod, které jsou vědomě řízeny nějakým modelem dat.

U nich je totiž úkolem najít mapování mezi atributy v modelu, a právě listy reprezentující ony atributy, což lze úspěšně provést zpracováním zdola nahoru. [2]

Kapitola 3

Extrakce informací založená na vizuálních vzorech

Extrakce informací založená na vyhledání vizuálních vzorů, jak byla popsána v dokumentu [2], a na které je založena tato práce, se snaží nahlížet na webovou stránku podobně, jako to dělá člověk svým lidským zrakem. Nepracuje tedy přímo se zdrojovým kódem, ale využívá vizuální reprezentaci webové stránky ve formě upraveného box modelu. V něm je základní jednotkou box neboli obdélníková oblast na webové stránce, jak již bylo popsáno v rámci sekce 2.2.4.

Tento box model je následně dále zpracován segmentačním algoritmem, který vyprodukuje vizuální oblasti skládající se z jednoho nebo více boxů. Oblasti jsou poté označeny štítky, které odpovídají extrahovaným datovým položkám, a až v takto označených oblastech se začínají hledat vizuální vzory. Nakonec už je nutné pouze nalézt co nejvhodnější mapování mezi označenými oblasti a vstupním ontologickým modelem dat.

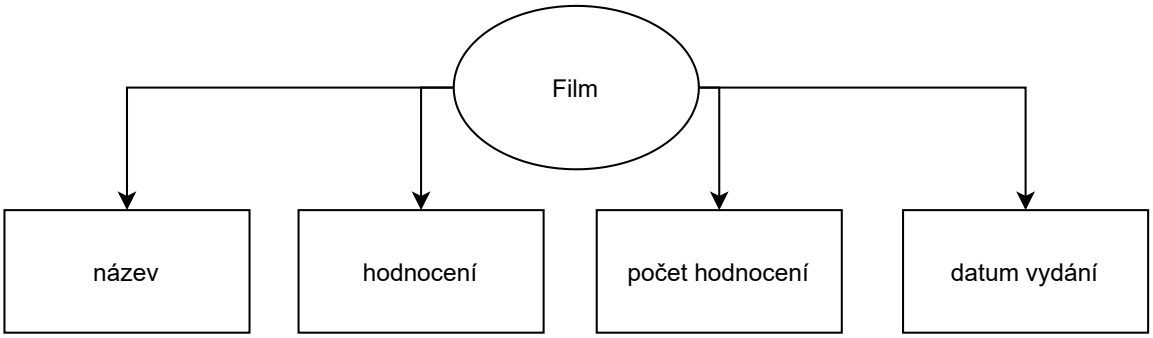
3.1 Doménový model jako vstup

Kromě samostatné zpracovávané vstupní webové stránky, kterou si extraktor interně vhodně reprezentuje, je vstupem extraktoru ještě navíc doménový model, schéma dat nebo ještě lépe ontologický datový model. Ten se ve zjednodušeném pojetí, jak lze odvodit například z dokumentace webového ontologické jazyka OWL [11], skládá zejména z jedinců formujících třídy a z jejich vlastností, přičemž:

- **jedincem**, individuem, objektem či instancí třídy je myšlena konkrétní „věc“ podobně jako v případě objektu v objektově orientovaném programování,
- **třídou**, entitou či konceptem je myšlena skupina jedinců stejného typu a
- **vlastností** nebo atributem je myšlena určitá charakteristika jedince, kde datová vlastnost má jako hodnotou jednoduchý typ (číslo, řetězec, ...) a objektová vlastnost má jako hodnotu jedince, čímž efektivně tvoří vazby mezi nimi.

Zjednodušeně si lze ontologický datový model používaný v extraktoru představit jako ER diagram nebo diagram tříd. Ukázkovou ontologii pro filmovou doménu je možné vidět na obrázku 3.1.

Ontologie se používají zejména v rámci tzv. sémantického webu, což je podle [13] web dat, ve kterém jsou data vzájemně propojena a nabývají tak na významu neboli sémantice.



Obrázek 3.1: Ilustrace jednoduché filmové ontologie s třídou Film a datovými vlastnostmi název, hodnocení, počet hodnocení a datum vydání.

Jejich explicitní použití není u extrakcí nutnost, doménový model může být specifikován i libovolným jiným způsobem, jako například na míru vytvořeným vlastním jazykem pro definici doménové modelu nebo přímým svázáním doménového modelu se zdrojovým kódem. Ačkoliv však může být jejich použití z počátku komplikovanější, při správném návrhu mají ontologie tu výhodu, že jejich výstup může být celkem snadno začleněn po případném očištění dat do již existujících databází znalostí.

3.2 Segmentace webové stránky na oblasti

Boxy, které byly získány v rámci vykreslování vstupní webové stránky, mohou tvořit i opravdu malé části stránky. Jednat se může například o jediné slovo nebo i písmeno, které bylo z různých důvodů zvýrazněno HTML značkou `<strong>`. Proto se ještě obvykle před samotnou detekcí vizuálních vzorů uplatňuje proces zvaný segmentace webové stránky.

Úkolem procesu segmentace je rozdělit webovou stránku na oblasti přibližně podobně, jako by tyto oblasti vnímal člověk. Oblast se tedy ve výsledku může skládat z jednoho nebo více boxů v závislosti na jejich detekci a slouží jako základ pro další zpracování. Boxy se navíc mohou nacházet pouze v jedné z oblastí, tedy průnik libovolných dvou množin s boxy musí být prázdný.

Výstupem segmentačních metod tedy většinou bývá jednoduchý jednorozměrný seznam detekovaných oblastí, jako je tomu například u metody *Box Clustering Segmentation*. Například metoda *Vision based Page Segmentation* nebo i další metody ale mohou mít oblasti uspořádány do stromové struktury a reprezentovat tak určitou vizuální hierarchii, tedy že jedna oblast je podoblastí jiné oblasti. Takto organizované oblasti nemusí nutně odpovídat hierarchii, která je tvořena zdrojovým HTML kódem, ale mohou tuto hierarchii vytvořit i jinak a vhodněji. Pro jednoduchost je možné do stromové struktury uložit i výstupy metod s jednorozměrným seznamem, kdy v takovém případě bude mít strom pouze jedinou úroveň.

3.3 Označování oblastí

Před samotnou detekcí vizuálních vzorů probíhá ještě proces, který by se dal česky nazvat jako označování oblastí, přiřazování štítků oblastem, případně označování obsahu (anglicky *content tagging*), kde oblastmi jsou myšleny ony vizuální oblasti vytvořené při segmentaci

vstupní webové stránky. Tento proces má za úkol alespoň částečně redukovat množinu prohledávaných oblastí, ve kterých se v dalším kroku budou hledat vizuální vzory, a také, a to zejména, označit případné oblasti, které mohou být alespoň částečně relevantní pro danou extrakci, tedy odpovídat atributům hledaných jedinců.

Ve své podstatě se jedná o typický problém klasifikace z oblasti strojového učení, kdy je snaha oblastem přiřadit správnou třídu, resp. štítek v tomto případě. Tento problém je zjednodušen tak, že jedna oblast může mít přiřazených více štítků, a stačí tedy mít k dispozici velmi jednoduchý binární klasifikátor, přičemž případné striktní rozhodnutí, který štítek je ten správný, je ponecháno na další fázi extrakce.

Přiřazení štítků oblastem nemusí být striktně binární, tedy případ, kdy oblast daný štítek buď má, nebo nemá, a nic mezi tím, ale může mít ještě například pravděpodobnost, s jakou daná oblast odpovídá danému štítku, resp. atributu, který je daným štítkem reprezentován. Takové pravděpodobnosti se pak říká například podle [2] podpora (anglicky *support*) a nabývá hodnot v intervalu od 0 do 1. Hodnota 0 pak logicky znamená, že štítek zcela určitě nepřísluší dané oblasti, a hodnota 1, že štítek zcela určitě dané oblasti přísluší.

Pro označování oblastí je možné využít mimo jiné i následující formy značkovačů oblastí (anglicky *taggers*):

- regulární výrazy, které mohou být jak velmi obecné, tedy například požadovat pouze počáteční velké písmeno u nadpisů, tak i specifičtější, kdy lze například s celkem vysokou úspěšností identifikovat email nebo URL adresu,
- již natrénované pokročilejší značkovače oblastí typu rozpoznávání pojmenovaných entit (anglicky *named-entity recognition*, zkráceně *NER*) například pro jména lidí,
- značkovače oblastí založené na vizuálních vlastnostech, například tučnosti písma, jeho velikosti, barvě, zda je kurzívou a podobně,
- jiné účelně natrénované značkovače oblastí.

3.4 Detekce vizuálních vzorů v označených oblastech

Poté, co jsou oblasti označeny a jsou jim tedy přiřazeny příslušné štítky, je nutné nějakým způsobem jednoznačně určit, který z přiřazených štítků je pro danou oblast ten správný, případně zda oblast má skutečně mít některý štítek. Tento proces by se dal česky nazvat jako odstranění nejednoznačnosti štítků (anglicky *tag disambiguation*) a jeho výsledkem jsou tak vizuální oblasti, které odpovídají atributům jednotlivých jedinců ve zdrojovém doménovém modelu.

Tohoto odstranění nejednoznačnosti je dosaženo v [2] vyhledáním vizuálních vzorů v označených oblastech. Vizuálními vzory jsou přitom myšleny v zásadě 2 věci:

1. opakující se oblasti se stejným štítkem a vizuálním stylem zobrazení textu, tedy například se stejnou tučností a velikosti písma nebo
2. opakující se dvojice oblastí ve vzájemné poloze na webové stránce, tedy například první oblast je na stejném řádku vždy za druhou oblastí nebo vždy pod danou oblastí a podobně.

3.4.1 Oblasti se stejným stylem textu

Vizuálním stylem textu je myšleno jeho vizuální zobrazení uživateli, tedy zejména různé vlastnosti písma v něm obsaženém. Například v [2] nebo v [3] je takový styl pro danou vizuální definován jako pětice obsahující:

- **průměrná velikost písma** v dané oblasti na absolutní škále od 0 do teoreticky nekonečna ve zvolených jednotkách, jakou může být například CSS pixel,
- **průměrná tučnost písma** v dané oblasti na relativní škále od 0 do 1, kde 0 znamená, že oblast obsahuje pouze nejméně tučné písmo vyskytující se v rámci zpracovávané webové stránky, což může být typicky běžný netučný text, pokud nejsou na stránce použity tenčí písma, a 1 znamená, že oblast obsahuje pouze nejtučnější písmo, což bývají typicky nadpisy nebo části označené HTML značkou `<strong>`,
- **průměrný styl písma** v dané oblasti na relativní škále od 0 do 1, kde 0 znamená, že oblast obsahuje pouze obyčejný text, a 1 znamená, že oblast obsahuje pouze text psaným kurzívou,
- **barva písma** a **barva pozadí**, které v dané oblasti převažují, například v hexadecimální notaci.

Taková definice může být ještě dále dle potřeby případně rozšířena například o podtržení, které je na webových stránkách typické pro odkazy, poměr velkých písmen například pro detekci oblastí obsahující pouze samá velká písmena, což může být použito například pro dekoraci nadpisů, a další okrajové vlastnosti. Zmíněných 5 vlastností, které byly použity v odkazovaných článcích, by ale mělo stačit pro velmi dobré výsledky.

Oblasti, kterým byl přiřazen alespoň jeden stejný štítek, pak mohou být posouzeny z hlediska stejnosti vizuálního stylu textu na základě stejnosti posuzovaných vlastností. Taková stejnost nemusí být zcela striktní, ale mohou být povoleny určité odchylky. Aby tedy byly oblasti považovány za oblasti se stejným vizuálním stylem, mohou se například lišit v předem stanoveném počtu vlastností. Z takto stejných oblastí poté může být vytvořena množina, která tak logicky bude obsahovat vizuálně stejné oblasti se stejným štítkem.

3.4.2 Oblasti ve vzájemné poloze

Druhou část detekce vizuálních vzorů v [2] i v [3] tvoří detekce vzájemné polohy dvou oblastí, což efektivně znamená nalezení vazeb mezi vlastnostmi jedince či jedinců. Poloha oblasti je přitom určena ve 2D prostoru počátečními souřadnicemi x a y a šířkou a výškou oblasti, ze kterých je poté možné vypočítat i souřadnice koncové. Vzájemná poloha je pak reprezentována, formálně vzato, binární relací mezi oblastmi, která určuje, zda se oblasti ve specifické vzájemné poloze nacházejí nebo nikoliv.

Různých relacích reprezentujících vzájemné polohy dvou oblastí může být celá řada a pro tento účel by se v podstatě daly použít pro základní relace předložky před, za, pod a nad. Ve zmíněných článcích jsou pak použity například následující relace:

- **napravo od sebe**, kdy je jedna oblast umístěna na stejném řádku vpravo od druhé bez velké mezery mezi nimi,
- **následující za oblastí**, kdy je jedna oblast umístěna na stejném řádku kdekoliv za druhou oblastí,

- **těsně pod oblastí**, kdy je jedna oblast umístěna vertikální pod druhou oblastí bez velké mezery mezi nimi,
- **kdekoliv pod oblastí**, kdy je jedna oblast umístěna vertikální kdekoliv pod druhou oblastí nebo
- **na stejném řádku**, kdy obě oblasti leží na stejném řádku nezávisle na tom, jestli je to vpravo či vlevo od sebe.

Jak je zřejmé, některé relace jsou také podrelacemi jiných relací, například pokud je oblast těsně pod jinou oblastí, pak je jistě i kdekoliv pod touto oblastí. Zároveň je k těmto relacím často možné pro další analýzu definovat něco, co by se dalo nazvat jako inverzní relace. Tedy například oblast A je pod oblastí B právě tehdy, když je oblast B nad oblastí A, kde „být nad oblastí“ je ona inverzní relace k relaci „být pod oblastí“.

3.5 Mapování oblastí na doménový model

Posledním krokem extrakce informací založené na detekci vizuálních vzorů je nalezení co nejvhodnějšího mapování mezi označenými oblastmi a vstupním ontologickým modelem dat. Toho je dosaženo v [2] nalezením největších možných množin dvojic vizuálních oblastí, které odpovídají stejné dvojici vlastností ve vstupním ontologickém datovém modelu. Dvojice oblastí v rámci jedné množiny přitom musí splňovat všechny následující podmínky:

- štítky obou oblastí ve dvojici musí být daným oblastem přiřazeny se stejnou minimální podporou,
- štítky obou oblastí ve dvojici musí být pro všechny oblasti stejné, tzn. že pokud první oblast ve dvojici má štítek 1 a druhá oblast štítek 2, pak i všechny ostatní dvojice v dané množině musí mít první oblast ve dvojici se štítkem 1 a druhou oblast ve dvojici se štítkem 2,
- styly obou oblastí ve dvojici musí být pro všechny oblasti stejné, tzn. že pokud první oblast ve dvojici má styl 1 a druhá styl 2, pak i všechny ostatní dvojice v dané množině musí mít první oblast ve dvojici se stylem 1 a druhou oblast ve dvojici se stylem 2,
- všechny dvojice oblastí se musejí nacházet ve stejné vzájemné poloze.

Texty vizuálních oblasti ze všech nalezených množin je poté možné po případném post-processingu přímo přiřadit podle příslušných štítků oblastí vlastnostem ze vstupního ontologického datového modelu.

Kapitola 4

Aplikační rámec FitLayout

Aplikační rámec (anglicky *framework*) FitLayout popsaný detailněji v [6] slouží pro podporu při vytváření aplikací a algoritmů pro vizuální zpracování a analýzu webových stránek a PDF dokumentů. Ačkoliv je napsán v jazyce Java a nabízí v tomto jazyce bohaté aplikační rozhraní včetně již připravených datových struktur, lze jeho výstupy použít i v jiných jazycích prostřednictvím rozhraní příkazové řádky nebo skrze aplikační rozhraní REST.

Tato kapitola se věnuje popisu hlavních aspektů tohoto aplikačního rámce pro jeho pozdější použití v práci jakožto základu pro analýzu a zpracování webových stránek, ze kterých chce uživatel provést extrakci informací.

4.1 Artefakty

Základem aplikačního rámce FitLayout jsou tzv. artefakty (anglicky *artifact*), což jsou serializovatelné výstupy jednotlivých fází analýzy vstupní webové stránky, se kterými je možné dále v aplikaci nebo i externě pracovat. Ty mohou a většinou i tvoří stromovou strukturu, kdy jeden artefakt vychází z druhého artefaktu a dále ho zpracovává. Výjimkou je počáteční kořenový artefakt, který většinou vzniká přímo ze vstupní webové stránky a nemá tedy žádného rodiče.

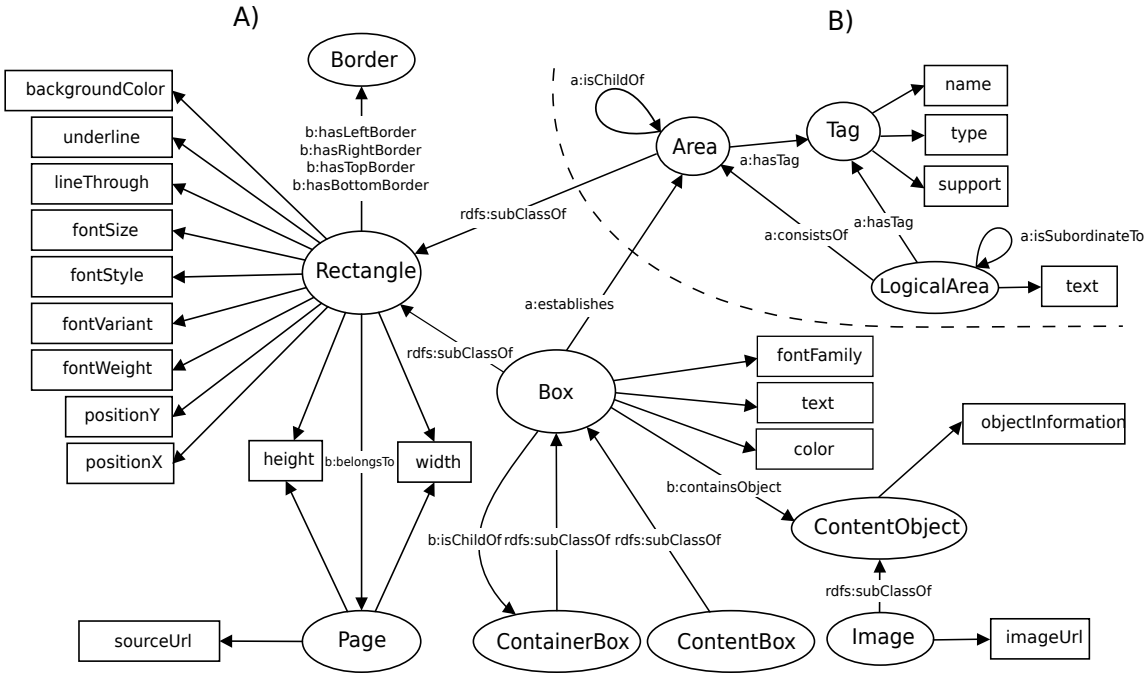
Artefakty jsou v aplikačním rámci FitLayout vytvářeny a konzumovány výhradně prostřednictvím tzv. *artifact service* a v době psaní tohoto textu mohou být následujících typů:

- **stránka** (anglicky *page*) vycházející přímo z vykresleného dokumentu,
- **strom oblastí** (anglicky *area tree*) obsahující hierarchicky uspořádané oblasti s jedním nebo více boxy jakožto výstup procesu segmentace,
- **strom logických oblastí** (anglicky *logical area tree*) s logickými oblastmi formujícími jednotlivé entity,
- **množina bloků textu** (anglicky *text chunk set*) formující posloupnost textových obdélníkových oblastí.

4.2 Reprezentace dokumentů

Reprezentace dokumentů v aplikačním rámci FitLayout je definována jako ontologie a vychází z CSS box modelu, který rozšiřuje o další třídy a vlastnosti užitečné pro vizuální

analýzu příslušné webové stránky. Ontologie v aplikačním rámci FitLayout zahrnuje boxy, obdélníky, oblasti, štítky a další třídy a jejich vlastnosti, jak ukazuje obrázek 4.1.



Obrázek 4.1: Ontologie aplikačních rozhraní aplikačního rámce FitLayout použitá mimo jiné také pro reprezentaci zpracovávaného dokumentu. Část A) znázorňuje ontologii boxů a část B) ontologii vizuálních oblastí. Převzato z [10].

Jak již bylo zmíněno v sekci 2.2.4, takováto reprezentace dokumentů není přímo vázána na HTML kód a webové stránky obecně, ale lze ji použít například i na dokumenty ve formátu PDF, k čemuž je dostupné vykreslovací jádro CSSBox, viz sekce 4.4.

4.3 Aplikační rozhraní

Aplikační rámec FitLayout nabízí v zásadě 3 aplikační rozhraní, pomocí kterých lze s tímto rámcem pracovat či využívat jeho výstupů:

- **aplikační rozhraní v jazyce Java** pro využití aplikačního rámce jako knihovny,
- **rozhraní příkazového řádku** pro využití například člověkem, případně jinou externí aplikací zejména v jiném jazyce než v jazyce Java bez potřeby serveru nebo
- **aplikační rozhraní REST** pro využití aplikačního rámce jako webové služby například v architektuře mikroslužeb.

4.3.1 Aplikační rozhraní v jazyce Java

Zřejmě nejpokročilejší rozhraní, které aplikační rámec FitLayout nabízí, je aplikační rozhraní pro jazyk Java, ve kterém je aplikační rámec napsán. Díky němu lze přímo z jazyka pracovat s artefakty, definovanými metodami těchto artefaktů nebo službami vytvá-

řející artefakty a je možné si dle potřeby definovat vlastní implementaci nabízených rozhraní. Navíc je možné přímo v rámci aplikačního rámce definovat například pomocí třídy `ServiceManager` postup nebo postupy (anglicky *workflow*), jakými budou dokumenty zpracovávány, tedy zejména jaké služby artefaktů budou v rámci řetězce zpracování použity.

4.3.2 Rozhraní příkazového řádku

V případě použití rozhraní příkazového řádku je možné analyzovat různé dokumenty pomocí aplikačního rámce FitLayout i z jiných jazyků než přímo z jazyka Java. Rozhraní nabízí na druhou stranu ale jen menší množství příkazů oproti přímému použití aplikačního rozhraní v jazyce Java, ačkoliv jednotlivé příkazy jsou celkem dobře konfigurovatelné.

Příkazy, které je možné použít, zahrnují dle [6] příkazy pro vykreslení stránky, provedení segmentace, export artefaktů ve formátech XML, Turtle, HTML, PNG nebo PNGi a příkazy pro práci s úložišti artefaktů v podobě načítání, ukládání nebo dotazování. Příkazy je možné také řetězit a dosáhnout tak například vykreslení, provedení segmentace a exportu do výše zmíněných formátů v jednom běhu programu.

4.3.3 Aplikační rozhraní REST

Pokud je žádoucí, aby aplikační rámec běžel jako webová služba, například pro využití mikroslužbami nebo i integraci v jiných jazycích než v jazyce Java, může být využito také aplikačního rozhraní REST. Samo o sobě je toto aplikační rozhraní mikroslužbou v jazyce Java, která může běžet na podporovaném aplikačním serveru a která je implementovaná v rámci projektu *FitLayoutWeb*.

Podle [6] nabízí v současné době aplikační rozhraní REST aplikačního rámce FitLayout následující koncové body (anglicky *endpoints*):

- **administrativní koncové body** sloužící například pro vytváření, mazání a zobrazení všech repozitářů i detailů o jednotlivých repozitářích,
- **koncové body artefaktů** sloužící například pro vytváření nových artefaktů, jejich čištění a mazání nebo získání detailů o nich,
- **koncový bod autentizace** vracející informace o uživateli na základě jeho tokenu,
- **koncové body repozitáře** sloužící například pro přidávání, čtení či mazání jmenných prostorů repozitáře, tvrzení, RDF popisů a dalších a
- **koncové body služeb artefaktů** sloužící pro získání výchozí konfigurace služeb artefaktů, jejich seznamu, seznamu operátorů nebo pro vykonání služby a vrácení artefaktu.

4.4 Vykreslovací jádra dokumentů

Vykreslovací jádra zpracovávaných dokumentů jakožto jeden typ služeb artefaktů mají za úkol převést vstupní webovou stránku nebo PDF dokument na interní reprezentaci v podobě artefaktu typu stránka. V aplikačním rámci FitLayout je v současné době k dispozici vykreslovací jádro nebo také vykreslovací backend (anglicky *rendering backend*):

- **CSSBox** sloužící pro vykreslování jednoduchých webových stránek bez podpory JavaScriptu a zejména PDF dokumentů a

- **Puppeteer** sloužící pro vykreslování komplexních webových stránek s využitím strojově ovládaného prohlížeče založeného na Chromiu.

4.4.1 Vykreslovací jádro CSSBox

Vykreslovací jádro CSSBox, které je jako jedno ze dvou vykreslovacích jader využito v aplikačním rámci FitLayout, je odlehčená alternativa vykreslovacích jader, které lze nalézt v běžných webových prohlížečích, jako například Blink, Gecko, Trident nebo Webkit. Toto vykreslovací jádro je napsáno výhradně v jazyce Java a za cíl si klade poskytnout o obsahu a rozložení zpracovávané webové stránky ve formátu (X)HTML se stylováním v jazyce CSS co nejvíce informací užitečných pro další zpracování či analýzu [1], což ho dělá skvělým kandidátem pro účely vizuální analýzy. Kromě webových stránek navíc podporuje i zpracování PDF dokumentů, které v případě vykreslovacího jádra Puppeteer není k dispozici.

Výhodou vykreslovacího jádra CSSBox je tak především jeho jednoduchost na zprovoznění, rychlost a nezávislost na externích komponentách, stejně tak jako možnost vykreslovat PDF dokumenty. Jeho nevýhodou je zejména absence podpory JavaScriptu, který dnes vyžaduje čím dál tím více webů i pro základní funkčnost, a také zřejmě ne úplně dokonalé vykreslování složitějších vzhledů webových stránek [5]. Vykreslovací jádro se tak hodí spíše pro jednodušší webové stránky, které nepoužívají složitá rozvržení stránky a nevyžadují pro svůj běh JavaScript, nebo případně pro PDF dokumenty, kde je vykreslovací jádro CSSBox v současné době jedinou volbou.

4.4.2 Vykreslovací jádro Puppeteer

Puppeteer je knihovna pro běhové prostředí Node.js jazyka JavaScript, která skrze vývojářský protokol dostupný v prohlížečích Chrome či Chromium poskytuje aplikační rozhraní pro jejich ovládání a automatizaci úloh [9]. Využívá se také často pro extrakci dat z webu, jak je naznačeno například v [4]. V aplikačním rámci FitLayout je na této knihovně postaveno vykreslovací jádro celým názvem *FitLayout - Puppeteer*, které je alternativou k vykreslovacímu jádru CSSBox.

Díky využití plnohodnotného webového prohlížeče, pro který jsou webové stránky primárně navrhovány, dokáže vykreslovací jádro zpracovávat i komplexní a dynamické webové stránky, a to zejména ty, které závisí na podpoře jazyka JavaScript, přičemž takto zpracovávané webové stránky jsou pak jednou z nejvěrnějších reprezentací, kterou lze získat. Nevýhodou pak může být složitější zprovoznění, které na rozdíl od vykreslovacího jádra CSSBox nefunguje ihned po instalaci aplikačního rámce FitLayout, nutnost napojení na další jazyk, chybějící podpora pro PDF dokumenty a také náročnost na výpočetní prostředky včetně místa na disku [5].

4.5 Segmentační algoritmy

Aplikační rámec FitLayout v základu obsahuje 3 algoritmy určené pro segmentaci webové stránky na oblasti:

- algoritmus segmentace založený na vidění, anglickým názvem *Vision-based page segmentation* (zkráceně *VIPS*),
- algoritmus segmentace založený na shlukování boxů, anglickým názvem *Block clustering segmentation* (zkráceně *BCS*) a

- algoritmus seskupující vizuální oblasti, anglickým názvem *Visual area grouping* nebo také *Basic page segmentation*.

Pro práci je zásadní zejména poslední zmíněný algoritmus, který je založen na základní a dále konfigurovatelné segmentaci zdola nahoru. Algoritmus detekuje větší vizuální oblasti, které následně seskupuje do uměle vytvořených uzlů, přičemž dále neprovádí nějakou složitější segmentaci, jako je tomu v případě prvních dvou algoritmů.

Kapitola 5

Návrh aplikace pro extrakci informací z webu

Na základě výše prezentovaných poznatků byla navržena aplikace, která provádí extrakci informací z webových stránek a PDF dokumentů metodou detekce vizuálních vzorů nacházejících se ve vstupních dokumentech, která je založena na metodách prezentovaných v [2] a částečně v [3] upravených pro potřeby aplikace. Aplikace byla pojmenována jako *VizGet* z českého slova *vizuální* a anglického slova *get*, tedy česky *získat*. Kapitola popisuje detaily návrhu aplikace VizGet.

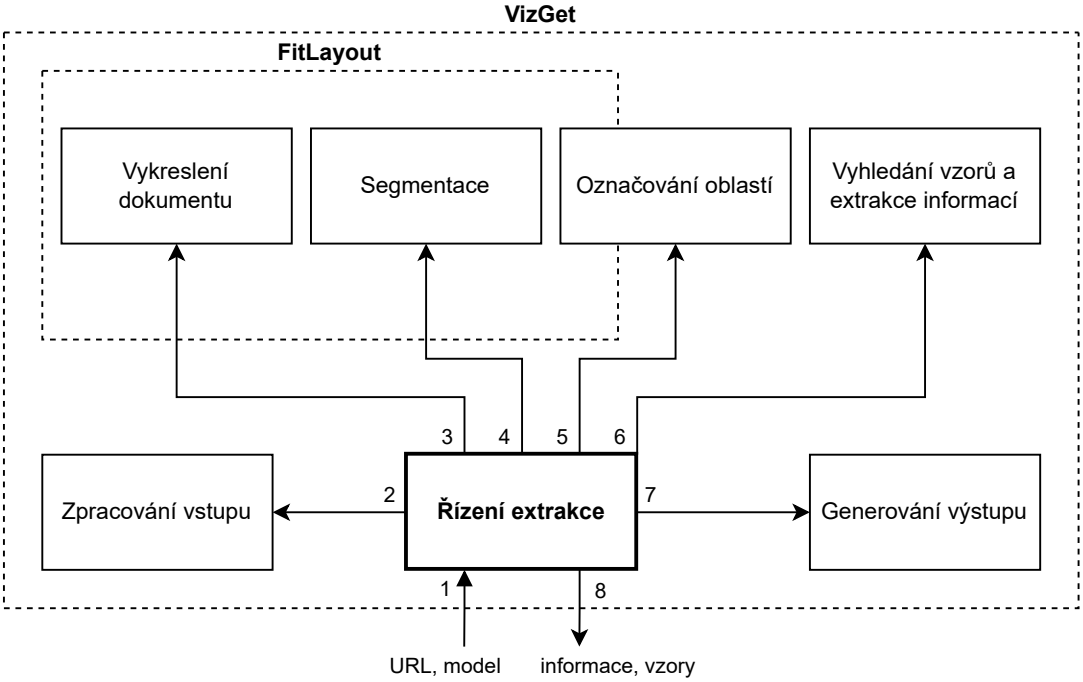
5.1 Architektura aplikace

Z hlediska architektury je aplikace VizGet konzolová aplikace napsaná v jazyce Java, jejíž architektura je ovlivněna využitím aplikačního rámce FitLayout. Součástí architektury, kterou lze ve zjednodušeném náhledu možné vidět na obrázku 5.1, je:

- hlavní komponenta řadiče řídící celou extrakci,
- zpracování vstupu zahrnující zpracování argumentů a načtení vstupního ontologického datového modelu,
- aplikační rámec FitLayout provádějící stahování dokumentů ze sítě včetně jejich vykreslení a segmentace zvolenou metodou,
- označování oblastí prováděné různými značkovači oblastí například pro datum a čas nebo značkovačem využívajícím regulárního výrazu,
- hlavní jádro aplikace v podobě algoritmu pro vyhledávání vizuálních vzorů a extrakce informací na základě vstupního modelu a
- komponenta pro generování výstupu aplikace na základě získaných informací.

5.2 Vstupní parametry aplikace

Jelikož je aplikace VizGet konzolová aplikace, jsou jejím vstupem zejména argumenty příkazové řádky, které specifikují povinně a v nejjednodušším případě použití:



Obrázek 5.1: Základní součásti architektury aplikace VizGet blíže popsané v sekci 5.1. Čísla značí pořadí volání jednotlivých komponent.

- **soubor s ontologickým datovým modelem**, k němuž se budou hledat ve vstupních dokumentech požadované informace, zadaný pomocí parametru `-m` nebo `--model`, případně skrze standardní vstup, není-li parametr zadán,
- **URL adresy**, ze kterých se budou požadované informace navázané na příslušný ontologický datový model extrahovat.

Příklad nejjednoduššího použití aplikace pro filmový ontologický datový model a se specifikací pouze povinných vstupních argumentů je možné vidět na obrázku 5.2.

```
1 ./vizget https://www.csfd.cz/zebricky/filmy/nejlepsi/ \
2   < movie.ttl \
3   > extracted.ttl
```

Obrázek 5.2: Ukázka nejjednoduššího použití aplikace VizGet, kdy jsou specifikovány pouze povinné vstupní parametry.

Ačkoliv i při nejjednodušším použití by měla mít aplikace přednastavené použitelné hodnoty a pokud možno automaticky detekovat například formát vstupního modelu či dokumentu, je možné aplikaci specifikovat i následující parametry:

- **vykreslovací jádro** zadané pomocí parametru `--renderer`, které se má použít pro vykreslení dokumentu, například pro použití vykreslovacího jádra CSSBox v případě jednoduché HTML stránky pro dosažení rychlejšího zpracování,

- **šířka** a **výška** viditelného výřezu (anglicky *viewport*) vykreslovaného dokumentu zadané parametry `-w` nebo `--width` pro šířku a `-h` nebo `--height` pro výšku,
- **perzistence**, vytrvalost či „rychlost“ při vykreslování komplexního dokumentu zadaná parametrem `-s` nebo `--speed`,
- **formát vstupního modelu** zadaný pomocí parametru `--formatIn`, který nemusel být správně detekován,
- **formát výstupního souboru** zadaný pomocí parametru `--formatOut` například pro použití v aplikaci nepodporující výchozí výstupní formát,
- **výstupní soubor s extrahovanými informacemi** zadaný pomocí parametru `-o` nebo `--output`, není-li žádoucí posílat extrahované informace skrze standardní výstup,
- **obrázek vizualizace** nalezených vizuálních vzorů v dokumentu zadaný pomocí parametru `-p` nebo `--patterns`, pokud je žádoucí zkoumat programem nalezené vizuální vzory,
- **zapnutí/vypnutí/přenastavení** některých/všech **reduktorů** a **metrik** zadané pomocí parametrů `--enable-reducer`, `--disable-reducer`, `--reducer-pipeline`, `--enable-metric`, `--disable-metric` a `--metrics`,
- **vypnutí detekce víceřádkových oblastí** zadané pomocí parametru `--no-multiline`.

Příklad použití téměř všech dostupných parametrů při pokročilém použití aplikace opět s filmovou ontologií ukazuje obrázek 5.3.

```
./vizget --renderer cssbox \
  --formatIn TURTLE \
  --model movie.ttl \
  --formatOut RDFXML \
  --output extracted.xml \
  --patterns patterns.png \
  --width 1280 \
  --height 720 \
  --disable-reducer ClosestIdAreaReducer \
  --enable-reducer IdAreasReducer \
  --disable-metric UniqueWordsMetric \
  https://www.csfd.cz/zebricky/filmy/nejlepsi/
```

Obrázek 5.3: Ukázka pokročilejšího použití aplikace VizGet s téměř všemi podporovanými parametry.

5.3 Formát vstupního modelu dat

Povinným vstupem aplikace je ontologický datový model, na základě kterého bude extrakce řízena. Takovým modelem je tedy v podstatě seznam tříd a vlastností, které mají být ze vstupního dokumentu extrahovány a které se tím pádem vyskytnou na výstupu jako jedinci těchto tříd. K takovému účelu slouží mimo jiné i ontologický jazyk OWL [11], jehož omezená podmnožina je využita pro specifikaci vstupního datového modelu. Podporována je zejména specifikace tříd a jejich datových a objektových vlastností, domény vlastností, dále maximální či přesné omezení kardinality a použití všech vlastností či tříd definovaných v ontologii aplikace VizGet.

Zjednodušený příklad minimální kostry vstupního datového modelu pro filmovou ontologii je možné vidět na obrázku 5.4.

```
1  @prefix vizget: <https://vizget.supermartas.cz/#> .
2
3  :Movie rdf:type owl:Class ;
4         rdfs:subClassOf [ rdf:type owl:Restriction ;
5                           owl:onProperty :name ;
6                           owl:maxCardinality "1"^^xsd:nonNegativeInteger
7                         ] ,
8                         [ rdf:type owl:Restriction ;
9                           owl:onProperty :rank ;
10                          owl:maxCardinality "1"^^xsd:nonNegativeInteger
11                        ] ;
12        vizget:idProperty :name .
13
14 :name rdf:type owl:DatatypeProperty ;
15       rdfs:domain :Movie ;
16       vizget:tagger [ rdf:type vizget:SentenceTagger ;
17                       vizget:languages "cs", "en"
18                     ] .
19
20 :rank rdf:type owl:DatatypeProperty ;
21       rdfs:domain :Movie ;
22       vizget:tagger [ rdf:type vizget:RegexTagger ;
23                       vizget:regex "\\d+(?=\\.)"
24                     ] .
```

Obrázek 5.4: Ukázka minimální kostry zjednodušeného ontologického datového modelu pro filmovou ontologii v jazyce OWL a formátu Turtle. Prefix `vizget:` odkazuje na ontologii definovanou pro aplikaci VizGet. Zbytek vynechaných prefixů tvoří standardní prefixy užívané v ontologiích.

Specifikem datového modelu použitého v aplikaci VizGet je možnost využít pro účely aplikace vytvořenou ontologii s doporučeným prefixem `vizget:`, která obsahuje anotační vlastnost `tagger`. Touto anotací lze nepovinně určit, jaký značkovač oblastí bude využit pro danou vlastnost. Pokud tedy například nějaká oblast obsahuje text podobný datu,

bude tato oblast označena štítkem signalizujícím skutečnost, že se může jednat o vlastnost `releaseDate` a podobně. Je vyloženě povinné tuto anotaci uvést, jinak nelze očekávat žádné smysluplné výsledky.

Dalším specifikem je užití anotační vlastnosti `idProperty`, která specifikuje pro danou třídu identifikující vlastnost. Pro dobré výsledky je v zásadě nutné ji vhodně zvolit a v ontologii specifikovat. Správnou volbu vlastnosti popisuje dále podrobněji sekce 6.2.3.

5.4 Formát extrahovaných informací

Hlavním výstupem aplikace jsou informace extrahované z dokumentů dostupných na zadaných URL adresách, které jsou zapsány jako vlastnosti jedinců příslušných tříd ze vstupního ontologického datového modelu, čímž jsou s tímto modelem jedinci efektivně propojeni. V zásadě tak jde tedy více méně o trojice subjekt–predikát–objekt, které jsou základem v Resource Description Frameworku (zkráceně *RDF*) [8]. Tyto trojice mohou být pak zapsány ve stejném formátu, jako je tomu u vstupního ontologického datového modelu. Příklad takového zkráceného zápisu ve formátu Turtle pro jeden film ve filmovém ontologickém datovém modelu lze vidět na obrázku 5.5.

```
1 <https://www.csfd.cz/zebricky/filmy/nejlepsi/#Movie-1> rdf:type :Movie ;
2   :name "Vykoupení z věznice Shawshank" ;
3   :rank "1" ;
4   :rating "95,3" ;
5   :ratingCount "106 470" ;
6   :releaseDate "1994" .
```

Obrázek 5.5: Zjednodušená ukázka relevantních částí extrahovaného filmu z Česko-Slovenské filmové databáze ve formátu Turtle. Prefix : je odkaz na vstupní ontologický datový model, na který je extrahovaný film navázán.

Kapitola 6

Zpracování vstupních dat

Jak již bylo popsáno v sekci 5.2, vstup aplikace VizGet tvoří zejména parametry příkazové řádky, které zahrnují URL adresu pro provedení extrakce z ní, specifikaci souboru s modelem, jeho formátem a další, a hlavně vstupní ontologický model dat. Aby aplikace věděla, co má přesně provádět za činnost a kde má tuto činnost provádět, je nutné tato vstupní data nejprve zpracovat a přenést do vhodnějších datových struktur.

Tato kapitola se zabývá právě tímto zpracováním vstupním dat. V první sekci kapitoly je vysvětleno zpracování parametrů příkazové řádky, na což navazuje druhá sekce s popisem získání informací o ontologickém datovém modelu.

6.1 Zpracování vstupních parametrů

Parametry či argumenty příkazové řádky tvoří jeden z nejzákladnějších způsobů, jak programu na příkazové řádce předat vstupní data a modifikovat jeho chování. I proto existují již hotová řešení, která se snaží do určité míry co nejvíce zjednodušit zpracování těchto parametrů a celkově také vystavět určitou podporu pro psaní konzolových aplikací.

Pro jazyk Java existuje spousta knihoven či dokonce aplikačních rámců s takovým zaměřením. Od tradičnějších Apache Commons CLI[1] po novější JCommander[2] nebo aplikační rámec picocli[3]. Právě aplikační rámec picocli ve verzi 4.6.2 byl vybrán pro svou jednoduchost a nabízené funkce s minimálním kódem navíc, který by byl nutný napsat.

V aplikačním rámci picocli jsou základem příkazy (anglicky *commands*) v podobně tříd anotovaných pomocí `@Command` a implementujících metodu příslušného rozhraní, která vrací návratový kód typický pro konzolové aplikace. V rámci této třídy poté anotované soukromé členské proměnné tvoří vstupní parametry aplikace, které mohou být v základu získány buď jako nepoziční parametry (anglicky *options*) nebo naopak poziční parametry (anglicky *parameters* nebo *arguments*).

Aplikace VizGet definuje jediný stejnojmenný příkaz `vizget`, který provádí veškerou funkčnost aplikace. Obecný výčet pozičních i nepozičních parametrů odpovídá parametrům definovaným v sekci 5.2. Nejběžnější datové typy jako číslo, soubor, URL adresa nebo dokonce seznamy těchto typů dokáže aplikační rámec picocli sám z řetězce převést, ovšem pro vlastní či jiné pokročilejší datové typy je nutné definovat si vlastní převodník, který se o celou konverzi z řetězce sám postará.

[1] https://commons.apache.org/proper/commons-cli/
[2] https://jcommander.org/
[3] https://picocli.info/

6.1.1 Převodník na instance tříd

Převodník na instance tříd `ClassConverter` ze zadaného textového jména třídy vytváří instance z bezparametrických konstruktorů těchto tříd a kontroluje, zda tyto třídy implementují rozhraní příslušné pro daný parametr. Zadat je možné jak celé plně kvalifikované jméno třídy[4] tak i pouze holé jméno třídy[5], která se nachází ve stejném balíčku jako rozhraní daného parametru.

Převodník je využit pro třídy implementující rozhraní metrik a reduktorů popsaných dále v textu, tedy parametrů `--enable-metric`, `--disable-metric`, `--metrics`, `--enable-reducer`, `--disable-reducer` a `--reducer-pipeline`.

6.1.2 Převodník ontologického datového modelu

Převodník ontologického datového modelu `ModelConverter` načítá soubor s modelem v zadaném formátu a převádí ho na příslušnou třídu modelu z aplikačního rámce Apache Jena[6] prostřednictvím metod z tohoto rámce. Před samotnou konverzí ale do modelu navíc přimíchá ontologii aplikace VizGet, která definuje některé základní ontologické třídy například pro značkovače oblastí, reduktory či metriky, přičemž metadata těchto tříd jsou později využita pro tvorbu příslušných instancí těchto tříd přímo v kódu.

Po načtení modelu ještě převodník provádí jeho základní validaci. Ta zahrnuje vydání varování v případě, že se v ontologii nachází třída bez jakékoliv datové či objektové vlastnosti, a tedy že třída nebude nikdy extrahována a bude tak efektivně ignorována. Zároveň jsou zkontrolovány všechny datové i objektové vlastnosti a vydáno varování v případě, že vlastnost nemá definovanou žádnou deklarující třídu a bude tedy ignorována. Aplikace také ukončí svůj běh, pokud má vlastnost definovanou více než jednu deklarující třídu, jelikož takový případ není podporován a pro správný běh musí mít vlastnost definovanou právě jednu deklarující třídu.

Tento převodník není využit přímo jako převodník specifikovaný v anotaci volby, nýbrž je volán z hlavního těla programu. Pro svou správnou funkci totiž vyžaduje zpracování dvou parametrů zároveň: parametru `-m` či dlouze `--model` pro specifikaci souboru s ontologickým datovým modelem (nebo případně řetězec ze standardního vstupu, pokud nebyl model specifikován jinak) a parametru `--formatIn`, který určuje formát modelu.

6.1.3 Ostatní převodníky

Pro sjednocení výstupu aplikace jak na standardní výstup, tak i do souboru, existuje převodník `OutputStreamConverter`, který z cesty k souboru zadaného parametrem `-o` či `--output` vytváří standardní výstupní rozhraní v jazyce Java.

Vytvoření instance příslušné třídy pro vykreslovací jádro zadané parametrem `-r` či `--renderer` zase zajišťuje převodník `RendererConverter` typu `if-else` pro jediné možnosti `cssbox` a `puppeteer`.

A nakonec formát výstupního souboru zadaného parametrem `--formatOut` je konvertován v převodníku `RdfFormatConverter`, který vrátí skrze reflexi příslušnou konstantu ze třídy `RDFFormat` z aplikačního rámce Apache Jena.

[4] Například `cz.supermartas.vizget.patterns.reducers.BiggestClusterReducer`.
[5] Například `BiggestClusterReducer` v tomto ukázkovém případě.
[6] https://jena.apache.org/

6.2 Získání informací o modelu

Po základním zpracování vstupních parametrů přichází důležitý krok získání informací o vstupních ontologickém datovém modelu. Tento krok má za cíl extrahovat informace o třídách, vlastnostech a párech vlastností, které následně budou hledány na požadované webové stránce. Díky informací specifikovaným přímo v modelu lze s úspěchem použít další techniky při hledání vzorů jako například předpoklady o kardinalitě mezi vlastnostmi apod.

6.2.1 Tvorba instancí z jedinců

Jedinci, a to zejména ti anonymní, jsou ve vstupní ontologii použiti výhradně jako nástroj pro vytvoření nakonfigurovaných instancí tříd přímo v kódu aplikace. To umožňuje v ontologii vytvořit například značkovače oblastí, které mají různé parametry ovlivňující, jak bude označování oblastí probíhat a co se případě bude v oblastech hledat.

Vytváření instancí z jedinců probíhá ve třídě `IndividualParser` skrze vyhledání třídy, jejíž instanci daný jedinec odpovídá a nalezení anotační proměnné `vizget:javaClass`, ve které je zapsáno plně kvalifikované jméno třídy jazyka Java. Jméno třídy může být i relativní vzhledem k balíčku rozhraní, které má třída implementovat. Následně je v programu nalezena třída tohoto jména a pokud tato třída implementuje ono příslušné očekávané rozhraní pro daný typ třídy, je vytvořena instance z bezparametrického konstruktoru třídy. Samotné vytváření instance ze jména třídy je tedy shodné s vytvářením instancí v případě převodníku na instance tříd, který je popsán v sekci 6.1.1.

Rozdílem oproti vytváření instancí z jedinců oproti převodníku ze sekce 6.1.1 je podpora nastavování parametrů, které mohou následně modifikovat chování dané instance třídy. Toho je docíleno skrze nutnost implementovat rozhraní `ParametrizedOperation` z aplikačního rámce FitLayout, které poskytuje mimo jiné možnost dynamicky nastavovat proměnné skrze jména uložená v proměnných. Názvy a hodnoty těchto proměnných pak pochází přímo z datových vlastností přiřazených jednotlivým jedincům. Název proměnné je tedy celé URI predikátu v případě predikátu nepocházejícího z ontologie VizGet nebo jen jeho lokální jméno pro predikáty z ontologie VizGet a hodnota je literál na místě objektu příslušného primitivního datového typu.

Jednoduchý příklad vytvoření instance třídy z jedince ve vstupní ontologii ukazuje obrázek 6.1.

```
1 vizget:RegexTagger rdf:type owl:Class ;
2   vizget:javaClass "cz.supermartas.vizget.tagging.taggers.RegexTagger" .
3
4 movie:name vizget:tagger [ rdf:type vizget:RegexTagger ;
5                            vizget:regex "[A-Z].+"
6                          ] .
```

Obrázek 6.1: Na ukázce ontologie v jazyce Turtle je zachycena část s anonymním jedincem, který reprezentuje značkovač oblastí pomocí regulárního výrazu. V kódu je z něj vytvořena instance třídy `cz.supermartas.vizget.tagging.taggers.RegexTagger`, které je předán parametr s názvem `regex` typu řetězec znaků s hodnotou `[A-Z].+`.

6.2.2 Získání informací o vlastnostech

Aplikace VizGet se snaží hledat data pro všechny datové a objektové vlastnosti, které jsou specifikované ve vstupním modelu a které mají definovanou jedinou deklarující třídu. Existence jediné deklarující třídy je klíčová, protože samotná třída slouží pro algoritmus jako určitá obálka nad několika vlastnostmi. Ty mají všechny určitý prostorový vztah, jak byl definován v sekci 3.4.2, s jedinou hlavní, třídu identifikující vlastností.

Vlastnosti jsou tak ve třídě `PropertyParser` hledány na základě získání seznamu všech pojmenovaných tříd, které nejsou ze jmenného prostoru ontologie VizGet, ani nejsou potomky pomocných tříd, které ontologie VizGet definuje. K těmto třídám jsou následně vyhledány všechny datové a objektové vlastnosti, které mají danou třídu určenou jako deklarující (skrze predikát `rdfs:domain`), čímž je efektivně získán seznam všech podstatných vlastností.

Každá takto získaná vlastnost je uložena do vlastní třídy reprezentující právě jednu vlastnost k extrakci. Pokud je navíc pomocí anotační vlastnosti `vizget:styleHints` specifikováno jedno či více vodítek, které určují možné styly oblastí dané vlastnosti, jsou do této třídy také uloženy instance tříd těchto vodítek, které byly vytvořeny z jedinců podle algoritmu ze sekce 6.2.1.

Pro každou objevenou vlastnost jsou navíc ještě získány značkovače oblastí z hodnoty anotační vlastnosti `vizget:tagger`, přičemž zadat je možné nanejvýše jeden značkovač oblastí pro každou vlastnost. Značkovače oblastí jsou opět instance tříd vytvořené z jedinců podle algoritmu, který byl popsán v sekci 6.2.1, a je nutné a velmi důležité je zadat pro správný běh programu. Každá jedna instance značkovače a štítek, který značkovač přiřazuje, jsou v programy spojeni vždy právě s jednou vlastností, díky čemuž je později možné identifikovat u označeného oblasti vlastnost, kterou byla oblast označena.

6.2.3 Získání dvojic vlastností k extrakci

Extrakce v aplikaci VizGet je založena na hledání vzorů, přičemž jeden vzor v hledané množině vzorů odpovídá nějaké dvojici či páru vlastností ze vstupního ontologického datového modelu. Aplikace tedy potřebuje vědět, jaké dvojice má na stránce hledat, a musí tedy umět nějakým způsobem tyto dvojice ze vstupního modelu sestavit. Jak bylo zmíněno v předešlé sekci, určitými obálkami pro vlastnosti jsou právě třídy a přesně toho je využito při sestavování dvojic vlastností k extrakci.

Ačkoliv ne vždy to musí být pravda, aplikace zakládá na, jak se ukázalo, dost dobře použitelném předpokladu, že bude vždy existovat alespoň jeden z definovaných prostorových vztahů právě mezi vlastností, která byla označena pro danou třídu jako identifikující (dále také jako „identifikující vlastnost“), a libovolnou další vlastností dané třídy (dále také jako „závislá vlastnost“). Z toho plynou mimo jiné následující důsledky:

- Aplikace nedokáže pro jednu třídu vyhledat pouze její jedinou (a tím pádem i identifikující) vlastnost, ale třída musí mít alespoň dvě datové vlastnosti nebo musí pro zkoumanou třídu existovat jiná třída, která má tuto třídu jako hodnotu své objektové vlastnosti.
- Aplikace nedokáže nalézt jiné potenciálně existující prostorové vztahy mezi dvěma závislými vlastnostmi, tedy vlastnostmi, kdy ani jedna z vlastností není pro danou třídu identifikující vlastnost.

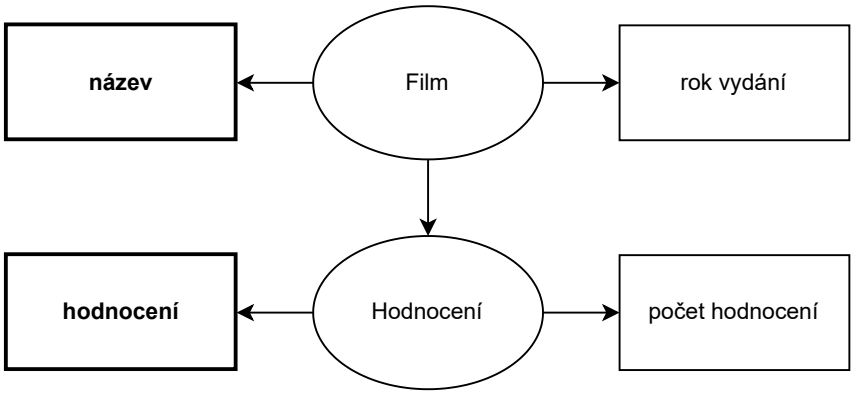
Samotné sestavení či nalezení dvojic vlastností tedy ve třídě `PropertyPairParser` zahrnuje jako první krok nalezení všech identifikujících vlastností pro každou jednu třídu.

Nelze vyloučit, že nalezení té „správné“ identifikující vlastnosti, tedy nejspíše té vlastnosti, která by znamenala nejlepší výsledky extrakce, by bylo možné do určité míry zautomatizovat, ovšem pro jednoduchost tuto informaci zadává uživatel pomocí anotační vlastnosti `vizget:idProperty` u každé jedné třídy modelu. Zadání této informace není po uživateli vyžadováno, ale v případě absence se použije jako identifikující vlastnost libovolná jiná vlastnost třídy a extrakce nemusí být tak efektivní jako při manuální volbě.

Aplikace VizGet ve výchozím nastavení počítá s tím, že je zvolena jako identifikující vlastnost taková vlastnost, jejíž hodnoty jsou pokud možno pro danou třídu unikátní nebo alespoň co nejvíce unikátní, tedy že neobsahují duplicitní hodnoty, přičemž zvýhodňována jsou i unikátní slova. Takové zvýhodňování probíhá pomocí metrik hodnotící množinu vzorů, které jsou popsané dále v textu. Zvolit jako identifikující vlastnost je však možné i vlastnost, jejíž hodnoty jsou všechny stejné. Jen to poté bude znamenat nižší zvýhodňování množin vzorů s takovými vlastnostmi a extrakce tak nemusí poskytovat nejlepší výsledky, nebyly-li tyto zvýhodňující metriky explicitně vypnuty.

V následujícím kroku už stačí pouze projít všechny v předchozí sekci nalezené vlastnosti, které nejsou pro žádnou třídu identifikující, a z nich vytvořit ony dvojice vlastností. Jako identifikující vlastnost dvojice se při takovém vytváření použije vždy identifikující vlastnost třídy, do které vlastnost spadá. V případě datových vlastností se pak procházená vlastnost jednoduše použije přímo jako vlastnost závislá. Komplikace nastává pouze u objektových vlastností, kde se prochází jejich povinně definovaný rozsah hodnot (hodnoty vlastnosti `rdfs:range`), tedy možné třídy jako hodnoty, a jako závislé vlastnosti se použijí identifikující vlastnosti těchto tříd.

Příklad vytvoření dvojic vlastností k extrakci ukazuje na rozšířené filmové ontologii obrázek 6.2.



Obrázek 6.2: Rozšířená filmová ontologie s třídami v oválech, vlastnostmi v obdélnících a identifikujícími vlastnostmi dané třídy zvýrazněnými tučně. Pro takovou ontologii vzniknou dvojice vlastností *název – rok vydání*, *název – hodnocení* a *hodnocení – počet hodnocení*.

6.2.4 Získání kardinality dvojic vlastností

Důležitým předpokladem pro úspěšnou extrakci je specifikace kardinality dvojic vlastností. Na základě této specifikace totiž fungují dále v textu popsané reduktory, které extrakci významně vylepšují. Vzhledem k tomu, že identifikující vlastnost reprezentuje ve dvojici celou třídu, jedná se vlastně o kardinalitu mezi třídou a konkrétní závislou vlastností.

Pro tento účel existují v jazyku OWL tzv. restrikce, které se používají pro tvrzení, že nějaká třída je podtřídou právě této restrikce. Těmito restrikcemi je možné mimo jiné specifikovat i kardinalitu vlastností třídy, a to skrze minimální počet výskytů, maximální počet výskytů a také přesný počet výskytů. Ukázku specifikace kardinality pro část filmové ontologie je možné vidět na obrázku 6.3.

```
:Movie rdf:type owl:Class ;
  rdfs:subClassOf [ rdf:type owl:Restriction ;
                    owl:onProperty :releaseDate ;
                    owl:maxCardinality "1"^^xsd:nonNegativeInteger ;
                    owl:onDataRange xsd:string
                  ] .
```

Obrázek 6.3: Ukázka zápisu maximální kardinality vlastnosti *datum vydání* na třídě reprezentující film, které rozumí aplikace VizGet.

Na příkladu s filmovou ontologií může mít každý film nanejvýš jedno datum vydání (vztah 1:1) a protože nebylo specifikováno jinak, tak třeba i neomezený počet herců (vztah 1:n). Na dvojicích vlastností se to pak projeví tak, že název filmu v jedné dvojici bude mít nanejvýš jedno datum vydání a zároveň v jiné dvojici bude mít název filmu neomezený počet herců. Relace m:n není pro svou komplikovanost aplikací VizGet podporována.

Zároveň aplikace nijak zvlášť nezohledňuje minimální počet výskytů, ovšem maximální počet výskytů je možné specifikovat, a to včetně čísel větších jak 1 a menších než nekonečno. Specifikace jiných hodnot maximální kardinality než 1 a nekonečno (vynecháním specifikace kardinality) se hodí v případech, kdy je tento počet předem znám, jako například v hodinovém výpisu nějakých informací pro každý den, kdy v každém dnu bude vypsáno nanejvýše 24 hodin/hodnot.

Aplikace VizGet se bude maximální kardinalitou do značné míry řídit a jedná se tak o další z klíčových způsobů, jak ovlivnit kvalitu výsledků extrakce. Specifikovat tuto maximální kardinalitu je nutné zejména pro vztahy 1:1, jelikož ve výchozím nastavení mají vztahy v ontologiích kardinalitu 1:n. Vztahů 1:1 je navíc, na rozdíl od jiných oblastí užití těchto vztahů, ze zkušenosti většina, takže je nutné na to při tvorbě ontologie pamatovat.

Implementačně lze kardinality vztahů v modelu najít skrze vyhledání všech nadtříd extrahovaných tříd, které jsou právě typu restrikce. Z těchto restrikcí jsou dále odstraněny všechny restrikce, které nejsou typu minimální, maximální nebo přesná kardinalita. Tyto restrikce pak je možné v rámci jedné vlastnosti třídy sloučit, čímž vzniká omezení minimálního a maximálního počtu výskytů pro každou vlastnost. Při tvorbě dvojic vlastností je poté v každé dvojici využito omezení vlastnosti, která ve vztahu vystupuje jako závislá, jelikož identifikující vlastnost se vyskytuje vždy jen jednou.

6.2.5 Získání ostatních informací o dvojicích vlastností

Stejně jako v případě kardinality se používají i pro ostatní dvojice vlastností hodnoty, které byly specifikovány právě u závislé vlastnosti, tentokrát jako anotační vlastnosti. Hodnotami těchto anotačních vlastností jsou jedinci, kteří jsou zpracováváni opět podle algoritmu ze sekce 6.2.1. Podporováno je:

- **zapnutí/vypnutí/přenastavení** některých/všech **reduktorů** a **metrik** skrze jedince v hodnotách anotačních vlastností:
  `vizget:enableReducers`, `vizget:disableReducers`, `vizget:reducerPipeline`, `vizget:enableMetrics`, `vizget:disableMetrics` a `vizget:metrics`,
- **vynucení prostorových vztahů** mezi dvojicí vlastností skrze jedince v hodnotách anotační vlastnosti `vizget:areaRelations`.

Speciální zacházení nastává pouze v případě tzv. „zapínatelných“ anotačních vlastností reduktorů a metrik. Ve výchozím nastavení má aplikace povolenou určitou množinu reduktorů a metrik, které je možné kromě lokální změny u konkrétní dvojice vlastností měnit také na globální úrovni pro všechny dvojice vlastností zároveň. To lze učinit skrze parametry příkazové řádky `--enable-reducer`, `--disable-reducer`, `--reducer-pipeline`, `--enable-metric`, `--disable-metric` a `--metrics`.

Pokud je tedy například reduktor X ve výchozím nastavení povolen, na globální úrovni se zakáže, ale vlastnost A ho opět povolí, bude reduktor X použit pouze pro vlastnost A a všechny ostatní vlastnosti tento reduktor nepoužijí. Přepíše-li navíc vlastnost B reduktory na X a Y, použijí se u této vlastnosti právě a pouze tyto reduktory a žádné jiné.

Kapitola 7

Příprava pro hledání vizuálních vzorů

Poté, co jsou získány všechny potřebné informace ze vstupních dat v podobě parametrů příkazové řádky a vstupního ontologického datového modelu, může extrakce pokročit do další fáze. Tuto fázi zastává z velké části aplikační rámec FitLayout ve verzi 2.0.1 a jde o hrubé získání obsahu webové stránky, její následná segmentace, předzpracování takto vzniklých oblastí, přiřazování štítků oblastem, a nakonec extrakce shluků textu z označených a předzpracovaných oblastí. Souhrnně byla tato fáze nazvána jako příprava pro hledání vizuálních vzorů a o vyjmenovaných dílčích podproblémech pojednává právě tato kapitola.

7.1 Vykreslování dokumentů

O celé vykreslení a získání CSS boxů webových stránek a PDF dokumentů se stará zvolené vykreslovací jádro z aplikačního rámce FitLayout, a to buď vykreslovací jádro CSSBox nebo Puppeteer. Ačkoliv zvolit je možné obě vykreslovací jádra, veškerý vývoj probíhal výhradně na vykreslovacím jádru Puppeteer.

Jak již totiž bylo zmíněno v sekci 4.4.2, jádro CSSBox se hodí pouze na PDF dokumenty, jelikož Puppeteer tyto dokumenty nepodporuje, a na jednoduché webové stránky. Za komplexnější stránky, se kterými má CSSBox potíže, jsou považovány i stránky, které používají dnes velmi rozšířený CSS flexbox či grid. Ty způsobují pády[1] vykreslovacího jádra a nejsou tak zatím vykreslovacím jádrem podporovány[2].

Kromě výběru vykreslovacího jádra je možné vykreslování dokumentů ovlivnit i skrze další parametry vykreslovacího jádra. Těmito parametry jsou nastavení šířky (parametr `-w` nebo `--width`) a výšky (parametr `-h` nebo `--height`) viditelného výřezu dokumentu a v případě vykreslovacího jádra Puppeteer také tzv. perzistence, vytrvalosti či určité „rychlosti“ vykreslování (parametr `-s` nebo `--speed`).

Po hojném využívání tradičních technik extrakce mohou být mírným překvapením právě parametry šířky a výšky, které ale dávají smysl u vizuálních technik extrakce, kde záleží na pozici elementů na webové stránce. Zejména pomocí šířky pak lze u responzivních webů ovlivnit, zda jsou informace extrahovány z verze webu pro počítače, tablety či mobilní telefony. Tyto verze nemusí být vždy ekvivalentní ať už co se prezentovaného obsahu týče nebo i úspěšnosti extrakce v prostředí potenciálně jinak uspořádaných elementů.

[1] Diskuze o pádech je dostupná na https://github.com/FitLayout/FitLayout/issues/5.
[2] Vyplývá z komentáře https://github.com/radkovo/CSSBox/issues/60#issuecomment-702721273.

Parametr perzistence koresponduje s parametrem perzistence u vykreslovacího jádra Puppeteer a jde o určitý režim úsilí, které je vyvinuto při snaze vytěžit ze stránky maximum informací[3]. Parametr nabývá hodnot 0 až 3, kde výchozí hodnotou je hodnota 1, a čím je číslo větší, tím větší úsilí je vynakládáno při snaze získat kompletní informace. Ve zkratce lze tyto režimy popořadě popsat jako:

- `-s 0` – čekání na událost `DOMContentLoaded`,
- `-s 1` – čekání na událost `load`,
- `-s 2` – čekání na 2 a méně aktivních síťových spojení a
- `-s 3` – čekání na 0 aktivních síťových spojení.

7.2 Segmentace oblastí

Segmentace oblastí jako taková není pro aplikaci VizGet zcela klíčová věc. Využita je zejména z důvodu, že další operátory aplikačního rámce VizGet vyžadují pro svou práci oblasti ze stromu oblastí, které vznikají právě procesem segmentace. Z toho důvodu byl využit nejjednodušší dostupný segmentační algoritmus nazvaný *Basic page segmentation* a to tak, jak byl popsaný v sekci 4.5. Tento algoritmus má tu výhodu, že listové uzly prakticky odpovídají nejmenším boxům na stránce a může tak být dosaženo vyhledávání vzorů a jejich prostorových vztahů v co nejmenších boxech.

7.2.1 Detekce víceřádkových oblastí

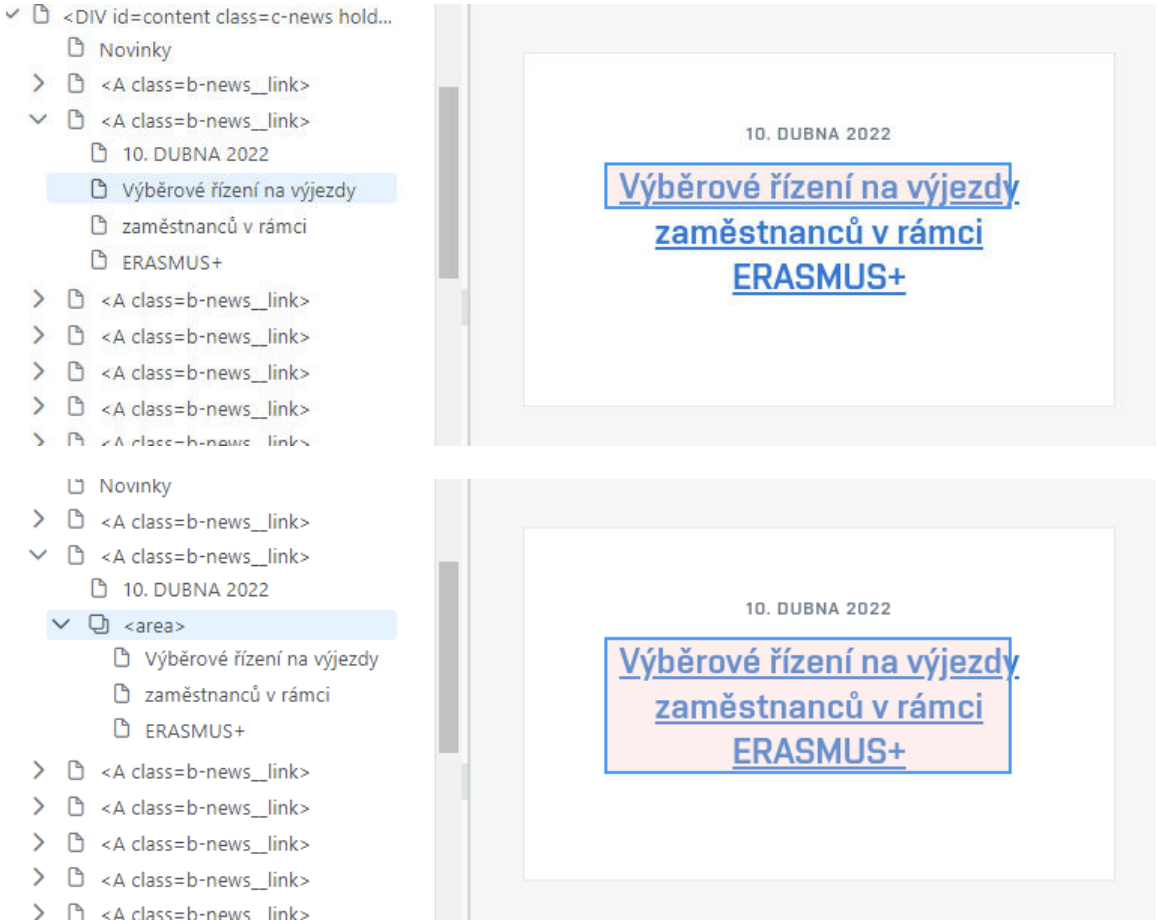
Výjimku v důležitosti při segmentaci tvoří nutnost zpracovávat i boxy, které obsahují text přes více řádků. Boxy v aplikačním rámci FitLayout by totiž při správném fungování měly obsahovat pouze text na jednom řádku a původně víceřádkový text je tak rámcem rozdělen do více boxů a tím pádem i do více oblastí. Typickým příkladem, který by uživatel mohl chtít zpracovávat, jsou víceřádkové popisky produktů, perexy článků, texty novinek, odstavce článků a další.

Pro účel spojení boxů do větších umělých oblastí (dále jen *superoblasti*) existuje v aplikačním rámci FitLayout operátor homogenních listů `HomogeneousLeafOperator` jakožto speciální případ operátoru „superoblastí“ `SuperAreaOperator`. Tento operátor funguje na principu detekce sousedících listových oblastí, které nejsou vizuálně oddělené a mají stejný styl textu.[4] Tyto detekované oblasti operátor následně slučuje do superoblastí, čímž efektivně vytváří z oblastí zhruba reprezentující jednotlivé řádky textu větší oblasti, které již zhruba reprezentují kýžené všechny řádky textu. Příklad víceřádkové oblasti před a po použitím operátoru je možné vidět na obrázku 7.1.

Problémem jsou pak pouze řádky textu, které nemají stejný vizuální styl, ačkoliv pohledem lidského čtenáře reprezentovaly pokračování extrahovaného textu. Takové řádky jsou aplikací VizGet stále považovány za jednotlivé oblasti a žádaná extrakce z nich tak není přímo podporována, resp. musí být činěna s větším rozmyslem. Do budoucna se jedná o jedno z míst aplikace, které by mohlo být vylepšeno.

[3] Vyplývá z dokumentace na stránce https://github.com/FitLayout/FitLayout/tree/main/fitlayout-render-puppeteer#services-and-parameters.

[4] Vyplývá z dokumentace ve zdrojového kódu aplikačního rámce FitLayout.



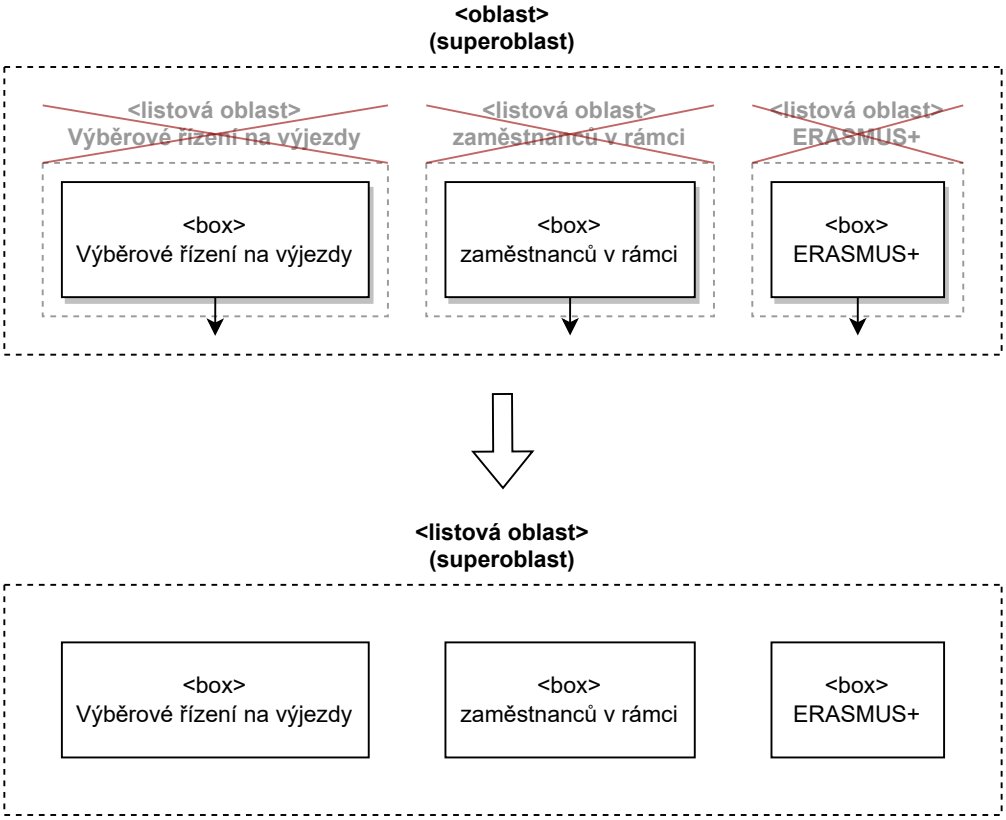
Obrázek 7.1: Příklad víceřádkové oblasti před a po sloučení pomocí operátoru homogenních listů na snímku obrazovky z nástroje `https://layout.fit.vutbr.cz/`. V levém panelu je vidět strom oblastí po segmentaci a sloučení.

7.2.2 Přesun boxů do superoblastí

Vytvořené superoblasti reprezentující víceřádkový text nejsou listovými oblastmi, ale obsahují právě ony původní oblasti, kdy až tyto oblasti jsou listové. Většina značkovačů oblastí popsaných v následující sekci ale pracuje pouze s listovými oblastmi a nemělo by tak takovéto vytváření superoblastí žádný význam. Navržen byl proto vlastní operátor `CollapseArtificialAreasOperator` provádějící určité sbalení oněch superoblastí nebo také převod nelistových superobolastí na listové superoblasti.

Operátor funguje tak, že vyhledá všechny vytvořené superoblasti, pro které platí, že všichni přímí potomci této oblasti jsou listovými oblastmi. Pro každou takovouto superoblast následně operátor přesune všechny boxy ze všech jejich potomků do právě zpracovávané, a tedy nadřazené superoblasti. Potomci, ze kterých byly boxy přesunuty, jsou poté z nadřazené superoblasti odstraněny, čímž z původně nelistové superoblasti vznikne oblast listová. To má za následek, že je pak již možné tyto superoblasti standardně zpracovávat existujícími značkovači oblastí.

Jak zhruba tento proces přesunu boxů z listových oblastí do superoblastí probíhá ukazuje ilustrační obrázek 7.2.



Obrázek 7.2: Ilustrace přesunu boxů do superoblastí pro superoblast z obrázku 7.1. Oblasti uvnitř superoblasti jsou zrušeny a jejich boxy jsou přesunuty přímo do superoblasti.

7.3 Označování oblastí

Poté, co jsou segmentací vytvořeny oblasti a předzpracovány superoblasti, přichází na řadu přiřazování štítků oblastem pomocí značkovačů oblastí. Samotná logika volání značkovačů oblastí je přenechána na aplikačním rámci FitLayout a na jeho operátoru pro přiřazování štítků entitám `TagEntitiesOperator`. Operátoru jsou pouze předány všechny značkovače, které byly získány podle postupu popsaného v sekci 6.2.2, a o aplikaci značkovačů se operátor postará již sám.

V základu je v aplikaci VizGet možné použít všechny značkovače oblastí, které nabízí aplikační rámec FitLayout, tedy značkovače:

- data – `vizget:DateTagger`,
- času – `vizget:TimeTagger`,
- míst – `vizget:LocationsTagger`,
- osob – `vizget:PersonsTagger`.

Navíc přidává aplikace VizGet dva vlastní značkovače:

- značkovač používající regulární výrazy – `vizget:RegexTagger`,
- značkovač vět a slov – `vizget:SentenceTagger`.

7.3.1 Značkovač používající regulární výrazy

Značkovač používající regulární výrazy je nejvíc univerzální a zřejmě i nejpoužívanější značkovač, který aplikace VizGet nabízí. Jak už název napovídá, tento značkovač funguje na základě vyhledávání podřetězců textu v listových oblastech, kdy tyto podřetězce musí odpovídat uživatelem zadanému regulárnímu výrazu. Pokud tedy uživatel hledá například cenu produktu v českých korunách, může zadat regulární výraz `\d+(?= Kč)` značící jakékoliv číslice následované mezerou a zkratkou měny, a tento značkovač mu najde všechny takové výskyty.

Značkovač může fungovat prakticky ve dvou režimech: buď jako značkovač tzv. bílé listiny (anglicky *whitelist*), nebo jako značkovač tzv. černé listiny (anglicky *blacklist*), případně i kombinací obou těchto režimů. V případě bílé listiny je regulární výraz zadán vlastností `vizget:regex` a odpovídá to případu, kdy chce uživatel najít všechny výskyty odpovídající regulárnímu výrazu a žádné jiné. V případě černé listiny je regulární výraz zadán vlastností `vizget:blacklistRegex` a odpovídá to naopak případu, kdy chce uživatel najít všechny výskyty, které tomuto regulárnímu výrazu neodpovídají.

Režimy je možné i zkombinovat, v takovém případě se ale nejprve najdou všechny výskyty, které odpovídají bílé listině a až z těchto výskytů se odstraní výskyty, které odpovídají regulárnímu výrazu pro černou listinu. Myslet na toto pořadí je nutné v tom případě, kdy množiny řetězců, které jsou nalezeny každým z regulárních výrazů, obsahují nějaké společné řetězce, tedy nejsou disjunktní. V takovém případě nebudou kvůli zvolenému pořadí tyto společné řetězce nikdy nalezeny.

7.3.2 Značkovač vět a slov

Značkovač vět či spíše slov se snaží hledat listové oblasti, které jako text obsahují posloupnost slov, které dokáže značkovač rozeznat. Zároveň značkovač zvýhodňuje oblasti s textem, který začíná velkým písmenem, tedy takové oblasti, které zhruba vypadají jako věty. To se hodí v případech nějakých větných nadpisů, popisků produktů, názvů složených ze známých slov a podobně.

Ve výchozím nastavení značkovač nerozpoznává žádná slova, ale je nutné mu rozpoznávaná slova zadat. To je možné učinit jedním ze dvou způsobů: buď specifikací identifikátorů jazyků slovníků, ze kterých má značkovač slova brát, nebo skrze zadání jednotlivých rozpoznávaných slov. Tyto způsoby lze i libovolně kombinovat a lze tak například přidat k seznamu českých slov libovolná další slova, která použitý český slovník nezná.

Kromě seznamu povolených slov lze zadat podobně jako v případě značkovače používajícího regulární výrazy také seznam zakázaných slov, tedy slov, která nebudou počítána do celkového skóre či podpory, kterou značkovač oblasti přiřazuje. 20 % této podpory přitom tvoří informace, zda text oblasti začíná velkým písmenem a zbylých 80 % podíl všech rozeznaných slov oproti celkovému počtu slov v textu. Tato skutečnost je zachycena v následujícím vzorci, kde $s(A)$ značí podporu oblasti A, $w_r(A)$ počet rozeznaných slov a $w_n(A)$ celkový počet slov textu oblasti A:

$$u(A) = \begin{cases} 1 & \text{pokud začíná oblast } A \text{ velkým písmenem} \\ 0 & \text{jinak} \end{cases}$$

$$s(A) = 0.8 * \frac{w_r(A)}{w_n(A)} + 0.2 * u(A)$$

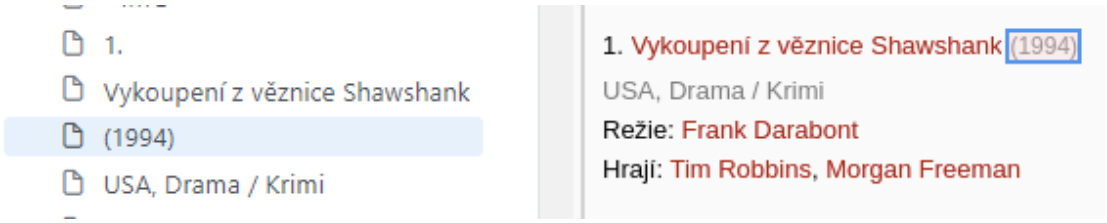
Značkovač lze i poměrně široce konfigurovat prostřednictvím vlastností jedince. Všechny konfigurovatelné parametry shrnuje tabulka 7.1.

Značkovač `vizget:SentenceTagger`	
Parametr	**Popis parametru**
`vizget:languages`	Identifikátory jazyků slovníků (`cs` a/nebo `en`)
`vizget:blacklistWords`	Seznam zakázaných slov textu oblasti
`vizget:whitelistWords`	Seznam povolených slov textu oblasti
`vizget:minWordCount`	Minimální počet libovolných slov textu oblasti
`vizget:maxWordCount`	Maximální počet libovolných slov textu oblasti
`vizget:wordCount`	Přesný počet libovolných slov textu oblasti
`vizget:minRecognizedWordCount`	Minimální počet rozeznaných slov textu oblasti
`vizget:maxRecognizedWordCount`	Maximální počet rozeznaných slov textu oblasti
`vizget:recognizedWordCount`	Přesný počet rozeznaných slov textu oblasti
`vizget:minLength`	Minimální počet znaků textu oblasti
`vizget:maxLength`	Maximální počet znaků textu oblasti
`vizget:length`	Přesný počet znaků textu oblasti

Tabulka 7.1: Výčet všech parametrů, kterými lze ovlivnit chování značkovače vět a slov.

7.4 Extrakce textu z oblastí

Pokud má oblast přiřazený nějaký štítek pomocí značkovačů z předchozího kroku, lze z takovéto oblasti extrahovat text, který má potenciál odpovídat příslušným datovým položkám. Ačkoliv je typicky extrahován celý text oblasti, existují případy, kdy se kýžená hodnota či hodnoty klidně i více datových položek nachází jen v části textu oblasti, jako je tomu například na obrázku 7.3.



Obrázek 7.3: Ukázka potřeby částečné extrakce při extrakci roku vydání filmu na snímku obrazovky z nástroje https://layout.fit.vutbr.cz/. Text oblasti obsahuje kolem roku nežádoucí závorky, ale extrahovat je potřeba pouze onen rok.

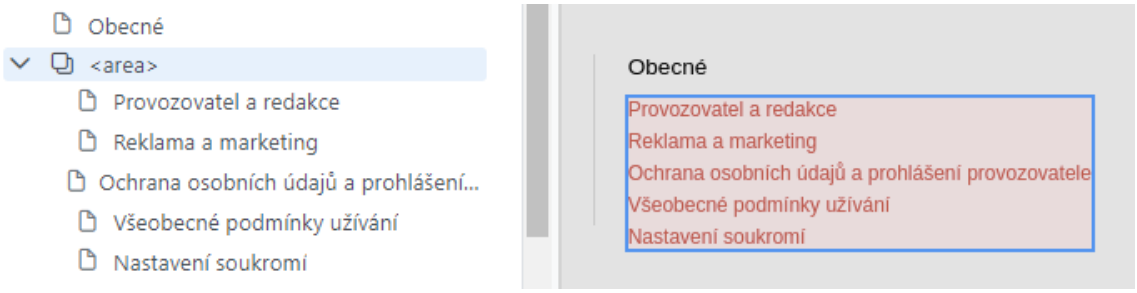
Pro tento případ existuje v aplikaci před samotným hledáním vzorů ještě proces, který extrahuje právě tyto části textu a rozděluje tak oblasti se štítkem na jeden nebo více bloků textu (anglicky *text chunks*). Tento krok zároveň transformuje pro další práci komplikovanější stromovou strukturu oblastí se štítky i bez nich na jednodušší plochou množinu těchto bloků textu, kdy tato množina obsahuje už pouze bloky textu odpovídající nějakému štítku.

Aplikační rámec FitLayout obsahuje přesně pro tento účel vlastní extraktor těchto bloků textu `TextChunksProvider`. Ten v současné době odvádí skvělou práci pro případy, kdy

jedna oblast odpovídá právě jednomu boxu a není tak nutné řešit formování textu oblasti z více boxů. Problém nastává s oblastmi, které odpovídají více řádkům, jelikož tyto oblasti obsahují více boxů, viz způsob formování těchto superoblastí v sekci 7.2.1. Extraktor v aplikačním rámci toto řeší tak, že extrahuje a zpracovává texty z každého boxu zvlášť, čímž efektivně neguje snahy o zpracování textů z boxů jedné oblasti jako jednoho velkého textu.

Kvůli těmto limitacím vznikl vlastní extraktor bloků textu `AreaTextChunksSource`, který vychází z funkce extraktoru z aplikačního rámce FitLayout. Odlišnost extraktoru spočívá kromě zpracovávání pouze oblastí s přiřazeným štítkem, které jediné jsou v aplikaci dále využity, zejména ve způsobu, jakým extraktor nahlíží na text oblasti. Tento extraktor totiž nezpracovává boxy oblasti jednotlivě, ale text všech boxů spojuje bez oddělovače do jednoho velkého textu. Tímto jednoduchým krokem je zajištěno, že na jednotlivé texty boxů ze superoblastí je nahlíženo právě jako na jeden velký text a jsou tedy korektně zpracovány tyto víceřádkové oblasti.

Spojování textu boxů bez použití oddělovače je něco, co by si zasloužilo ještě další zkoumání. Při experimentování totiž bylo zjištěno, že oddělovač – nejčastěji ve formě mezery – je již přítomen jako první znak v každém boxu superoblasti kromě boxu prvního. To má ten následek, že nelze použít standardní metodu aplikačního rámce FitLayout pro získání textu oblasti. Tato metoda totiž text z boxů odděluje jednou mezerou, která ale tvoří nežádoucí text navíc – nejčastěji tedy dvě mezery za sebou. Ovšem zajímavějším poznatkem je, že tento již existující oddělovač v boxech byl při pozorování přítomen pouze v případech, kdy superoblast vznikla skutečně z boxů, které by lidský čtenář považoval za pokračování víceřádkového textu, a nenacházel se naopak právě v těch boxech, které algoritmus pro tvorbu superoblastí ze sekce 7.2.1 vytvořil chybně.[5]



Obrázek 7.4: Ukázka chybně detekované superoblasti na snímku obrazovky z nástroje https://layout.fit.vutbr.cz/. Jednotlivé boxy nemají, na rozdíl od správně detekované superoblasti například na obrázku 7.1, mezi sebou žádnou mezeru.

[5]Diskuze k tomuto chování je dostupná na https://github.com/FitLayout/FitLayout/issues/9.

Kapitola 8

Algoritmy vyhledávání vizuálních vzorů

Krok samotného vyhledávání vizuálních vzorů přichází zhruba uprostřed funkce celé aplikace, a to po zpracování vstupu a přípravě před vyhledávání vizuálních vzorů. Aplikace již má dostupné informace, co za vlastnosti, resp. dvojice vlastností, se bude vyhledávat a ví o nich doplňující informace jako je například jejich kardinalita nebo vynucené reduktory. Zároveň je již provedeno základní získání obsahu ze stránky a jeho předzpracování ve formě segmentace, nalezení víceřádkových textů, hrubé označení oblastí štítky a extrakce bloků textu, které jsou již vstupem algoritmu vyhledávajícího vizuální vzory společně s informacemi o dvojicích vlastností.

Tato kapitola podrobně rozebírá přístupy, které jsou uplatněny při transformaci těchto vstupních dat na výsledné vizuální vzory v prezentované základní smyčce hledání. Představeny jsou i nové přístupy, které pomáhají nalézt ty správné a uživatelem žádané vizuální vzory. Tyto přístupy zahrnují zejména reduktory množin vizuálních vzorů a částečně i metriky, které pomáhají hodnotit kvalitu nalezených vzorů.

8.1 Základní smyčka hledání

Vyhledávání vizuálních vzorů probíhá pro každou dvojici vlastností naprosto odděleně, takže nedochází k žádnému ovlivnění vzorů pro jednu dvojici nalezenými vzory jiné dvojice, a to jak v negativním slova smyslu, tak bohužel ani v tom pozitivním. Pro případné budoucí vylepšení je tak toto místo, které by mohlo dále s přesností výsledků pomoci, kdy by se například statisticky určily vlastnosti identifikujících oblastí ze všech vzorů a na základě tohoto určení se přepočítaly nové a potenciálně lepší vzory.

Základní smyčka hledání, kterou je možné nalézt ve třídě `PatternFinder`, vychází ze smyčky podobné té z publikace [2] a prochází tak pro každou dvojici vlastností všechny možné trojice či konfigurace množiny vzorů, které se skládají z:

1. vodítek stylu identifikující oblasti,
2. vodítek stylu závislé oblasti a
3. prostorový vztah mezi oblastmi.

V každém průchodu jsou z každé takové konfigurace následně vypočítány odpovídající redukované množiny vzorů, tedy dvojic odpovídajících oblastí, a s těmito množinami je pracováno podle příslušného kroku či průchodu algoritmu.

Celá hledací smyčka se totiž v aplikaci VizGet provádí pro každou dvojici vlastností celkem dvakrát: v prvním případě za účelem zjištění globálních statistik o vzorech a ve druhém případě pro vybrání konfigurace s nejlepším skóre. Globální statistiky o vzorech momentálně zahrnují jedinou položku, kterou je maximální počet nalezených vzorů v množině. Tato informace je využita právě v metrikách, které se používají pro hodnocení množiny vzorů ve druhém průchodu algoritmu.

Po nalezení konfigurace s nejlepším skóre jsou konečně vypočítány vzory odpovídající této konfiguraci, ale do výpočtu jsou zahrnuty také identifikující i závislé oblasti, které mají přiřazeny příslušné štítky s menší podporou, než která je využita při běžném vyhledávání. Tyto nalezené vzory jsou sjednoceny spolu s nalezenými vzory pro ostatní dvojice vlastností do jedné velké množiny vzorů a tato množina je vrácena uživateli pro další zpracování.

8.2 Zpětný převod bloků textu na oblasti se stylem

Jak je možná patrné ze základní smyčky hledání, použitý algoritmus pracuje spíše na úrovni celých oblastí, ačkoliv vstupem algoritmu jsou bloky textu, které z těchto oblastí vznikly. Bloky textu tedy musí být před vyhledáváním převedeny zpět na jejich zdrojové oblasti (resp. instance třídy `TaggedArea`) s tím rozdílem, že je u nich zachována informace právě o extrahované části textu. U každé takovéto oblasti je také vypočítán její vizuální styl, který je dále využit pro vodítka stylů oblastí a obecně pro nalezení dalších oblastí, které mají stejný styl.

Základních pět složek stylu oblasti je stejných jako bylo popsáno v sekci 3.4.1, a to až na následující odlišnosti:

- jednotka velikosti písma je skutečně CSS pixel,
- uvažovány jsou všechny barvy písma a barvy pozadí všech boxů v oblasti,
- navíc jsou uvažovány také podtržení a přeškrtnutí textu opět na relativní škále od 0 do 1.

Kromě barev jsou všechny části tohoto stylu již připraveny v aplikačním rámci FitLayout, a stačí je pouze použít. Dodefinovat ale bylo potřeba porovnání, kdy jsou dva styly považovány za stejné neboli ekvivalentní. V aplikaci VizGet jsou styly ekvivalentní právě tehdy, když:

- jsou hodnoty průměrné velikosti písma, tučnosti, stylu, podtržení a přeškrtnutí matematicky zaokrouhlené na tisíciny ekvivalentní s příslušnými zaokrouhlenými hodnotami druhého stylu a zároveň
- jsou množiny prvního a druhého stylu s barvami písma ekvivalentní a zároveň
- jsou množiny prvního a druhého stylu s barvami pozadí ekvivalentní.

Není tedy připuštěna žádná odchylka v ani jedné z položek stylu až na odchylky, které vzniknou při zaokrouhlování.

8.3 Vodítka stylů oblastí

V konfiguraci množiny vzorů figuruje něco, co bylo nazváno jako vodítka stylů oblastí (anglicky *style hints*). Tato vodítka mají za úkol rozhodnout, jestli nějaký styl oblasti odpovídá

tomuto vodítku nebo nikoliv. Toho je využito při výpočtu množiny vzorů, kdy je požadováno, aby všechny oblasti odpovídaly příslušnému vodítku. Tedy všechny identifikující oblasti musí odpovídat právě zkoumanému vodítku pro identifikující oblasti a stejně tak musí závislé oblasti odpovídat právě zkoumanému vodítku pro oblast závislou.

Bez přičinění uživatele, tedy pokud uživatel u dané vlastnosti explicitně nezadá vodítko, si aplikace generuje vlastní interní vodítka pro každou oblast. To probíhá tak, že se ze všech oblastí zkoumané vlastnosti získají unikátní styly, tedy takové styly, pro které platí, že podle výše uvedeného algoritmu neexistuje v této unikátní množině stylů žádná dvojice stylů, která by byla považována ekvivalentní. Následně je každý tento unikátní styl převeden na vodítko, které rozhoduje, že jiný testovaný styl odpovídá tomuto vodítku právě tehdy, když testovaný styl odpovídá onomu unikátnímu stylu, ze kterého bylo vodítko vytvořeno. Tím je efektivně dosaženo situace, že jsou v rámci jedné konfigurace brány v potaz pouze oblasti se stejným stylem, a tedy případu jako v publikaci [2].

Pokud uživatel zadá jedno nebo i více vodítek, aplikace již negeneruje vlastní vodítka, ale použije pro danou vlastnost jediné vodítko, které vznikne logickým součtem všech vodítek zadaných uživatelem. To například znamená, že pokud uživatel zadá dvě vodítka, musí styl oblasti odpovídat alespoň jednomu z nich. Nezáleží přitom na tom, kterému z vodítek styl oblasti odpovídá, v rámci množiny vzorů můžu odpovídat kterémukoliv z nich.

Aplikace VizGet v základu nabízí uživateli pouze jeden typ vodítka, který může explicitně specifikovat. Jde o obecné vodítko stylů `vizget:StyleHint`, které ale nabízí rozsáhlé možnosti konfigurace. Testovaný styl oblasti bude odpovídat tomuto vodítku právě tehdy, když je alespoň n vlastností stylu ve vodítku nezadaných nebo sice zadaných, ale v rámci rozmezí stanoveného pro vlastnost příslušnou proměnnou ε.

Pokud tedy uživatel například zadá vodítku, že požaduje, aby oblast měla písmo o velikosti 13 pixelů s odchylku $\varepsilon = 3$, písmo o tučnosti 1 s výchozí odchylkou $\varepsilon = 0.5$ a hodnotu n ponechá rovnu 7, musí oblast, která má odpovídat danému vodítku, mít písmo o velikosti 10 až 16 pixelů, tučnost mezi 0.5 a 1 (maximální hodnota). Zbytek vlastností stylu mohou být libovolné, protože nebyly uvedeny a budou do hodnoty 7 počítány vždy bez ohledu na jejich hodnotu. V případě, že by uživatel snížil hodnotu n na 1, stačilo by, aby oblast měla klidně jen velikost písma nebo tučnost v příslušných rozsazích a styl bude i přesto vodítku odpovídat.

Všechny parametry uživatelem zadaného vodítka, které je možné libovolně konfigurovat, ukazuje tabulka 8.1.

8.4 Prostorové vztahy mezi oblastmi

Jednou z položek konfigurace množiny vzorů je i prostorový vztah mezi dvěma oblastmi, který musí všechny dvojice oblastí jedné množiny vzorů splňovat. Vyhodnocení těchto vztahů probíhá pro každou dvojici oblastí, které odpovídají vodítkům stylů z předchozí sekce. Hledá-li se tedy tento prostorový vztah mezi dvojicí vlastností jméno filmu a rok vydání, musí jméno filmu odpovídat právě testovanému vodítku stylů pro jméno filmu a stejně tak rok vydání musí odpovídat právě testovanému vodítku stylů pro rok vydání. Až tehdy poté je testován prostorový vztah mezi dvojicí takto odpovídajících oblastí.

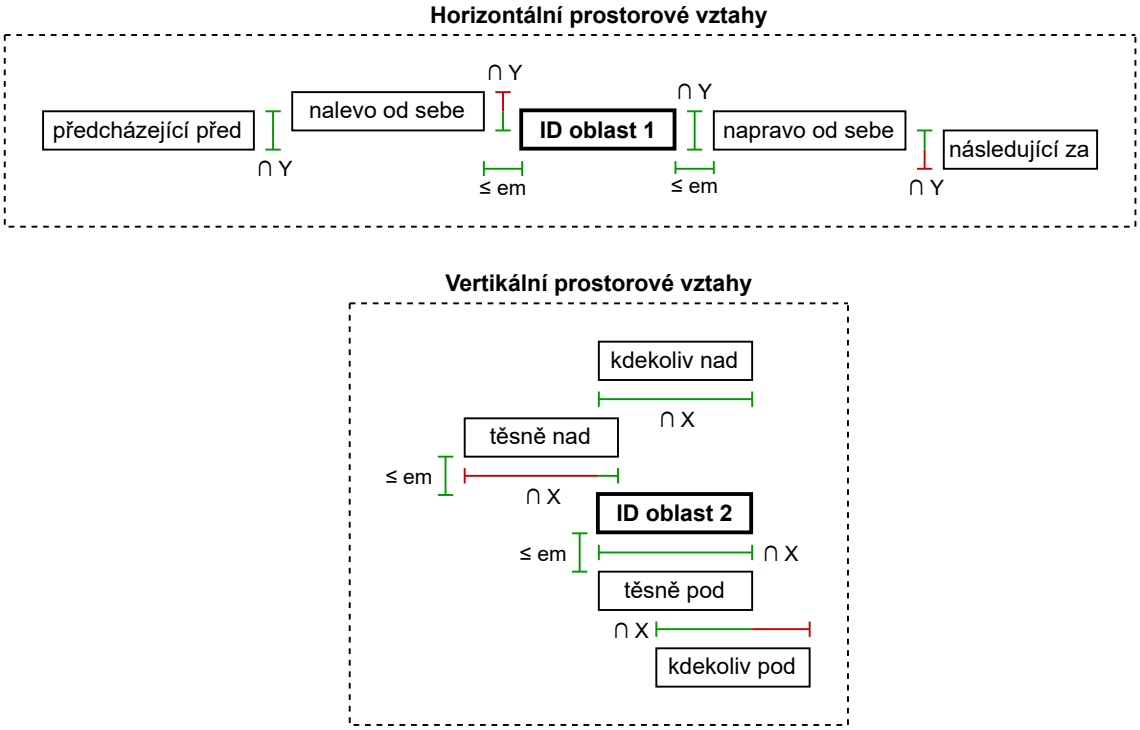
Aplikace VizGet definuje celkem 8 prostorových vztahů, které mohou být nalezeny nebo pevně specifikovány ve vstupním ontologickém datovém modelu. První 4 takto definované prostorové vztahy zhruba vychází z publikace [2] a zbylé 4 prostorové vztahy jsou pouhým opakem či jistou inverzí vztahů předchozích tak, jak byla popsána v sekci 3.4.2. Tyto prostorové vztahy či relace jsou definovány následovně:

Vodítko stylu `vizget:StyleHint`	
Parametr	**Popis parametru**
`vizget:fontSize`	Průměrná velikost písma
`vizget:fontWeight`	Průměrná tučnost písma
`vizget:fontStyle`	Průměrný styl písma
`vizget:lineThrough`	Průměrné přeškrtnutí písma
`vizget:underline`	Průměrné podtržení písma
`vizget:XEpsilon`	ε vlastnosti výše (X = např. `fontSize`)
`vizget:otherEpsilon`	ε všech vlastností výše kromě velikosti
`vizget:fontColors`	Barvy písma formátu #RRGGBB
`vizget:backgroundColors`	Barvy písma formátu #RRGGBB
`vizget:fontColorYEpsilon`	ε složky barvy písma (Y = `Red`/`Green`/`Blue`)
`vizget:backgroundColorYEpsilon`	ε složky barvy pozadí (Y = `Red`/`Green`/`Blue`)
`vizget:ZColorEpsilon`	ε všech složek barvy (Z = `font`/`background`)
`vizget:colorEpsilon`	ε všech složek všech barev

Tabulka 8.1: Výčet všech konfigurovatelný parametrů obecného vodítka stylů. Pro úsporu místa jsou zavedeny proměnné X, Y a Z, které je nutné nahradit příslušnými hodnotami v závorkách.

- oblasti jsou v relaci **napravo od sebe** právě tehdy, když je průnik jejich souřadnic na vertikální ose Y neprázdný a rozdíl mezi počáteční souřadnicí horizontální osy X druhé oblasti a koncové souřadnice horizontální osy X první oblasti je v intervalu $\langle 0; em \rangle$, kde em je velikost písma v pixelech počítána jako průměr velikostí písma z obou oblastí; inverzní relací je relace **nalevo od sebe**,
- oblasti jsou v relaci **následující za oblastí** právě tehdy, když je průnik jejich souřadnic na vertikální ose Y neprázdný a počáteční souřadnice horizontální osy X druhé oblasti je větší než koncová souřadnice horizontální osy X první oblasti; inverzí relací je relace **předcházející před oblastí**,
- oblasti jsou v relací **těsně pod oblastí** právě tehdy, když je průnik jejich souřadnic na horizontální ose X neprázdný a rozdíl mezi počáteční souřadnicí vertikální osy Y druhé oblasti a koncové souřadnice vertikální osy Y první oblasti je v intervalu $\langle 0; em \rangle$, kde em je velikost písma v pixelech počítána jako průměr velikostí písma z obou oblastí; inverzní relací je relace **těsně nad oblastí**,
- oblasti jsou v relaci **kdekoliv pod oblastí** právě tehdy, když je průnik jejich souřadnic na horizontální ose X neprázdný a počáteční souřadnice vertikální osy Y druhé oblasti je větší než koncová souřadnice vertikální osy Y první oblasti; inverzní relací je relace **kdekoliv nad oblastí**.

Inverzní prostorový vztah v kódu vzniká pouhým prohozením argumentu metody reprezentující neinverzní vztah, tedy $\overline{R}(A_1, A_2) = R(A_2, A_1)$. Jak takové vztahy mohou vypadat ukazuje obrázek 8.1. Identifikátory vztahů pro vynucení prostorového vztahu ve vstupním ontologickém datovém modelu potom shrnuje tabulka 8.2.



Obrázek 8.1: Ilustrace všech prostorových vztahů vztažených k příslušné identifikující oblasti, která je u každé kategorie označena tučně.

Název vztahu	**Identifikátor v ontologii**
Napravo od sebe	`vizget:OnRightSideRelation`
Nalevo od sebe	`vizget:OnLeftSideRelation`
Následující za oblastí	`vizget:AfterRelation`
Předcházející před oblastí	`vizget:BeforeRelation`
Těsně pod oblastí	`vizget:UnderRelation`
Těsně nad oblastí	`vizget:OverRelation`
Kdekoliv pod oblastí	`vizget:BelowRelation`
Kdekoliv nad oblastí	`vizget:AboveRelation`

Tabulka 8.2: Seznam všech prostorových vztahů a jejich identifikátorů pro vynucení prostorového vztahu ve vstupním ontologickém datovém modelu.

8.5 Hrubý výpočet množiny vizuálních vzorů

Výpočet jedné množiny vzorů pro danou konfiguraci probíhá v podstatě ve dvou krocích: nejprve je vypočítána hrubá množina vizuálních vzorů přibližně tak, jak to bylo popsáno v publikaci [2], a až poté jsou aplikovány příslušné reduktory z následující sekce pro danou dvojici oblastí. Hrubý výpočet množiny vizuálních vzorů tedy probíhá pomocí následujícího algoritmu:

1. Z předpočítané hashovací tabulky mapující štítky na oblasti, které mají tento štítek přiřazený, jsou vybrány oblasti pro štítek identifikujících oblastí a oblasti pro štítek závislých oblastí.

2. V množinách identifikujících i závislých oblastí jsou ponechány pouze ty oblasti, které odpovídají vodítkům stylů z konfigurace pro dané oblasti.

3. Množina vzorů je kartézský součin množiny identifikujících oblastí a množiny závislých oblastí vyjma případů, kdy je identifikující oblast ve dvojici shodná se závislou oblastí.

4. V množině vzorů jsou ponechány pouze ty vzory, jejichž oblasti mezi sebou mají z konfigurace daný prostorový vztah.

V ideálním případě se v množině vzorů nacházejí pouze relevantní vzory a jejich počet by tak tedy byl lineárně závislý na počtu závislých oblastí. Bohužel ale už samotné vyhodnocování prostorových vztahů je záležitostí dvojic oblastí, a tedy i v podstatě kvadratické časové složitosti, minimálně v průběhu výpočtu množiny vzorů. Pro zlepšení této skutečnosti by tak musely být nejspíš pročištěny již množiny identifikujících i závislých oblastí, bylo-li by to vůbec možné, případně alespoň využít paralelismus[1].

8.6 Reduktory množiny vizuálních vzorů

Jak již bylo zmíněno dříve, reduktory množiny vizuálních vzorů nastupují v těsném závěsu za hrubým výpočtem množiny vizuálních vzorů. Jejich cílem je odstranit z množiny vizuálních vzorů všechny případné nežádoucí vzory čili redukovat tuto množinu a dosáhnout tak lepších výsledků extrakce zejména v oblasti přesnosti (anglicky *precision*). Na druhou stranu ale zase může z množiny odstranit některé relevantní vzory a zhoršit tím úplnost (anglicky *recall*), takže je potřeba i s tímto nástrojem pracovat opatrně.

Samotný reduktor je definován jako funkce či metoda v jazyce Java, která obdrží na vstupu množinu vzorů a na výstup vyprodukuje opět množinu vzorů. Velikost této vyprodukované množiny vzorů musí být menší nebo rovna velikost vstupní množiny vzorů a všechny vzory v této výstupní množině musí být vzory z množiny vstupní. Tedy formálněji řečeno, musí platit, že sjednocení vstupní a výstupní množiny musí být rovno vstupní množině. Jakmile platí tyto podmínky, může již reduktor provádět téměř libovolnou činnost, ačkoliv je vhodné držet časovou složitost ideálně nanejvýš na lineární úrovni.

Protože každý reduktor dělá jen nějakou svou omezenou malou činnost, jsou reduktory skládány do sériových posloupností reduktorů (anglicky *pipeline*), kdy výstup jednoho reduktoru je vstupem dalšího reduktoru v řadě. Každý reduktor má pevně přiřazené pořadí či určitou prioritu a pokud je pak takový reduktor použit, jeho aplikace proběhne až po reduktorech s nižší číslem pořadí a zároveň před reduktory s vyšším číslem pořadí. Tedy všechny

[1] V současné době je vše v aplikaci vyhodnocováno sériově.

povolené reduktory (viz sekce 6.2.5) jsou vykonávány podle definovaného čísla v pořadí od nejmenšího čísla po největší.

Aplikace VizGet definuje celkem 7 reduktorů, z nichž ve výchozím nastavení jsou aktivní reduktory 4. Jeden z neaktivních reduktorů je prázdný reduktor, který na výstup vrací nezměněnou vstupní množinu vzorů a hodí se například pro testovací „vypnutí“ všech reduktorů přenastavením posloupnosti reduktorů. Zbylé dva neaktivní reduktory duplicitních oblastí byly nahrazeny lépe pracujícími reduktory nejbližších oblastí. Seznam všech těchto reduktorů spolu s pořadovými čísly a identifikátory v ontologii ukazuje tabulka 8.3.

#	**Název reduktoru**	**Identifikátor v ontologii**
0	Prázdný reduktor	`vizget:EmptyReducer`
1	Reduktor největších shluků	`vizget:BiggestClusterReducer`
2	Reduktor oblastí na obou stranách	`vizget:AreasOnBothSidesReducer`
3	*Reduktor identifikujících oblastí*	`vizget:IdAreasReducer`
4	*Reduktor závislých oblastí*	`vizget:DependentAreasReducer`
5	Reduktor nejbližších závislých oblastí	`vizget:ClosestDependentAreasReducer`
6	Reduktor nejbližších ID oblastí	`vizget:ClosestIdAreaReducer`

Tabulka 8.3: Seznam všech reduktorů s jejich pořadovými čísly, resp. prioritami, a identifikátory pro použití ve vstupním ontologickém datovém modelu. Kurzívou jsou vyznačeny ve výchozím nastavení neaktivní reduktory.

8.6.1 Reduktor oblastí na obou stranách

Reduktor oblastí na obou stranách vychází z předpokladu, že oblast vystupující jako identifikující už nemůže být použita jako oblast závislá a naopak. Je totiž třeba stále myslet na to, že každá z oblastí by měla reprezentovat jinou vlastnost ze vstupního ontologického datového modelu a shodné oblasti tak pravděpodobně představují chybu či nedostatek v přesném určení, jakou vlastnost daná oblast vlastně reprezentuje. Tyto oblasti na obou stranách (zkráceně duplicitní oblasti) jsou formálněji definované jako průnik množiny všech identifikujících oblastí všech vzorů z množiny vzorů s množinou všech závislých oblastí všech vzorů z množiny vzorů. Reduktor r je tedy pro množinu vzorů P a funkce $a_i(p)$ a $a_d(p)$ vracející identifikující, resp. závislou oblast vzoru p definován jako:

$$i(P) = \{a_i(p) \,|\, p \in P\} \cap \{a_d(p) \,|\, p \in P\}$$
$$r(P) = \{p \in P \,|\, a_i(p) \notin i(P) \wedge a_d(p) \notin i(P)\}$$

Je otázkou, co se vzory obsahující takovéto duplicitní oblasti správně udělat a zda by nebylo možné použít nějakou aproximaci, na které straně se vlastně oblast má správně nacházet, jako je tomu například v případě reduktorů nejbližších oblastí. Pro absenci lepšího řešení byl proto zvolen postup prostého odstranění všech vzorů, které tyto duplicitní oblasti obsahují na kterékoliv straně vzoru. Vyloučení stejných oblastí na obou stranách je už částečně prováděno i při hrubém výpočtu množiny vizuálních vzorů, ovšem toto odstraňování funguje pouze pro vzory, které mají stejnou oblast v rámci jediného vzoru a na rozdíl od reduktoru nehledí vůbec na skutečnost, že se například identifikující oblast může vyskytovat jako závislá v úplně jiném vzoru.

Uměle vytvořený příklad odstranění vzorů, které obsahují stejnou oblast na obou stranách, z množiny vzorů P může vypadat například takto:

$$P = \Big\{ \cancel{(\text{Pelíšky}, \cancel{\text{Jan Hřebejk}})}, \cancel{(\text{Sedm}, \cancel{\text{Pelíšky}})}, (\text{Sedm}, \text{David Fincher}) \Big\}$$

8.6.2 Reduktory duplicitních oblastí jedné strany

Jak již bylo zmíněno v sekci 6.2.4, je velmi důležité ve vstupním ontologickém modelu specifikovat kardinalitu dvojic vlastností. Na základě její znalosti je totiž možné sestavit reduktory, které s touto kardinalitou pracují, jako je tomu i v případě reduktorů duplicitních oblastí. Ačkoliv tyto reduktory nejsou již v aplikaci ve výchozím stavu aktivní, dali vzniknout reduktorům nejbližších oblastí, které jsou popsány v následující sekci.

Pro množinu vzorů reprezentující jednu z dvojic vlastností je možné identifikovat dvě omezení, která vznikla právě z kardinality tohoto vztahu a která jsou patrná i ze sekce 6.2.4:

1. oblast se může vyskytovat v multimnožině závislých oblastí nanejvýš jednou,
2. oblast se může vyskytovat v multimnožině identifikujících oblastí nanejvýš tolikrát, kolik je maximální hodnota kardinality.

Pro příklad, pokud existuje vztah s kardinalitou 1:24 jako je tomu u vztahu den–hodina, pak se stejná identifikující oblast reprezentující den vyskytne v multimnožině identifikujících oblastí nanejvýš 24krát, přičemž oblast pro hodinu vždy pouze jednou.[2]

Pro obě tato omezení vznikly samostatné reduktory: reduktor závislých oblastí využívající první omezení a reduktor identifikujících oblastí využívající druhé omezení. Oba reduktory uvnitř pracují shodně, ale oba nad jinými daty – první pracuje nad závislými oblastmi a druhý nad oblastmi identifikujícími – a s jinou kardinalitou podle příslušného omezení. Duplicitní oblasti v tomto kontextu jsou potom oblasti na příslušné straně dvojice, které se v multimnožině vyskytují vícekrát, než je v daném omezení povoleno.

I zde vyvstává otázka, co udělat se vzory obsahující duplicitní oblasti, a stejně jako v předchozí sekci i tyto reduktory šly původně z důvodu absence lepšího řešení cestou odstranění těchto vzorů. To ale znamenalo odstranění některých i validních výsledků, a proto byly vymyšleny reduktory popsané v následující sekci, které nahradily tyto původní reduktory. Opět uměle vytvořený příklad odstranění vzorů z množin vzorů P_1 a P_2 podle prvního, resp. druhého omezení může vypadat například takto, pokud by film mohl mít pouze jednoho režiséra:

$$P_1 = \Big\{ (\text{Pelíšky}, \text{Jan Hřebejk}), \cancel{(\text{Kmotr}, \cancel{\text{David Fincher}})}, \cancel{(\text{Sedm}, \cancel{\text{David Fincher}})} \Big\}$$

$$P_2 = \Big\{ \cancel{(\text{Pelíšky}, \cancel{\text{Jan Hřebejk}})}, \cancel{(\text{Pelíšky}, \cancel{\text{Frank Darabont}})}, (\text{Sedm}, \text{David Fincher}) \Big\}$$

8.6.3 Reduktory nejbližších oblastí

Reduktory nejbližších oblastí jsou ve své podstatě ekvivalentní s reduktory duplicitních oblastí jedné strany v tom smyslu, že využívají stejná dvě omezení a stejný výpočet duplicitních oblastí. Liší se ale v přístupu, co se vzory obsahující tyto duplicitní oblastmi

[2] Na první pohled může být matoucí, že identifikující oblast se několikrát opakuje a závislá nikoliv, při druhém zamyšlení ale toto dává smysl a není to tak chybou – ostatně více rozdílných závislých oblastí má jednu společnou identifikující oblast, která se tak vyskytuje při rozepsání či duplikování u každé závislé oblasti vícekrát a nikoliv naopak.

následně udělat. Jak již bylo řečeno, reduktory předchozí sekce všechny tyto vzory odstraňovaly, ovšem reduktory nejbližších oblastí tyto vzory neodstraňují, ale z množiny vzorů obsahující duplicitní oblasti vybírají maximální možný počet vzorů dle příslušného omezení, resp. odstraňují vzory, které jsou dle omezení navíc.

Výběr těchto vzorů k ponechání probíhá, jak už název reduktoru napovídá, podle vzdáleností mezi oblasti, resp. konkrétněji podle euklidovské vzdálenosti mezi centrálními body oblastí ve vzoru. Pokud tedy existuje například nějaký název filmu a k němu několik roků vydání, ponechá se pouze ten vzor, jehož vzdálenost mezi názvem a rokem je nejmenší. Z kardinality totiž vyplývá, že film může mít maximálně jeden rok vydání a že tedy ostatní nalezené roky vydání jsou nejspíše parazitní roky vydání například z jiných vzorů.

Aplikace VizGet nabízí opět dva takovéto reduktory pro každou stranu dvojice: reduktor nejbližších závislých oblastí, který vybírá n závislých oblastí dle kardinality pro každou identifikující oblast, a reduktor nejbližších identifikujících oblastí, který naopak vybírá jedinou identifikující oblast pro každou závislou oblast. Příklad se stejnými množinami vzorů P_1 a P_2 jako v předchozí sekci, kdy je tentokrát vybrána jedna oblast a zbytek odstraněn, může vypadat například takto pro vzdálenosti oblastí uvedené u dvojic:

$$P_1 = \left\{(\text{Pelíšky}, \text{Jan Hřebejk})_{25}, \cancel{(\text{Kmotr}, \text{David Fincher})_{100}}, (\text{Sedm}, \text{David Fincher})_{20}\right\}$$

$$P_2 = \left\{(\text{Pelíšky}, \text{Jan Hřebejk})_{25}, \cancel{(\text{Pelíšky}, \text{Frank Darabont})_{150}}, (\text{Sedm}, \text{David Fincher})_{20}\right\}$$

8.6.4 Reduktor největších shluků

Posledním přidaným reduktorem je reduktor největších shluků, který je založený na jednoduché shlukové analýze. Reduktor vznikl na základě pozorování, že euklidovské vzdálenosti mezi oblastmi jednotlivých vzorů zůstávají ve spoustě případů přibližně stejné a ty vzdálenosti, které se liší od většiny, pravděpodobně patří parazitním vzorům, které nejsou v množině vzorů očekávané. Tento předpoklad je obzvlášť patrný u tabulek, kdy například vzdálenosti zarovnaných stran mezi oblastmi pro název předmětu a zkratku předmětu budou stejné a budou tak tvořit jeden shluk vzdáleností, a vzdálenosti názvu předmětu a ústavu předmětu budou sice opět stejné, ale rozdílné od předchozích vzdáleností, a budou tedy tvořit vlastní shluk.

Příklad s předměty byl vybrán z toho důvodu, že zkratka předmětu a jeho ústav vypadají velmi podobně a lze je regulárním výrazem rozlišit jen velmi složitě. Zároveň tyto datové položky mají stejný styl a odpovídají i stejnému prostorovému vztahu, a z toho důvodu se mohou vyskytnout v množině vzorů současné. Proto vznikl právě reduktor největších shluků, který předpokládá, že sice tento případ může nastat, ale použitý značkovač oblastí přece jen do množiny vzorů zahrne o trochu více správných vzorů než těch nesprávných.

Samotná shluková analýza probíhá nad euklidovskými vzdálenostmi mezi centrálními body oblastí vzorů, tedy nad stejnými vzdálenosti, jako v případě reduktorů nejbližších oblastí. Shlukování vychází z algoritmu [12], který má být podle autora obdobou algoritmu DBSCAN. Celý algoritmus funguje následovně:

1. Shlukované body, tedy v případě reduktoru vzdálenosti, jsou seřazeny od nejmenší po největší.

2. Vytvořen je jediný shluk o jediném první prvku ze seřazené posloupnosti.

3. Zbylé body jsou postupně popořadě procházeny a přidávány do aktuálního shluku do té doby, dokud je rozdíl mezi aktuálním a předchozím bodem menší než předem zvolené ε.

4. Jakmile je rozdíl větší nebo je cyklus u konce, je aktuální shluk finalizován, uzavřen a uložen do množiny nalezených shluků, jako aktuální shluk je vytvořen nový shluk s jediným aktuálním prvkem.

5. Výsledkem je množina nalezených shluků.

První otázkou, kterou je nutné vyřešit, je volba hodnoty ε, tedy určitý práh, kdy je mezera mezi vzdálenostmi natolik velká, že už algoritmus začne považovat bod za patřící do jiného shluku. Hodnota ε v případě reduktoru představuje vzdálenost v pixelech a vzhledem k absenci lepšího řešení byla experimentálně zvolena hodnota 30. Uživatel přitom může tuto hodnotu přepsat ve vstupním ontologickém datovém modelu vlastností reduktoru `vizget:epsilon`, viz sekce 6.2.1.

Druhou a poslední otázkou k vyřešení je volba shluku nebo i shluků, které obsahují očekávané vzory. Jak již bylo řečeno, reduktor je založen na předpokladu, že správných vzorů bude většina a shluk s největším počtem by tedy měl odpovídat očekávaným vzorům. To platí pouze pro kardinalitu 1:1, ovšem pro kardinalitu 1:n, kde n je větší než 1, porostou vzdálenosti pravděpodobně lineárně s tím, jak se další a další závislé oblasti budou postupně vzdalovat od své identifikující oblasti. Proto bylo zvoleno, že se do výsledku reduktoru dostane nanejvýš n největších shluků, kdy budou v pro reduktor „nejhorším“ případě nekonečného n vybrány vzory ze všech shluků, a v „nejlepším“ případě právě jeden největší shluk.

Pro množinu vzorů P s názvy a zkratkami předmětů by mohl uměle vytvořený příklad, kde jsou zkratky a názvy zarovnány ve sloupcích tabulky, vypadat například takto:

$$P = \left\{ (\text{Algoritmy}, \text{IAL})^{1}_{310}, \cancel{(\text{Algoritmy}, \text{K3D})^{2}_{1000}}, (\text{Bioinformatika}, \text{BIF})^{1}_{300} \right\}$$

8.7 Metriky hodnocení množiny vizuálních vzorů

Jakmile je hotový sběr globálních statistik o vzorech, tedy momentálně pouze maximální počet již redukovaných vzorů v množině, přichází na řadu hodnocení jednotlivých množin vizuálních vzorů. K tomu jsou v aplikaci VizGet využívány tzv. metriky, kde metrika je funkce, resp. metoda jazyka Java, která na vstupu dostává množinu vzorů spolu s globálními statistikami a na výstupu vrací desetinné číslo z intervalu $\langle 0; 1 \rangle$. Vrácená hodnota 0 znamená naprosto nekvalitní množinu vzorů, resp. množinu vzorů, která měřené metrice vůbec neodpovídá, a naopak hodnota 1 znamená nejkvalitnější množinu vzorů.

Každá metrika má navíc svou určitou definovanou váhu, kterou je možné změnit vlastností jedince `vizget:weight` pro každou extrahovanou dvojici vlastností. Tato váha určuje podíl metriky na celkovém skóre pro danou dvojici vlastností a celkové skóre množiny vzorů je tak určeno jako vážený průměr všech vrácených hodnot ze všech metrik konkrétní dvojice.

V aplikaci VizGet byly definovány celkem 4 metriky, které jsou všechny ve výchozím nastavení povoleny. Jejich seznam spolu s vahami a identifikátory ontologie ukazuje tabulka 8.4, přičemž podrobněji jsou všechny metriky rozebrány v následujících sekcích.

Váha	Název metriky	Identifikátor v ontologii
65 %	Metrika nejlepšího počtu vzorů	`vizget:PatternCountMetric`
20 %	Metrika unikátní hodnoty identifikující oblasti	`vizget:UniqueIdMetric`
10 %	Metrika unikátních slov identifikující oblasti	`vizget:UniqueWordsMetric`
5 %	Metrika podpor značkovačů	`vizget:TagSupportMetric`

Tabulka 8.4: Seznam všech dostupných metrik, jejich vah a identifikátorů pro použití ve vstupním ontologickém datovém modelu.

8.7.1 Metrika nejlepšího počtu vzorů

Metrika nejlepšího počtu vzorů v množině vychází z předpokladu, že již redukovaná množina bude mít nejvíce vzorů ze všech testovaných množin. Ve své podstatě jde o stejný předpoklad jako v publikaci [2], ve které je taktéž maximalizován počet vzorů v množině při výběru nejlepší množiny. Samotný vzorec metriky je jednoduchý podíl počtu vzorů v testované množině vůči nejlepšímu počtu vzorů (oné globální statistice), kde nejlepší množina je množina s maximální počtem vzorů a nejhorší množina neobsahující žádný vzor:

$$m(P) = \frac{|P|}{|P_{max}|}$$

8.7.2 Metrika unikátní hodnoty identifikující oblasti

Jak už bylo řečeno v sekci 6.2.3, aplikace VizGet počítá s tím, že je pokud možno jako identifikující vlastnost zvolena taková vlastnost, která je pokud možno co nejvíce unikátní pro danou třídu. Takové zvýhodňování probíhá právě pomocí metriky unikátní hodnoty identifikující oblasti. Vzhledem k použití více metrik není celé vyhodnocování kvality množiny vzorů na tomto předpokladu závislé, ale právě z určité části dokáže metrika množinu s vyhovujícími vzory zvýhodnit i přes nižší počet vzorů nebo naopak vyloučit množinu s větším počtem vzorů, ale menší unikátností.

Vzorec metriky je tedy podíl unikátních hodnot unikátních identifikujících oblastí vůči celkovému počtu unikátních identifikujících oblastí v množině vzorů, kde nejlepší množina je množina bez opakujících se hodnot identifikujících oblastí a nejhorší množina obsahující pouze jedinou hodnotu identifikující oblasti. Použití unikátních identifikujících oblastí je důležité, protože v případě kardinality 1:n, kde n je větší než 1, se mohou, jak už bylo dříve rozebíráno, identifikující oblasti opakovat, čímž by docházelo ke znehodnocování metriky pro vyšší počty opakování.

Pro množinu vzorů P, funkci $a_i(p)$ vracející identifikující oblast vzoru p a funkci $v(a)$ vracející textovou hodnotu oblasti a je pak metrika $m(P)$ definována následujícím vzorcem:

$$\begin{aligned} a(P) &= \{a_i(p) \,|\, p \in P\} \\ v'(P) &= \{v(a) \,|\, a \in a(P)\} \\ m(P) &= \frac{|v'(P)|}{|a(P)|} \end{aligned}$$

8.7.3 Metrika unikátních slov identifikujících oblastí

Podobně jako předchozí metrika i metrika unikátních slov identifikujících oblastí pracuje s předpokladem určité unikátnosti hodnot identifikujících oblastí. Tentokrát však není povětšinou brán celý text identifikující oblasti jako celek, ale počítána je unikátnost jednotlivých slov. Často je totiž extrahovaná identifikující oblast formována ze slov či větných celků, o nichž se předpokládá, že budou mít vysokou vypovídající hodnotu a nebudou tedy slova příliš často opakovat.

Slovo v metrice je definováno jako libovolná posloupnost čísel a písmen dle UTF-8 a oddělena jsou čímkoliv, co není číslo nebo písmeno. Při výpočtu se nehledí na velikost písmen, malá písmena jsou tak ekvivalentní svým velkým variantám a naopak. Vzorcem metriky je poté podíl unikátních slov unikátních identifikujících oblastí vůči celkovému počtu slov unikátních identifikujících oblastí, kde nejlepší množina je množina bez opakujících se slov v identifikujících oblastech a nejhorší množina obsahující ve všech oblastech velký/nekonečný počet výskytů jediného slova. Formálněji zapsáno vypadá vzorec metriky $m(P)$ s množinou vzorů P, kde funkce $w(P)$ vrací množinu slov v identifikujících oblastech a funkce $w_n(P)$ celkový počet i opakujících se slov v identifikujících oblastech, takto:

$$m(P) = \frac{|w(P)|}{w_n(P)}$$

Jak je možné odhadnout, v případě, že každá oblast má podle definice pouze jedno slovo a tato slova jsou všechna unikátní, chová se tato metrika velmi podobně jako metrika předchozí. Význam tak má především právě u víceslovných větných celků, jakým může být například název filmu.

8.7.4 Metrika podpor značkovačů

Některé značkovače nevrací pouze jedinou hodnotu podpory, ale obsahují určité stupně podpory, tedy jak pravděpodobně oblast obsahuje danou datovou položku. Vznikla proto metrika podpor značkovačů, která se snaží tuto skutečnost reflektovat a upřednostňovat ty oblasti, které mají co největší hodnoty podpory štítků pro identifikující i závislé oblasti. Vzorec této metriky je tak jednoduchý aritmetický průměr všech podpor všech oblastí všech vzorů a pro množinu vzorů P může tedy hodnota metriky $m(P)$, kde funkce $s_i(p)$ a $s_d(p)$ vrací podporu identifikující, resp. závislé oblasti vzoru p, vypadat následovně:

$$\begin{aligned} s(p) &= \{s_i(p), s_d(p)\} \\ s'(P) &= \bigcup_{p \in P} s(p) \\ m(P) &= \frac{1}{|s'(P)|} \sum_{x \in s'(P)} x \end{aligned}$$

Ačkoliv při interním testování se neukázalo, že by tato metrika nějak pomáhala nebo škodila, a to ani například u značkovače vět a slov, který obsahuje spojitou složku, metrika v aplikaci zůstala povolená a zakázat ji je možné stejně jako ostatní metriky případně kdykoliv později.

Kapitola 9

Generování výstupních dat

Hlavním výstupem aplikace jsou samotné extrahované informace, které budou v drtivé většině případů zajímat uživatele nejvíce. Jak již bylo popsáno v sekci 5.4, tyto výstupní informace mají podobou jedinců, kteří odpovídají třídám ze vstupního ontologického datového modelu. Tyto jedince je potřeba nějakým způsobem z nalezených vizuálních vzorů vytvořit, což popisuje sekce 9.1 této minikapitoly.

Extrahované jedince je už následně možné serializovat do textového formátu, který je výstupem aplikace. Nad ním už může uživatel provádět například dotazy pomocí jazyka SPARQL, případě extrahovaná data využít ve svém programu či jinde dle potřeby. Generování tohoto souboru v požadovaném formátu popisuje sekce 9.2. A nakonec, pokud o to uživatel má zájem, může aplikace vygenerovat také vizualizaci vizuálních vzorů, tedy obrázek webové stránky s vyznačenými vizuálními vzory a prostorovými vztahy mezi nimi, což je popsáno v sekci 9.3.

9.1 Extrakce jedinců z vizuálních vzorů

Výstupem fáze vyhledávání vizuálních vzorů je množina všech nalezených vzorů všech datových vlastností. Vzhledem k tomu, že v aplikaci neprobíhá nějaké následné zpracování či čištění dat, je už jednoduché takový výstup převést na jedince tříd vstupního ontologického datového modelu. Nutné je tedy nejprve převést všechny unikátní identifikující oblasti na jedince a následně přiřadit příslušným jedincům hodnoty závislých oblastí.

Vytvoření jedince z identifikující oblasti ve třídě `IndividualExtractor` znamená v první řadě zavolání příslušné tovární metody ontologické třídy aplikačního rámce Apache Jena, která vyžaduje identifikátor jedince. Ten je vytvářen ze zdrojové URL adresy, ze které probíhá extrakce, a to nastavením jejího fragmentu na lokální jméno ontologické třídy, pro kterou se identifikátor vytváří, spojovník a sekvenční číslo specifické opět pro ontologickou třídu. Tedy například pro první procházený film v žebříčku z webu ČSFD.cz může identifikátor vypadat následovně: `https://www.csfd.cz/zebricky/filmy/nejlepsi/#Movie-1`.

Nově vytvářenému jedinci je dále přiřazena hodnota identifikující oblasti, která pochází z bloku textu pro danou oblast, a také hodnota anotační vlastnosti[1], která obsahuje URL adresu, ze které byl jedinec extrahován. Takto vytvořený jedinec je poté uložen do hashovací tabulky, která mapuje identifikující oblast na právě vytvořeného jedince. Toto uložení je důležité pro pozdější vyhledávání jedinců podle identifikujících oblastí, neboť tyto oblasti

[1] V kódu `vizget:sourceUrl`.

se ve vzorech několikrát opakují a hodnoty závislých oblastí je tedy nutné koncentrovat do jediného jedince pro danou identifikující oblast.

Toto prvotní vytváření jedinců probíhá postupně při procházení všech vzorů tak, že pokud pro danou identifikující oblast jedinec ještě neexistuje, je vytvořen, jinak je získán již existující jedinec a je tedy zaručena ona koncentrace do jednoho jedince. Protože závislé oblasti jsou všechny unikátní, resp. je nutné se k nim jako k unikátním oblastem chovat, přiřazení hodnoty každé závislosti příslušnému jedinci je prováděno u každého vzoru. Zároveň tak není nutné hledět na neopakování hodnoty jako v případě identifikujících oblastí.

Přiřazovaná hodnota závislé oblasti se liší podle typu vlastnosti. V případě datové vlastnosti je situace jednoduchá a jde stejně jako u identifikující oblasti o přiřazení příslušného textu oblasti vzniklého z bloku textu. V případě objektové vlastnosti je situace mírně složitější, protože hodnotou už není prostý literál, ale odkaz na jiného jedince. Protože závislá oblast v této situaci reprezentuje identifikující oblast pro jiného jedince, stačí pouze vytvořit nebo případně získat již vytvořeného jedince podle stejného algoritmu jako u čistě identifikující oblasti a přiřadit takto vytvořeného či získaného jedince jako hodnotu vlastnosti.

Tímto jednoduchým způsobem jsou vytvořeni všichni jedinci ze všech vzorů, kde platí, že všechny identifikující oblasti odpovídají právě jednomu jedinci, tito jedinci jsou řádně a unikátně identifikováni, z anotační vlastnosti zdrojové adresy lze zjistit, z jaké URL adresy byli extrahováni, a hlavně mají správně přiřazeny všechny hodnoty vlastností jak z textů identifikujících oblastí, tak i z oblastí závislých, přičemž funguje i propojení objektových vlastností.

9.2 Generování souboru požadovaného formátu

Protože vytváření jedinců a vůbec celé zpracování ontologického vstupního datového modelu, tedy souhrnně modelování, probíhá výhradně pomocí aplikačního rámce Apache Jena, je vytvoření výstupního souboru třídou `RdfOutput` ve formátech podporovaných aplikačním rámcem pouhé triviální zavolání příslušné metody s předaným modelem, formátem a cílem zápisu. Ve výchozím nastavení se v aplikaci VizGet model převádí na formát Turtle s možností uživatele zvolit si libovolný formát podporovaný[2] aplikačním rámcem pomocí parametru `--formatOut`. Výstup je poté zapisován na standardní výstup nebo do souboru zadaného parametrem `-o` či `--output`.

Mírnou nevýhodou tohoto přístupu je nízká konfigurovatelnost podoby výstupu a zejména fakt, že výstup kromě nutných informací obsahuje i spoustu pomocných, a ne úplně vyžadovaných informací. Zároveň jsou k jedincům, jejichž vlastnosti jsou tou zásadní výstupní informací, přimíchány také informace o vstupním ontologickém datovém modelu i o ontologii aplikace VizGet. Hlavní informace jsou ale ve výstupu obsaženy, a tedy tyto informace navíc nemají vliv na úspěšnost extrakce a znamená to tak spíše pouze horší případnou orientaci uživatele ve výstupu aplikace.

9.3 Generování vizualizace vizuálních vzorů

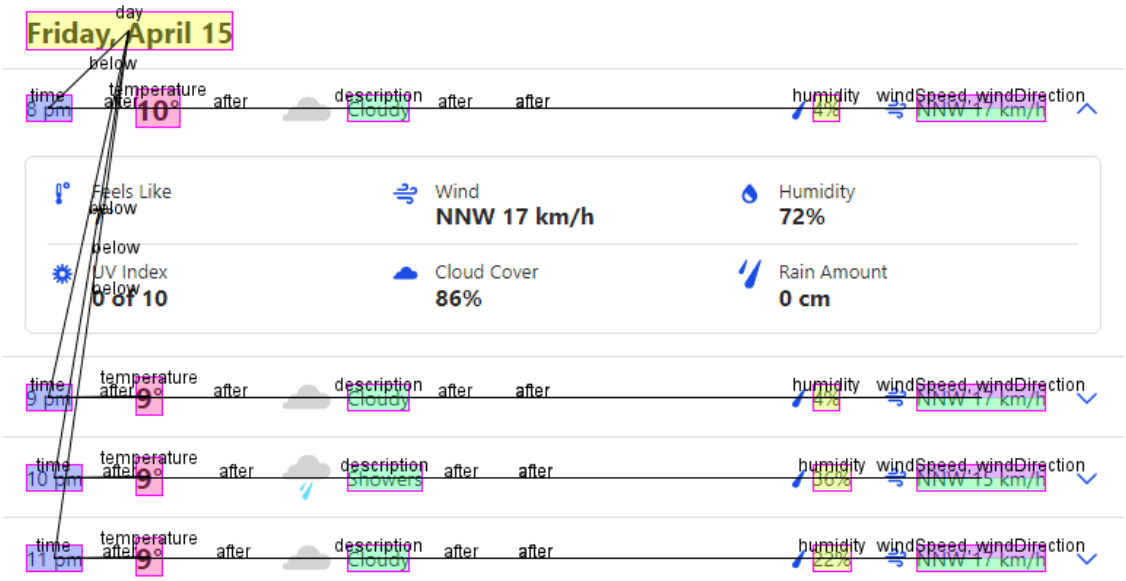
Volitelným výstupem aplikace VizGet je obrázek vykreslené webové stránky, na který je nanesena vrstva s vizualizací vizuálních vzorů. Tento výstup je spíše jenom experimentálním doplňkem aplikace a slouží zejména pro rychlou kontrolu člověkem. Samotné vykreslování

[2] Viz identifikátory konstant typu `RDFFormat` na https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/riot/RDFFormat.html.

grafických výstupů má podporu již v aplikačním rámci FitLayout, takže spousta vykreslovacích činností je otázka zavolání jedné metody. V repozitáři aplikačního rámce FitLayout je také dostupná ukázka vykreslování, kterou se tento výstup implementovaný ve třídě `ImageOutput` inspiroval.

Vykreslovaná stránka jakožto pozadí obrázku pochází ze „snímku obrazovky“, který produkuje použité vykreslovací jádro jako obrázek ve formátu PNG a je do obrázku zakreslována přímo aplikačním rámcem FitLayout. Vizualizace vzorů na tento obrázek nanášená se poté skládá pro každý vzor a každou oblast vzoru z vizualizace všech štítků oblasti, vykreslení hranic oblasti a z vykreslení všech názvů oblasti ve formě lokálního názvu datové položky, která je danou oblastí reprezentována. Nakonec už je vykreslena nad to vše vrstva s prostorovými vztahy mezi oblastmi vzoru společně s názvy těchto vztahů.

Aplikačním rámcem FitLayout jsou kromě stránky napřímo vykreslovány už jen hranice oblastí jakožto fialový rámeček na souřadnicích oblasti. Zbytek tvoří vlastní implementace kreslící přímo čáry či texty na příslušné souřadnice, případně upravená forma vykreslování jako v případě dále popisované vizualizaci štítků oblasti. Vykreslování jako takové neřeší konflikty ve formě překrývajících se částí vizualizace, jednotlivé prvky jsou umísťovány na absolutní pozici a přepisují původní pixely. Proto se může stát a často i stává, že některé prvky bohužel nemusí být dobře čitelné. Pro jednoduchou vizualizaci je ale toto zobrazení dostatečné, jak ukazuje například celkem pěkně povedený obrázek 9.1.



Obrázek 9.1: Vizualizace části nalezených vzorů na stránce s počasím, viz test 10.8. Barevně jsou vyznačeny jednotlivé štítky fialově ohraničených oblastí a čáry reprezentují vztahy mezi těmito oblastmi.

9.3.1 Vizualizace štítků oblasti

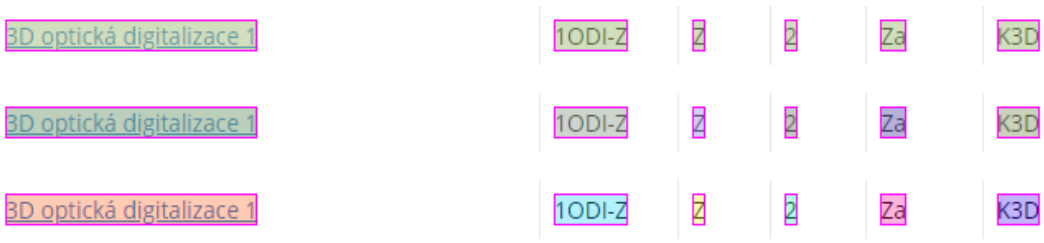
Vizualizace štítků oblasti spočívá ve vykreslení poloprůhledného barevného obdélníku na místo, kde se nachází oblast. Tuto činnost dokáže dělat již v základu aplikační rámec FitLa-

yout, nutné je tedy předem pouze seskupit všechny štítky podle oblastí, aby mohl FitLayout vykreslit všechny tyto štítky oblasti v barevných pruzích.

Mírným problémem se ukázala hashovací funkce použitá uvnitř funkce pro získání barvy, která při použití identifikátorů štítků ze vstupního ontologického datového modelu převáděla tyto identifikátory na prakticky totožné barvy. Přepsána proto byla konverze názvu na barvu, která využívá vhodnější hashovací funkci SHA-1, ze které si bere dva nejvýznamnější bajty, tyto bajty převádí operací modulo na hodnoty 0 až 359 a tuto hodnotu používá jako odstín barvy v modelu HSB. Hodnoty sytosti a jasu jsou zafixovány na maximálních hodnotách, průhlednost byla zvolena 30 %, kde 100 % znamená neprůhlednou barvu.

Použitím tohoto přístupu je dosaženo rozdílných a rozmanitých hodnot barev oproti standardní implementaci v aplikačním rámci FitLayout. Dále se od sebe barvy liší pouze odstínem, a nikoliv sytostí a jasem, takže je pro člověka snadnější zpozorovat rozdíl mezi barvami, ačkoliv i tak se občas ukáže obtížněji rozpoznatelná kombinace barev. Rozdíl oproti standardní implementaci ukazují obrázky 9.2 a 9.3.

Obrázek 9.2: Srovnání standardní implementace při použití celých URI jako hodnoty štítku (vlevo), standardní implementaci při použití lokálního názvu (uprostřed) a vlastní implementace (vpravo) s celými URI. Zobrazen je produkt z webu Alza.cz, viz test 10.6.



Obrázek 9.3: Srovnání standardní implementace při použití celých URI jako hodnoty štítku (nahoře), standardní implementaci při použití lokálního názvu (uprostřed) a vlastní implementace (dole) s celými URI. Zobrazen je vyučovaný předmět na FIT VUT, viz test 10.5.

Kapitola 10

Testování aplikace na webových stránkách

Kapitola o testování aplikace VizGet na reálných webových stránkách představuje zautomatizovanou metodiku testování, podle které probíhalo testování úspěšnosti extrakce aplikace již v průběhu vývoje aplikace, a metriky testování, podle kterých je tato úspěšnost extrakce hodnocena. Představeny jsou jednotlivé testovací případy, z nichž některé byly použity také pro změnu a vylepšení určitých částí implementace. Nakonec jsou prezentovány výsledky testování a ke každému výsledku je poskytnut krátký rozbor.

10.1 Metodika testování

Pro pohodlné testování úspěšnosti extrakce a z důvodu nestálosti a proměnnosti webových stránek, ze kterých jsou informace extrahovány, byla vytvořena částečně automatizovaná metodika, která je součástí testovací části zdrojového kódu aplikace VizGet ve formě vlastního testovacího rámce `ReferenceTester`. Metodika se skládá z ručního sestavení vstupního ontologické datového modelu, z části získání referenčního nebo také očekávaného modelu dat z testované webové stránky, resp. ze získání jednotlivých referenčních tvrzeních RDF navázaných na vytvořený vstupní ontologický datový model, a ze získání skutečného modelu dat, tedy výstupu aplikace VizGet. Tyto modely jsou následně pomocí specifikovaného algoritmu, který zahrnuje převod na jednotlivá tvrzení, porovnány a z tohoto porovnání jsou vypočítány metriky úspěšnosti extrakce.

Vstupní ontologický datový model je pro každý test a každou webovou stránku, ze které mají být informace extrahovány, ručně vytvořen tak, jak by to udělal uživatel při klasické využití aplikace VizGet. Takto vytvořený model je poté využit jak pro vstup aplikace VizGet, tak i pro navázání jedinců na třídy tohoto vstupního modelu při tvorbě referenčního modelu.

10.1.1 Získání modelů k porovnání

Referenční model je pro každý test získán ručním vytvořením specifického kódu, který:

- načte pomocí testovacího rámce ručně vytvořený ontologický datový model pro možnost tvorby navázaných jedinců,

- zpracuje testovanou webovou stránku tradičními metodami specifické extrakce; v aplikaci VizGet jde o použití knihoven jsoup[1] a Playwright[2],
- získá očekávané informace a předzpracuje je do očekávaného formátu,
- ze získaných očekávaných informací ručním voláním metod aplikačního rámce Apache Jena vytvoří jedince odpovídající třídám z vytvořeného ontologického datového modelu,
- předá vytvořený referenční model aplikačního rámce Apache Jena pro porovnání se skutečným modelem.

Skutečný model je pro každý test získán voláním hlavní vstupní metody aplikace VizGet, které je jako parametr předán vytvořená ontologický datový model, URL adresa k extrakci a další parametry, které jsou ručně zadané a specifické pro příslušný test. Serializovaný výstupní řetězec se skutečným modelem je automaticky testovacím rámcem převeden na model aplikačního rámce Apache Jena a tento model je následně testovacím rámcem porovnán s referenčním modelem.

10.1.2 Tvorba porovnatelných tvrzení

Pro účel porovnání modelů je u každého testu ručně zavedena řetězcová identifikace jednotlivých jedinců či tzv. hash jedince tak, aby bylo možné porovnat vlastnosti jedinců mezi modely. Identifikátor používaný v ontologiích a generovaný aplikací VizGet je pro tento účel nepoužitelný, protože je generován ze sekvenčního čísla, které se při špatné extrakci může posunout, případně může být jedinci přiřazeno jiné sekvenční číslo na základě změny v pořadí dokumentu.

Ve většině případů je u testů jako identifikátor zvolena hodnota identifikující proměnné, ale v některých případech toto nemusí být dostatečné a je tak zvolena kombinace hodnot více vlastností. Důsledek této volby znamená, že pokud se nepodaří jedince spárovat pomocí identifikátoru, je celý jedinec považován za neočekávaný a chybějící. To ovlivňuje příslušné metriky a ty tak spíše mohou vykazovat horší výsledky, než jaké ve skutečnosti jsou.

Porovnání modelů spočívá v převodu tvrzení o jedincích na množinu tzv. porovnatelných tvrzení. Porovnatelné tvrzení je tvrzení ve formě trojice řetězců subjekt–predikát–objekt, kde subjekt je hash jedince, predikát je plně kvalifikované jméno predikátu tvrzení a objekt je hash jedince v případě, že hodnotou je jedinec, nebo objekt převedený na řetězec, tedy nejčastěji textová hodnota datové vlastnosti. Porovnatelná tvrzení jsou pak ekvivalentní právě tehdy, když se všechny řetězce trojice rovnají, přičemž jako hash porovnatelného tvrzení jsou použity opět tyto tři řetězce.

Jednotlivá tvrzení pro převod na porovnatelná tvrzení jsou získána tak, že jsou získání všichni jedinci v modelu, kteří nejsou anonymní, a u nich jsou získána všechna jejich tvrzení, která nemají jako objekt anonymní objekt. Nutno podotknout, že takováto množina tvrzení obsahuje i pomocná tvrzení generovaná aplikačním rámcem Apache Jena, a také například anotační vlastnost zdrojové URL adresy, která je přidávána jedinci interně aplikací VizGet. To může tedy na rozdíl od nespárování jedinců zlepšovat metriky, přestože nebylo fakticky nic příliš užitečného extrahováno, resp. tato tvrzení jsou do značné míry pouze případ správného chování kódu aplikace VizGet. Příklad těchto tvrzení ukazuje tabulka 10.1.

[1] https://jsoup.org/
[2] https://playwright.dev/

Subjekt	Predikát	Objekt
Pelíšky	`rdf:type`	:Movie
Pelíšky	`rdf:type`	rdfs:Resource
Pelíšky	`:name`	Pelíšky
Pelíšky	`:rank`	10
Pelíšky	`:releaseDate`	1999
Pelíšky	`:movieRating`	109 624
Pelíšky	`vizget:sourceUrl`	https://www.csfd.cz/zebricky/filmy/nejlepsi/
109 624	`rdf:type`	:Rating
109 624	`rdf:type`	rdfs:Resource
109 624	`:rating`	91,2%
109 624	`:ratingCount`	109 624
109 624	`vizget:sourceUrl`	https://www.csfd.cz/zebricky/filmy/nejlepsi/

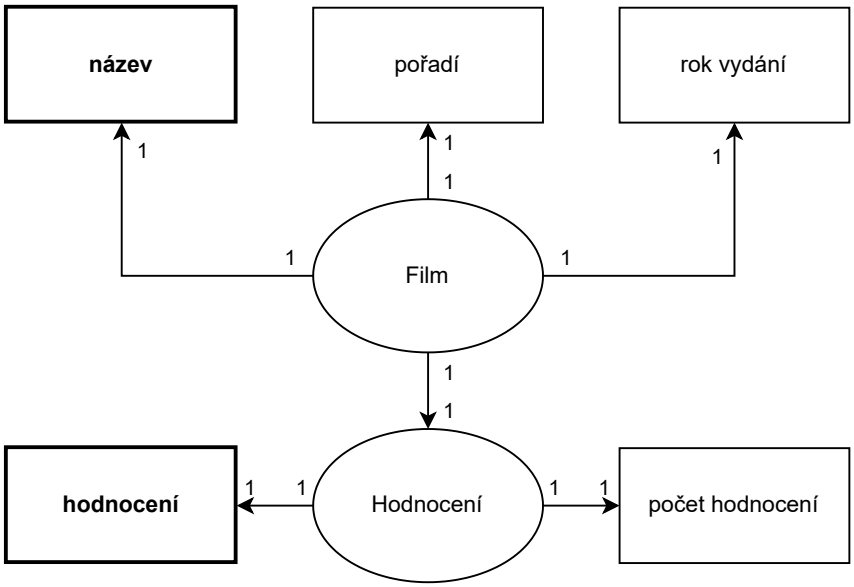
Tabulka 10.1: Ukázka všech porovnatelných tvrzení, které jsou testovacím rámcem vygenerovány pro film Pelíšky a jeho hodnocení. URI v predikátech a objektech jsou pro úsporu místa zkráceny na standardní prefixy.

10.2 Metriky měření úspěšnosti extrakce

Pro vyhodnocování testů byly zvoleny následující metriky, které patří v kontextu extrakce informací mezi často používané:

- **počet skutečně pozitivních výsledků** (anglicky *true positives*),
  $TP = |\text{expected} \cap \text{actual}|$
- **počet falešně pozitivních výsledků** (anglicky *false positives*),
  $FP = |\text{actual} \setminus \text{expected}|$
- **počet falešně negativních výsledků** (anglicky *false negatives*),
  $FN = |\text{expected} \setminus \text{actual}|$
- **přesnost** (anglicky *precision*) – odpovídá na otázku „Když už bylo tvrzení aplikací získáno, kolik procent z nich jsou očekávaná tvrzení?“
  $P = \frac{TP}{TP+FP}$
- **úplnost** (anglicky *recall*) – odpovídá na otázku „Kolik procent z očekávaných tvrzení bylo skutečně aplikací získáno?“
  $R = \frac{TP}{TP+FN}$
- **F-skóre** (anglicky *F-score*) – harmonický průměr přesnosti a úplnosti; kombinuje obě předchozí metriky do jednoho čísla.
  $F_1 = 2 * \frac{P*R}{P+R}$

Množina *expected* je množina očekávaných porovnatelných tvrzení a *actual* množina skutečných porovnatelných tvrzení.



Obrázek 10.1: Ontologický datový model filmů na ČSFD.cz. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

10.3 Žebříček nejlepších filmů

Úplně prvním testem, na kterém byla aplikace testována a který zásadně ovlivnil implementaci, je žebříček nejlepších filmů na Česko-Slovenské filmové databázi[3]. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.1, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **název filmu** značkovačem vět a slov s českým a anglický slovníkem,
- **pořadí v žebříčku** regulárním výrazem hledajícím číslo následované tečkou,
- **rok vydání** pomocí regulárním výrazem hledajícím číslo o 4 číslicích,
- **procentuální hodnocení** regulárním výrazem hledajícím desetinné číslo následované symbolem procenta,
- **počet hodnocení** regulárním výrazem hledajícím kombinaci číslic a mezer.

Jako hash jedince byl použit název filmu pro třídu Film a počet hodnocení pro třídu Hodnocení. Stejný model byl využit také pro extrakci žebříčku nejlepších seriálů[4] ze stejného webu, přičemž jak u filmů a seriálů bylo vybráno několik stránek žebříčku. Výsledky extrakce ukazuje tabulka 10.2.

Horší výsledek u poslední stránky filmů způsobil reduktor největších shluků, který vyhodnotil nejspíše příliš dlouhý název filmu „Pytlákova schovanka aneb Šlechetný milionář“ jako parazitní záznam a vyloučil ho. Je-li reduktor globálně zakázán pomocí parametru `--disable-reducer BiggestClusterReducer`, vyšplhá se úplnost a F-skóre na 100 %.

[3] https://www.csfd.cz/zebricky/filmy/nejlepsi/
[4] https://www.csfd.cz/zebricky/serialy/nejlepsi/

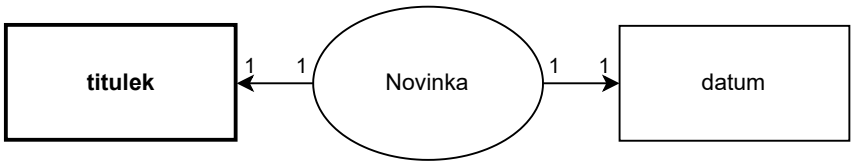
Stránka	TP	FP	FN	P	R	F_1
Filmy 1–99	1188	0	0	100,0 %	100,0 %	100,0 %
Filmy 900–999	1205	0	7	100,0 %	99,4 %	99,7 %
Seriály 1–99	1089	456	99	70,5 %	91,7 %	79,7 %
Seriály 900–999	1101	481	107	69,6 %	91,1 %	78,9 %

Tabulka 10.2: Výsledky testu žebříčku filmů a seriálů pro první a poslední stránky žebříčku ze 14. 4. 2022.

Jiná situace je u seriálů. Zde totiž dochází k chybnému nalezení názvu filmu (resp. seriálu) pro rok vydání a jako tento název jsou voleni režiséři a herci místo skutečného názvu filmu. To se děje z důvodu absence slov z názvu filmu ve slovníku značkovače, kdy nejsou vůbec rozpoznány některé názvy filmů, resp. mají nízkou podporu, a aplikace tedy upřednostní jako názvy filmů herce a režiséry, kterých je pro daný prostorový vztah více.

V případě, že jsou alespoň některá nerozpoznaná slova přidána do slovníku, tedy například v případě první stránky slova „Černobyl“, „Simpsonovi“, „MINDHUNTER“, „Ajťáci“, „Devadesátky“, „Fauda“, „Haikjú“ a „Miliardy“, zvyšuje se F-skóre první stránky na 100 %. V případě druhé stránky je nutné přidat slova „Sorjonen“, „Mirai“, „Garrowův“, „taeyang“, „Matrjoški“, „Šiki“, „Agui“ a „kjóšicu“ pro 100% přesnost, 99,4% úplnost a 99,7% F-skóre. Poslední chybou je zde opět dlouhý název filmu „Hadžime no Ippo: The Fighting – New Challenger“, ovšem vypnutí reduktoru největších shluků zde znamená stejné skóre jako bez přidání slov a pro 100% skóre by bylo nejspíše nutné přidat do slovníku více slov.

10.4 Novinky z fakultního webu



Obrázek 10.2: Ontologický datový model novinek na hlavní stránce FIT VUT v Brně. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

Druhým testem, který ovlivnil implementaci zavedením detekce víceřádkových oblastí, je test extrakce novinek na hlavní stránce webu FIT VUT v Brně[5]. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.2, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **titulek novinky** značkovačem vět a slov s českým slovníkem,
- **datum novinky** regulárním výrazem hledajícím řetězce začínající čísly následované tečkou, libovolnými znaky a končící čtyřmi číslicemi.

[5] https://www.fit.vut.cz/.cs

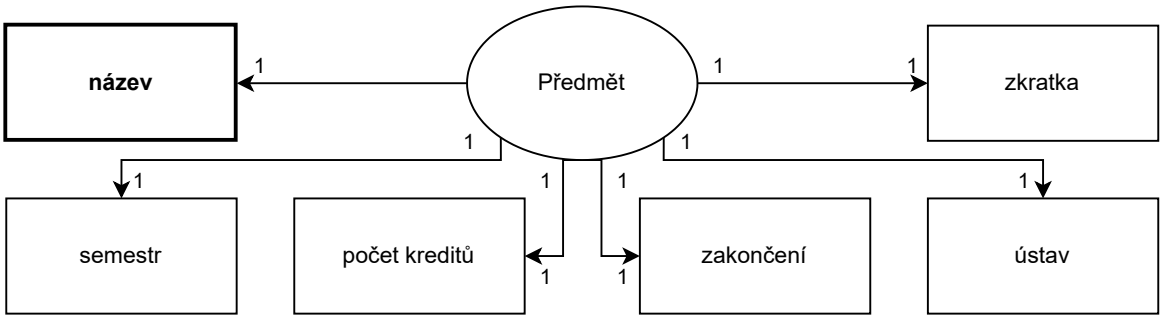
Jako hash jedince byl použit titulek novinky. Stejný model byl využit také pro extrakci všech novinek fakulty[6], přičemž výsledky extrakce ukazuje tabulka 10.3.

Stránka	TP	FP	FN	P	R	F_1
Hlavní stránka	45	0	0	100,0 %	100,0 %	100,0 %
Všechny novinky	2464	0	0	100,0 %	100,0 %	100,0 %

Tabulka 10.3: Výsledky testu novinek hlavní stránky a všech novinek webu FIT VUT ze 14. 4. 2022.

U tohoto testu není co řešit, aplikace dokázala korektně extrahovat všechny požadované informace. Korektně jsou spojeny všechny víceřádkové oblasti a u stránky se všemi novinkami jsou také díky laxnímu regulárnímu výrazu korektně extrahovány rozsahy dat, kdy daná novinka platí, resp. kdy se koná akce, na kterou novinka či aktualita odkazuje.

10.5 Předměty fakulty



Obrázek 10.3: Ontologický datový model předmětů vyučovaných na FIT VUT v Brně. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

Třetím testem, který ovlivnil implementaci zavedením reduktoru největších shluků a prvním zavedením značkovače vět a slov, je test extrakce vyučovaných předmětů na FIT VUT[7]. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.3, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **název předmětu** značkovačem vět a slov s českým slovníkem,
- **zkratka předmětu** regulárním výrazem hledajícím kombinaci malých a velkých písmen, číslic, spojovníku a podtržítka o délce 2 až 10 znaků, přičemž vyloučeny jsou pouhé číslice a zkratky zakončení předmětu „Zk“, „ZaZk“, „Za“ a „Klz“,
- **semestr**, kdy je předmět vyučován, regulárním výrazem hledajícím písmena „L“ nebo „Z“,
- **počet kreditů**, které je možné za předmět získat, regulárním výrazem hledajícím libovolná čísla,

[6] https://www.fit.vut.cz/fit/news/.cs
[7] https://www.fit.vut.cz/study/courses/.cs

- **zakončení předmětu** regulárním výrazem hledajícím zkratky „Za“, „Zk“, „ZaZk“, „Klz“ a spojovník,
- **ústav**, pod který předmět spadá, regulárním výrazem hledajícím kombinaci velkých písmen a mezery o dvou a více znacích, přičemž navíc jsou povoleny ústavy „K3D“ a „Knihovna“.

Jako hash jedince byl zvolen název předmětu. Stejný model byl využit také pro extrakci předmětů z roku 2002/2003[8], přičemž výsledky extrakce ukazuje tabulka 10.4.

Stránka	TP	FP	FN	P	R	F_1
Předměty aktuálního roku	3047	0	0	100,0 %	100,0 %	100,0 %
↑ bez reduktoru největších shluků	2865	188	182	93,8 %	94,0 %	93,9 %
Předměty roku 2002/2003	1360	0	18	100,0 %	98,7 %	99,3 %
↑ bez reduktoru největších shluků	1283	97	95	93,0 %	93,1 %	93,0 %

Tabulka 10.4: Výsledky testu vyučovaných předmětů na FIT VUT v aktuálním roce a roce 2002/2003 z 14. 4. 2022.

Jak je možné si všimnout, regulární výrazy použité pro hledání jsou v případě tohoto testu hodně specifické. To je způsobeno tím, že není jednoduché na první pohled univerzálně odlišit například zkratku předmětu od ústavu. Triky jako to, že ústav začíná písmenem „U“ nejsou stoprocentní, protože existuje spousta výjimek z tohoto pravidla. Stejně tak příliš obecnému regulárnímu výrazu pro zkratku jako například libovolná kombinace písmen a číslic odpovídá jak většina zkratek, tak i semestr, kredity, zakončení a ústav. Proto musely být zavedeny právě některé specifické konstrukce v jednotlivých regulárních výrazech, které se snaží počet společných prvků napříč jednotlivými množinami co nejvíce redukovat.

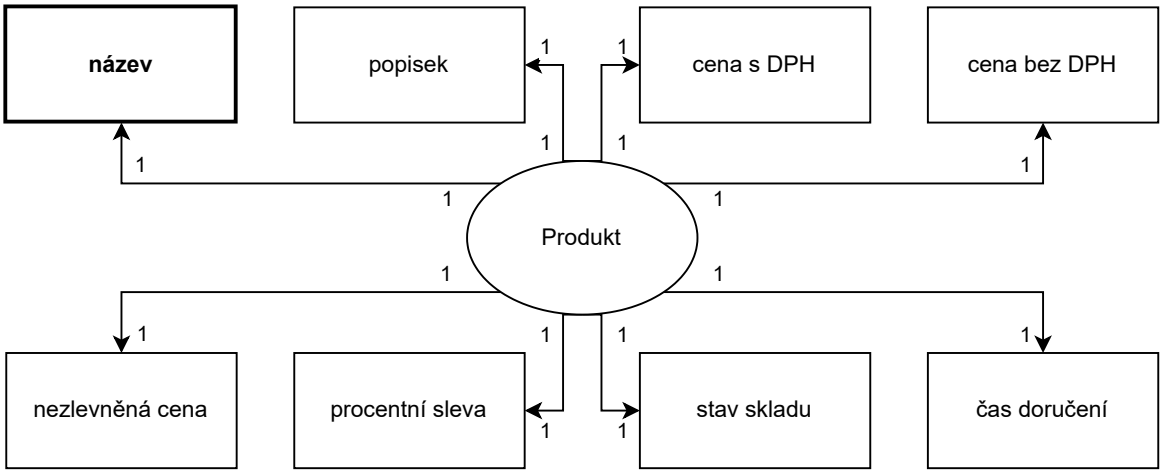
Aby přece jenom regulární výrazy mohly zůstat více obecné a nemusely se hledat regulární výrazy produkující zcela disjunktní množiny, byl v tomto testu poprvé zaveden reduktor největších shluků, který umožnil použít regulární výrazy tak, jak jsou v tomto testu prezentovány. Pokud by byl tento reduktor vypnut, znamenalo by to u obou testů pokles úspěšnosti extrakce, jak je možné vidět v tabulce 10.4. Tento pokles je způsoben nalezením vzoru, který považuje za ústav zkratku předmětu.

Horší výsledek předmětu z roku 2002/2003 je paradoxně nejspíš způsoben opět reduktorem největších shluků, který nejspíše opět chybně odstranil předměty „Přípravný kurs pro všeobecnou státní zkoušku z němčiny: Staatsexamen 2/2“ a „Přípravný kurs pro všeobecnou státní zkoušku z ěmčiny: Staatsexamen 1/2“[9], které mají příliš dlouhý název. Pro odstranění této chyby by nejspíš bylo nutné experimentovat s hodnotou ε, použít specifičtější regulární výrazy, případně počítat v reduktoru se zarovnáním textu.

10.6 Produkty internetového obchodu

Čtvrtým testem, který přímo ovlivnil implementaci zavedením možnosti použít vodítka stylů, vynutit prostorové relace a změnit složení reduktorů, je test extrakce produktů z kategorie počítače internetového obchodu Alza.cz[10]. Konceptuální model tříd a extrahovaných

[8] https://www.fit.vut.cz/study/courses/.cs?year=2002&type=ALL
[9] Chybějící písmeno „n“ u němčiny je převzatá chyba z webu FIT VUT.
[10] https://www.alza.cz/pocitace/18852653.htm



Obrázek 10.4: Ontologický datový model produktů na Alza.cz. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

datových položek ukazuje obrázek 10.4, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **název produktu** regulárním výrazem hledajícím text začínající volitelně malým písmenem následovaným velkým písmenem podle UTF-8 a libovolnými znaky o délce 1–80,
- **popisek produktu** regulárním výrazem hledajícím libovolný text o délce alespoň 50 znaků,
- **cenu s DPH** regulárním výrazem hledajícím volitelně libovolnou kombinaci číslic a bílých znaků podle UTF-8 následovanou povinně číslicemi a textem „,-“ nebo volitelně libovolnými bílými znaky podle UTF-8 a povinně textem „Kč“; navíc je použito vodítko stylů odpovídající textu o velikost 21 pixelů ± 1 pixel,
- **cena bez DPH** regulárním výrazem stejným jako u ceny s DPH, ale s použitým vodítkem stylů odpovídající textu o velikosti 11 pixelů ± 1 pixel,
- **nezlevněná cena** regulárním výrazem hledajícím libovolnou kombinaci číslic a bílých znaků podle UTF-8, které předchází znak procenta,
- **procentní sleva** regulárním výrazem hledajícím libovolné číslo, kterému předchází znak mínus a za kterým následuje znak procent,
- **stav skladu** regulárním výrazem hledající libovolný text, který obsahuje jako podřetězec text „skladem“ nebo „Skladem“,
- **předpokládaný čas doručení** regulárním výrazem hledajícím libovolný text, kterému předchází text „Můžete mít “ nebo „můžete mít “.

Jako hash jedince byl zvolen název produktu. Stejný model byl využit také pro extrakci produktů z kategorie dílny internetového obchodu Alza.cz[11], kategorie počítačů z internetového obchodu CZC.cz[12], a kategorie praček internetového obchodu MALL.CZ[13]. U obchodu CZC.cz a MALL.CZ se očekává extrakce názvu, popisku, ceny s DPH a stavu skladu. Navíc mají všechny testy vypnutý reduktor největších shluků. Výsledky extrakce ukazuje tabulka 10.5.

Stránka	**TP**	**FP**	**FN**	**P**	**R**	**F_1**
Alza počítače	191	11	10	94,6 %	95,0 %	94,8 %
↑ s reduktorem největších shluků	191	11	10	94,6 %	95,0 %	94,8 %
Alza dílna	204	12	14	94,4 %	93,6 %	94,0 %
↑ s reduktorem největších shluků	128	76	90	62,7 %	58,7 %	60,7 %
CZC.cz počítače	114	40	65	74,0 %	63,7 %	68,5 %
↑ s reduktorem největších shluků	62	60	117	50,8 %	34,6 %	41,2 %
↑ s odsouhlasenými cookies	154	0	25	100,0 %	86,0 %	92,5 %
MALL.CZ pračky	168	27	24	86,2 %	87,5 %	86,8 %
↑ s reduktorem největších shluků	168	27	24	86,2 %	87,5 %	86,8 %

Tabulka 10.5: Výsledky testu extrakce produktů z různých kategorií různých internetových obchodů se základním modelem navrženým pro obchod Alza.cz ze 14. 4. 2022.

U domény produktů bylo zpozorováno, že úspěšnost extrakce se v čase mění s tím, jak se mění nabídka produktů. Vykazované výsledky jsou spíše mírně horší než obvykle, u počítačů na Alza.cz se v čase objevovalo F-skóre blízko 100 % nebo i přesně 100 %, dílna měla k 17. 3. 2022 96,8 %, CZC.cz 71,8 % a MALL.CZ 89,1 %. Na Alza.cz se navíc začalo objevovat v poslední době vyskakovací okno přes celou stránku, přičemž je známou chybou, že tato okna způsobují aplikačnímu rámci potíže, viz dále rozbor testu CZC.cz. I přes toto horší skóre se stále jedná o uspokojivé hodnoty.

Počítače na Alza.cz měly horší výsledek, protože byly špatně odhaleny prostorové vztahy mezi názvem produktu, nezlevněnou cenou a procentní slevou, kdy se tato cena a sleva chybně přiřazují názvu pod cenou a slevou, místo správného přiřazování názvu nad cenou a slevou. Toto bylo způsobeno faktem, že bylo správnému vztahu přiřazeno nižší skóre, protože obsahoval duplicitní název produktu „Alza BattleBox Core RTX3070 Ti Quiet“, přestože šlo o dva rozdílné produkty, a tím pádem zde bylo i spoustu neunikátních slov, což ovlivnilo další metriku. Objevil se i případ u produktu „Alza GameBox Core GTX1650“, kdy se do předpokládaného času doručení dostal nesmyslný a na stránce neexistující údaj, což mohlo být způsobeno problémem s vyskakovacím oknem.

Dílna na Alza.cz měla horší výsledek opět zřejmě kvůli nesmyslnému textu u produktu „KETER Skládací pracovní stůl“, který byl způsoben pravděpodobně vyskakovacím oknem. Popisek u produktu „BOSCH AdvancedLevel 360 Set“ se lišil od referenčního modelu pouze v nezlomitelných mezerách, které bohužel knihovna jsoup použitá k referenční extrakci nezachovává, takže tento záznam není chybou. Chybou ale je opět špatné odhalení prostorového vztahu u nezlevněné ceny a procentní slevy, tentokrát ale mají nalezené vztahy stejná skóre se správným vztahem, a proto se využije první nalezená množina vzorů. Poslední chybou je

[11] https://www.alza.cz/hobby/vybaveni-dilny/18862227.htm
[12] https://www.czc.cz/pocitace/produkty
[13] https://www.mall.cz/pracky

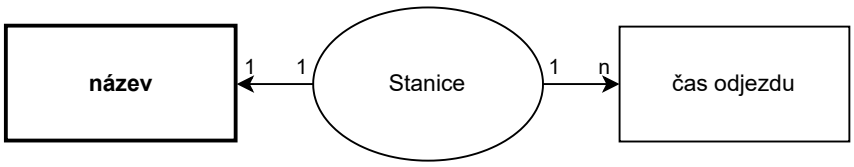
absence produktu „3M™ Dual-Lock™ Samolepící suchý zip SJ355B, 25mm x 2,5m“, jehož název začíná číslem, a proto nebyl regulárním výrazem vůbec odhalen.

Situace se mění u obchodu CZC.cz, kde za primární špatné skóre může vyskakovací okno pro odsouhlasení cookies. V průběhu vývoje byla totiž objevena chyba[14] v aplikačním rámci FitLayout, kdy tato chyba občas způsobuje vracení pouze částečných textů, když se na stránce objevuje právě ono vyskakovací okno. Jak je možné vidět v tabulce 10.5, pokud je použit profil prohlížeče Chromium, ve kterém byly cookies odsouhlaseny, skóre se výrazně vylepšuje a jediným problémem zůstává neodhalení stavu skladu. Tento problém se objevil u více testů a webů, a tyto weby jsou tedy v současné době označeny jako nepodporované.

Hlavním problémem u obchodu MALL.CZ jsou dva dílčí podproblémy. Jedním z nich je v podstatě nejspíše stejná chyba[15] v aplikačním rámci FitLayout, jako v případě vyskakovacích oken, a tedy nevracení celých popisků produktů. Rozdílem je, že nejspíše není způsobena kvůli vyskakovacímu oknu, ale kvůli ořezávání přetečení, resp. zkracování popisku. Druhým podproblémem je již problém aplikace VizGet, kdy se v některých popiscích produktu nachází tučné pasáže, které pak právě z důvodu jiného stylu nejsou spojeny do víceřádkové oblasti. Přesnost navíc mírně zhoršují dva nesprávně nalezené záznamy původní ceny před slevou, které se neměly ve výsledku vůbec objevit.

Všechny testované weby mají společné to, že produkty zobrazují v mřížce (anglicky *grid*) o několika produktech na řádek. Toto ale velmi nevyhovuje reduktoru největších shluků, protože v rámci rozdílných řádků nemusí být dodržená stejná vzdálenost mezi názvem produktu a závislými datovými položkami. Pokud se například hodně liší délka popisků, reduktor může začít takové záznamy odstraňovat, což se negativně projeví na úspěšnosti extrakce. Proto byl reduktoru ve výchozím stavu pro všechny tyto testy a weby vypnut a není tak obecně doporučeno ho nechávat zapnutý pro záznamy umístěné v mřížce, případně je nutné otestovat, která varianta vychází lépe. Jak se změní úspěšnost je možné vidět opět v tabulce 10.5.

10.7 Jízdní řád veřejné dopravy



Obrázek 10.5: Ontologický datový model časů odjezdu ze stanic. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

Pátým testem, který mírně ovlivnil implementaci zavedením parametru pro rychlost či vytrvalost extrakce, je test jízdního řádu vlakové linky Alamein Line v australské státě Victoria, který je zajišťován agenturou Public Transport Victoria[16]. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.5, z nějž vyplývá, že extrahovány tedy byly datové položky:

[14] https://github.com/FitLayout/fitlayout-puppeteer/issues/2
[15] https://github.com/FitLayout/fitlayout-puppeteer/issues/2#issuecomment-1069305962
[16] https://www.ptv.vic.gov.au/route/timetable/1/alamein/

- **název stanice**, na které vlak zastavuje, značkovačem vět a slov s anglickým slovníkem,
- **časy odjezdu** ze stanice regulárním výrazem hledajícím dvě čísla oddělená dvojtečkou, za kterými následuje mezera a text „am“ nebo „pm“.

Jako hash jedince byl zvolen název stanice. Stejný model byl využit také pro extrakci jízdního řádu vlakové linky V/Line Ballarat-Wendouree – Melbourne via Melton[17] a tramvajové linky 1 East Coburg - South Melbourne Beach[18]. Všechny testy mají navíc nastavenou rychlost či vytrvalost na maximální hodnotu 3 a šířku stránky na hodnotu 25 000 pixelů. Výsledky extrakce ukazuje tabulka 10.6.

Stránka	TP	FP	FN	P	R	F_1
Alamein Line	921	0	0	100,0 %	100,0 %	100,0 %
Ballarat-Wendouree	332	0	0	100,0 %	100,0 %	100,0 %
East Coburg	3669	0	0	100,0 %	100,0 %	100,0 %

Tabulka 10.6: Výsledky testu extrakce jízdních řádů různých linek z 15. 4. 2022.

Na první pohled může být divné, proč nebyl použit pro čas odjezdu značkovač specializovaný právě na extrakci času. Tento značkovač ale obsahuje chybu[19], která špatně extrahuje 12hodinový čas, kdy je u tohoto času určení části dne AM či PM odděleno mezerou. Proto musel být využit vlastní regulární výraz, který tímto problémem netrpí.

Zároveň proto, že jízdní řád je donačítán na stránce dynamicky prostřednictvím jazyka JavaScript, byla do aplikace přidána možnost ovlivnit již existující parametr rychlosti či vytrvalosti extrakce, který nabízí vykreslovací jádro Puppeteer v aplikačním rámci FitLayout. Tento parametr je nutné využít, jinak extrakce skončí neúspěchem, protože se jízdní řád nestačí donačíst.

Posledním specifikem je nastavení obří šířky stránky. To je způsobeno použitím horizontálního posuvné lišty u jízdního řádu, která brání aplikačnímu rámci FitLayout extrahovat všechny položky, protože jednoduše nejsou vidět. Díky tomu, že se šířka této oblasti s jízdním řádem přizpůsobuje šířce stránky, je možné nastavit takto obrovskou šířku, která je již schopna pojmout všechny časy jízdního řádu a extrahovat tak korektně všechny položky.

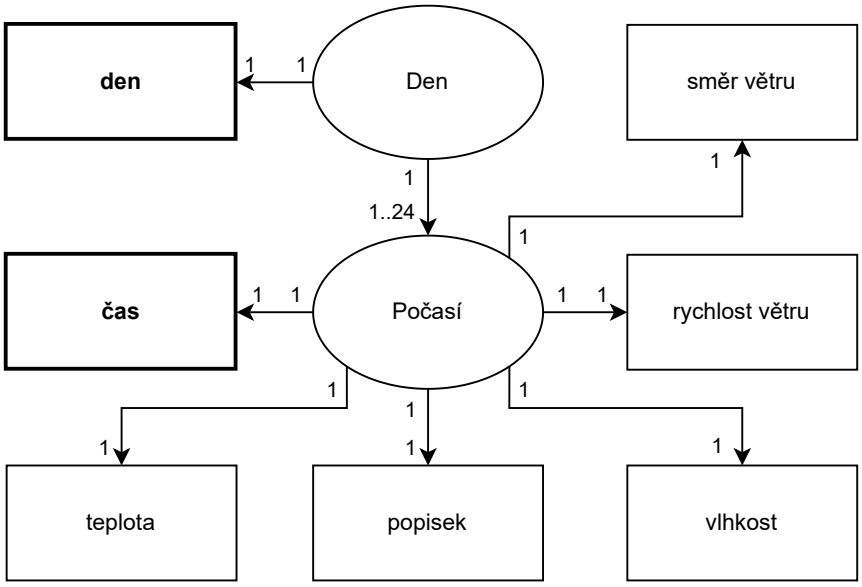
Co se týče výsledků, tak zde není co řešit, aplikaci VizGet se podařilo korektně nalézt všechny záznamy, a to i přes skutečnost, že se jedná o první použití kardinality různé od 1:1, konkrétně kardinality 1:n.

10.8 Počasí města po hodinách

Testem, který nijak neovlivnil implementaci, je test extrakce předpovědi počasí v Praze po hodinách v nejbližších dnech z webu The Weather Channel[20]. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.6, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **datum dne**, pro který je určena daná předpověď počasí, značkovačem data,

[17] https://www.ptv.vic.gov.au/route/timetable/1728/ballarat-melbourne-via-melton/
[18] https://www.ptv.vic.gov.au/route/timetable/721/1/
[19] https://github.com/FitLayout/FitLayout/issues/8
[20] https://weather.com/en-US/weather/hourbyhour/l/2dcf05dd14240a44e1a853ad63211c01b751cc20647bcd75f3bbc4bd51b9649a?unit=m



Obrázek 10.6: Ontologický datový model počasí po hodinách. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

- **čas**, resp. hodina předpovědi v daném dni, regulárním výrazem hledajícím číslo následované libovolný počtem bílých znaků podle UTF-8 a textem „am“ nebo „pm“,
- **teplota** ve °C regulárním výrazem hledajícím celá čísla, tedy i čísla záporná se znaménkem mínus na začátku, následovaná symbolem stupně,
- **popisek počasí** značkovačem vět a slov s anglickým slovníkem,
- **vlhkost** v procentech regulárním výrazem hledajícím číslo následované znakem procenta,
- **směr větru** regulárním výrazem hledajícím libovolnou kombinaci znaků „N“, „S“, „W“ a „E“,
- **rychlost větru** v kilometrech za hodinu regulárním výrazem hledajícím číslo následované libovolným počtem bílých znaků podle UTF-8 a textem „km/h“.

Jako hash jedince bylo zvoleno datum dne pro třídu Den a zřetězení data dne a času pro třídu Počasí. Stejný model byl využit také pro extrakci počasí pro Brno[21], Nairobi[22] a Stockholm[23]. Výsledky extrakce ukazuje tabulka 10.7.

Zde opět není co řešit, identifikovat se podařilo úspěšně všechny záznamy a vzory ve všech testovaných městech. Použity byly i jiné značkovače než regulární výrazy a když už

[21] https://weather.com/weather/hourbyhour/l/154614828289f8b588a587328b4c5ab4f74302d9fbbb598a13169fdc9982543f?unit=m

[22] https://weather.com/weather/hourbyhour/l/e1d0bb735632de2df082e02f88493ef295908714761728c5c7d5d6c76cb2f83e?unit=m

[23] https://weather.com/weather/hourbyhour/l/dde274874b4d0ebf41c3de25c75bbb59b96ba1b2e0210b78bc7ff530086e6598?unit=m

Stránka	TP	FP	FN	P	R	F_1
Praha	490	0	0	100,0 %	100,0 %	100,0 %
Brno	490	0	0	100,0 %	100,0 %	100,0 %
Nairobi	490	0	0	100,0 %	100,0 %	100,0 %
Stockholm	490	0	0	100,0 %	100,0 %	100,0 %

Tabulka 10.7: Výsledky testu extrakce hodinové předpovědi počasí pro různá města světa z 15. 4. 2022.

byly regulární výrazy použity, nejednalo se o nijak složité a příliš specifické výrazy. Tabulky se zdají být jako pěkná ukázka případu, který aplikaci VizGet nedělá příliš problémy.

10.9 Program rozhlasových stanic

Obrázek 10.7: Ontologický datový model programu rozhlasových stanic. Ovály tvoří třídy a obdélníky datové vlastnosti, tučně jsou vyznačené identifikující datové vlastnosti a čísla značí kardinalitu vztahu.

Posledním testem, který mírně ovlivnil implementaci přidáním možnosti zakázat detekci víceřádkových oblastí, je test extrakce programu vybraných rozhlasových stanic pro aktuální den. Konceptuální model tříd a extrahovaných datových položek ukazuje obrázek 10.7, z nějž vyplývá, že extrahovány tedy byly datové položky:

- **název položky programu** regulárním výrazem hledajícím text začínající volitelně libovolným počet číslic, povinně následovaný velkým písmenem podle UTF-8 a dalším libovolným textem,
- **čas**, ve kterém je položka programu vysílána, značkovačem času.

Jako hash jedince byl zvolen název položky programu. Model byl již od začátku navrhován pro seznam stanic poskytnutý vedoucím práce, kdy tento seznam zahrnuje program rozhlasových stanic Antena 2[24], BBC Radio 3[25], France Musique[26], MDR KULTUR[27], NPO Radio 4[28], Radio Clásica[29] a rádií instituce WDR[30]. Stanice MDR KULTUR a WDR mají vypnutý reduktor největších shluků. Výsledky extrakce ukazuje tabulka 10.8.

Prvním možným překvapením je volba regulárního výrazu pro název položky programu. V aplikaci pro tento účel existuje značkovač vět a slov, ovšem ten v tomto případě nemohl

[24] https://www.rtp.pt/antena2/programacao
[25] https://www.bbc.co.uk/schedules/p00fzl8t
[26] https://www.radiofrance.fr/francemusique/grille-programmes
[27] https://www.mdr.de/kultur/radio/ipg/mdr-kultur-programm-100.html
[28] https://www.nporadio4.nl/gids
[29] https://www.rtve.es/play/guia-rne/radioclasica/
[30] https://www.wdr.de/programmvorschau/alle/uebersicht/2022-04-15/

Stránka	TP	FP	FN	P	R	F_1
Antena 2	81	0	0	100,0 %	100,0 %	100,0 %
BBC Radio 3	76	0	0	100,0 %	100,0 %	100,0 %
France Musique	0	0	85	0,0 %	0,0 %	0,0 %
MDR KULTUR	55	0	0	100,0 %	100,0 %	100,0 %
NPO Radio 4	0	15	50	0,0 %	0,0 %	0,0 %
Radio Clásica	70	0	15	100,0 %	82,4 %	90,3 %
WDR	133	0	0	100,0 %	100,0 %	100,0 %

Tabulka 10.8: Výsledky testu extrakce programu různých rozhlasových stanic z 15. 4. 2022.

být použit, protože obsahuje pouze český a anglický slovník, přičemž stránky jsou ale v různých jazycích od těchto dvou. Proto byl zvolen tento jednoduchý, ale přesto celkem účinný regulární výraz s detekcí velkého písmena.

Druhým překvapením je nulová úspěšnost u dvou stanic. V případě stanice France Musique je tento neúspěch způsoben faktem, že značkovač času nedokáže rozpoznat formát času, který je na stránce použit a nemůže tak být nalezen žádný vzor. Vyhráno ale není ani v situaci, kdy je značkovač změněn na regulární výraz, který tomuto formátu času odpovídá. Čas vysílání položky programu není z hlediska dostupných prostorových vztahů aplikace VizGet totiž nijak zarovnán s názvem položky, a tedy nemůže být nalezen korektní vztah a místo toho je nalezen jeden parazitní záznam.

Nulová úspěšnost u stanice NPO Radio 4 je způsobena v podstatě selháním detekce víceřádkových oblastí. Názvy položek programu na stejném řádku jsou totiž spojeny do jedné oblasti a tato oblast je následně nejspíše vyloučena jedním z reduktorů. Je-li zakázána detekce víceřádkových oblastí, stoupá F-skóre na 100 %.

Poslední mírně horší výsledek u stanice Radio Clásica je způsoben absencí osoby u položky programu. Osoby u těchto položek jsou totiž při detekci víceřádkových oblastí spojeny do jedné oblasti spolu s časem začátku i konce programu, ale protože mají jinou barvu písma (#000 u osoby a #111 u času), jedná se o jiný celkový styl oblasti než v případě, že by v oblasti byl pouze čas. Z toho důvodu nemohou být oblasti s osobou i s časem a oblasti pouze s časem v jedné množině vzorů a vybrána je tak množina s největším počtem oblastí.

Výsledek se ale nezlepší ani při zakázání detekce víceřádkových oblastí. V takovém případě je chybně detekován čas konce místo času začátku, nejspíše z důvodu větší blízkosti středu oblasti s názvem položky programu, a F-skóre klesá na hodnotu 80 %.

Kapitola 11

Závěr

Cílem této práce bylo vytvořit prakticky použitelnou aplikaci, která by prováděla extrakci informací na základě vyhledávání vizuálních vzorů. V práci tedy vznikla konzolová aplikace VizGet jazyka Java, která využívá aplikační rámec FitLayout a s ním spojenou techniku vyhledávání vizuálních vzorů založenou na již publikované práci. Vyhledávání je rozšířeno o reduktory a metriky hodnocení těchto vizuálních vzorů, vlastní značkovače oblastí a další.

Uživateli stačí aplikaci předat pouze ontologický datový model obohacený o několik specifických prvků a URL adresu, ze kterých se mají informace extrahovat, a aplikace mu dokáže vrátit extrahované informace navázané na tento vstupní model dat. Vizuální zpracování zajišťuje menší závislost na zdrojovém kódu a extrakce je jednoduchá i z webové stránky s ne příliš přívětivým zdrojovým kódem.

Testování aplikace ukázalo, že ji lze celkem dobře využít na mnoha rozdílných doménách, ačkoliv občas je třeba aplikaci pomoci dostupnými parametry, které výsledky extrakce dokáží mnohdy vylepšit až na 100% úroveň. Případem takového parametru je vypínání reduktoru největších shluků, který přestože prokazatelně dokáže některé výsledky vylepšit, často naopak způsobí výsledky horší. Cílem ale nikdy nebylo dosáhnout 100% úspěšnosti za všech okolností, a i proto tedy prezentované výsledky ukazují velmi dobrou praktickou použitelnost pro případy, kdy občasné zaváhání aplikace není kritické.

Do budoucna existuje v aplikaci spousta dalších možností, jak ji vylepšit. Použit by například mohl být reduktor s pokročilejším shlukovacím algoritmem, reduktor největších shluků by mohl počítat se zarovnáním oblastí nebo třeba reduktory založené na kardinalitě by mohli vybírat správné oblasti na základě jiných způsobů než jen nejkratší vzdálenosti. Vzniknout by mohly nové značkovače oblastí například pro čísla, procenta nebo značkovače využívající prefixu a sufixu, dále určité předběžné odstraňování štítků, které nemohou existovat společně, nebo převod dat na jiné než textové datové typy ve výstupu aplikace.

Vyhledávání vizuálních vzorů by také do budoucna mohlo probíhat ve dvou kolech, kdy by se nejdříve nalezly pomocí standardního algoritmu všechny množiny vzorů a následně by se v těchto množinách hledaly například společné identifikující oblasti, přičemž množiny, které by nějakým způsobem neodpovídaly, by mohly být na základě prvního kola přepočítány. Vyhledávání by také mohlo probíhat i mezi neidentifikujícími datovými položkami.

Pro lepší použitelnost by také mohla vzniknout aplikace s grafickým uživatelským rozhraním, která by zásadně zlepšila definici vstupního ontologického modelu dat. Aplikace pro definici ontologií sice již existují a lze je pro tento účel použít, ale často se v nich pro účely aplikace VizGet tvoří ontologie komplikovaným způsobem. Člověk musí přesně vědět, co dělá, a i přesto je často nutné výsledný model upravit přímo v kódu.

Literatura

[1] Burget, R. Java HTML rendering engine. *CSSBox* [online]. [cit. 2021-12-26]. Dostupné z: http://cssbox.sourceforge.net/.

[2] Burget, R. Information Extraction from the Web by Matching Visual Presentation Patterns. In: *Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia*. Springer International Publishing, 2017, s. 10–26. Lecture Notes in Computer Science vol. 10579. DOI: 10.1007/978-3-319-68723-0_2. ISBN 978-3-319-68722-3. Dostupné z: https://www.fit.vut.cz/research/publication/11218.

[3] Burget, R. Model-Based Integration of Unstructured Web Data Sources Using Graph Representation of Document Contents. In: *15th International Conference on Web Information Systems and Technologies*. SciTePress - Science and Technology Publications, 2019, s. 326–333. ISBN 978-989-758-386-5. Dostupné z: https://www.fit.vut.cz/research/publication/12003.

[4] Burget, R. *Extrakce dat z webu*. Říjen 2020. [cit. 2021-11-05]. Dostupné z: https://www.fit.vutbr.cz/~burgetr/upa/05_webscraping/.

[5] Burget, R. *FitLayout/2: Web Page Analysis Framework* [online]. 2020. Aktualizováno 1. 12. 2021 [cit. 2021-12-26]. Dostupné z: https://github.com/FitLayout/FitLayout/tree/dcfbfc17739ca2744684aa169eb2e4a2c671d572.

[6] Burget, R. *FitLayout/FitLayout Wiki* [online]. 2021. Aktualizováno 3. 11. 2021 [cit. 2021-12-26]. Dostupné z: https://github.com/FitLayout/FitLayout/wiki.

[7] Etemad, E. *CSS Box Model Module Level 3*. Candidate Recommendation. W3C, prosinec 2020. Dostupné z: https://www.w3.org/TR/2020/CR-css-box-3-20201222/.

[8] Lassila, O. a Swick, R. R. *Resource Description Framework (RDF) Model and Syntax Specification*. Proposed Recommendation. W3C, leden 1999. Dostupné z: https://www.w3.org/TR/1999/PR-rdf-syntax-19990105/.

[9] Lushnikov, A., Franklin, J. et al. *Puppeteer* [online]. 2017. Aktualizováno 22. 12. 2021 [cit. 2021-12-26]. Dostupné z: https://github.com/puppeteer/puppeteer/blob/71cef32f6d0420883baef87c1639dbc2def819a0/README.md.

[10] Milička, M. a Burget, R. Information Extraction from Web Sources based on Multi-aspect Content Analysis. In: *Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015*. Springer International Publishing, 2015, sv. 2015, č. 548, s. 81–92. Communications in Computer and Information Science. DOI:

10.1007/978-3-319-25518-7_7. ISBN 978-3-319-25517-0. Dostupné z: https://www.fit.vut.cz/research/publication/10840.

[11] Schreiber, G. a Dean, M. *OWL Web Ontology Language Reference*. W3C Recommendation. W3C, únor 2004. Dostupné z: https://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[12] tyrex. Stack Overflow. *1D Number Array Clustering* [online]. Červenec 2021 [cit. 2022-04-07]. Dostupné z: https://stackoverflow.com/a/68271073/13187896.

[13] W3C. W3C. *Semantic Web* [online]. Aktualizováno 24. 6. 2021 [cit. 2021-12-26]. Dostupné z: https://www.w3.org/standards/semanticweb/.

[14] Williams, J. PromptCloud. *Scraping Data comparison:Site-specific Extractors vs. Generic Extractors* [online]. Září 2013. Aktualizováno 22. 5. 2020 [cit. 2021-10-29]. Dostupné z: https://www.promptcloud.com/blog/site-specific-extraction-vs-generic-extraction-after-scraping/.

Příloha A

Obsah paměťového média

- `.git/` – složka verzovacího systému Git.
- `.idea/` – složka vývojového prostředí IntelliJ IDEA.
- `doc/` – zdrojové kódy semestrální prezentace, LaTeXu a zkompilovaná PDF verze této práce.
- `fitlayout-puppeteer/` – inicializovaný repozitář s vykreslovacím jádrem Puppeteer pro Windows.
- `out/` – složka pro kompilaci LaTeXu a uložení výsledků testů.
- `results/` – výsledky testování zahrnující statistiky, nalezené vizuální vzory a extrahované informace.
- `src/` – veškeré zdrojové kódy aplikace včetně testovacího rámce, jednotlivých testů a testovacích ontologií.
- `target/` – zkompilovaná aplikace nástrojem Maven.
- `.gitattributes` – soubor s určením konců řádků některých souborů ve verzovacím systému Git.
- `.gitignore` – seznam ignorovaných souborů a složek verzovacího systému Git.
- `.gitmodules` – seznam submodulů verzovacího systému Git.
- `pom.xml` – seznam závislostí a kroků sestavení aplikace nástrojem Maven.
- `README.md` – základní informace o požadavcích, instalaci a spuštění aplikace.
- `vizget` – shell skript pro spuštění aplikace na Linuxu.
- `vizget.bat` – skript příkazové řádky pro spuštění aplikace na Windows.
- `vizget.iml` – soubor vývojového prostředí IntelliJ IDEA.