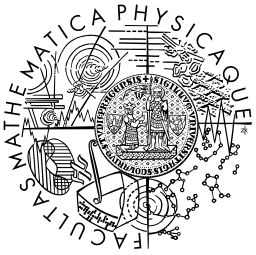


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Štěpán Stenchlák

Vizuální browser grafových dat

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové a datové inženýrství

Praha 2020

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Vizuální browser grafových dat

Autor: Štěpán Stenchlák

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., katedra

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Visual browser of graph data

Author: Štěpán Stenchlák

Department: Name of the department

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., department

Abstract: Abstract.

Keywords: key words

Obsah

Úvod

Na internetu je dnes možné najít téměř cokoli, kupříkladu stav počasí, encyklopedické informace, odborné publikace, jízdní řády a podobně. Tyto informace jsou uloženy jako webové stránky systému WWW (World Wide Web) a kterýkoli uživatel internetu má k nim přístup a může z nich čerpat.

Pro lidi je systém WWW vyhovující zdroj informací, nicméně narazíme na problém, pokud chceme tyto informace číst strojově. WWW totiž nepopisuje, jak by měly být informace na internetu prezentovány. Každý publikovatel si může data zveřejnit jinak, například tabulkou, odrážkovým seznamem, nebo ve větách. Tyto informace jsou stále snadno čitelné pro člověka, ale obtížně čitelné pro stroj.

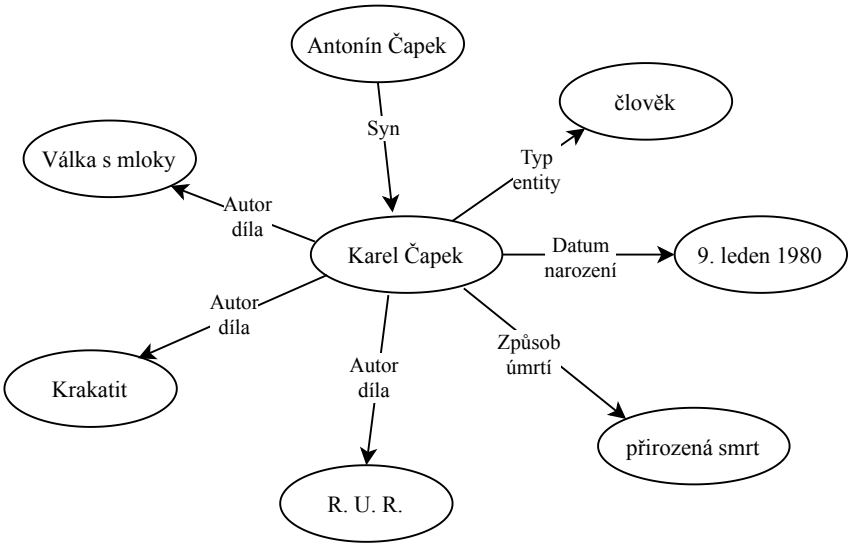
Možností řešení tohoto problému je zveřejňovat data i ve strojové podobně. Nabízí se například tabulky ve formátu CSV, nebo komplikovanější data ve formátu JSON a XML. Zde je již snadné číst data a dál je zpracovávat, ale programátor je stále nucen pochopit a přizpůsobit program na rozhraní těchto dat.

Konečným řešením se nabízí popsat data pomocí RDF frameworku, jež je popsán důkladněji v následující kapitole.

Tato práce se zabývá reprezentací dat popsancýh právě pomocí RDF. RDF reprezentuje entity z reálného světa jako vrcholy grafu a jejich vlastnosti pomocí hran propojující tyto entiy. Data uložená v grafových databázích jsou snadno čitelná počítačovými programy a je jednoduché propojovat různé datové zdroje do větších a provádět nad nimi dotazy.

Jako veškerá strojová data je potřeba i tyto grafová data vizualizovat. Příkladem může být datový analytik, který se chce přesvědčit, že jsou data uložena tak, jak bylo zamýšleno.

Jako modelový příklad vizualizace uveďme entitu Karla Čapka a některá jeho data, jež má o něm uložená Wikipedie.

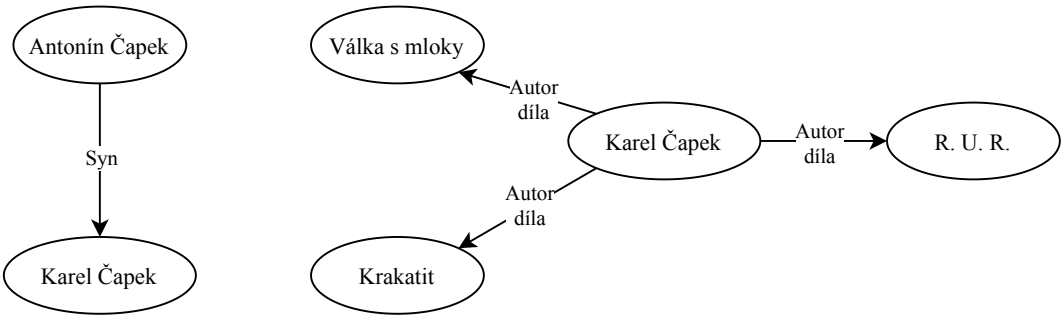


Obrázek 1: Ukázka části grafu jež může reprezentovat Karla Čapka.

Jak vidíme z obrázku 1, zobrazit všechna data k uzlu může být velmi nepřehledné a takovýchto dat může být stovky. Pokud nás například zajímají pouze

díla Karla Čapka, určitě v grafu nebudeme chtít mít informaci, že Karel Čapek je člověk. Možností řešení tohoto problému je dívat se na entity pouze určitým pohledem a tedy zobrazit pouze ty vztahy k vrcholu, jež odpovídají pohledu.

Kupříkladu pokud bychom chtěli procházet rodokmenem, je vhodné se na Karla Čapka dívat jako na osobu, jež má rodiče a sama může být rodičem ostatních osob. V tuto chvíli nás nezajímají ostatní vlastnosti jako knihy, které napsal, nebo ocenění, která získal. Na Karla Čapka se ale můžeme dívat i jako na spisovatele, kde nás pak zajímají pouze jeho díla a rodinné vztahy můžeme skrýt. Příklad tekovýchto pohledů je na obrázku 2.



Obrázek 2: Pohled na Karla Čapka jako na osobu mající rodinu (vlevo) a na spisovatele jež je autorem literárních děl (vpravo)

Tento způsob procházení grafových dat sice vyžaduje předem nadefinovat pohledy, ale procházení dat je pak jednoduché a velmi přehledné.

0.1 Cíl práce

Cílem této práce bylo vyrobit webovou aplikaci, jež by na základě předem definovaných pohledů vizualizovala grafová data a umožňovala uživateli procházet tento graf a objevovat nové vrcholy. Protože je žádoucí mít více pohledů na konkrétní typ vrcholu, pohledy jsou seskupeny do takzvaných konfigurací. Konfigurace pak popisuje nejen pohledy na různé typy vrcholů, ale i jaké vrcholy lze takto vizualizovat a z jakých zdrojů čerpat.

Konfigurace, jež jsou použité v aplikaci jsou například:

- **Procházení živočichů na Wikidatech** - Uživatel může vizualizovat jednotlivé rostlinné a živočičné druhy a procházet je podle rozdělení do taxonů.
- **Procházení slavných osobností z Wikidat** - Uživatel může procházet slavné osobnosti a nechat si načíst jejich filmová a literární díla a rodinné vztahy.

Kromě procházení si bude uživatel moci zobrazit detailní informace ke konkrétnímu vrcholu v závislosti na pohledu.

1. Technické předpoklady

1.1 Resource Description Framework

Resource Description Framework zkráceně RDF je rodina specifikací (tedy sada pravidel) která se používá na popis informací na internetu. RDF popisuje data jako graf, konkrétně vrcholy grafu jsou entity, jež ztvářňují nějaké věci (fyzické předměty, díla, myšlenky) a hrany tyto entity propojují a dávají jim vztahy.

Základem je takzvaný **statement** (česky trojice). Statement se skládá v tomto pořadí ze subjektu, predikátu a objektu přičemž subjekt a objekt jsou vrcholy (anglicky **nodes**) a predikát je orientovanou hranou jdoucí od subjektu k objektu.

Vrcholem v RDF může být IRI, literál, nebo prázdný uzel.

Jako vrchol IRI (Internationalized Resource Identifier) se rozumí vrchol jež přiřazuje entitě nějaký identifikátor ve tvaru IRI. Tímto jsme schopni v rámci WWW identifikovat různé entity a pracovat s nimi napříč různými zdroji. Ku příkladu Karel Čapek, zmíněný v úvodu, je v rámci Wikipedie jednoznačně identifikován jako `https://www.wikidata.org/wiki/Q155855`. IRI kromě identifikátoru nenese žádné další informace včetně jména. IRI tedy reprezentuje nějakou neznámou entitu, která musí být popsaná statementy aby dostala význam.

Literálem je pak uzel nesoucí nějakou hodnotu. Může se jednat o číslo popisující věk osoby, její jméno atp. V rámci RDF kromě hodnoty má literál i svůj typ, jež je opět vyjádřen pomocí IRI. Uveďme kupříkladu `http://www.w3.org/2001/XMLSchema#integer` jež vyjadřuje obecně celé číslo. Speciálním typem je `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString` který vyžaduje ještě takzvaný **laguage tag** a popisuje text v nějakém konkrétním jazyce.

Literály nám tedy dávají možnost přidat IRI uzlům různé hodnoty a podporují právě i multijazyčnost. Taktovéto grafy pak dokáží popsat celou řadu věcí z reálného světa.

Nakonec, hrana je opět popsána pomocí IRI jež reprezentuje typ vztahu.

1.1.1 Turtle jazyk

RDF graf může být popsán pomocí Turtle jazyka[1]. Ten je využit při zápisu konfigurací a metakonfigurací, jež jsou popsány dále v tomto dokumentu.

Trojice (statementy) se zapisují jako tři slova oddělená mezerami a zakončená tečkou. Chceme-li zapsat IRI, musíme jej dát do špičatých závorek `<>`. Na začátku dokumentu lze kromě dalších věcí definovat i prefixy, které pak umožňují zkrátit zapisované IRI do namespacu a zbytku `namespace:zbytek`.

Pokud se nám u statementů opakuje subjekt a predikát, můžeme jednotlivé objekty oddělovat čárkou (`,`). Obdobně, pokud se nám opakuje jen subjekt, můžeme dvojice predikát-objekt oddělovat středníkem (`;`).

Uveďme několik příkladů RDF grafu.

Příklad. Statement „Karel Čapek se narodil 9. ledna 1890“ vyjádřený v rámci wikidat.

[1] `https://www.w3.org/TR/turtle/`

```
@prefix wd: <https://www.wikidata.org/wiki/> .
@prefix wdt: <https://www.wikidata.org/wiki/Property:> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

wd:Q155855 wdt:P569 "1890-01-09"^^xsd:dateTime .
```

Kód popisuje jeden statement kdy se Karlu Čapkovi `https://www.wikidata.org/wiki/Q155855` přiřadí přes property datum narození `https://www.wikidata.org/wiki/Property:P569` literál s jeho datem narození, jež má typ `http://www.w3.org/2001/XMLSchema#dateTime`.

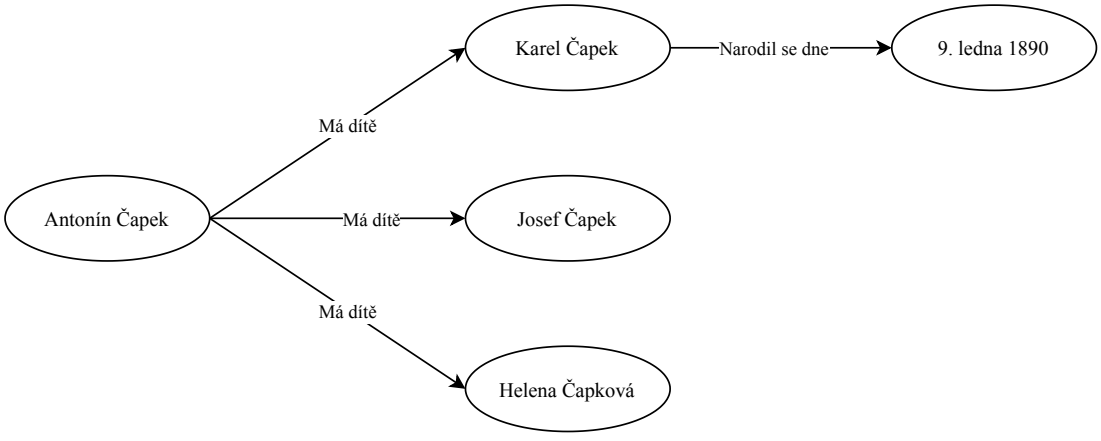
Jak lze vidět z příkladu, statementy nemusí odkazovat jen na svůj dataset, ale mohou odkazovat i mimo něj. To nám dovoluje stavět již na existujících datasetech a jednoduše se na ně okazovat přes IRI. Můžeme tak mít vlastní knihovní databázi jež ke knize přiradí autora z datasetu Wikidat. Tímto jsme propojili dva datasety, což nám umožní se nad oběma dotazovat. Příkladem takového dotazu by mohlo být „Chci seznam knih v naší knihovně, jež byly napsány českými autory.“ Takový dotaz pak současně prohledá dva různé datasety a vrátí očekávané výsledky.

Uveďme ještě příklad statementu, jež se odkazuje na další entity.

Příklad. Statement „Děti Antonína Čapka jsou Karel, Josef a Helena“ vyjádřený v rámci wikidat.

```
@prefix wd: <https://www.wikidata.org/wiki/> .
@prefix wdt: <https://www.wikidata.org/wiki/Property:> .

wd:Q6657059 wdt:P40 wd:Q155855 ,
                    wd:Q454568 ,
                    wd:Q4532606 .
```



Obrázek 1.1: Příklad grafu který získáme z předešlých dvou ukázek

Pro úplnost zmiňme ještě například následující graf popisující vztahy mezi lidmi, které jsou vyjádřeny ontologií FOAF (friend of a friend). Pokud by všechny zdroje popisující lidi využívali tuto ontologii, měli bychom jednotný interface jak přistupovat k lidským vstahům.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix example: <http://example.org/> .

<http://example.org/Jan-Novák> foaf:name "Jan Novák" .
<http://example.org/Ondřej-Novák> foaf:name "Ondřej Novák" .

example:Pavel-Novotný foaf:name "Pavel Novotný" ;
                      foaf:knows example:Jan-Novák ,
                                 example:Ondřej-Novák .
```

1.1.2 SPARQL jazyk

Jakyk SPARQL slouží na dotazování nad RDF daty a jejich manipulaci a je použit v definici konfigurací, kde jsou pomocí něj definovány dotazy na získání dalších vrcholů grafu.

2. Analýza požadavků

Tato kapitola popisuje požadavky, jež byly kladeny na vývoj systému.

Jak již bylo zmíněno v úvodu, cílem práce je vyrobit webovou aplikaci, jež komunikuje se serverem a je schopna číst konfigurace na základě kterých pak vizualizuje grafová data. Konfigurace a serverová část byla napsána mým vedoucím a tedy sloužila jako specifikace systémových požadavků na klientskou část. V následující podkapitole nejprve popíšu strukturu konfigurace.

2.1 Konfigurace

2.1.1 Rozšíření požadavků

Původní návrh konfigurací byl mírně jednodušší a považoval konfigurace pouze jako odkazy na seznam pohledů. Klientská aplikace tedy musela mít seznam konfigurací s jejich názvy počátečními vrcholy a dalšími parametry. Na základě debaty s mým vedoucím jsem se rozhodl konfigurace rozšířit o tyto parametry (název, počáteční vrcholy) a přidat meta konfigurace, jež slouží jako složky pro konfigurace. Níže je tedy popsán již nový návrh konfigurací.

V následujících sekcích jsou použity tyto RDF namespacy:

Prefix	IRI
browser	`https://linked.opendata.cz/ontology/knowledge-graph-browser/`
dct	`http://purl.org/dc/terms/`

2.1.2 Meta konfigurace

Meta konfigurace je skupina pro další konfigurace a meta konfigurace. Díky tomuto uživatel může procházet desítky různých konfigurací uspořádaných do složek obdobně jak v souborovém systému na počítači. Meta konfigurace má tyto vlastnosti:

- `dct:title` Název metakonfigurace *(je možné zadat ve více jazycích)*
- `dct:description` Detailnější popis, co daná metakonfigurace obsahuje *(je možné zadat ve více jazycích)*
- `browser:image` Odkaz na URL soubor s obrázkem reprezentujícím metakonfiguraci. Obrázek je pak zobrazen v aplikaci při procházení konfigurací.
- `browser:hasMetaConficuration` Meta konfigurace, které spadají pod tuto meta konfiguraci. *(je očekáváno více objektů)*
- `browser:hasConfiguration` Konfigurace, které spadají pod tuto metakonfiguraci. *(je očekáváno více objektů; popsáno dále)*

Meta konfigurací může být například „Wikidata“ nebo „Otevřená data ČR“.

2.1.3 Konfigurace

Konfigurace popisuje, jak by měla aplikace procházet datasety. Dvě konfigurace již byly zmíněny v úvodní kapitole. Aktuálně může mít konfigurace tyto vlastnosti:

- `dct:title` Název konfigurace *(je možné zadat ve více jazycích)*
- `dct:description` Detailnější popis, čeho je možné s konfigurací dosáhnout *(je možné zadat ve více jazycích)*
- `browser:hasVisualStyleSheet` Určuje, jak mají uzly v aplikaci vypadat. *(popsáno dále)*
- `browser:startingNode` Doporučený uzel nebo uzly, kde začít s procházením grafu. *(je očekáváno více objektů)*
- `browser:resourceUriPattern` Regulární výraz popisující, jak by mělo vypadat IRI uzlu. Používá se v aplikaci jako nápověda uživateli, zda zadal správné IRI ještě než se pošle požadavek.
- `browser:hasViewSet` View sety. *(popsáno dále)*
- `browser:autocomplete` JSON soubor se seznamem RDF uzlů podle kterých probíhá hledání. *(je očekáváno více objektů; popsáno dále)*

2.1.4 ViewSet

View set reprezentuje skupinu pohledů. Pravý smysl pohledů pochopí čtenář dále v textu. View set má následující vlastnosti:

- `dct:title` Název view setu *(je možné zadat ve více jazycích)*
- `browser:hasView` Pohledy které patří pod tento view set. *(je očekáváno více objektů; popsáno dále)*
- `browser:hasDefaultView` Výchozí pohled ze seznamu výše.
- `browser:hasCondition` SPARQL ASK dotaz jež určí, zda tato množina pohledů je aplikovatelná na konkrétní uzel.
- `browser:hasDataset` Dataset vůči kterému probíhá ASK dotaz. *(popsáno dále)*

2.1.5 View

Konkrétní pohled na uzel jež byl vysvětlen v úvodu. View má následující vlastnosti:

- `dct:title` Název pohledu *(je možné zadat ve více jazycích)*
- `dct:description` Popis pohledu *(je možné zadat ve více jazycích)*

- `browser:hasExpansion` Odkaz na expanzi - určuje jaké uzly lze získat z daného uzlu *(popsáno dále)*
- `browser:hasPreview` Odkaz na preview - určuje jaká data se mají získat pro ostylování konkrétního uzlu *(popsáno dále)*
- `browser:hasDetail` Odkaz na detail - určuje která data se zobrazí v detailu konkrétního uzlu *(popsáno dále)*

2.1.6 Expansion

Expansion popisuje jak lze daný uzel expandovat, tedy jedná se o operaci kdy se stahují nové uzly jež jsou nějak příbuzné expandovanému uzlu. Expanze vrací graf, tedy expandované uzly nemusí být přímými sousedy expandovaného. Jako expanzi si můžeme představit například „Zobraz všechny knihy co napsala daná osoba“. Expanze formálně partří k pohledu (view).

- `dct:title` Název expanze *(je možné zadat ve více jazycích, aktuálně se nepoužívá)*
- `browser:hasDataset` Popisuje dataset vůči kterému se dotazuje na data. *(popsáno dále)*
- `browser:query` Popisuje SPARQL SELECT dotaz který bude spuštěn na endpointu datasetu.

2.1.7 Preview

Preview popisuje která data v rámci daného pohledu (view) popisují daný uzel. Popisem se myslí taková data, kretá mají na svědomí stylování uzlu. V aplikaci je možné mít uzly různých barev a tvarů, to právě popisuje preview. Obdobně jako expanze, preview patří ke konkrétnímu pohledu.

Preview má stejné vlastnosti jako expanze. `browser:query` v tomto případě popisuje SPARQL dotaz, který vrátí graf obsahující daný uzel a jeho literály a z těchto literálů bude sestaven preview.

Konkrétně pro preview je predikát `browser:class` považován na třídy uzlu a ty jsou nastaveny jako třídy v knihovně Cytoscape[1], která je využíváná v klientské části na krelsení grafů.

2.1.8 Detail

Detail je poslední z trojice a poskytuje dodatečné informace k uzlu. Může se jednat o literály které nemá smysl v dané konfiguraci vykreslit do grafu jako uzly, proto budou zobrazeny v bočním panelu aplikace.

Detail má stejné vlastnosti včetně `browser:query` jako preview.

[1] `https://js.cytoscape.org/`

2.1.9 Dataset

Dataset popisuje SPARQL endpoint vůči kterému probíhá dotazování na data.

- `dct:title` Název datasetu *(aktuálně se nepoužívá)*
- `void:sparqlEndpoint` URL adresa SPARQL endpointu na kterou se posílají SPARQL dotazy
- `browser:accept` Popisuje HTTP Accept type v jakém formátu by měl endpoint svá data poskytnout. TODO: nemělo by to čistě záležet na serveru?

2.1.10 Visual style sheet

Popisuje jakž styl mají mít uzly v dané konfiguraci. Popis je založen na datech pocházejících právě z **preview**. Forma zápisu stylů aktuálně odpovídá stylům jaké používá knihovna Cytoscape.

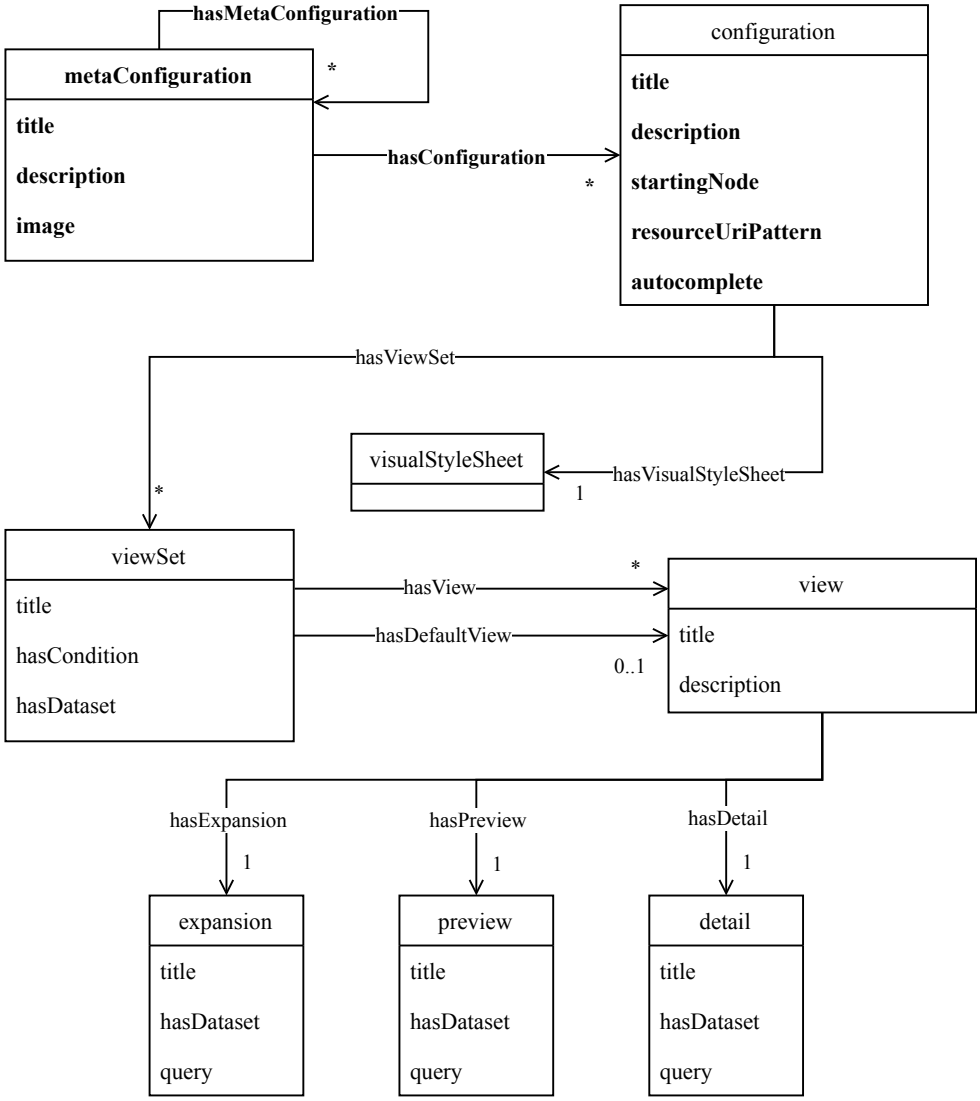
Visual style sheet má pouze jednu vlastnosti

- `browser:hasVisualStyle` Jedno pravidlo jak se má provést stylování, odkazuje na **Visual style**.

Visual style

Visual style má pak:

- `browser:hasSelector` Selector pro Cytoscape knihovnu, jež vybírá na které uzly nebo hrany daný styl bude aplikován.
- `browser:*` Jednotlivá pravidla jak má být daný uzel nebo hrana ostylována. Používají se přesně ty názvy, které používá knihovna Cytoscape, kupříkladu `browser:border-width` nebo `browser:background-color`.



Obrázek 2.1: Class diagram konfigurací. Tučně jsou zvýrazněné části jež nebyli součásti původní specifikace.

2.2 Server

Funkcionálním požadavkem bylo, aby aplikace komunikovala se serverem, který provádí dotazy a vrací data pro aplikaci. Server bude detailněji popsán v příští kapitole.

Server zprostředkovává následující požadavky:

- `/meta-configuration` - Vrátí informace o meta konfiguraci
- `/configuration` - Vrátí informace o konfiguraci
- `/stylesheet` - Vrátí visual style sheet
- `/view-sets` - K vrcholu a konfiguraci vrátí seznam view setů
- `/preview` - K vrcholu a pohledu vrátí preview
- `/detail` - K vrcholu a pohledu vrátí detail
- `/expand` - K vrcholu a pohledu vrátí expandované uzly

2.3 Uživatelské a systémové požadavky

Níže je sepsán seznam původních požadavků na klientskou aplikaci.

2.3.1 Konfigurace a stylesheet

Uživatel musí být schopen vybrat IRI konfigurace a visual style sheetu a lze je kdykoli změnit. IRI konfigurace může být vybráno jen jedno.

Analýza Konfigurace popisuje, jak je graf zobrazen a které pohledy jsou na uzly aplikovatelné. Není tedy možné mít dvě konfigurace v jednom grafu a změnou konfigurace tedy dojde ke smazání původního grafu.

2.3.2 Vložení vrcholů

Uživatel může ručně zadat IRI vrcholu který se následně zobrazí v grafu.

Analýza Pro úspěšné zobrazení vrcholu musí aplikace znát preview vrcholu. Preview lze ale získat pouze z konkrétního pohledu a je tedy nutné nejprve stáhnout view sety daného vrcholu, z nich následně vybrat defaultní a zvolit výchozí pohled. Pak je možné zavolat metodu `preview` na serveru a získat data o uzlu.

IRI uzlu budeme považovat za chybné, pokud server vrátí prázdnou množinu view sets. V takovém případě totiž nejde aplikovat žádné pohledy na uzel a tedy zřejmě nepatří do dané konfigurace.

Vrchol může být již v grafu přítomen, pak bude viditelně zvýrazněn. Pokud je uzel skrytý, bude odkryt. Pokud jsou zapnuté filtry a uzel je skrytý filtrem, uzel se v grafu nezobrazí. Pokud se uzel podaří načíst, nebo již existuje, bude vybrán a zobrazí se jeho detail v pravém panelu.

2.3.3 Detail vrcholu

Pokud uživatel klikne na vrchol, zobrazí se panel s podrobnými informacemi o uzlu. Bude zobrazen detail uzlu voláním metody `detail` na serveru a budou zobrazeny veškeré pohledy vrcholu, možnost je přepínat a provádět expanze podle daného pohledu.

Detail bude zobrazen jako dvousloupcová tabulka klíč-hodnota.

Panel bude zobrazovat také další možné akce k vrcholu.

Analýza Vrchol nemusí mít načtené view sets, preview ani detail, je tedy nutné po zobrazení panelu tyto informace stáhnout a během stahování zobrazit informaci, že se data stahují.

Může se stát, že `view-sets` vrátí prázdný výsledek. V takovém případě vrchol v grafu necháme i přes to, že podle původního požadavku bychom jej smazali.

Mezi akcemi bude lokalizace vrcholu, smazání, znovunačtení všech dat, zafixování pozice.

2.3.4 Skrývání uzlů

Uživatel může skrývat uzly v panelu s detailem. Skrytý uzel se v grafu skryje společně se všemi hranami a bude možné zobrazit seznam skytých uzlů ze kterého bude možné skryté uzly opět zviditelnit.

Analýza Bude přidáno tlačítko k detailu pro skrytí/zobrazení uzlu. Současně pro vetší přehlednost bude do detailu přidána hláška informující uživatele, že je uzel skrytý.

Skryté uzly nebude možno lokalizovat ale stále bude možné s nimi dále pracovat. Podle prvního požadavku se uzel opět zobrazí, pokud ho uživatel bude chtít explicitně vložit.

V panelu se skrytými uzly bude možnost uzly přímo zobrazit, nebo si zobrazit jejich detail. Detail skrytého uzlu funguje stejně jako pro viditelné uzly.

2.3.5 Expanze

Po dvojkliku na uzel se uzel expanduje podle aktuálního pohledu. Uzel je také možné expandovat v panelu s detailem kliknutím na tlačtko expanze u příslušného pohledu. Expanzí se zavolá metoda `expand` na serveru a v grafu se zobrazí nové vrcholy.

Analýza Pokud přidávaný vrchol v grafu již je a je skrytý, pak skrytým zůstane. Na rozdíl od přidávání jednoho vrcholu totiž explicitně neříkáme, že chceme daný vrchol přidat. Protože je možné vrcholy mazat, povolíme uživateli provádět expanzi znova, i když byla provedena.

Poznámka Aktuálně se expanze ukládají k vrcholům a je tedy možné zjistit, které vrcholy vznikly ze které expanze bez nutnosti znovu expanzi volat. Bohužel tato funkcionalita ještě není implementována a je řešena v poslední kapitole.

2.3.6 Filtrování vrcholů

Uživatel může přidat filtr, jež skryje nevyhovující vrcholy. Takovéto skryté vrcholy se pak chovají stejně jako uživatelem skryté vrcholy. Požadované filtry jsou

- Zobraz jen ty vrcholy, jež mají stupeň v daném intervalu nebo rozmezí.
- Zobraz vrcholy jen s konkrétním typem nebo třídou.

Analýza Aplikace by měla podporovat snadné přidání filtrů do budoucna a podporu pro pluginy, které dodávají vlastní možnosti filtrování. Aby bylo možné vyjádřit libovolné filtry, každý filtr by měl mít přístup k celému grafu a všem vrcholům.

Uzel může být skrytý jak filtrem, tak i uživatelem. Pro přehlednost by měl být uživatel informován, že nemůže ručně zobrazit vrchol který je skrytý filtrem. Tyto vrcholy budou stále zobrazeny v seznamu skrytých vrcholů pro možnost přístupu k nim.

Každý filtr určí, zda je vrchol podle tohoto filtru viditelný. Vrchol pak bude viditelný pokud všechny filtry určí, že viditelný je. Jedná se tedy o konjunkci.

Typy a třídy vrcholů nejsou známy předem a API serveru neumožňuje získat celou množinu typů a tříd. Je tedy nutné přizpůsobit filtry tak, aby si dokázaly poradit s novými uzly. Proto filtry na typ a třídu budou mít dva módy, v jednom módu explicitně skrývají zvolené vlastnosti a ve druhém explicitně zobrazují vrcholy s danými vlastnostmi. Toto chování pak ovlivní přidání nových, neznámých, vrcholů. V prvním případě nový vrchol bude zobrazen (pokud jeho typ, resp. třídy ještě nejsou známy) a ve druhém případě bude ihned skryt.

2.3.7 Vícejazyčné uživatelské rozhraní

Uživatel může přepínat mezi více jazyky uživatelského rozhraní. Aktuálně bude podporována pouze angličtina a čeština.

Analýza Kromě uživatelského rozhraní by měl být vícejazyčný i graf. Aktuální API serveru toto ještě neumožňuje a problém je předmětem poslední kapitoly. Multijazyčnost zatím podporují metody `meta-configuration` a `configuration` na serveru. Protože obecně můžou existovat překlady do spousty světových jazyků, je vhodné stahovat pouze požadovaný jazyk a při jeho změně stáhnout nový jazyk. Podrobněji je toto rozebráno v kapitole s implementací. TODO: ref

Pro velké grafy může být překlad problémem, protože změna jazyku může vyvolat spoustu požadavků na datové zdroje. Pro překlad grafu by tedy bylo nejlepší stáhnout překlad až když si uživatel zobrazí detail, nebo explicitně neoznačí vrcholy na překlad. Taktéž je vhodné oddělit výběr jazyka pro obsluhu aplikace a výběr konfigurací od překladu jednotlivých vrcholů. Příkladem může být procházení měst v Japonsku anglicky mluvícím uživatelem, kdy je žádoucí, aby názvy měst byly v originále.

2.3.8 Stažení grafu do souboru

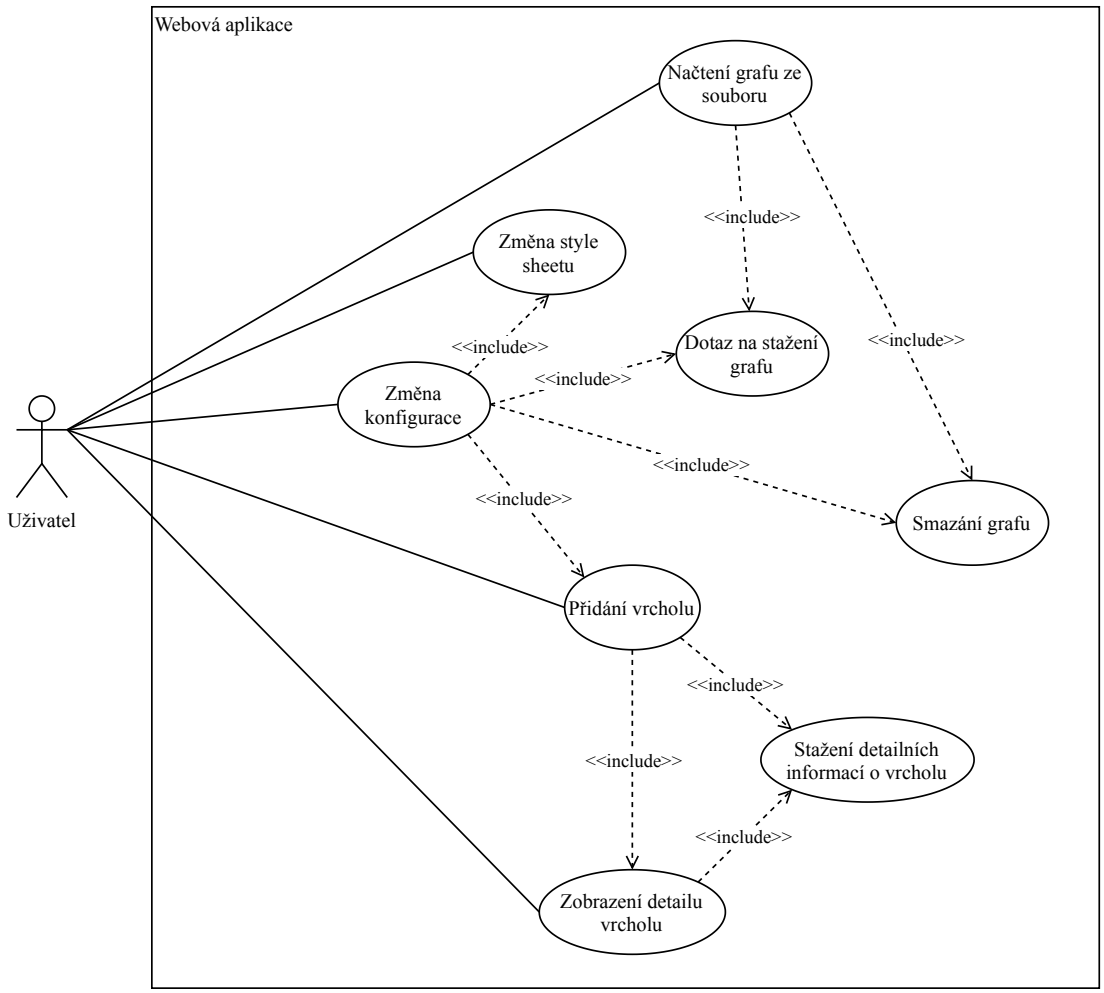
Uživatel má možnost stáhnout aktuální graf do souboru a později ho ze souboru načíst zpět.

Analýza Protože je aplikace ve vývoji, je třeba dbát na zpětnou kompatibilitu. Při každé nové verzi je tedy třeba ověřit, že soubor pochází ze staré verze a použít staré metody pro jeho zpracování a převedení do nového systému. Výsledný soubor může být zkomprimován pro menší velikost a nabízí se i možnost stáhnout jen základní informace tak, aby zbytek mohl být při načtení stažen ze serveru. Toto je opět řešeno v poslední kapitole.

Pokud uživatel bude chtít zavřít aplikaci, bude dotázán, zda chce aktuální graf uložit do souboru. Uložený graf již nebude blokovat stránku o uzavření dokud uživatel nesmaže, nebo nepřidá nové uzly. Stažení detailu, nebo změna pohledu nebude považována za změnu hodnou k uložení.

Pokud uživatel zvolí načtení nového souboru, starý graf bude zahozen a uživatel tedy bude požádán o uložení.

Kromě otevření souboru ze systému bude možnost soubor našíst i z webu.



Obrázek 2.2: Use case diagram vytváření grafu a základní práce s grafem.

2.3.9 Odstranění vrcholu

Uživatel může z grafu vrchol odstranit.

Analýza Odstraněním vrcholu se musí odstranit i všechny hrany patřící vrcholu.

Odstranit vrchol bude možné pomocí tlačítka v panelu s detailem, popřípadě skupinu vrcholů bude možné odstranit obdobným způsobem.

2.3.10 Vyhledávání vrcholů s pomocí autocomplete

Pokud to konfigurace umožňuje, bude možné přidávat nové uzly do grafu s pomocí autocomplete.

Analýza Data pro vyhledávání se budou stahovat až když uživatel bude chtít poprvé vyhledávat.

2.3.11 Podpora layoutů

Aplikace bude umožňovat několik způsobů layoutování grafu, které si bude moci uživatel volit.

- Pokud to layout umoňuje, bude možné ukotvit vrchol. To lze provést přesunutím vrcholu nebo pravým kliknutím myši. U ukotvených vrcholů bude zobrazena ikonka. Takovéto vrcholy pak nebudou layoutem ovlivněny.
- Layout reaguje na různé události, jako vytvoření skupiny, expanze vrcholu, přidání nového vrcholu do grafu, přesunutí vrcholu a na základě těchto událostí provádí layoutování grafu.
- Pokud to layout umožňuje, bude v pravém dolním rohu obrazovky tlačítko které spustí layoutování explicitně.
- Layout má vlastní nastavení.

2.3.12 Seskupování vrcholů

Vrcholy bude možné seskupovat do skupin které budou ve vizuálním grafu reprezentovány jedním uzlem. Pokud expanze vrátí větší množství nových vrcholů, vzniknou jako skupina. Skupinu bude možné rozbít dvojklikem.

- Z grafového hlediska skupina vzikne kontrakcí hran mezi vrcholy skupiny. To znamená, že pokud vedla hrana mezi vrcholem mimo skupinu a vrcholem ve skupině, povede tato hrana mezi vrcholem mimo skupinu a skupinou. Všechny násobné hrany stejného typu budou odstraněny. Obdobně toto platí i mezi dvěma skupinama. Pokud je ve skupině vrchol skrytý (ať uživatelem, nebo filtrem), jeho hrany se nepodílejí na utváření skupiny. Pokud jsou všechny vrcholy skupiny skryty, je skrytá i skupina.
- Skupina se na grafu chová jako obyčejné vrhcholy, je ji možné skrýt, přesouvat, ukotvit atp.

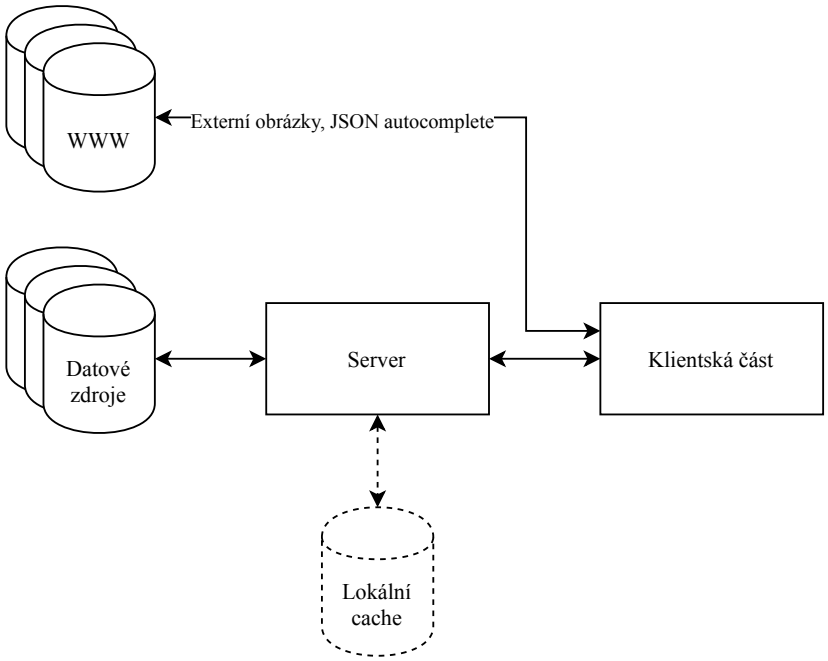
- Označením několika vrcholů se nabídne možnost vytvořit skupinu. Tato skupina pak vzikne na místě, kde byly původní vrcholy.
- Rozbít skupinu je možné dvojklikem, nebo tlačítkem z detailu skupiny. Vrchol skupiny bude odstraněn a vziknou místo něj původní vrcholy, které budou layoutovány.

3. Návrh architektury

Bylo zamýšleno, že klientská aplikace bude komunikovat se serverovou částí, která bude stahovat data z různých datových zdrojů a výsledky posílat zpět klientské aplikaci. Ta v případě potřeby bude stahovat přímo z internetu obrázky, pokud na ně uzly odkazují. Idea rozdělení klienta a serveru byla v tom, že v budoucnu může být na serveru implementováno chachování, aby aplikace byla plynulejší a neprobíhalo zatěžování datových zdrojů.[1]

Server tak slouží na odstínění od link datové vrstvy a v budoucnu pro cachování dat.

Chachování aktuálně implementováno ještě není, server je tedy kompletně bezestavový.



Obrázek 3.1: Komunikace mezi klientem, serverem a datovými zdroji. Cachování na serveru ještě implemontováno není.

3.1 Server

Jak již bylo zmíněno dříve, převážná část serveru byla napsána mým vedoucím Martinem Nečaským a sloužila i jako technická specifikace pro klientsou část. Server je napsaný v JavaScriptu a běží v Node.js[2]. Server je bezestavový a odpovídá na požadavky zmíněné dále.

[1] Při příliš rychlém stahování z Wikidat lze bohužel snadno dostat IP ban, tedy alespoň pro Wikipedii je cachování nezbytné.

[2] `https://nodejs.org/`

3.1.1 Jazyková podpora

Server jsem částečně přepsal, aby podporoval dotazování na data z více světových jazyků. Některé požadavky přijímají parametr `languages` obsahující čárkou (`,`) oddělené ISO 639-1 jazykové kódy. Server pak vrací objekty jejíž klíčem je jazykový kód a hodnotou daný překlad do jazyku. Pokud překlad neexistuje, hodnotou je `null`. V případě, že na všechny jazyky bylo vráceno `null`, server se pokusí přidat další jazyk, který existuje. Který jazyk takto bude vybrán není určeno.

Příklad. Pro `languages=cs,en` může server vrátit například

```
{
    cs: null,
    en: "Kankakee County"
}
```

ale pokud nezná překlad ani do češtiny, ani do angličity, může vrátit

```
{
    cs: null,
    en: null,
    sk: "Jazero Beňatina"
}
```

3.1.2 API

Server vrací data ve formátu JSON, požadavky jsou posílány metodou GET a parametry jsou kódovány do URL adresy.

/metaconfiguration

parametry: `iri` a `languages`

Vrátí informace o metakonfiguraci zadané podle `iri`.
Vrátí všechna data (viz kapitola 2.1.2) o metakonfiguraci, veškerá data o dceřiných konfiguracích (viz kapitola 2.1.3) a základní data o dceřiných metakonfiguracích (vše kromě seznamu konfigurací a metakonfigurací).

```
interface ResponseMetaConfiguration extends
ResponseMetaConfigurationBase {
    has_meta_configurations: ResponseMetaConfigurationBase[],
    has_configurations: ResponseConfiguration[],
}

interface ResponseMetaConfigurationBase {
    iri: string,
    title: {[language: string]: string},
    description: {[language: string]: string},
    image: string,
}
```

/configuration

parametry: `iri` a `languages`

Vrátí informace o konfiguraci zadané podle `iri`.
Vrátí stejná data jako `/metaconfiguration` o svých sceřiných konfiguracích.
Toto volání se používá pouze když uživatel ručně zvolí IRI konfigurace, v opačném případě si aplikace vystačí s voláním `/metaconfiguration`.

```
interface ResponseConfiguration {
    iri: string,
    stylesheet: string[],
    title: {[language: string]: string},
    description: {[language: string]: string},
    autocomplete: string[],
    starting_node: string[],
    resource_pattern: string|null,
}
```

/stylesheet

parametry: `stylesheet`

Vrátí kompletní visual style sheet (viz kapitola 2.1.10) na základě jeho IRI jako parametr `stylesheet`.

```
interface ResponseStylesheet {
    styles: {
        selector: string;
        properties: {
            [property: string]: string;
        }
    }[];
}
```

/view-sets

parametry: `config` a `resource`

Vrátí seznam možných view setů (viz kapitola 2.1.4) které odpovídají uzlu s IRI `resource` při dané konfiguraci `config`.

```
interface ResponseViewSets {
    viewSets: {
        iri: string;
        label: string;
        defaultView: string;
        views: string[];
    }[];
    views: {
        iri: string;
        label: string;
    }[];
}
```

/preview

parametry: `view` a `resource`

Vrátí data z dotazu preview (viz kapitola 2.1.7) na uzel s IRI `resource` při daném pohledu `view`.

```
interface ResponsePreview {
    nodes: ResponseElementNode[];
    types: ResponseElementType[];
}
```

/detail

parametry: `view` a `resource`

Vrátí data z dotazu detail (viz kapitola 2.1.8) na uzel s IRI `resource` při daném pohledu `view`.

```
interface ResponseDetail {
    nodes: {
        iri: string;
        data: {
            [IRI: string]: string;
        };
    }[];
    types: ResponseElementType[];
}
```

/expand

parametry: `view` a `resource`

Vrátí expandované uzly (viz kapitola 2.1.6) pro uzel s IRI `resource` při daném pohledu `view`. Tyto expandované uzly již obsahují data o detailu a tedy není třeba žádného dalšího volání.

```
interface ResponseExpand {
    nodes: ResponseElementNode[];
    edges: ResponseElementEdge[];
    types: ResponseElementType[];
}
```

Pomocné interfaces pak jsou

```
interface ResponseElementType {
    iri: string;
    label: string;
    description: string;
}

interface ResponseElementEdge {
    source: string;
```

```
    target: string;
    type: string;
    classes: string[];
}

interface ResponseElementNode {
    iri: string;
    type: string;
    label: string;
    classes: string[];
}
```

4. Implementace

4.1 Vue.js framework

Klientská část aplikace je postavena nad Vue.js frameworkem, jež je populární JavaScriptový framework na stavbu uživatelských rozhraní. Protože některé z jeho funkcionalit byly použity v klíčových částech aplikace, je nutné čtenáře obeznámit alespoň se základním principem fungování frameworku.

4.1.1 Vuex

Vue.js framework, podobně jako konkurenční React[1] (Facebook) nebo Angular[2] (Google), využívají principu sledování stavu aplikace (jejich dat) pro automatickou změnu DOMu webové stránky. V praxi to znamená, že programátor může velice snadno napsat kód, který generuje uživatelské rozhraní na základě dat, která mohou být libovolně měněna bez nutnosti řešit problém, zda ke změně vůbec došlo a které části aplikace mají být o změně stavu informovány. Ve Vue tuto funkcionalitu zastává právě Vuex[3], jež je možný používat samostatně.

Vuex drží stav aplikace jako jeden objekt (tedy slouží jako centrální úložiště dat pro celou aplikaci). Tento objekt se nazývá **store**. Změny ve storu mohou být sledovány Vuexem pro vykonání libovolných akcí, například překreslení textu na stránce, jež byl vykreslen Vue frameworkem.

Vrátíme-li se k původnímu příkladu, programátorovi stačí přiřadit do proměnné, jež je spravovaná Vuexem, novou hodnotu a Vuex se postará o zavolání všech komponent, které tuto proměnnou využívají a tyto komponenty na stránce překreslí původní hodnotu na novou. Překreslení přitom proběhne až poté, co skončí průběh aktuální funkce. Tohoto je docíleno pomocí `Window.requestAnimationFrame()`. Díky tomuto můžeme stav v rámci průběhu jedné funkce modifikovat vícekrát se skoro nulovým dopadem na celkový výkon aplikace.

Computed properties

Kromě této funkcionality Vuex nabízí takzvané **gettery**, jež jsou ve Vue frameworku nazývány jako **computed properties**. Jedná se o funkce, které využívají data ze storu pro výpočet dat nových. Výhoda takovýchto getterů je ta, že Vuex dokáže výsledky těchto funkcí cachovat a přepočítává je pouze tehdy, změní-li se data původní. Interně gettery fungují tak, že při zavolání klientské funkce Vuex sleduje které části storu byly dotázany a ty pak sleduje na změnu jež invaliduje cache konkrétního getteru. Při příštím požadavku na hodnotu se pak klientská funkce volá znovu a celá operace se opakuje.

Tyto computed properties jsou v aplikaci využívány často. Kupříkladu funkce, která počítá, zda je sousední uzel vybrán. Na takovouto hodnotu se v aplikaci mohu ptát libovolně krát, ale počítá se pouze tehdy, když se množina sousedních uzlů vrcholu změní, nebo se změní právě označení uzlu z množiny.

[1] `https://reactjs.org/`
[2] `https://angularjs.org/`
[3] `https://vuex.vuejs.org/`

Watchers

Ve Vue lze využívat i **watch**ery, které umožňují registraci callbacku na změnu určité proměnné ve stavu. Watchery má smysl využívat tam, kde již data přestávají být spravována Vue frameworkem, tedy u knihoven třetích stran. Watchery, podobně jako překreslení komponent, jsou volány až po skončení probíhající funkce.

Změna stavu

Jak již bylo zmíněno, stav lze měnit přiřazením do proměnné, popřípadě voláním metod jako `.push()` na poli. Vuex dokáže tyto změny sledovat nahrazením původní proměnné (máme na mysli položku objektu) JavaScriptovým setterem. U polí dojde k obalení metody `.push()` jež registruje změnu stavu.

Tento přístup má několik nevýhod, které se promítly i při vývoji klientské aplikace:

- Vue není plně kompatibilní s novými **ES6 kontejnery** `Map` a `Set` a proto jsou v aplikaci používány jen v rámci lokálních proměnných a mapa je nahrazena klasickým objektem.
- Protože je v JavaScriptu nemožné sledovat **vytvoření nové property objektu**, musí programátor v tomto případě volat ručně `Vue.set(target, propertyName/index, value)` a `Vue.delete`, popřípadě vytvořit nový objekt který nahradí ten původní.
- Je důležité mít na paměti, že data ve storu již nejsou původními objekty v pravém slova smyslu, ale veškeré fieldy a metody u polí byly nahrazeny, jak je zmíněno výše. Proto předání objektů a polí ze storu knihovnám třetích stran je nutné ošetřit **oklonováním objektu**, jinak může dojít k zaseknutí aplikace.
- Instance tříd **knihoven třetích stran může zaseknout aplikaci**, pokud ji uložíme do storu. Bohužel, tohoto je velmi snadné ve Vue frameworku dosáhnout omylem. Problém je rozebrán v následující kapitole.

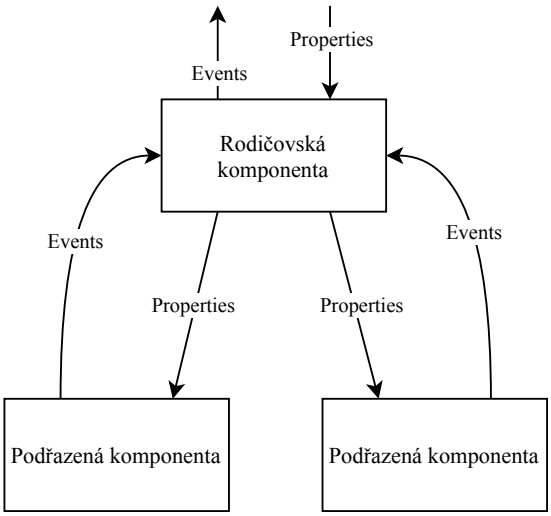
4.1.2 Vue framework

Vue framework využívá takzvané komponenty. Komponentou se rozumí prvek na stránce se kterým má smysl pracovat samostatně. Každá komponenta má vlastní HTML, CSS a JS. Komponenty se mohou do sebe zanořovat a vytvářet tak větší komponenty z menších. Příkladem může být komponenta *seznam* jež dokáže na stránku vykreslit senznam prvků. Takováto komponenta by pak mohla mít podkomponenty jako *prvek seznamu*.

Každé komponentě lze předat data formou **properties** přičemž komponenta na základě těchto dat může vyrobit pod sebou další komponenty a předat jim část dat, která dostala.

Tyto předávána data jsou právě data ze storu. Vue framework doporučuje, aby data co koponenta dostane formou properties neupravovala přímo, ale místo toho posílala události rodičovské komponentě, která data upraví. Ve své práci jsem se

rozhodl toto doporučení ignorovat, neboť by tímto vzrostla náročnočnost na správu aplikace.



Obrázek 4.1: Doporučený způsob komunikace mezi komponentami ve Vue frameworku

4.1.3 Loaders

Kód Vue komponent se zapisuje do souborů s příponou `.vue`. Při sestavování aplikace se pak použije `vue-loader` který ze souboru vyextrahuje zvlášť CSS, JS a HTML a ty předá dál na zpracování. HTML kód komponent není ve skutečnosti pravý HTML. Jedná se nadstavbu umožňující psát speciální značky, jež rozhodují kolikrát a jestli vůbec se tag na stránce vyrenderuje. Tato HTML nadstavba je pak předána `vue-template-compiler` který vyrobí optimalizovaný JS kód jež renderuje HTML na základě stavu komponenty.

Příklad. Ukázka jednoduché Vue komponenty `IndexedList` která má parametr `list` očekávající pole stringů. Tato komponenta vypíše pole v odrážkovém seznamu ve formě `index: hodnota`. Komponenta se sama stará o překreslení DOMu, když se změní data. Komponentu můžeme v jiné komponentě použít vložením `<indexed-list :list="inputData"/>` kde `inputData` je proměnná obsahující pole stringů.

```
<template>
  <ul>
    <li v-for="(item, index) in list" :key="index">
      {{index}}: {{ item }}
    </li>
  </ul>
</template>

<script lang="ts">
  import {Component, Prop} from "vue-property-decorator";
  import Vue from "vue";
  @Component
  export default class IndexedList extends Vue {
    @Prop() private list: string[];
  }
</script>

<style scoped lang="scss">
  ul {
    color: red;
  }
</style>
```

Scoped styly

Můžeme si povšimnout `scoped` stylů v ukázce. Vue má mechanismus, že styly které zde nastavíme se aplikují jen na tuto komponentu. Nastavením červené barvy na seznam jsme tedy skutečně nastavili červenou barvu jen této komponentě a ostatní seznamy jsou netknuté. Toto má nespornou výhodu pokud pracujeme s velkým množstvím komponent a hrozilo by, že bychom museli používat složitě pojmenované css třídy aby nedošlo ke kolizi.

Pokud bychom chtěli ovlivnit styly vnořených komponent a máme nastavené scoped styly, musíme použít pseudoselektor `::v-deep`, kupříkladu `.actions ::v-deep .v-input-selection-controls`. Tohoto je hodně používáno poukud je potřeba upravit styly Vuetify frameworku (viz dále).

5. Uživatelské testování

Závěr

Anděl (2007, str.29)

Seznam použité literatury

Anděl, J. (2007). *Základy matematické statistiky.* Druhé opravené vydání. Matfyzpress, Praha. ISBN 80-7378-001-1.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

A. Přílohy

A.1 První příloha