Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

WebGephi - Webové rozhraní pro Gephi

Bc. Václav Čokrt

Vedoucí práce: Ing. Jaroslav Kuchař

23. června 2014

Poděkování

Děkuji Ing. Jaroslavu Kuchařovi za možnost pracovat na zajímavém tématu, své přítelkyni za podporu a trpělivost během mé studijní vytíženosti a v neposlední řadě samozřejmě své rodině za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 23. června 2014 .....

Odkaz na tuto práci

Čokrt, Václav. *WebGephi - Webové rozhraní pro Gephi.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstrakt

Cílem této práce je vytvoření webové aplikace *WebGephi*, která zpřístupní funkcionalitu aplikace *Gephi*[1] pomocí RESTful webových služeb.

První část práce je zaměřena na analýzu aplikace *Gephi* a specifikaci požadavků nové aplikace. Na základě těchto požadavků je navrhnuta architektura aplikace a struktura REST rozhraní. Protože aplikace bude sloužit především jako poskytovatel služeb jiným aplikacím, práce se podrobně zabývá i možnými způsoby autorizace a autentizace. Závěrečná část práce se zabývá samotnou implementací *WebGephi*.

Součástí řešení je i implementace ukázkové klientská aplikace - grafické nadstavby - demonstrující funkcionalitu *WebGephi*.

Klíčová slova Gephi, Gephi Toolkit, graf, REST, RESTful, webové služby, webová aplikace

[1] *Gephi*[1] je opensource desktopová aplikace sloužící k vizualizaci a manipulaci s grafy.

Abstract

The aim of this thesis is to create the *WebGephi* application that exposes the *Gephi*[2] application functionality as a RESTful web service.

The first part is focused on *Gephi* analysis and requirements specification of the new application. Based on these requirements, application architecture and structure of the REST interface is designed. Because the main purpose of *WebGephi* is to provide services to other applications, this thesis focuses on possible ways of authorization and authentication too. The final part discusses the *WebGephi* implementation.

Part of the solution will also be a sample client application - graphical interface - which demonstrates the *WebGephi* functionality.

Keywords Gephi, Gephi Toolkit, graph, REST, RESTful, web services, web application

[2] *Gephi*[1] is an open source desktop application for graphs visualization and manipulation.

Obsah

Seznam obrázků

Seznam ukázek zdroj. kódu

Úvod

„Graf je základním objektem teorie grafů. Jedná se o reprezentaci množiny objektů, u které chceme znázornit, že některé prvky jsou propojeny. Objektům se přiřadí vrcholy a jejich propojení značí hrany mezi nimi. Grafy slouží jako abstrakce mnoha různých problémů. Často se jedná o zjednodušený model nějaké skutečné sítě (například dopravní), který zdůrazňuje topologické vlastnosti objektů (vrcholů) a zanedbává geometrické vlastnosti, například přesnou polohu.[2]“

Graf tedy můžeme chápat jako zjednodušený obraz nějaké skutečnosti - seznam lidí a vztahů mezi nimi, množina webových stránek a hypertextových odkazů, cokoli, co lze znázornit jako množinu uzlů a hran (vztahů mezi nimi).

Gephi[1] je desktopová platforma sloužící k analýze grafů. Umožňuje interaktivně měnit rozložení grafů na základě jejich struktury, vypočítávat metriky (PageRank, shlukovací koeficient, ...), filtrovat uzly podle jejich vlastností a mnoho dalšího. To vše slouží k tomu, aby uživatel byl schopný v grafu najít skryté závislosti a mohl lépe pochopit (a vizualizovat) strukturu grafu.

Jednou z hlavních výhod *Gephi* je jeho rozšiřitelnost. Kdokoli může vytvořit svou vlastní funkci k manipulaci s grafem a ve formě pluginu ji přidat do aplikace.

Aplikace *WebGephi* by měla zachovat tyto vlastnosti a navíc přidat výhody plynoucí ze standardizovaného rozhraní webové aplikace.

Kapitola 1

Cíl práce

Cílem této práce je vytvořit webovou aplikaci poskytující funkcionalitu *Gephi*. Hlavní rozhraní této aplikace bude založeno na architektuře REST.

„ REST (Representational State Transfer) – je architektura rozhraní, navržená pro distribuované prostředí. REST navrhnul a popsal v roce 2000 Roy Fielding (jeden ze spoluautorů protokolu http) v rámci disertační práce Architectural Styles and the Design of Network-based Software Architectures. V kontextu práce je nejzajímavější kapitola 5, ve které Fielding odvozuje principy RESTu na základě známých přístupů k architektuře. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). REST je tedy na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.“[3]

WebGephi bude sloužit jen jako poskytovatel služeb pro jiné (klientské) aplikace. Tyto aplikace budou pomocí webových služeb přistupovat k *WebGephi* (nahrávat grafy, aplikovat na ně funkce, exportovat výsledky, ...) a komunikovat s koncovým uživatelem přes vlastní grafické rozhraní. Klientské aplikace mohou být webové, desktopové i mobilní aplikace. Přístup klientských aplikací samozřejmě bude také třeba řídit. *WebGephi* tedy kromě REST rozhraní bude poskytovat také grafické rozhraní pro registraci a správu uživatelů a klientských aplikací.

Hlavní výhody *WebGephi* budou jednoduché, standardizované, deklarativní rozhraní. *Gephi* sice poskytuje knihovnu pro práci s grafy, *Gephi Toolkit*[4], ta je však dosti složitá na použití. Klientské aplikace budou moci jednoduše přistupovat k tomuto rozhraní pomocí HTTP protokolu, na kterém je v naprosté většině založena REST architektura. To umožní jak jednoduché prozkoumávání rozhraní (např. pomocí webového prohlížeče), tak strojové zpracování koncovými klientskými aplikacemi.

Pro koncového uživatele je hlavní výhodou možnost použití bez nutnosti instalace (ve spolupráci s klientskými aplikacemi), použití jako centrálního

úložiště grafů a možnost sdílení s jinými uživateli.

WebGephi bude postaveno nad již zmíněnou knihovnou *Gephi Toolkit*. Ta obsahuje základní moduly *Gephi* (bez GUI[3] funkcionality) ve formě jednoduché Java knihovny. V prvním kroku bude třeba určit, jaká funkcionalita bude implementována ve *WebGephi*. Následně bude třeba analyzovat strukturu knihovnu *Gephi Toolkit* a s její pomocí tuto funkcionalitu implementovat. Dále bude potřeba navrhnout a implementovat strukturu REST rozhraní a způsob autentizace uživatelů.

Součástí práce je i implementace ukázkové klientské aplikace. Ta bude využívat naprostou většinu funkcionality WebGephi a demonstrovat její funkčnost.

[3]Graphical User Interface - grafické uživatelské rozhraní

Kapitola 2

Analýza

Cílem této kapitoly je hlubší náhled do problematiky a analýza dostupných nástrojů. Na konci kapitoly bychom měli mít jasno v tom, zda a jak je tento úkol řešitelný.

Nejdříve si představíme aplikaci *Gephi* a její možnosti. Určíme, jakou funkcionalitu bude možné a vhodné přenést do *WebGephi*. Dále představíme strukturu knihovny *Gephi Toolkit* a nastíníme, zda a jakým způsobem bude možné požadovanou funkcionalitu implementovat. Aplikace bude také muset řešit autentizaci a autorizaci přístupu. V poslední části si tedy také představíme standardní řešení tohoto problému.

2.1 Gephi

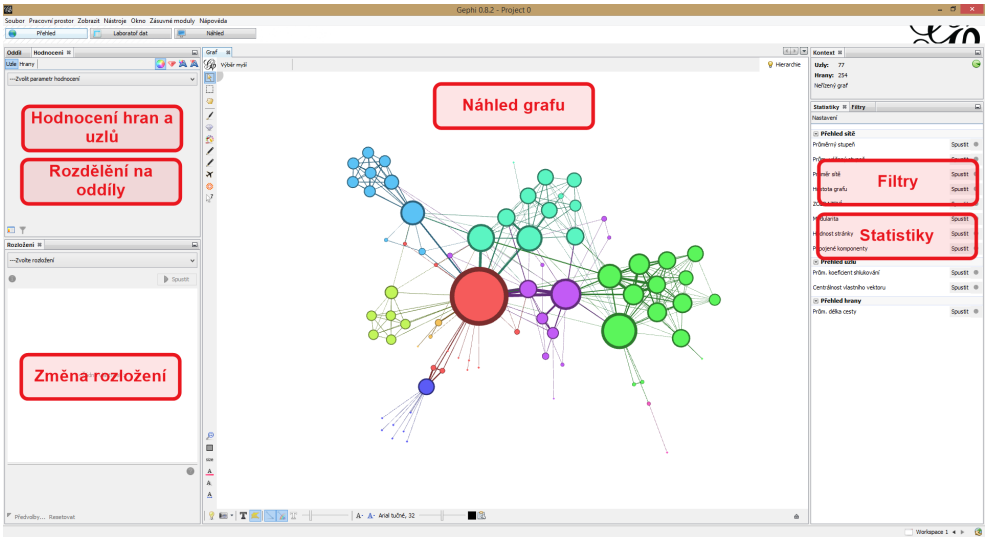
„Gephi je platforma pro vizualizaci a prozkoumávání všech druhů sítí a komplexních systémů, dynamických a hierarchických grafů. Je dostupný pro Windows, Linux a Mac OS x. Gephi je open-source a zdarma.“[4]

Jedná se tedy o volně dostupný, bezplatný nástroj pro práci s grafy. První veřejně dostupná verze měla číslo 0.6alpha1 a vyšla v roce 2008. Původně se jednalo o studentskou práci francouzských studentů[5]. V současnosti se o jeho vývoj stará „Gephi Consortium“[5].

Přesto, že i současná verze (0.8.2-beta) je stále betaverze, jedná se o široce používaný software s rozsáhlou funkcionalitou. Budoucí verze číslo 0.9 by měla být výrazným krokem v evoluci *Gephi*, bude obsahovat změny v samotném jádru aplikace. Jedná se o velmi živý projekt, který je v neustálém vývoji.

[4]Volný překlad z oficiálních stránek *Gephi*[1]. Orig.: „Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. Runs on Windows, Linux and Mac OS X. *Gephi* is open-source and free.“

[5]The University of Technology of Compiègne (Université de Technologie de Compiègne or UTC)



Obrázek 2.1: Záložka „Přehled“

Gephi je napsáno v programovacím jazyku Java. Je založeno na *NetBeans*[6] platformě, grafické rozhraní využívá framework Swing.

2.1.1 Struktura aplikace

Zde si představíme strukturu aplikace *Gephi* a její uživatelské rozhraní.

Základní jednotkou při práci s *Gephi* je „Projekt“. Aplikace může pracovat vždy jen s jedním projektem. Projekt je nejvyšší jednotkou v *Gephi*, může být ukládán a načítán z disku a může obsahovat jeden nebo více „Pracovních prostorů“ (Workspace). V aplikaci je vždy aktivní jen jeden Workspace. Workspace představuje pracovní plochu, na které se mohou vytvářet a editovat grafy.

Uživatelské rozhraní se skládá ze tří záložek - Přehled, Laboratoř dat a Náhled.

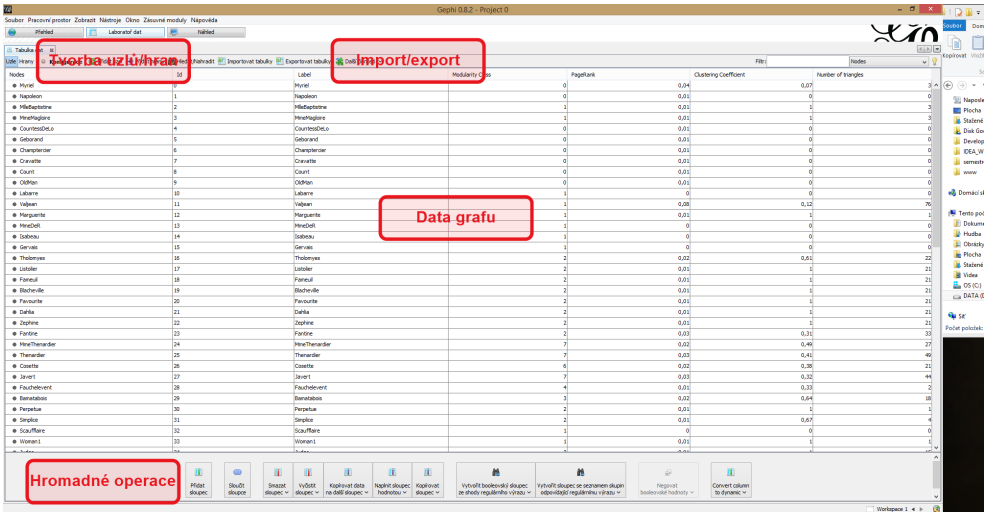
2.1.1.1 Záložka Přehled

Jedná se o nejdůležitější záložku (obr. 2.1). Na středu je vidět náhled grafu a po stranách jsou dostupné funkce, které lze na graf aplikovat. Po aplikaci funkce je graf okamžitě překreslen. S grafem lze manipulovat i ručně pomocí nástrojů na svislé liště na levé straně náhledu grafu.

2.1.1.2 Záložka Laboratoř dat

Tato záložka (obr. 2.2) slouží k úpravě zdrojových dat grafu. Uzly a hrany lze vkládat ručně nebo importovat z různých zdrojů (soubor, databáze, ...). Na

[6] https://*NetBeans*.org/



Obrázek 2.2: Záložka „Laboratoř dat“

středu je vidět seznam uzlů a hran aktuálního grafu. S daty lze provádět po sloupcích (atributech) i hromadné operace.

2.1.1.3 Záložka Náhled

Poslední záložka (obr. 2.3) slouží k vizuální úpravě grafu. Typicky se používá před exportem výsledku do obrázku. Lze nastavovat velikost hran, uzlů, nastavení popisků, průhlednost, . . . Výsledný graf lze exportovat ve formátu png, svg a pdf.

2.1.2 Funkcionalita

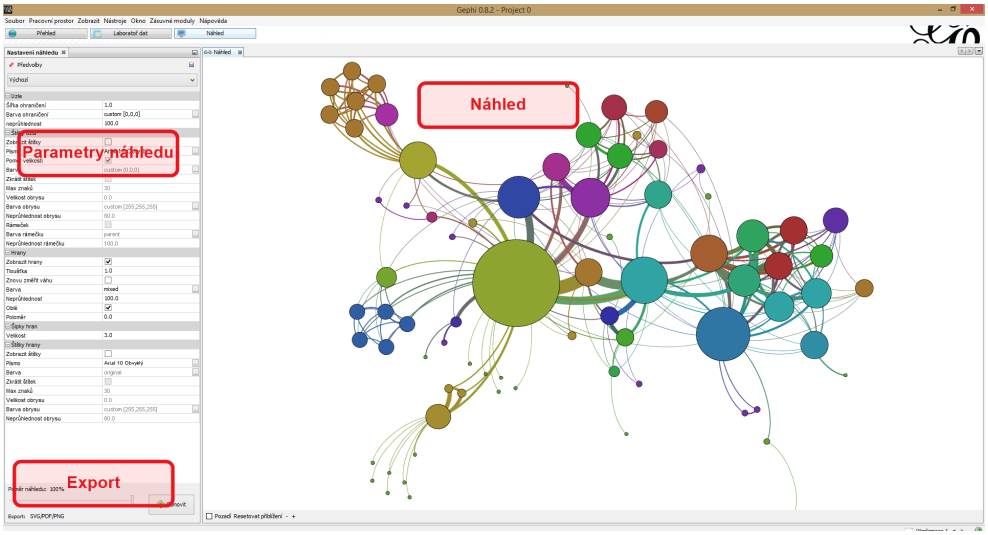
Zde podrobně popíšeme funkcionalitu *Gephi*.

2.1.2.1 Import a export

Gephi ukládá projekty do svého interního formátu `gephi`. Ten kromě samotného grafu ukládá i strukturu projektu a jeho nastavení. Pro výměnu dat mezi aplikacemi jsou důležitější standardizované formáty. *Gephi* umí načítat téměř jakýkoli formát vhodný pro ukládání grafu, včetně `csv`, `graphML` a `gexf`. Do těchto formátů umí samozřejmě grafy i exportovat.

Nejvhodnější z těchto formátů je jednoznačně formát `gexf`[6]. Ten je schopen uložit všechny důležité informace včetně hierarchických a dynamických grafů a je prakticky standardem to tomto oboru. Jedná se o xml formát s relativně jednoduchou strukturou[7]. Základem je seznam uzlů a hran (včetně

[7]Přesná struktura je definována standardně pomocí xml schématu: http://www.gexf.net/1.2draft/gexf.xsd



Obrázek 2.3: Záložka „Náhled“

jejich atributů).

V příloze B.1 můžete vidět porovnání dostupných formátů.

Gephi je schopno načítat grafy také SQL databáze (stačí specifikovat potřebný SQL dotaz).

Kromě datového exportu je dostupný taká export vizuální reprezentace grafu. Dostupné jsou formáty `pdf`, `svg` a `png`.

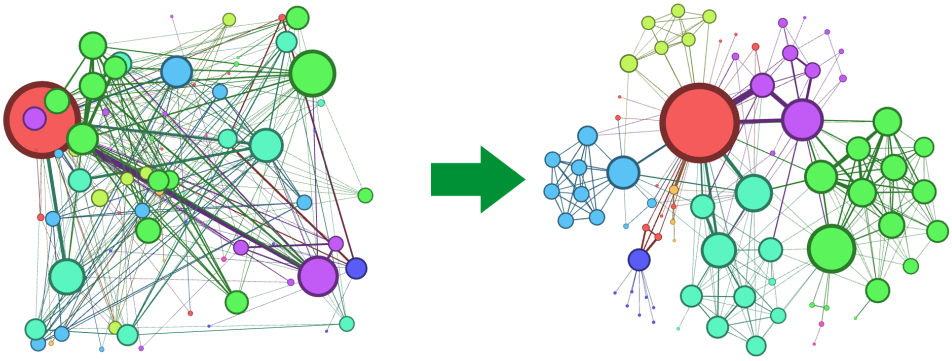
2.1.2.2 Grafové funkce

Hlavní silou *Gephi* je manipulace s grafem jako s celkem. *Gephi* už v základu obsahuje množství takovýchto funkcí a je možné je dále rozšířit pomocí pluginů. Funkce jsou podle zaměření rozděleny do několika skupin.

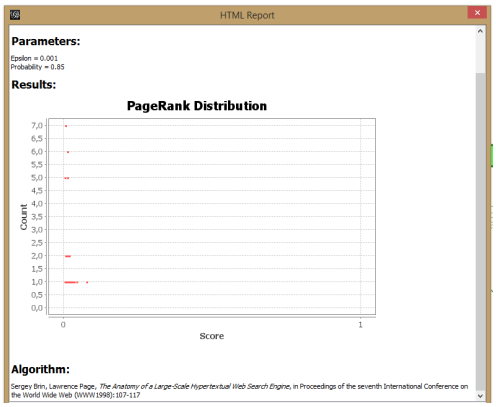
Rozložení Funkce pro změnu rozložení slouží k systematickému přeskupení uzlů. Může se jednat o velmi jednoduché funkce, např. `Otočení podle směru hodinových ručiček`, která jen otočí graf o definovaný úhel, ale i o velmi specifikované funkce, jako je `Force Atlas`, který přeskupuje uzly podle jejich vazeb k okolí. Obecně se jedná o jakékoli funkce, které mění poziční atributy uzlů.

Některé funkce podporují interaktivní režim, kdy se funkce spustí v nekonečné smyčce, takže okamžitě vidíme, jaký vliv na rozložení grafu má změna atributů. Když jsme s rozložením spokojeni, stačí funkci zastavit. Ukázku změny rozložení můžete vidět na obr. 2.4.

Statistiky a metriky Tako kategorie funkcí je určená k výpočtu metrik nad grafem. Výstupem funkce je přidání jednoho nebo více atributů k uzlům



Obrázek 2.4: Změna rozložení podle funkce `Foce Atlas 2`.



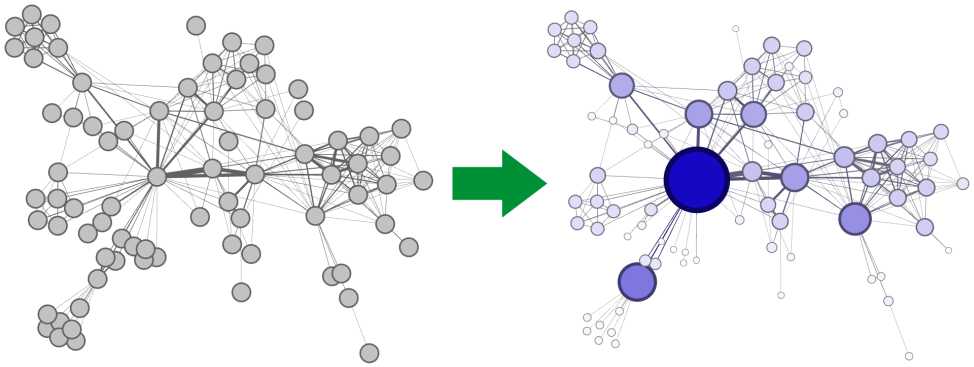
Obrázek 2.5: Report po aplikaci funkce `PageRank`PageRank

nebo hranám. Navíc je zobrazena shrnující zpráva (obr. 2.5), která typicky prezentuje rozložení hodnot atributů v grafu. Jako příklad statistické funkce lze uvést např. `Hodnost stránky` (PageRank[8]).

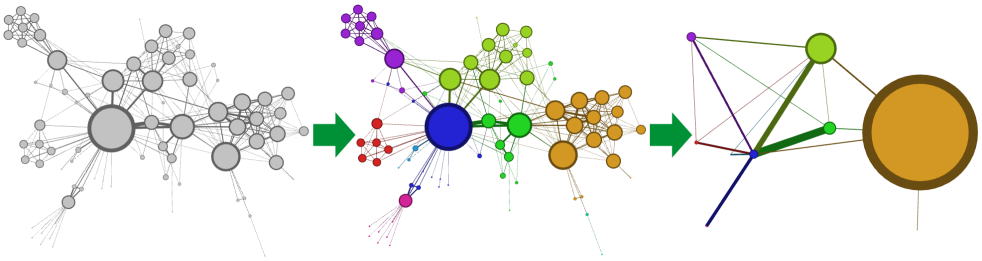
Hodnocení Hodnocení slouží k lepší vizuální reprezentaci hodnot atributů (vypočtených např. pomocí již zmíněných statistických funkcí). Podle hodnoty atributu lze měnit barvu nebo velikost uzlů a hran. Aplikaci „hodnocení“ můžeme vidět na obr. 2.6.

Rozdělení na oddíly Rozdělení na oddíly má podobnou funkci jako „hodnocení“. V tomto případě se obarvují uzly se stejnou hodnotou atributu na stejnou barvu (lze použít jen na celočíselné atributy). Tyto uzly lze také seskupit do

[8] Algoritmus pro ohodnocení důležitosti webových stránek na základě struktury hypertextových odkazů.



Obrázek 2.6: Hodnocení barvou a velikostí podle funkce `PageRank`.



Obrázek 2.7: Rozdělení na oddíly podle hodnoty atributu `Modularity Class`.

jednoho uzlu, jehož velikost je závislá na počtu uzlů, které obsahuje. Ukázku můžete vidět na obr. 2.7.

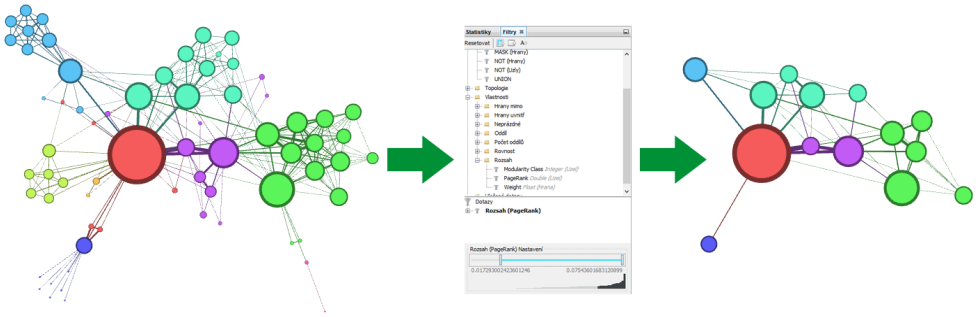
Filtry Ve velkých grafech je pro uživatele těžké se orientovat, je zahlcen velkým množstvím uzlů a hran. Pro tento účel obsahuje *Gephi* filtry. Na základě nejrůznějších podmínek, většinou závislých na hodnotě atributů, lze uzly filtrovat (odstranit) z grafu. Jednoduché filtry lze navíc řetězit pomocí logických operátorů do složitějších výrazů.

Příklad aplikace jednoduchého filtru můžete vidět na obr. 2.8.

Generování grafu Pokud nemáme žádný vhodný graf (např. pro testování vyvíjeného pluginu), *Gephi* poskytuje možnost vygenerování grafu. V základu je možnost generování jen náhodného grafu, kde můžeme nastavit pouze požadovaný počet hran a uzlů. Existují však pluginy, které možnosti *Gephi* v tomto směru dále rozšiřují.

2.1.2.3 Dynamické grafy

Gephi podporuje také dynamické grafy. Jedná se o běžný graf, kde každý uzel a hrana má nastavený interval platnosti. Většinou se jedná o časový rozsah.



Obrázek 2.8: Aplikace filtru `Rozsah` na základě hodnoty funkce `PageRank`.

Tímto způsobem lze zachytit vývoj skutečnosti, kterou graf představuje, v čase.

Gephi, kromě běžných operací, umí pomocí časové osy takovýto graf přehrát. Jedná se vlastně o specifický filtr na základě hodnoty intervalu.

2.1.2.4 Manuální editace

Gephi poskytuje také možnosti manuální editace, a to jak grafického znázornění grafu, tak datového modelu. V okně „Graf“ je v levém svislém panelu množství funkcí, které umožňují ruční, interaktivní editaci grafu - změnu velikosti uzlu, barvy, pozice, popisku, ... V „Laboratoři dat“ je zase kromě importu a exportu také možné ručně přidávat a odebírat uzly a hrany, nastavovat ručně hodnoty atributů a popisků.

2.1.2.5 Instalace pluginů

Jak už bylo několikrát zmíněno, *Gephi* poskytuje možnost snadného rozšíření funkcionality pomocí pluginů. Využívá přitom možností platformy *NetBeans*, která je od základu přizpůsobena modulární architektuře. *Gephi* plugin je *NetBeans* plugin[9] využívající API *Gephi*. *Gephi* při startu automaticky načte všechny pluginy a tím rozšíří svou funkcionalitu. Využívá k tomu *NetBeans* specifické řešení, tzv. *Lookup API*[10]. Přesný způsob implementace není pro tuto práci podstatný, stačí vědět, že pouhou implementací nějakého rozhraní lze rozšířit funkcionalitu *Gephi*.

[9] Jedná se o zip archive obsahující deskriptor modulu a jednu nebo více Java knihoven, které implementují vlastní funkcionalitu.

[10] API pro volnou vazbu mezi moduly (pluginy). Modul deklaruje veřejné rozhraní SPI (*Service Provider Interface*). Jiné pluginy pak mohou toto rozhraní implementovat a deklarovat jako *Service Provider* a tím rozšířit funkcionalitu. Implementace *Lookup API* poté dokáže tyto implementace nalézt i během runtime a použít.

2.2 Komponenty relevantní pro WebGephi

Gephi poskytuje opravdu rozsáhlé možnosti práce s grafy, od manuální editace až po aplikaci komplexních funkcí a práci s dynamickými grafy. Ne všechna funkcionalita je však vhodná pro *WebGephi*. Zde si určíme, jakou funkcionalitu je vhodné (a možné) implementovat ve *WebGephi*.

2.2.1 Cílový uživatel

Nejdříve si musíme shrnout, jak vlastně bude vypadat cílový uživatel, k jakému účelu bude *WebGephi* primárně používáno.

2.2.1.1 Student, vývojář aplikací

Jedním z typických uživatelů bude pravděpodobně student vysoké školy, který v rámci nějakého projektu (např. semestrální práce) bude mít za úkol analýzu grafu. Většinou to nebude jediný smysl aplikace, pouze jedna z jejích částí. Místo vlastní implementace pomocí *Gephi Toolkit* využije API *WebGephi*. Nebude se tedy muset zabývat strukturou knihovny ani způsobem implementace. Navíc není omezen cílovou platformou (např. není nucen psát aplikaci v jazyku Java).

Nemusí se samozřejmě jednat pouze o studenty. Stejně tak se může jednat o jakéhokoli vývojáře, který chce v rámci své aplikace využít možností *Gephi*. *WebGephi* ho odstíní od detailů implementace *Gephi Toolkit* a poskytne platformovou nezávislost.

2.2.1.2 Tvůrce pluginů

Dalším typickým uživatelem bude tvůrce algoritmů pro práci s grafy. *Gephi* poskytuje možnost snadného rozšíření pomocí pluginů. Pokud někdo takovýto plugin vytvoří, pomocí *WebGephi* (a její klientské aplikace) může vytvořit demo pro prezentaci jeho funkcionality. Případní uživatelé pluginu tak nebudou nuceni instalovat plugin jen pro jeho vyzkoušení.

2.2.2 Požadovaná funkcionalita WebGephi

Je tedy vidět, že pro *WebGephi* není nutné ani vhodné pokrýt celou funkcionalitu *Gephi*. *WebGephi* bude zaměřeno na práci s již existujícími grafy, není nutné řešit manuální editaci grafu. Stejně tak není nutné podporovat všechny formáty pro import a export. Naopak je nutné zachovat většinu grafových funkcí. Klíčovou vlastností *Gephi* je jeho rozšiřitelnost, je nezbytné aby *WebGephi* tuto rozšiřitelnost zachovalo.

2.2.2.1 Import a export

Existuje mnoho různých formátů vhodných pro uložení grafu, včetně proprietárních formátů různých aplikací. *Gephi* obsahuje podporu pro většinu z nich. Aplikace *WebGephi* bude podporovat import grafu ve stejném rozsahu jako *Gephi*. Základním formátem bude však formát `gexf`. Tento formát je ve svém oboru standard a z dostupných formátů má největší možnosti (viz srovnání B.1). Jedná se relativně jednoduchý formát založený na `xml`, je tedy snadno strojově zpracovatelný a díky definici ve formátu `xsd` je jednoznačný.

Export grafu bude dostupný pouze ve formátu *gexf*, který je nejkomplexnější a je schopen uchovat všechny potřebné informace.

Vizualizace grafu bude zodpovědností především klientských aplikací. Přesto je vhodné, aby i *WebGephi* obsahovalo základní export v grafickém formátu. Nejvhodnější z podporovaných formátů[11] je `svg`. Jedná se o vektorový formát založený na `xml`, který je pro tento účel vhodnější než rastrový formát `png`. A oproti `pdf` je možná jeho další editace. Formát `svg` je podporován většinou grafických editorů a všechny hlavní webové prohlížeče obsahují nativní podporu pro jeho vizualizaci.

2.2.2.2 Grafové funkce

Grafové funkce budou nejdůležitější částí aplikace. WebGephi se nebude zabývat tvorbou grafu ve většině případů ani jeho vizualizací. Typické workflow bude *načtení grafu - aplikace funkcí - export ve formátu gexf.*

Nejdůležitější bude podpora pro funkce „Rozložení“ a „Statistiky“, které jsou nejpoužívanější. Pro lepší vizualizaci výsledků je také potřeba podpora pro funkce „Hodnocení“. „Filtry“ jsou zase nezbytnou funkcionalitou pro lepší orientaci v rozsáhlých grafech. Podpora pro „Generování grafů“ a „Rozdělení na oddíly“ nemusí být součástí této práce, jedná se o méně využívané funkce. Struktura *WebGephi* však musí umožnit snadné doplnění této funkcionality v budoucnu.

2.3 Gephi Toolkit

Gephi je aplikace určená pro koncového uživatele. Kromě modulů obsahujících logiku pro práci s grafy obsahuje také moduly související s uživatelským rozhraním, nápovědou... Není tedy vhodné pro další použití v jiných aplikacích. Naštěstí *Gephi* pro tyto účely poskytuje knihovnu *Gephi Toolkit*. Tato knihovna obsahuje jádro aplikace ve formátu jedné Java knihovny.

„Gephi Toolkit obsahuje základní moduly (Graf, Rozložení, Filtry, IO...) ve formě standardní Java knihovny, kterou může jakýkoli projekt použít pro své potřeby. Toolkit je jeden JAR, který může kdokoli znovupoužít v jiné Java aplikaci a například z příkazové řádky dosáhnout automatizovaně toho samého

[11] `png`, `pdf` a `svg`

jako v Gephi. Možnost využít takto schopností Gephi v jiných Java aplikací rozšiřuje jeho možnosti a zdá se být velmi užitečné.“[12]

Gephi Toolkit je součástí *Gephi* a je distribuováno pod stejnou licencí - duální licencí CDDL 1.0 a GNU General Public License v3. Může tedy být použit bezplatně i v rámci komerčního software.

2.3.1 Struktura

Gephi je založeno na platformě *NetBeans* a tudíž využívá jeho techniky pro práci s modulárním software. Ta je založena na `Lookup API`. Veřejné API poskytuje přístup pouze k rozhraním a jeho implementace může být získána pomocí třídy `Lookup`.

```
1 ProjectController pc = Lookup.getDefault().lookup(
    ProjectController.class);
```

Ukázka zdroj. kódu 2.1: Ukázka použití Lookup API

Práce s Gephi je prováděna s pomocí kontrolerů. Pro každou oblast funkcionality (modul) existuje kontroler, což je singleton poskytující příslušné metody. Základní kontrolery jsou:

ProjectController
: Vytváření a správa projektů a workspace.

ImportController
: Import grafu ze souboru, databáze, ...

ExportController
: Export grafu do obrázku, v `gexf` formátu, `pdf`

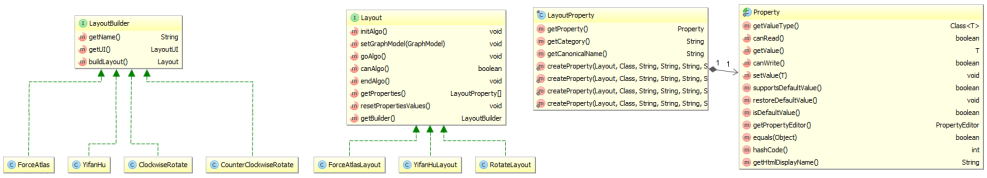
GraphController
: Manipulace s grafem, vytváření uzlů a hran.

AttributeController
: Přidávání, mazání a editace atributů u uzlů a hran.

LayoutController, StatisticsController, RankingController, ...
: Práce s příslušnými grafovými funkcemi.

Klíčovým požadavkem pro implementaci *WebGephi* bude možnost vylistování všech dostupných funkcí. To bohužel není možné udělat standardní

[12] Volný překlad z oficiálních stránek *Gephi Toolkit*[4]. Orig.: „The Gephi Toolkit project package essential modules (Graph, Layout, Filters, IO...) in a standard Java library, which any Java project can use for getting things done. The toolkit is just a single JAR that anyone could reuse in new Java applications and achieve tasks that can be done in Gephi automatically, from a command-line program for instance. The ability to use Gephi features like this in other Java applications boost possibilities and promise to be very useful.“



Obrázek 2.9: Struktura tříd pro výpočet funkcí „Rozložení“.

cestou, jak je to řešeno v *NetBeans* platformě[13]. *Gephi Toolkit* je už jen standardní knihovna bez deskriptorů, které jsou nezbytné pro *NetBeans* moduly.

Nejdříve analyzujeme strukturu potřebných grafových funkcí a poté popíšeme, zda a jak bude možné řešit nutnost výčtu dostupných funkcí.

2.3.1.1 Rozložení

Základní třídou pro výpočet „Rozložení“ je třída `LayoutBuilder` (strukturu tříd je znázorněna na obr. 2.9). Ta poskytuje přístup ke jménu funkce, grafické komponentě pro zadání parametrů a třídě `Layout`, která již provádí samotný výpočet nad grafem. Parametry rozložení jsou přístupné ve formě pole objektů `LayoutProperty`, což umožňuje jejich snadné vylistování a nastavení. Objekt `LayoutProperty` obaluje instanční proměnné a přistupuje k nim promocí reflexe[14].

```
YifanHu yifanHu = new YifanHu(); // Factory
YifanHuLayout layout = yifanHu.buildLayout(); // Layout function
layout.setGraphModel(graphModel); // Set the graph to work with
layout.initAlgo();
layout.resetPropertiesValues(); // Set default values
layout.getProperties()[0].getProperty().setValue(200f); // Change
    first param (OptimalDistance)
for (int i = 0; i < 100 && layout.canAlgo(); i++) {
    layout.goAlgo(); // Apply layout function
}
layout.endAlgo();
```

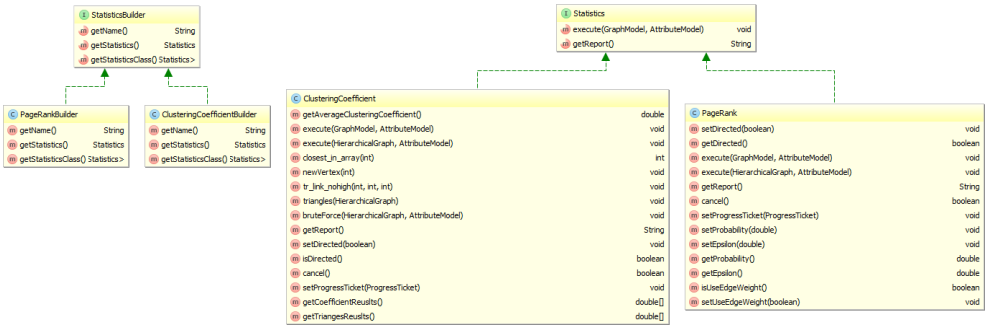
Ukázka zdroj. kódu 2.2: Aplikace funkce rozložení `YifanHu`

2.3.1.2 Statistiku a metriky

Struktura tříd (viz obr. 2.10) je obdobná jako u „Rozložení“. V tomto případě ale třída `Statistics` neobsahuje žádný seznam možných parametrů. Ty jsou nastavovány přímo z uživatelského rozhraní (implementace rozhraní `StatisticsUI`) pomocí getterů a settrů. Seznam parametrů bude tedy nutné získat na základě metod třídy za běhu programu pomocí reflexe.

[13] Pomocí `Service Provider Interface` a `Service Provider`, viz 2.1.2.5

[14] Možnost za běhu programu získat od objektu kompletní informace o jeho typu a dále s nimi pracovat.



Obrázek 2.10: Struktura tříd pro výpočet „Statistik a metrik“.

Výstupem „Statistik“ je kromě pozměněného grafu také html zpráva obsahující souhrnné informace o rozložení metrik. Ta je dostupná po aplikaci metriky pomocí metody `Statistics::getReport()`.

```
// Create statistics builder
StatisticsBuilder builder = new GraphDistanceBuilder();
// Get statistics function implementation
GraphDistance gd = (GraphDistance) builder.getStatistics();
gd.setDirected(true); // Set parameter directly using setter
gd.execute(graphModel, attributeModel); // apply function
String report = gd.getReport(); // Get html report

// We can iterate over metric values of nodes too
AttributeColumn col = attributeModel.getNodeTable().getColumn(
    GraphDistance.BETWEENNESS);
for (Node n : graphModel.getGraph().getNodes()) {
    Double centrality = (Double) n.getNodeData().getAttributes().
        getValue(col.getIndex());
}
```

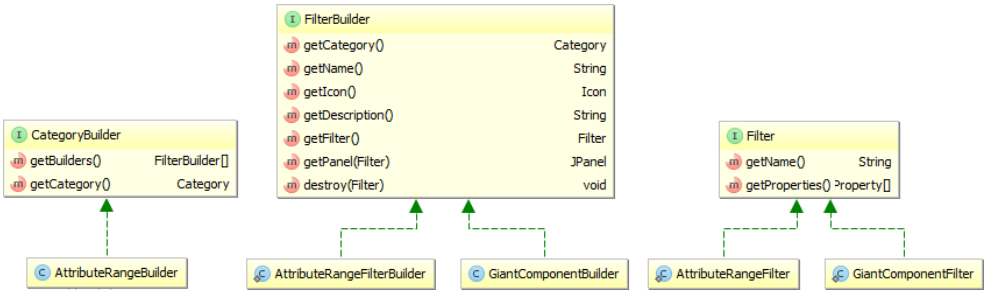
Ukázka zdroj. kódu 2.3: Aplikace statistické funkce `GraphDistance`

2.3.1.3 Filtry

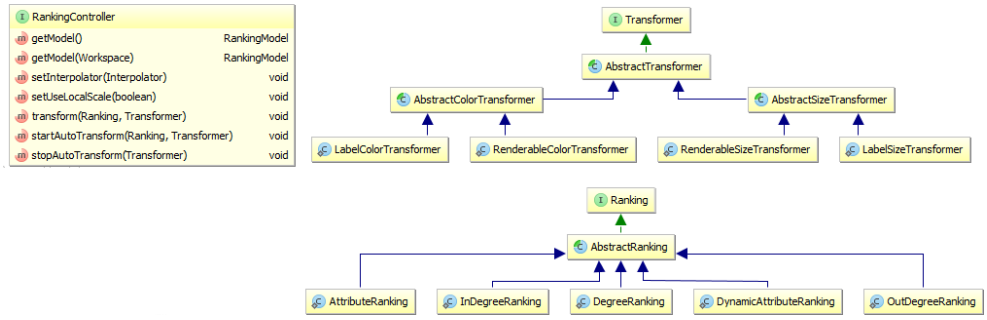
Základladní třídou pro aplikaci filtrů je třída `Filter` a k jejímu vytvoření slouží třída `FilterBuilder` (viz obr. 2.11). Avšak narozdíl od „Statistik“ a „Rozložení“, některé buildery jsou vytvářeny v závislosti na obsahu grafu za pomoci třídy `CategoryBuilder`. Například `AttributeRangeBuilder` (category builder) vytvoří příslušný filter builder pro každý číselný atribut.

`Filter` obsahuje seznam parametrů, s jehož pomocí lze jeho konfiguraci snadno vylistovat na nastavit. Z filtru se následně vytvoří objekt `Query`, který se aplikuje na graf (viz ukázka kódu 2.4).

```
// Create builder and filter
GiantComponentBuilder builder = new GiantComponentBuilder();
Filter filter = builder.getFilter();
```



Obrázek 2.11: Struktura tříd pro aplikaci „Filtrů“.



Obrázek 2.12: Struktura tříd pro výpočet „Hodnocení“.

```
// Create query from filter
Query query = filterController.createQuery(filter);
// Apply filter
FilterProcessor processor = new FilterProcessor();
Graph result = processor.process((AbstractQueryImpl) query,
    graphModel);
// Set result view as visible
graphModel.setVisibleView(result.getView());
```

Ukázka zdroj. kódu 2.4: Aplikace filtru `GiantComponent`

2.3.1.4 Hodnocení

K aplikaci hodnocení slouží třída `RankingController` a její metoda `transform(Ranking, Transformer)`. Stačí jen specifikovat podle čeho hodnotit a co bude výstupem. K tomu slouží implementace rozhraní `Ranking` (na základě čeho hodnotit) a `Transformer`. Jejich implementace můžete vidět na obr. 2.12. Hodnotit lze na základě jakéhokoli atributu uzlu nebo hrany (případně podle stupně uzlu). Výstupem hodnocení může být změna barvy (uzlu, hrany, popisku) nebo velikosti (uzlu, hrany).

```
// Column with centrality attribute, which we wat to use for
    ranking
```

```
AttributeColumn centralityColumn = attributeModel.getNodeTable().
    getColumn(GraphDistance.BETWEENNESS);
// Create ranking based on centrality attribute
Ranking centralityRanking = rankingController.getModel().
    getRanking(Ranking.NODE_ELEMENT, centralityColumn.getId());
// Create transformer - defines what we want to change (node size)
AbstractSizeTransformer sizeTransformer = (AbstractSizeTransformer
    ) rankingModel.getTransformer(Ranking.NODE_ELEMENT,
    Transformer.RENDERABLE_SIZE);
// Set transformer parameters
sizeTransformer.setMinSize(3);
sizeTransformer.setMaxSize(20);
// Apply ranking, node size is updated
rankingController.transform(centralityRanking, sizeTransformer);
```

Ukázka zdroj. kódu 2.5: Změna velikosti uzlu podle hodnoty atributu „betweenesscentrality“

2.3.2 Seznam dostupných funkcí

Jak už jsme zmínili, klíčovým požadavkem pro *WebGephi* je schopnost najít všechny dostupné grafové funkce, tzn. všechny implementace rozhraní `LayoutBuilder` a `StatisticsBuilder`. Tento list nemůže být statický. Musíme totiž zároveň zachovat rozšiřitelnost, pokud k aplikaci přidáme plugin obsahující např. funkci pro výpočet „Rozložení“, aplikace ho musí sama přidat do svého (REST) rozhraní.

Jelikož nemůžeme využít možností *NetBeans* platformy, musíme využít možností samotného jazyku Java. Pomocí reflexe nalezneme všechny implementace daných rozhraní (`LayoutBuilder` a `StatisticsBuilder`) a s jejich pomocí už máme dostupnou potřebnou funkcionalitu. Takto nalezneme všechny implementace dostupné na classpath aplikace, je tím tedy zajištěna požadovaná rozšiřitelnost. Z bezpečnostních důvodů nebude možné přidávat pluginy za běhu aplikace, proces nalezení všech dostupných funkcí tedy bude stačit provést pouze jednou po startu aplikace.

V případě „Hodnocení“ se nepočítá s dalším rozšiřováním, tudíž není třeba zjišťovat dostupné možnosti dynamicky. Bude stačit implementovat několik základních možností.

2.3.3 Víceuživatelský přístup

Architektura *Gephi* je od počátku navrhována pro potřeby desktopové aplikace. Je z velké části založena na návrhovém vzoru Singleton[15]. Např. všechny kontrolery jsou singletony. To není v zásadě problém, protože kontrolery nejsou

[15] *„Singleton (česky jedináček nebo také unikát) je název pro návrhový vzor, používaný při programování. Využijeme ho při řešení problému, kdy je potřeba, aby v celém programu běžela pouze jedna instance třídy. Tento návrhový vzor zabezpečí, že třída bude mít jedinou instanci a poskytne k ní globální přístupový bod.“*[7]

stavové. Problémy nastávají v momentě, kdy chceme pracovat s více Workspace najednou (ve vícevláknovém prostředí). Tento problém je ostatně zmíněn (ne příliš výrazně) i na stránkách s příklady použití *Gephi Toolkit*.

„(. . .) many modules just do their job on this current workspace and doesn't allow to specify which workspace to work on from the API. Therefore it is often required to set current workspace, as showed in export here.“[4]

Architektura *Gephi* je totiž navržena tak, že vždy právě jeden Projekt a Workspace je aktivní. A přestože některé moduly (třídy, metody) se tváří tak, že umí pracovat s Workspace předaným v parametru, v těle metody se používá „current Workspace“. Tento problém ilustruji na příkladu 2.6 třídy `DefaultProcessor`, která se využívá při importu grafu.

```
// Set current workspace to 'currentWorkspace'
projectController.openWorkspace(currentWorkspace);
DefaultProcessor defaultProcessor = new DefaultProcessor();
// Set processor to use different workspace than current
defaultProcessor.setWorkspace(notCurrentWorkspace);
// Import graph to Workspace 'currentWorkspace'
importController.process(container, new DefaultProcessor(),
    currentWorkspace);
```

Ukázka zdroj. kódu 2.6: Import grafu do Workspace který není nastavený jako „currentWorkspace“

Vše vypadá na první pohled v pořádku, graf se zdánlivě načte do Workspace `notCurrentWorkspace`. Pokud se však podíváme do implementace třídy `DefaultProcessor` (viz ukázka kódu 2.7) zjistíme, že se ze skutečnosti používá Workspace nastavený v Singletonu `ProjectController` jako „currentWorkspace“.

```
@ServiceProvider(service = Processor.class, position = 10)
class DefaultProcessor extends AbstractProcessor implements
    Processor {
    // ...
    // Method called during import
    public void process() {
        // ...
        // Get GraphModel from current workspace!!!
        GraphModel graphModel = Lookup.getDefault().lookup(
            GraphController.class).getModel();
        // It should be:
        // Lookup.getDefault().lookup(GraphController.class).
            getModel(workspace)
        // ...
    }
// ...
}
```

Ukázka zdroj. kódu 2.7: Výřez z implementace třídy `DefaultProcessor`

Tento problém lze vyřešit (jak ostatně zní doporučení v dokumentaci *Gephi Toolkit*) tím, že vždy nastavíme jeden workspace jako „current“ a pracujeme

pouze s ním. To může fungovat u desktopové aplikace, v serverovém prostředí se však pohybujeme ve vícevláknovém prostředí. Museli bychom prakticky veškerou práci s *Gephi Toolkit* provádět serializovaně, (v `synchronized` bloku nebo nějakou pokročilejší technikou zajišťující, že k danému úseku kódu přistupuje vždy jen jedno vlákno). To by samozřejmě způsobovalo úzké hrdlo aplikace a výrazně degradovalo výkon. Prakticky by mohl být v jednu chvíli zpracováván vždy jen jeden požadavek. V kombinaci s tím, že operace nad grafy mohou trvat relativně dlouho, by se aplikace stala naprosto nepoužitelnou už pro relativně malý počet uživatelů.

Druhé možné řešení je odstranění těchto chyb tím, že upravíme původní třídy tak, abychom místo s „currentWorkspace“ pracovali s konkrétním Workspace předaným komponentě. Jedná se o relativně malé množství komponent, tudíž bude toto řešení vhodnější. Tyto třídy budou součástí *WebGephi*, nebude se jednat o úpravy samotné knihovny.

2.4 Možnosti autentizace a autorizace

WebGephi bude multiuživatelská aplikace. Kdokoli se bude moci založit svůj účet, nahrávat a upravovat své grafy. Z toho přirozeně plyne nutnost autentizace (ověření identity uživatele) a autorizace (určení, zda má uživatel práva pro danou operaci).

WebGephi však bude poskytovat pouze REST rozhraní, které málokdo bude využívat přímo. Místo toho autorizuje nějakou klientskou aplikaci, aby mohla přistupovat k jeho účtu. Tato aplikace pak skrze vhodné grafické rozhraní umožní uživateli pracovat s jeho grafy.

Zde si představíme standardizovaná řešení této problematiky.

2.4.1 Basic autentizace[8]

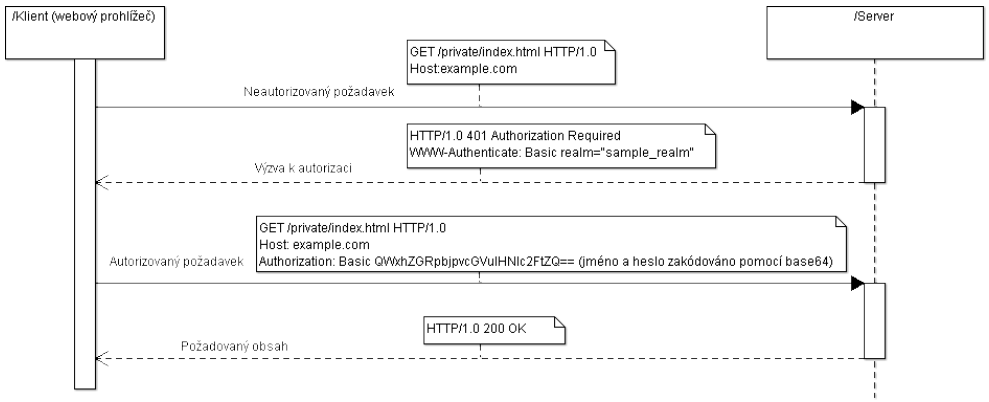
Basic autentizace je jednou z nejstarších metod autentizace. Princip, jak už název napovídá, je velmi jednoduchý. Klient (většinou webový prohlížeč) zašle v hlavičce HTTP požadavku své jméno a heslo[16]. Server ověří správnost údajů a na jejich základě poskytne nebo odmítne poskytnout požadovaný obsah.

Pokud požadované přihlašovací údaje chybí, vrátí server odpověď s požadavkem na Basic autentizaci.

Výhodou této metody je její jednoduchost. Snad každý HTTP klient obsahuje podporu pro Basic autentizaci. Všechny rozšířené webové prohlížeče si navíc pamatují zadané přihlašovací údaje a není tedy nutné je zadávat vždy znovu.

Nevýhodou je, že je nutné zasílat přímo přihlašovací údaje, navíc v nezašifrované podobě a tudíž kdokoli může údaje po cestě odposlechnout. Při použití HTTPS protokolu se však jedná o bezpečnou metodu autentizace.

[16] ve formě `uživatelskéJméno:heslo`, zakódováno pomocí base64



Obrázek 2.13: Průběh Basic autorizace.

2.4.1.1 Použití pro WebGephi

Jedná se o vhodnou metodu pro přímé procházení a prozkoumávání REST rozhraní. Není však vhodné pro poskytování přístupu aplikacím třetích stran (klientské aplikace). V tomto případě by bylo nutné předat klientské aplikaci přímo své uživatelské jméno a heslo. Tím by uživatel poskytl plný přístup ke svému účtu a nevěrohodná klientská aplikace by toho mohla zneužít (v krajním případě změnit heslo a tím „ukrást“ uživatelův účet). Uživatel by navíc nemohl odebrat aplikaci již jednou udělené oprávnění. Pro tento účel by jedině musel změnit své heslo.

Tento způsob autentizace je tedy vhodný jedině pro přímé prozkoumávání REST rozhraní aplikace. Mohl by být použit také v případě, kdy klientská aplikace využívá jen svůj vlastní účet (může případně řešit správu uživatelů na své straně).

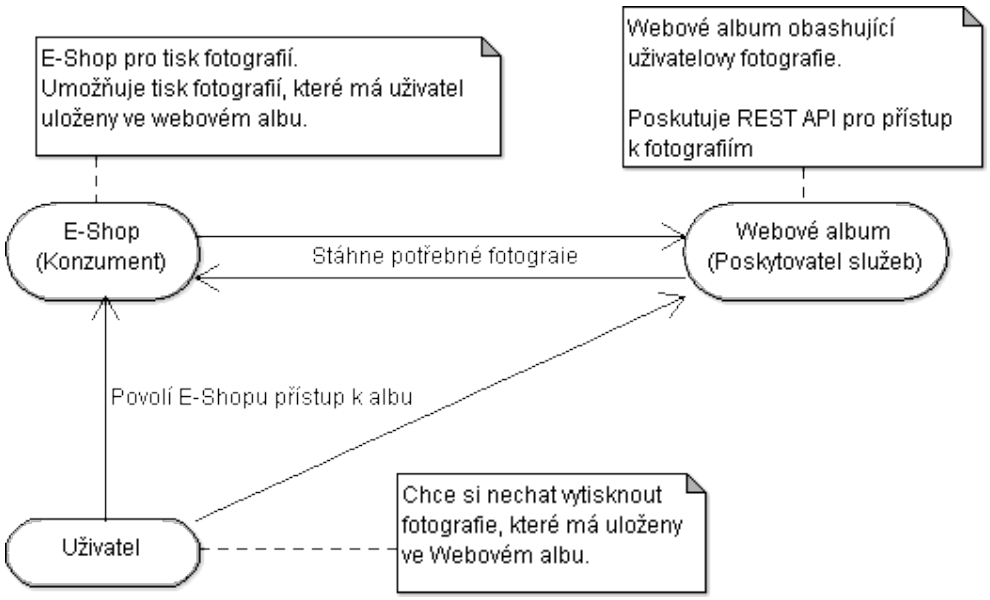
2.4.2 Digest autentizace[9]

Jedná se o mírnou evoluci Basic autentizace. Princip je úplně stejný, v tomto případě se však neposílá heslo přímo, ale jeho hash[17]. Z hlediska bezpečnosti se jedná o mírné zlepšení. Při odposlechnutí požadavku nelze (pokud je autentizace správně implementována) požadavek zopakovat. Nevýhodou je složitější implementace jak na straně serveru, tak na straně klienta.

2.4.2.1 Použití pro WebGephi

Tato metoda, za předpokladu že použijeme HTTPS, nepřináší prakticky žádné výhody oproti Basic autorizaci. Naopak je složitější na implementaci.

[17] Používá se `MD5` hash kombinace hesla, cílové uri, metody (GET, POST, ...), nonce (serverem poskytnutý náhodný řetězec, cnonce (klientův náhodný řetězec) a pořadí požadavku. Více viz [9])



Obrázek 2.14: Ukázkový příklad využití OAuth protokolu.

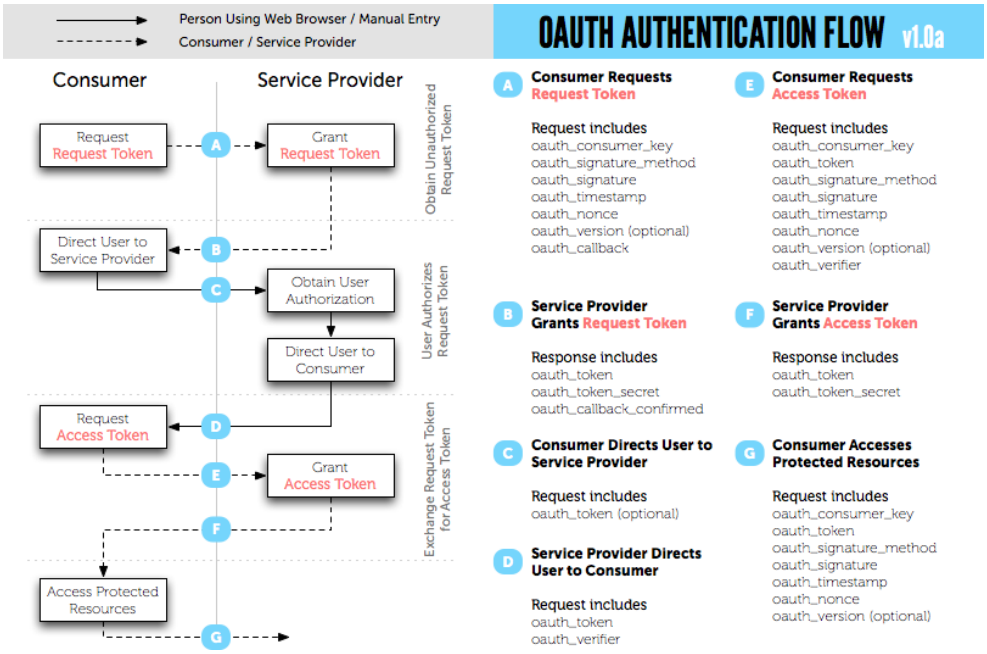
2.4.3 OAuth v1.0a

„Protokol OAuth umožňuje webovým stránkám nebo aplikacím (Konzumentům) přistupovat k chráněným zdrojům webových služeb (Poskytovatele služeb) skrz API bez nutnosti, aby uživatel prozradil své přihlašovací údaje k Poskytovateli služeb Konzumentům. Obecněji, OAuth vytváří volně implementovatelnou a obecnou metodiku pro autentizaci k API“ [18]

OAuth je standardizovaný způsob, jak uživatel může poskytnout přístup ke svému účtu (chráněným zdrojům) u nějaké aplikace (*Poskytovatel služeb*) jiné aplikaci (*Konzument*). A to bez toho, aby této aplikaci (*Konzumentovi*) musel prozradit své přihlašovací údaje. Ukázkový příklad pro využití OAuth protokolu je znázorněn na obr. 2.14.

Konzument musí být zaregistrován u *Poskytovatele služeb* a má svůj jednoznačný identifikátor (`consumer key`) a heslo (`consumer secret`). Pokud požaduje přístup k uživatelovým datům u *Poskytovatele služeb*, zažádá nejdříve o `Request token`. Ten specifikuje, k jakým zdrojům požaduje aplikace přístup. Poté přesměruje uživatele na stránky *Poskytovatele služeb*. Uživatel se zde přihlásí a autorizuje požadavek *Konzumenta*, následně je přesměrován zpět na stránky *Konzumenta*. Odpověď obsahuje `verifier`, který *Konzument* pou-

[18] Volný překlad. Orig.: *„The OAuth protocol enables websites or applications (Consumers) to access Protected Resources from a web service (Service Provider) via an API, without requiring Users to disclose their Service Provider credentials to the Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication.“*[10]



Obrázek 2.15: Průběh OAuth autorizace. Převzato z [10]

žije k výměně `Request tokenu` za `Access token`. `Access token` už může být použit přímo k přístupu k požadovaným zdrojům. Podrobný popis průběhu OAuth autorizace je znázorněn na obr. 2.15.

Autentizace a autorizace pomocí OAuth v1.0a protokolu je velmi bezpečná. Veškerá komunikace s API *Poskytovatele služeb* je podepisována pomocí informací z `Access tokenu`. Navíc všechny citlivé informace jsou při přenosu šifrovány. To umožňuje jeho bezpečné použití i bez použití SSL. Na druhou stranu je kvůli tomu OAuth náročný na implementaci jak na straně serveru, tak na straně klienta. Při použití HTTPS je navíc šifrování v podstatě duplikátní. Na druhou stranu díky přesné specifikaci existují hotové knihovny pro jeho použití.

Hlavní výhodou OAuth protokolu je, že nemusíme aplikacím třetích stran vyzradit své přihlašovací údaje jen kvůli tomu, abychom jim poskytli přístup k části svých chráněných zdrojů. Toto povolení můžeme navíc kdykoliv odebrat. Protokol také umožňuje poskytnout přístup pouze k části zdrojů (tzv. `scopes`).

2.4.3.1 Použití pro WebGephi

OAuth je ideální protokol pro *WebGephi*. Jeho součástí je částečně i způsob registrace klientských aplikací. Umožňuje uživatelům udělovat oprávnění klientským aplikacím (*Konzumentům*), aby mohli přistupovat k jejich účtům na *WebGephi* (*Poskytovatel služeb*). A to bez toho, aby museli prozradit své při-

hlašovací údaje. Navíc nemusí udělovat plný přístup ke svému účtu, některé aplikace mohou např. požadovat pouze práva pro čtení. Typicky žádná aplikace nebude mít práva pro úpravu uživatelova profilu (jména, hesla).

Jedinou nevýhodou je složitější implementace protokolu na straně serveru i klienta.

2.4.4 OAuth v2.0[11]

Jedná se o novou verzi, která vyšla v říjnu 2012 (RFC 6749). Již dlouho předtím však byla ve stadiu návrhu. Její cílem bylo zjednodušení protokolu a větší použitelnost pro jiné než webové aplikace (desktopové, mobilní, ...). Hlavní změnou je, že komunikace již není šifrována, plně se spoléhá na SSL šifrování (nutnost použití společně s HTTPS). Zavádí také více různých způsobů autorizace (kromě klasického „three-legged“ OAuth), které jsou vhodné např. pro mobilní aplikace.

Jedná se o volnější specifikaci než je OAuth 1.0a. Jednotlivé implementace se výrazně liší a většinou nejsou spolu kompatibilní.

2.4.4.1 Použití pro WebGephi

OAuth v2.0 nepřináší oproti první verzi z pohledu *WebGephi* žádné výrazné výhody. Volnější specifikace a nekompatibilita jednotlivých implementací ztěžuje jeho použití různými *Konzumenty*. Výhodou by byla lepší podpora desktopových a mobilních klientů.

2.4.5 Výběr autentizace a autorizace pro WebGephi

Pro *WebGephi* byla vybrána kombinace Basic autorizace a OAuth v1.0a. Preferována bude OAuth autorizace, která umožňuje bezpečnější a přesnější řízení přístupu. Basic autorizace se bude používat pouze pro prozkoumávání a vyzkoušení REST rozhraní a v případech, kdy klientská aplikace není (kvůli složitosti) schopná implememtovat OAuth klienta. To se může stát např. v případě čistě JavaScriptového klienta.

Kapitola 3

Návrh

V této kapitole se zaměříme na návrh struktury aplikace *WebGephi*. Představíme si základní workflow aplikace a její datový model. Nejdůležitější částí kapitoly bude popis REST API. Protože výsledná aplikace nebude monolitická, popíšeme si zde také význam jednotlivých modulů.

3.1 Procesní model

Na diagramu 3.1 je znázorněno, jak bude řešeno základní workflow ve *WebGephi*. Základem je úložiště uživatelových grafů. Na tyto grafy může klient aplikovat dostupné grafové funkce (`Layout`, `Statistics`, `Ranking`). Po každé aplikaci funkce se výsledek uloží jako nový graf a uživateli se vrátí odkaz na něj. To mimo jiné umožní zobrazenit historii práce s každým grafem.

Vytvořené grafy nebude možné mazat ani upravovat. To umožní snadnou kešovatelnost úložiště.

3.2 Datový model

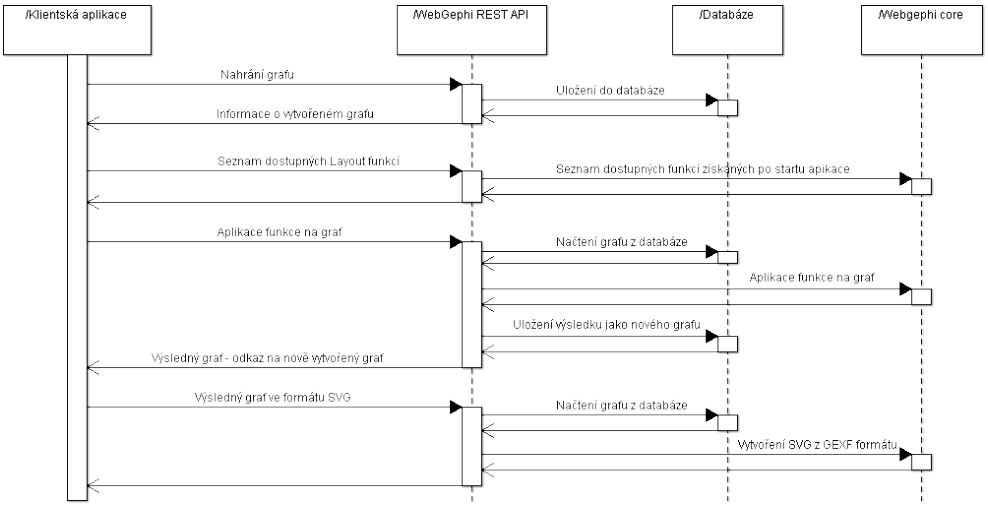
Datový model aplikace bude velmi jednoduchý (viz diagram 3.2). Zde si popíšeme jeho jednotlivé prvky.

3.2.1 User

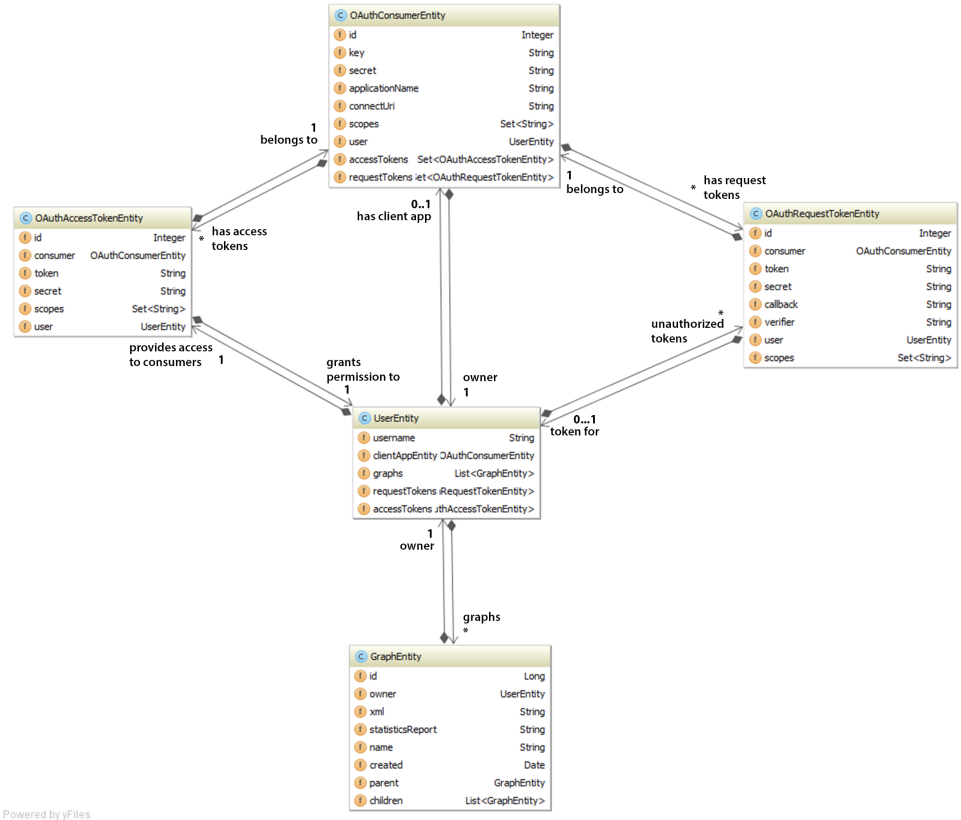
Entita `User` představuje běžného uživatele aplikace (případně admina, struktura rolí není v modelu znázorněna). Každý uživatel má přístup ke svým grafům a může spravovat jednu klientskou aplikaci.

3.2.2 Graph

Entita graph představuje jeden graph, jeho struktura je uložena ve formátu `gexf`. Pokud byla na graf aplikována statistická funkce, v atributu `statistics-`



Obrázek 3.1: Sekvenční diagram práce s WebGephi



Obrázek 3.2: Datový model WebGephi

`Report` je uložena shrnující zpráva (formát `html`). Kromě toho má každý graf své jméno a informaci o datu vytvoření.

3.2.3 OAuth

Pro potřeby OAuth autorizace musí datový model také obsahovat potřebné informace. Jedná se o entity `OAuthConsumer`, `OAuthAccessToken` a `OAuthRequestToken`.

3.2.3.1 OAuthConsumer

Tato entita představuje klientskou aplikaci (v OAuth termnologii se jedná o *Konzumenta*). Pro její vytvoření a správu musí být přiřazena k uživateli, který je pak její správce (vlastník). *Konzument* pak může od uživatelů požadovat přístup k jejich účtu.

3.2.3.2 OAuthRequestToken

Jedná se o dočasnou entitu potřebnou pro zažádání přístupu k cizímu účtu. Po autorizaci tokenu uživatele je `Request token` smazán a nahrazen `Access tokenem`.

3.2.3.3 OAuthAccessToken

Představuje udělené oprávnění pro *Konzumenta* přistupovat k účtu jiných uživatelů. Kromě identifikátoru (atribut `token`) a tajného klíče (atribut `secret`) obsahuje také seznam oprávnění, které token poskytuje (atribut `scopes`).

Uživatel může kdykoli smazat `Access token` a tím zrušit uděléné oprávnění.

3.3 REST rozhraní

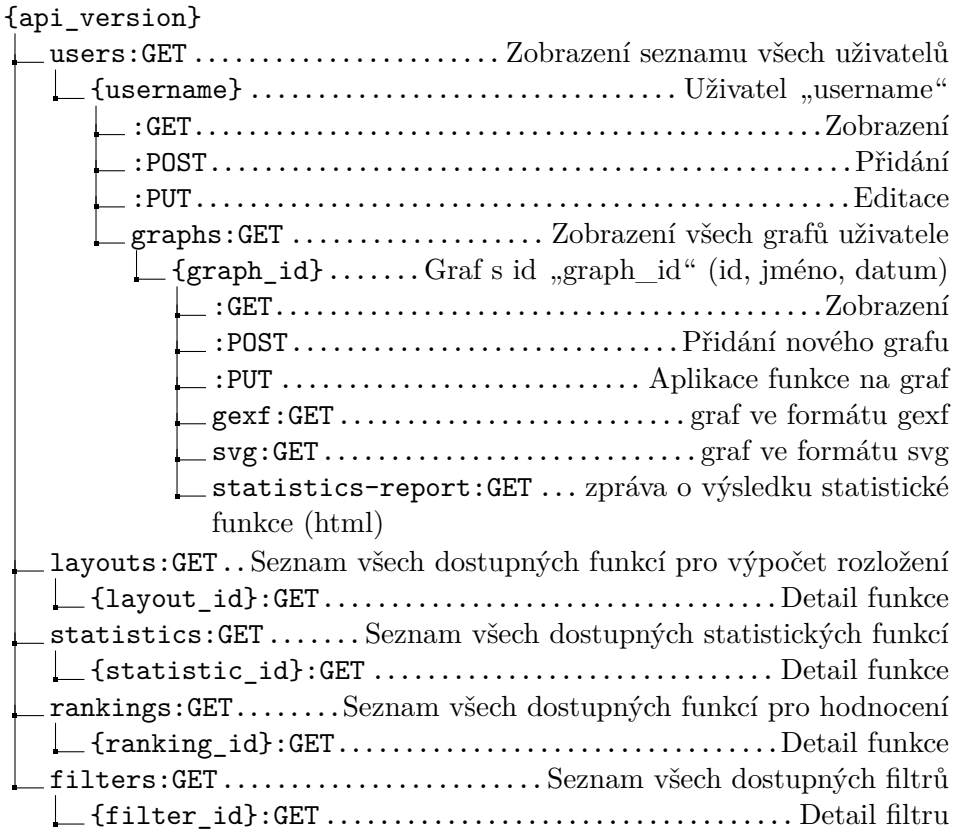
Rozhraní aplikace *WebGephi* bude vytvořeno jako RESTful[19] webová služba. REST architektura je založena na *zdrojích (resources)*. Každá URI představuje zdroj a webová služba poskutuje standardní metody pro práci s nimi. V případě použití protokolu HTTP (naprostá většina RESTful služeb) se používají přímo metody tohoto protokolu.

3.3.1 Metody RESTful webových služeb

GET Zobrazení konkrétní rerezentace zdroje

POST Vytvoření nového objektu - zdroje. Data jsou odesílána v tělě požadavku.

[19] Webová služba splňující principy REST architektury



Obrázek 3.3:

PUT Úprava existujícího zdroje na základě dat v těle požadavku.

DELETE Smazání existujícího zdroje.

HEAD Popis formátu zdroje.

3.3.2 Struktura WebGephi REST rozhraní

Zde podrobně popíšeme struktury RESTful webové služby *WebGephi*. Základní struktura je znázorněna diagramu 3.3

U každého zdroje je popsán popis možných parametrů a obsah těla požadavku a odpovědi. Pokud během zpracování požadavku dojede k nějaké nestandardní situaci (data požadavku jsou nevalidní, interní chyba systému, ...), místo standarní odpovědi s kódem 200 (OK) je vrácena odpověď s příslušným HTTP kódem a popisem problému (viz 3.1).

```
<error>
    <code number="409">Conflict</code>
    <message>User is invalid</message>
    <detail>Username test is already taken.</detail>
```

```
</error>

<html>
<head>
    <title>Error 404 (Not Found)</title>
</head>
<body>
    <h1>404 (Not Found)</h1>
    <ul>
        <li>Message: This graph has no statistics</li>
        <li>Detail: Statistics are set only if some statistic
            function was applied previously</li>
    </ul>
</body>
</html>
```

Ukázka zdroj. kódu 3.1: XML a HTML chybová odpověď

3.3.2.1 Seznam dostupných funkcí pro výpočet rozložení

URI [20] /layouts

Metoda GET

Potřebné oprávnění [21] Žádné

Tělo požadku Prázdné

Tělo odpovědi Seznam „Layout“ funkcí dostupných v systému (ukázka 3.2).

```
<layouts>
   <layout id="clockwise-rotate">
      ...
   </layouts>
   ...
</layouts>
```

Ukázka zdroj. kódu 3.2: Tělo odpovědi zdroje /layouts (GET)

3.3.2.2 Detail funkce pro výpočet rozložení

URI /layouts/{layout_id}

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

[20] Všechny URI jsou relativní k {server_url}/rest/v1/

[21] viz sekce 3.4

Tělo odpovědi Popis konkrétní fukce (jméno, parametry, ...) (ukázka 3.3).

```
<layout id="clockwise-rotate">
    <name>Clockwise Rotate</name>
    <properties>
        <property id="clockwise.angle.name">
            <name>Angle</wg:name>
            <description>Clockwise rotation angle in degrees</
                wg:description>
            <value>
                <double value="90.0"/>
            </value>
        </property>
    </properties>
</layout>
```

Ukázka zdroj. kódu 3.3: Tělo odpovědi zdroje /layouts/{layout_id} (GET)

3.3.2.3 Seznam dostupných statistických funkcí

URI /statistics

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Seznam statistických funkcí dostupných v systému (ukázka 3.4).

```
<statistics>
   <statistic id="clustering-coefficient">
      ...
   </statistic>
   ...
</statistics>
```

Ukázka zdroj. kódu 3.4: Tělo odpovědi zdroje /statistics (GET)

3.3.2.4 Detail statistické funkce

URI /statistics/{statistic_id}

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Popis konkrétní fukce (jméno, parametry, ...) (ukázka 3.5).

```
<statistic id="clustering-coefficient">
   <name>Clustering Coefficient</name>
   <properties>
      <property id="Directed">
            <name>Directed</wg:name>
            <value>
                <boolean value="false"/>
            </value>
        </property>
   </properties>
</statistic>
```

Ukázka zdroj. kódu 3.5: Tělo odpovědi zdroje /statistics/{statistic_id} (GET)

3.3.2.5 Seznam dostupných filtrů

URI /filters

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Seznam filtrů dostupných v systému (ukázka 3.6).

```
<filters>
   <filter id="attribute-non-null-filter">
      ...
   </filter>
   ...
</filters>
```

Ukázka zdroj. kódu 3.6: Tělo odpovědi zdroje /filters (GET)

3.3.2.6 Detail filtru

URI /filters/{filter_id}

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Popis konkrétního filtru (jméno, parametry, ...) (ukázka 3.7).

```
<filter id="attribute-non-null-filter">
    <description>
        Category: Non-null-Attributes; Keep nodes/edges with non-
            null values for a particular column
    </description>
    <name>Attribute Non Null Filter</name>
    <properties>
        <property id="attribute">
            <name>Attribute column</name>
            <description>Select attribute to filter</description>
            <value>
                <attribute attributeId="column"/>
            </value>
        </property>
    </properties>
</filter>
```

Ukázka zdroj. kódu 3.7: Tělo odpovědi zdroje /fiters/{filter_id} (GET)

3.3.2.7 Seznam dostupných funkcí pro hodnocení

URI /rankings

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Seznam funkcí pro hodnocení (ukázka 3.8).

```
<rankings>
   <ranking id="color-ranking">
       ...
   </ranking>
    ...
</rankings>
```

Ukázka zdroj. kódu 3.8: Tělo odpovědi zdroje /rankings (GET)

3.3.2.8 Detail funkce hodnocení

URI /rankings/{ranking_id}

Metoda GET

Potřebné oprávnění Žádné

Tělo požadku Prázdné

Tělo odpovědi Popis konkrétní fukce (jméno, parametry, ...) (ukázka 3.9).

```
<ranking id="edgeColor">
    <description>
        Ranking according to edge attribute, changes edge color.
    </description>
    <name>Edge color ranking</name>
    <properties>
        <property id="edgeAttribute">
            <name>Edge attribute</name>
            <description>
                Id of attribute which will be used for ranking. It
                    has to be one of already calculated edge
                    attributes (see GEXF format)
            </description>
            <value>
                <edgeAttribute attributeId="id-of-attribute-column
                    "/>
            </value>
        </property>
        <property id="startColor">
            <name>Start color</name>
            <description>
                Color of node with lowest value. In hexadecimal
                    format: RRGGBB, e.g. FEF0D9
            </description>
            <value>
                <color>FEF0D9</color>
            </value>
        </property>
        <property id="endColor">
            <name>End color</name>
            <description>
                Color of node with highest value. In hexadecimal
                    format: RRGGBB, e.g. B30000
            </description>
            <value>
                <color>B30000</color>
            </value>
        </property>
    </properties>
</ranking>
```

Ukázka zdroj. kódu 3.9: Tělo odpovědi zdroje /rankings/{ranking_id} (GET)

3.3.2.9 Seznam uživatelů

Zobrazí seznam všech uživatelů. Tento resource je dostupný pouze pro správce.

URI /users

Metoda GET

Potřebné oprávnění ADMIN

Tělo požadku Prázdné

Tělo odpovědi Seznam všech uživatelů v systému (ukázka 3.10).

```
<users>
    <user username="admin">
        ...
    </user>
    ...
</users>
```

Ukázka zdroj. kódu 3.10: Tělo odpovědi zdroje /users (GET)

3.3.2.10 Přidání uživatele do systému

Přidá nového uživatele do aplikace. Vytvořený uživatel bude mít roli USER (bežný uživatel)

URI /users

Metoda POST

Potřebné oprávnění Žádné

Tělo požadku Údaje o uživateli (ukázka 3.11)

Tělo odpovědi Nově vytvořený uživatel (ukázka 3.12).

```
<user username="test">
    <email>test@test.cz</email>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
    <password>password</password>
</user>
```

Ukázka zdroj. kódu 3.11: Tělo požadavku zdroje /users (POST)

```
<user username="test">
    <email>test@test.cz</email>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</user>
```

Ukázka zdroj. kódu 3.12: Tělo odpovědi zdroje /users (POST)

3.3.2.11 Detail uživatele

Zobrazí profil uživatele. Heslo se nikdy nezobrazuje.

URI /users/{username}

Metoda GET

Potřebné oprávnění PROFILE_READ

Tělo požadku Prázdné

Tělo odpovědi Profil uživatele `username` (ukázka 3.13).

```
<user username="test">
    <email>test@test.cz</email>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
</user>
```

Ukázka zdroj. kódu 3.13: Tělo odpovědi zdroje /users/{username} (GET)

3.3.2.12 Přihlášený uživatel

Zobrazí profil právě přihlášeného uživatele. Slouží klientským aplikacím, aby mohli zjistit, s jakým účtem právě pracují.

URI /users/logged

Metoda GET

Potřebné oprávnění PROFILE_READ

Tělo požadku Prázdné

Tělo odpovědi Profil právě přihlášeného uživatele.

3.3.2.13 Editace uživatele

Upraví profil uživatele podle zadaných informací.

URI /users/{username}

Metoda PUT

Potřebné oprávnění PROFILE_WRITE

Tělo požadku Údaje ke změně 3.14)

Tělo odpovědi Uživatel po editaci (ukázka 3.15).

```
<user username="test">
    <email>changed@changed.cz</email>
    <firstName>CHANGED John</firstName>
    <lastName>CHANGED Doe</lastName>
    <password>password2</password>
</user>
```

Ukázka zdroj. kódu 3.14: Tělo požadavku zdroje /users/{username} (PUT)

```
<user username="test">
    <email>changed@changed.cz</email>
    <firstName>CHANGED John</firstName>
    <lastName>CHANGED Doe</lastName>
</user>
```

Ukázka zdroj. kódu 3.15: Tělo odpovědi zdroje /users/{username} (PUT)

3.3.2.14 Seznam grafů uživatele

Zobrazí seznam všech grafů daného uživatele. Seznam je stránkovaný, stránkování lze řídit pomocí parametrů.

URI /users/{username}/graphs?[page=X]&[pageSize=Y]&[desc=B]

page číslo stránky. Výchozí hodnota je 1.

pageSize velikost jedné stránky. Výchozí hodnota je 50. 1 <= Y <= 50.

page řazení seznamu. Výchozí je false (vzestupně, nejstarší první).

Metoda GET

Potřebné oprávnění GRAPHS_READ

Tělo požadku Prázdné

Tělo odpovědi Seznam grafů uživatele (ukázka 3.16).

```
<graphs>
   <graph id="69">
       ...
   </graph>
    ...
</graphs>
```

Ukázka zdroj. kódu 3.16: Tělo odpovědi zdroje /users/{username}/graphs (GET)

3.3.2.15 Detail grafu

Zobrazí detail grafu s daným id. Graf může obsahovat odkaz na rodičovský graf, ze kterého vynikl. To umožňuje zobrazení historii editace grafu.

URI /users/{username}/graphs/{graph_id}

Metoda GET

Potřebné oprávnění GRAPHS_READ

Tělo požadku Prázdné

Tělo odpovědi Detail grafu (ukázka 3.17).

```
<graph>
    <created>2014-04-19T19:09:20+02:00</created>
    <name>
        Missereables_clockwise-rotate_force-atlas_force-atlas_page-
            rank
    </name>
    <parent id="59">
        <created>2014-04-19T19:09:05+02:00</created>
        <name>
            Missereables_clockwise-rotate_force-atlas_force-atlas
        </name>
    </parent>
</graph>
```

Ukázka zdroj. kódu 3.17: Tělo odpovědi zdroje /users/{username}/graphs/{graph_id} (GET)

3.3.2.16 Vytvoření nového grafu

Přidá nový graf do úložiště. Parametr name určuje jméno vytvořeného grafu.

URI /users/{username}/graphs[?name={graph_name}]

Metoda POST

Potřebné oprávnění GRAPHS_WRITE

Tělo požadku Graf ve formátu `gexf` (ukázka 3.18).

Tělo odpovědi Detail vytvořeného grafu.

```
<gexf version="1.2" xmlns="http://www.gexf.net/1.2draft" ...>
    ...
</gexf>
```

Ukázka zdroj. kódu 3.18: Tělo požadavku zdroje /users/{username}/graphs (POST)

3.3.2.17 Graf ve formátu `gexf`

Zobrazí graf v datovém formátu `gexf`.

URI /users/{username}/graphs/{graph_id}/gexf

Metoda GET

Potřebné oprávnění GRAPHS_READ

Tělo požadku Prázdné

Tělo odpovědi Reprezentace grafu ve formátu textttgexf.

3.3.2.18 Graf ve formátu `svg`

Zobrazí graf v grafickém formátu `svg`.

URI /users/{username}/graphs/{graph_id}/svg

Metoda GET

Potřebné oprávnění GRAPHS_READ

Tělo požadku Prázdné

Tělo odpovědi Reprezentace grafu ve formátu textttsvg (ukázka 3.19).

```
<svg contentScriptType="text/ecmascript" width="1026px"
     xmlns:xlink="http://www.w3.org/1999/xlink" zoomAndPan="
         magnify"
     contentStyleType="text/css" viewBox="-537 -514 1269 1040"
         height="841px"
     preserveAspectRatio="xMidYMid meet" xmlns="http://www.w3.org
         /2000/svg"
     version="1.1">
     ...
</svg>
```

Ukázka zdroj. kódu 3.19: Tělo odpovědi zdroje /users/{username}/graphs/{graph_id}/svg (GET)

3.3.2.19 Zpráva z výsledků statistické funkce

Zobrazí HTML zprávu, která je výsledkem aplikace statistické funkce. Dostupné pouze pokud poslední aplikovanou funkcí byla statistická funkce.

URI /users/{username}/graphs/{graph_id}/statistics-report

Metoda GET

Potřebné oprávnění GRAPHS_READ

Tělo požadku Prázdné

Tělo odpovědi HTML obsahující zprávu, většinou ve formě grafu rozložení hodnot.

3.3.2.20 Aplikace funkce na graf `svg`

Aplikuje funkci na graf (`Layout`, `Statistics` nebo `Ranking`). Výsledek je uložen jako nový graf a vrácen uživateli.

URI /users/{username}/graphs/{graph_id}

Metoda PUT

Potřebné oprávnění GRAPHS_WRITE

Tělo požadku Definice funkce (ukázka 3.20).

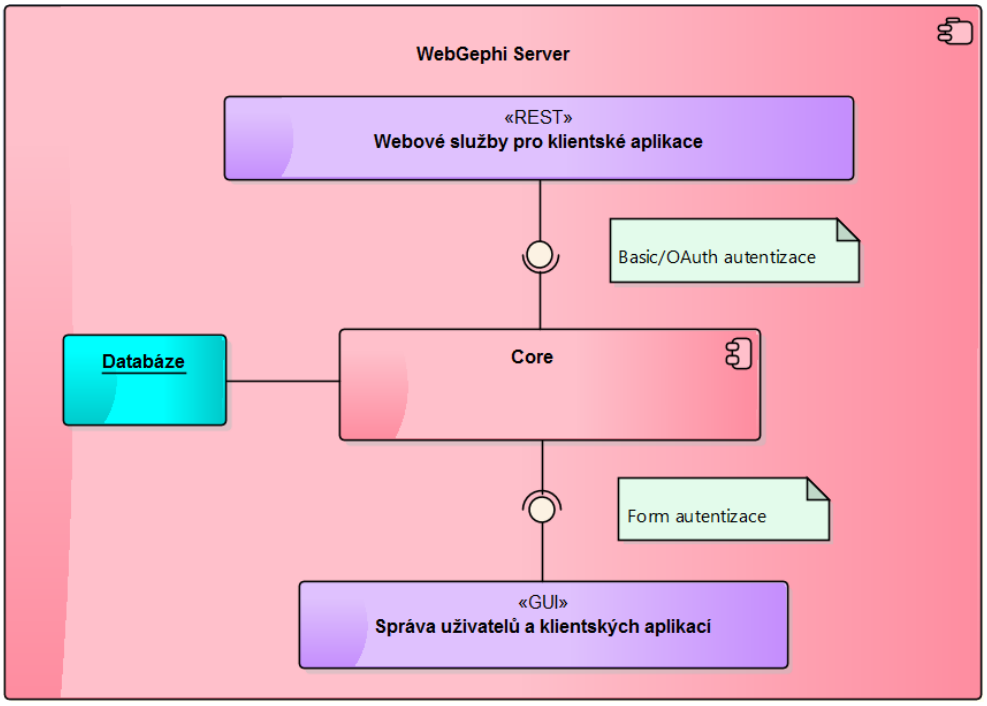
Tělo odpovědi Detail nově vytvořeného grafu.

```
<function xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <statistic id="clustering-coefficient">
        <name>Clustering Coefficient</name>
    <properties>
      <property id="Directed">
        <name>Directed</name>
        <value>
          <boolean value="false"/>
        </value>
      </property>
    </properties>
    </statistic>
</function>
```

Ukázka zdroj. kódu 3.20: Tělo požadavku zdroje /users/{username}/graphs/{graph_id} (PUT)

3.4 Autentizace a autorizace

WebGephi bude poskytovat dvě různá rozhraní pro interakci s uživateli: grafické rozhraní pro správu uživatelů a klientských aplikací a REST webové služby pro práci s grafy (viz diagram. 3.4). K oběma rozhraním bude řízen přístup na základě uživatelských oprávnění a rolí. Autentizace bude v případě REST rozhraní umožněna metodou Basic a OAuth protokolu. Přihlašování ke grafické části aplikace bude realizováno klasicky pomocí jména a hesla (tzv. Form autentizace).



Obrázek 3.4: Struktura WebGephi

3.4.1 Uživatelská oprávnění a role

Přístup k jednotlivým zdrojům aplikace bude řízen pomocí uživatelských oprávnění. Pokud přihlášený uživatel nebude mít potřebná práva, bude mu přístup ke zdroji zamezen. Skupiny oprávnění budou sdruženy do rolí, které představují kategorii uživatelů.

3.4.1.1 Oprávnění

Každé oprávnění umožňuje přístup k jednomu typu operace (zdroje).

PROFILE_READ Čtení profilu přihlášeného uživatele

PROFILE_WRITE Editace profilu přihlášeného uživatele

GRAPHS_READ Zobrazení všech grafů aktuálně přihlášeného uživatele

GRAPHS_WRITE Přidání nového grafu a editace grafů (aplikace funkcí) přihlášeného uživatele

ALL_PROFILES_READ Zobrazení všech uživatelů v aplikaci

ALL_PROFILES_WRITE Editace jakéhokoliv uživatele v aplikaci

3.4.1.2 Role

Role představuje skupinu oprávnění. Role uživatele bude určena při přihlášení na základě typu účtu a způsobu přihlášení.

ADMIN Správce stránek. Platná pouze pro Basic a Form autentizaci. Obsahuje všechna dostupná oprávnění: `PROFILE_READ`, `PROFILE_WRITE`, `GRAPHS_READ`, `GRAPHS_WRITE`, `ALL_PROFILES_READ`, `ALL_PROFILES_WRITE`.

USER Běžný uživatel aplikace, pokud je přihlášen pomocí jména a hesla (Form nebo Basic metodou). Má oprávnění editovat své vlastní zdroje (`PROFILE_READ`, `PROFILE_WRITE`, `GRAPHS_READ`, `GRAPHS_WRITE`).

CLIENT_APP Role pro klientské aplikace, tzn. pro uživatele přihlášeného z jiné aplikace přes protokol OAuth. Klientská aplikace může požadovat oprávnění `PROFILE_READ`, `GRAPHS_READ` a `GRAPHS_WRITE` (tzv. „scopes“ v OAuth terminologii). Požadovaná oprávnění se uživateli zobrazí při zobrazení žádosti o udělení přístupu (potvrzení `Request tokenu`). Některé klientské aplikace mohou např. požadovat pouze práva ke čtení grafů, jiné i k editaci grafu a zobrazení informací o uživateli. Uživatel vždy uvidí jaká práva aplikace požaduje a na základě toho se rozhodne, zda jí přístup udělí.

3.5 Správa uživatelů a klientských aplikací

Správa uživatelů a klientských aplikací bude realizována pomocí grafického rozhraní. Základní správa uživatelských účtů bude navíc umožněna pomocí REST webových služeb. Předpokládáme dva základní případy užití[22] (viz obr. 3.5).

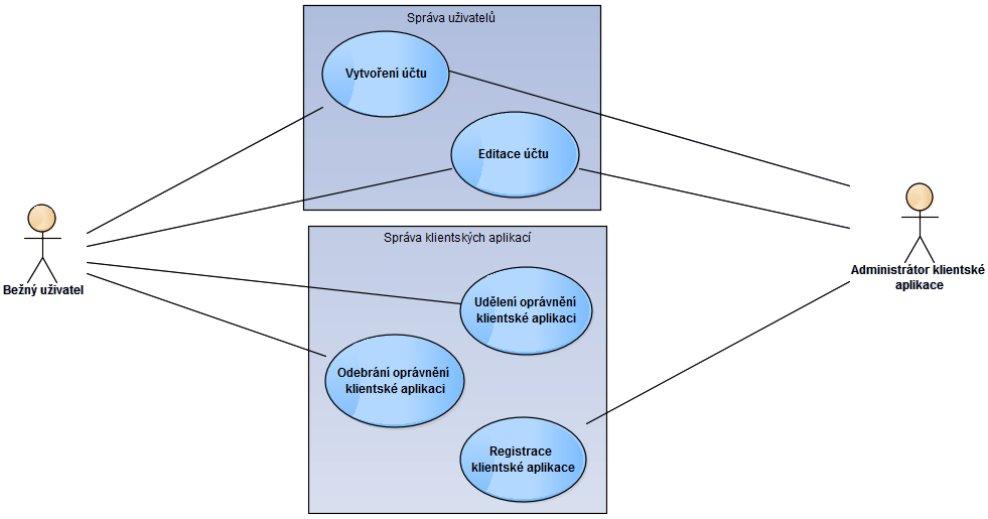
3.5.1 Běžný uživatel

Běžný uživatel je na WebGephi přesměrován z klientské aplikace. Založí si zde uživatelský účet, čímž získá možnost vytvářet a editovat vlastní grafy. To však nedělá přímo, místo toho udělí právo přístupu klietské aplikaci (pomocí OAuth protokolu). Tato oprávnění může kdykoli odebrat.

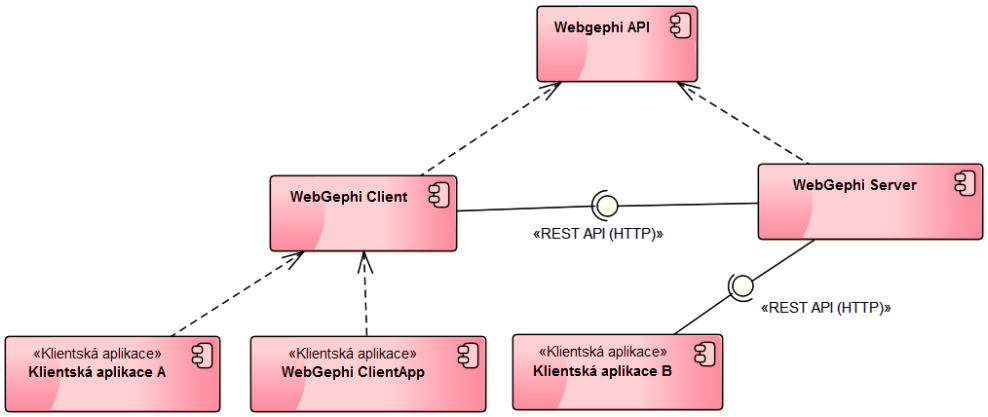
3.5.2 Administrátor klientské aplikace

Jedná se o správce, případně vývojáře nějaké aplikace, která chce využívat možností WebGephi. Nejdříve si musí založit běžný účet. V rámci svého uživatelského účtu si zaregistruje svou klientskou aplikaci, čímž získá potřebné informace pro přihlášování pomocí OAuth protokolu (`consumer_key` a `consumer_secret`). Následně může ze své aplikace od uživatelů požadovat přístup k jejich účtu na WebGephi.

[22] Use-case



Obrázek 3.5: Případy užití uživatelského účtu



Obrázek 3.6: Struktura WebGephi aplikací a jejich modulů

3.6 Struktura aplikace

Cílem této práce je kromě vytvoření serverové aplikace také vytvoření ukázkové klientské aplikace. V této kapitole si popíšeme strukturu a závislosti těchto aplikací a jejich modulů. Základní struktura je znázorněma na obr. 3.6.

3.6.1 Server

Serverová aplikace (dále jen *Server* nebo *WebGephi Server*) je základní aplikací, která realizuje samotné operace s grafy a řeší správu uživatelů. *WebGephi Server* poskutuje přístup ke své funkcionalitě pomocí REST rozhraní. Vytvoření této aplikace je hlavním cílem této práce.

3.6.2 Klient (Java konektor)

Přítup k *Serveru* je umožněn pomocí REST webových služeb a autorizace pomocí OAuth protokolu[23]. Implementace OAuth klienta však není triviální. Proto bude pro potřeby klientských aplikací vytvořen „konektor“ k *WebGephi Serveru* (dále jen *Client* nebo *WebGephi Client*). Ten bude umožňovat jednoduché vytvoření autorizovaného připojení k *Serveru* a poskytovat Java metody pro přístup ke zdrojům REST rozhraní.

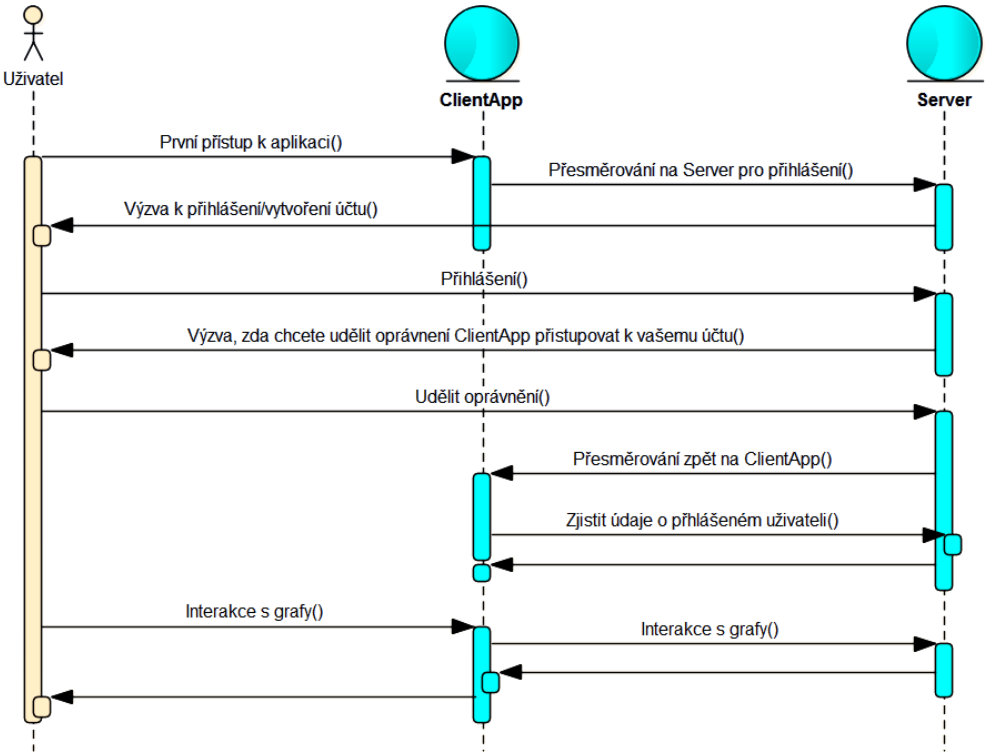
WebGephi Client bude standardní Java knihovnou, jeho použití tedy bude limitováno na klientské aplikace implementované v jazyku Java.

3.6.3 Klientská aplikace

Klientskou aplikací je míněna jakákoli aplikace využívající REST rozhraní *Serveru*. A to buď přímo (například JavaScriptové aplikace), nebo pomocí knihovny *WebGephi Client* (Java aplikace). Součástí práce bude vytvoření ukázkové klientské aplikace (dále jen *ClientApp* nebo *WebGephi ClientApp*). Ta bude se *Serverem* komunikovat pomocí knihovny *Client*. V podstatě se bude jednat o grafickou nadstavbu nad REST rozhranním serveru.

Na obr. 3.7 je znázorněn proces použití *ClientApp*. Klientská aplikce nejdříve přesměruje uživatele na *Server*, aby jí udělil oprávnění přistupovat k jeho účtu. Po přihlášení deleguje veškeré operace s grafy na REST rozhraní serveru a pouze zobrazuje výsledky.

[23] Basic autorizace je také možná, ale preferováno je použití OAuth protokolu.



Obrázek 3.7: Proces používání *ClientApp*

Kapitola 4

Realizace

Na základě předchozích kapitol (analýzy a návrhu) byla implementována aplikace *WebGephi* (*Server*, *Client* a všechny související moduly). V této kapitole popíšeme způsob implementace, použité technologie a co vedlo k jejich výběru.

4.1 Použité technologie

V současné době se téměř žádné aplikace nestaví na zelené louce, jen s prostředky poskytovanými samotným jazykem. U webových aplikací toto platí stonásobně. Během vývoje se řeší stále stejné problémy, vytvoření webového GUI, práce s databází a management transakcí, serializace a deserializace objektů do XML, ...

Pro zjednodušení a urychlení vývoje, aby nebylo nutné stále znovu vynalézat kolo, vznikly kolekce knihoven řešící tyto problémy - „frameworky“. Mnoho těchto frameworků bylo použito i při vývoji *WebGephi*.

4.1.1 JavaEE[12]

JavaEE[24] není framework v právém slova smyslu. Jedná se o verzi jazyka javy rozšířenou o technologie vhodné pro vývoj tzv. „Enterprise“ aplikací - podnikové a webové aplikace, pro které je charakteristická potřeba integrace s jinými systémy. Aplikace pro platformu Java EE jsou vyvíjeny na základě API a dalších fragmentů definovaných v jednotlivých specifikacích. Běhovým prostředím jsou tzv. aplikační servery, které jsou vyvíjeny různými společnostmi. Např. společnost Oracle vyvíjí aplikační server *GlassFish*, který je referenční implementací. Mezi další známe servery patří JBoss AS (nyní Wildfly), IBM WebSphere a WebLogic. Vyvíjené aplikace by měli být mezi těmito servery libovolně přenostitelné. Mnoho z nich však obsahuje svá proprietární rozšíření, která mohou tuto přenositelnost narušit.

[24] Java Platform, Enterprise Edition

Součástí specifikace je např. technologie Java Server Faces (JSF), Java Persistence API (JPA), Enterprise Java Beans (EJB) a Contexts and Dependency Injection (CDI).

Historickým konkurentem JavaEE je proprietární open-source framework Spring[13]. Ten byl populární především v době JavaEE 1.4, kdy byla velmi těžkopádná a složitá na konfiguraci. Od té doby prošla JavaEE velkým vývojem a převzala mnoho myšlenek právě od frameworku Spring. V dnešní době jsou tyto dvě platformy rovnocené a z velké části využívají i stejné frameworky.

Nejnovější verzí je JavaEE 7, která byla vydána v roce 2013. Tato verze také byla použita při vývoji aplikace *WebGephi*. Před platformou Spring jsem jí dal přednost především kvůli svým zkušenostem s touto technologií.

4.1.2 Wildfly 8[14]

Wildfly (dříve JBoss AS) je open-source aplikační server od společnosti JBoss. Osmá verze tohoto aplikačního serveru je plně kompatibilní s JavaEE 7. Mezi jeho hlavní rysy patří rychlý start, malá spotřeba paměti a modularita.

Komerční verze tohoto serveru se nazývá JBoss EAP[25]. Jedná se v zásadě o stejný server jako Wildfly, obsahuje však jen stabilní, ověřené verze modulů a zahrnuje také placenou podporu.

Tomuto serveru jsem dal přednost před jinými především kvůli svým dobrým zkušenostem s ním. Je využíván i v komerčním prostředí a jeho hlavním benefitem je rychlost a přehledná konfigurace. Migrace na jiný aplikační server však není problém, je pouze třeba dát pozor na používání proprietárních deployment deskriptorů.

4.1.3 JAX-RS[15]

JAX-RS je standardní JavaEE technologie pro deklarativní vývoj REST webových služeb. Pomocí anotací se označí metody, které mají být zveřejněny jako webové služby (viz ukázka 4.1). Aplikační server se pak postará o jejich zveřejnění. Tuto technologii používá *WebGephi Server* pro implementaci REST rozhraní.

```
@POST
@Consumes(MediaType.TEXT_XML)
@Produces(MediaType.TEXT_XML)
public GraphXml addGraph(String document, @PathParam("user")
    String user, @QueryParam("name") String name, @QueryParam("
    format") String format) {
    ...
}
```

Ukázka zdroj. kódu 4.1: Definice RESTful služby pro import grafu

[25] Enterprise Application Platform

JAX-RS je pouze specifikace, v současnosti existují dvě implementace: referenční Jersey od společnosti Oracle a RestEasy od společnosti JBoss. Aplikační server Wildfly přirozeně obsahuje RestEasy implementaci.

JAX-RS používá technologii JAXB[26]. Ta je přímo součástí JDK a slouží k jednoduché deklarativní serializaci a deserializaci objektů do XML. *WebGephi* JAXB modelové objekty jsem vyčlenil do modulu *WebGephi API*, aby mohly být znovupoužity v modulu *WebGephi Client* (viz diagram 3.6).

Od verze 7 je součástí JavaEE platformy také standardizovaný REST klient. Ten je použit v modulu *WebGephi Client*.

4.1.4 Java Persistence API

Objektový model aplikace založený na objektovém paradigma se příliž neslučuje se světem relačních databází. Pro jejich integraci bylo vyvinuto mnoho frameworků. Nejrozšířenějším frameworkem v Java světě je JPA, v současnosti ve verzi 2.0.

Java Persistence API (JPA) je technologie pro práci s databází založená na objektovém přístupu. Java objekty jsou mapovány na databázové tabulky, případně vztahy mezi nimi. Programátor je tak odstíněn od databáze a místo s ní pracuje s tzv. Entity manažerem. Ten poskytuje metody pro ukládání (`persist(entity)`, `merge(entity)`), načítání (`find(class, id)`) a dotazování pomocí `Query` objektu.

Kromě absence nutnosti psát SQL dotazy mezi hlavní výhody JPA patří jednoduchá výměna implementace databáze a jednoduché kešování na úrovni Entity manažera.

JPA je pouze specifikace, nejrozšířenější implementace jsou Hibernate, OpenJPA a EclipseLink. Aplikace *WebGephi Server* používá JPA (Hibernate) pro veškerou práci s databází.

4.1.5 Enterprise Java Beans[17]

Enterprise Java Beans jsou serverové komponenty, jejichž životní cyklus je spravován kontejnerem. Používají se především pro implementaci bussiness logiky Enterprise aplikací. Poskutují služby jako je transakční zpracování, řízení souběžného přístupu, události pomocí JMS, jmenné a adresářové služby JNDI, interceptory.

Existují tři základní typy EJB: `Stateles`, `Stateful` a `Singleton`. Aplikace *Gephi Server* využívá EJB pro inicializaci po deploymentu (`Singleton`), jako DAO[27] objekty a koncové body JAR-RS (`Stateles`). Využívá především schopností řízení souběžného přístupu, deklarativní správu transakcí a možnost navázání bussiness logiky po deploymentu aplikace.

[26] Java Architecture for XML Binding[16]

[27] Data Access Object

4.1.6 CDI

CDI[28] je součástí JavaEE od verze 6. Vzniklo na základě stížností, že koncept EJB je příliž těžkopádný a náročný na zdroje. Jedná se tedy o odlehčenou alternativu a doplněk k Enterprise Java Beans. Poskytují např. možnost navázání interceptorů k metodám, možnost injektu závislostí, vlastní messaging systém a od verze JavaEE 7 také schopnost deklarativného řízení transakcí.

WebGephi Server využívá CDI interceptorů pro ověřování autorizace uživatele. V aplikaci *ClientApp* za využívá CDI messaging systém pro notifikaci grafických komponent pokud došlo k nějaké globální akci (např. pokud se změnil aktuální graf).

4.1.7 GUI frameworky

Standardní technologií pro tvorbu GUI v JavaEE je komponentový framework Java Server Faces (JSF). Ten je vhodný především pro formulářové aplikace, které jsou běžné v podnikovém prostředí. Bohužel, jedná se o relativně starou technologii, do které byla např. podpora pro AJAX přidána až ve verzi 2.0. Není tedy příliš vhodný pro tvorbu moderních AJAXových aplikací, kde server slouží jen jako zdroj dat a prezentační logika je řízena přímo na klientovi (prohlížeči).

Proto jsem dal při implementaci *WebGephi* přednost frameworkům založeným na Google Web Toolkit (GWT)[19][20].

4.1.7.1 GWT

GWT je framework od společnosti Google, umožňující jednoduchou tvorbu RIA[29] aplikací v jazyku Java. Většina existujících technologií pro tvorbu RIA, jako je Flash, JavaFX nebo Silverlight, jsou nové programovací jazyky. Pro spuštění takovýchto aplikací je potřeba mít v prohlížeči nainstalovaný příslušný plugin.

GWT k problému přistupuje jinak. Všechny moderní prohlížeče mají nativní podporu skriptovacího jazyku JavaScript. Problémem JavaScriptu je, že původně byl konstruován jen pro drobné změny ve struktuře stránky. Je tedy velmi náročné udržovat rozsáhlejší aplikaci napsanou v JavaScriptu. GWT řeší problém tím, že poskytuje překladač, který dokáže Java kód přeložit do JavaScriptu.

V principu jednoduchá myšlenka, ale s obrovskými dopady. Programátoři téměř vůbec nemusí znát JavaScript, píší celou aplikaci (klientskou i serverovou část) v Javě. To jim umožňuje používat všechny výhody Javy, jako jsou typovost, objektový přístup, dědičnost a také možnost testování kódu např. pomocí frameworku JUnit. Část kódu, která běží na klientovi, je při překladu

[28] Contexts and Dependency Injection for Java EE[18]
[29] Rich Internet Application

přeložena do JavaScriptu (optimalizovaného na rychlé zobrazení a malou velikost souborů - není nutné udržovat kód přehledný) a může být spuštěna ve webovém prohlížeči.

GWT je velmi mocný framework, zároveň však dost nízkoúrovňový. Proto vznikly další frameworky, které obsahují komplexnější logiku a využívají GWT překladač pro překlad do JavaScriptu. Příkladem těchto frameworků je Errai a Vaadin.

4.1.7.2 Errai

Framework Errai[21] si vzal za cíl přinést výhody JavaEE i do prohlížeče. To samozřemě není možné v plném rozsahu. Mezi jeho hlavní výhody patří podpora injektu závislostí, mapování atributů kontrolerů do HTML šablon a podpora pro CDI messaging (i obousměrně mezi prohlížečem a serverem).

Tento framework jsem použil pro tvorbu grafického rozhraní aplikace *WebGephi Server* pro správu uživatelů a klientských aplikací. A to především z důvodu, že je paměťově velmi nenáročný - veškerá logika běží v prohlížeči v JavaScriptu a server slouží jen jako zdroj dat.

4.1.7.3 Vaadin

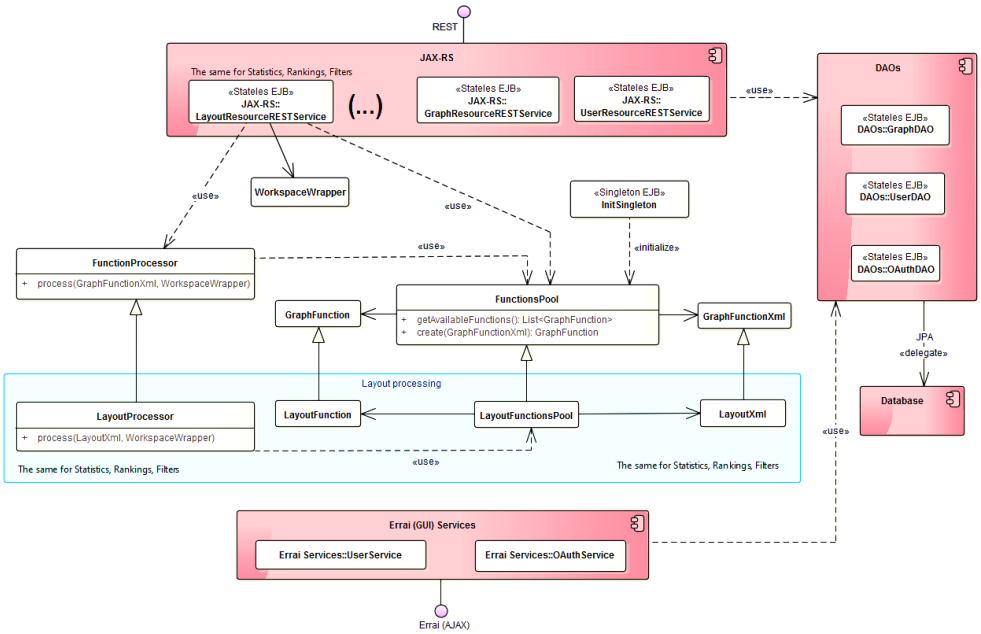
Vaadin[22] je na rozdíl od frameworku Errai serverový framework. GUI komponenty jsou umístěny na serveru a jejich stav se zrcadlí v prohlížeči. Pokud např. uživatel klikne v prohlížeči na tlačítko, informace o akci se přenese na server v krátkém ajaxovém dotazu, kde se vyhodnotí navázaná akce a poté se aktualizuje stav komponent v prohlížeči.

Tento model má výhody i nevýhody. Výhodou je možnost psát aplikace stejným způsobem jako desktopové aplikace, rychlý vývoj a možnost využití všech schopností JavaEE kontejneru. Nevýhodou je větší paměťová náročnost, protože všechny komponenty jsou drženy v session na serveru. Po každé uživatelské operací je také potřeba komunikace se serverem, což vede k prodlevě mezi akcí a reakcí systému. Na druhou stranu jsou AJAXové dotazy velmi krátké, což paradoxně může vést k rychlejší odezvě, než při použití jiných technologií. V případě nutnosti Vaadin také umožňuje vytvoření svých vlastních grafických komponent pomocí GWT.

Framework Vaadin jsem použil při implementaci aplikace *WebGephi ClientApp* a to především z důvodu rychlého a jednoduchého vývoje.

4.2 WebGephi Server

WebGephi Server se skládá z několika logických komponent. Jejich struktura a závislosti jsou znázorněny na diagramu 4.1.



Obrázek 4.1: Struktura aplikace *WebGephi Server*

.

4.2.1 DAOs

WebGephi Server využívá k přístupu technologii JPA. Přesto je její použití ještě obaleno použitím tzv. Data Access Objektů. To umožňuje v budoucnu např. záměnu úložiště grafů za NoSQL databázi.

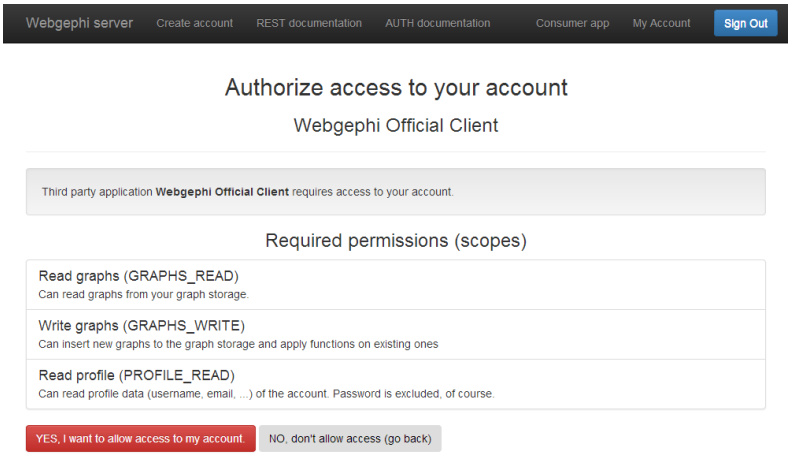
4.2.2 Třídy pro zpracování grafových funkcí

Tyto třídy jsou jádrem *WebGephi* a obsahují hlavní bussiness logiku. Po deploymentu aplikace proběhne scan tříd a do `FunctionsPool`ů se uloží všechny dostupné grafové funkce. O aplikaci funkce na graf se starají implementace třídy `FunctionProcessor`. Ty z xml definice funkce (JAXB třídy) vytvoří její instanci a aplikují ji na *Gephi* workspace (`WorkspaceWrapper`).

Všechny tyto třídy (`FunctionsPool`, `FunctionProcessor`, `GraphFunction` a `FunctionXml`) mají svou vlastní implementaci pro každou kategorii funkcí (Layouts, Statistics, Rankings a Filters). To v budoucnu umožňuje snadné rozšíření o další funkcionalitu, např. o „Generátory“.

4.2.3 REST rozhraní

REST rozhraní je tvořeno JAX-RS třídami, které obsahují funkce pro obsluhu jednotlivých zdrojů. Využívají jak DAO komponenty, tak i třídy pro zpracování grafových funkcí. Součástí aplikace *WebGephi Server* je i dokumentace



Obrázek 4.2: *WebGephi Server* - Ukázka GUI (žádost o autorizaci aplikace)

.

pro vývojáře popisující strukturu REST rozhraní a jednotlivých zdrojů. XML formát příchozích požadavků a odpovědí je navíc definován pomocí vlastního XSD schématu.

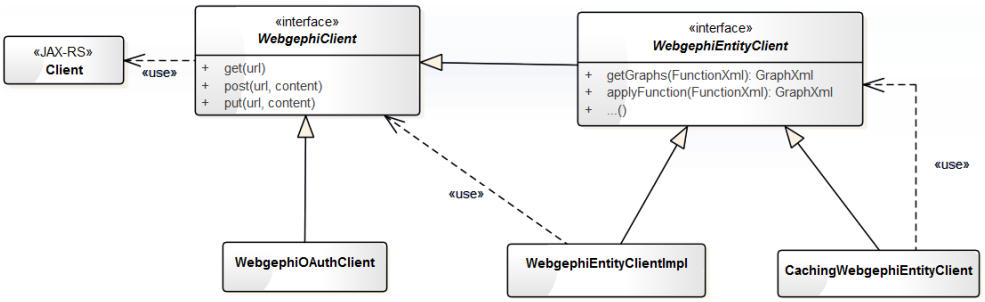
4.2.4 Správa uživatelů a klientských aplikací

Rozhraní pro správu uživatelů, OAuth autorizaci a správu klientských aplikací je implementováno pomocí frameworku Errai, který je rozšířením GWT. Skládá se z klientské (package `cz.cokrtvac.webgephi.webgephiserver.gwt.client`) a serverové (package `cz.cokrtvac.webgephi.webgephiserver.gwt.server`) části. Klientský kód je během buildu aplikace přeložen do html a JavaScriptu, běží ve webovém prohlížeči a vytváří grafické uživatelské rozhraní. Komunikuje se serverem za pomoci AJAXu a volá službu serverové části. Serverová část se stará o kontrolu autorizace a komunikuje s databází (DAO).

4.3 WebGephi Client

Webgephi Client je Java knihovna pro zjednodušení komunikace s REST rozhraním *WebGephi Serveru*. Klientské aplikace implementované v jazyku Java mohou tuto knihovnu využít, aby nemusely implementovat své vlastní řešení pro OAuth autentizaci a volání RESTful služeb.

K autorizaci (získání OAuth request tokenu) slouží třída `WebgephiAuthenticator`. Poté můžeme vytvořit samotný konektor - třídy implementující rozhraní `WebgephiClient`. Tyto třídy jsou struktutovány podobně jako Input/Output streamy v JDK (viz class diagram 4.3). Základní třída `WebgephiOAuthClient` umí pouze volat HTTP metody (`GET, POST, PUT`) podepsané



Obrázek 4.3: Struktura *WebGephi* konektorů (klientů)

.

OAuth Request tokenem. Třída `WebgephiEntityClientImpl` umí již volat konkrétní zdroje *WebGephi Server*, přičemž používá jednodušší `WebgephiClient` předaný v konstruktoru. Třída `CachingWebgephiEntityClient` umí navíc ještě kešovat neměné zdroje. Vytvoření kešovaného klienta je znázorněno v ukázce 4.2

```
WebgephiEntityClient client = new CachingWebgephiEntityClient(new
    WebgephiEntityClientImpl(new WebgephiOAuthClient("https://
    webgephi.local:8443/rest/v1", accessToken)));
```

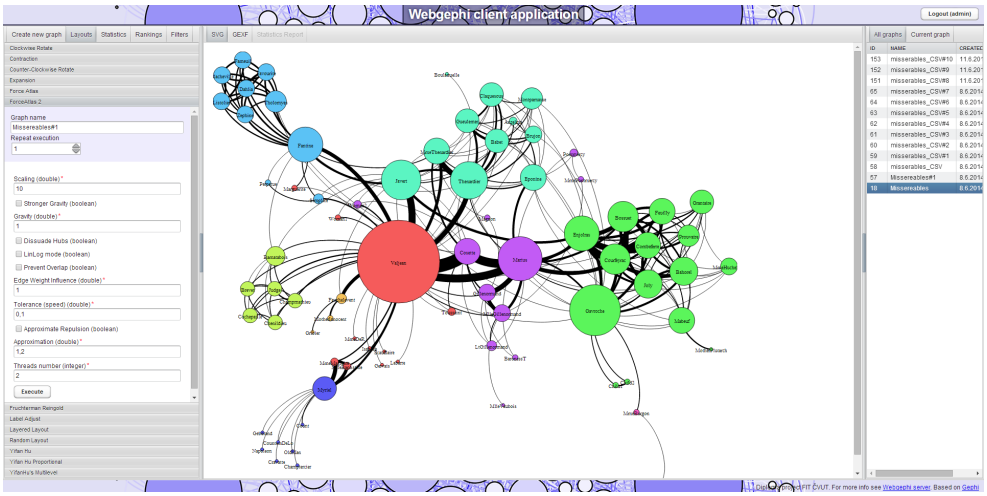
Ukázka zdroj. kódu 4.2: Vytvoření `CachingWebgephiEntityClient`

4.4 WebGephi ClientApp

Aplikace *ClientApp* slouží k demonstraci funkčnosti *WebGephi Serveru*. Zároveň slouží jako ukázková aplikace pro vývojáře klientských aplikací. Jedná se o grafickou nadstavbu nad REST rozhraním, která umožňuje interakci se *Serverem* téměř v plném rozsahu. Aplikace je implementována pomocí framworku pro tvorbu uživatelského rozhraní Vaadin (viz kapitola 4.1.7.3).

Uživatel se nejdříve musí přihlásit pomocí protokolu OAuth na *Serveru* - je přesměrován na žádost o udělení přístupu ke jeho účtu. Po schválení žádosti je přesměrován zpět a v grafické podobě je mu zobrazen obsah jeho grafového úložiště (obr. 4.4).

Výběrem grafu v pravé části obrazovky se zobrazí jeho detail ve středním panelu. Na aktuální graf je možno aplikovat libovolné grafové funkce - nabídka v levé části obrazovky. Aplikace také umožňuje nahrání vlastního grafu v libovolném formátu.



Obrázek 4.4: Náhled aplikace *WebGephi Client App*

.

Kapitola 5

Testování

Něco málo o testování

5.1 Jednotkové testy

5.2 Integrační testy

Testy REST rozhraní s použitím *WebGephi Client*.

5.3 Zátěžové testy

JMeter

Závěr

Aplikace *WebGephi* byla implementována v celém rozsahu zadání.

Literatura

[1] Gephi website. 2014. Dostupné z: http://gephi.org

[2] Wikipedie: Graf (teorie grafů) — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 21. 04. 2014]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Graf_(teorie_grafu)&oldid=10977342

[3] Wikipedie: Representational State Transfer — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 22. 04. 2014]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=10902570

[4] Gephi toolit website. 2014. Dostupné z: https://gephi.org/toolkit/

[5] Gephi Consortium. 2014. Dostupné z: https://consortium.gephi.org/

[6] GEXF formát. 2014. Dostupné z: http://gexf.net/format/

[7] Wikipedie: Singleton — Wikipedie: Otevřená encyklopedie. 2014, [Online; navštíveno 1. 05. 2014]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Singleton&oldid=11083315

[8] Wikipedie: Basic access authentication — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 3. 05. 2014]. Dostupné z: http://cs.wikipedia.org/w/index.php?title=Basic_access_authentication&oldid=9997988

[9] Wikipedia: Digest access authentication — Wikipedia, The Free Encyclopedia. 2013, [Online; accessed 3-May-2014]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Digest_access_authentication&oldid=588566752

[10] OAuth: OAuth Core 1.0. 2014. Dostupné z: http://oauth.net/core/1.0/

[11] OAuth. 2014. Dostupné z: http://oauth.net

[12] JavaEE. 2014. Dostupné z: http://www.oracle.com/technetwork/java/javaee

[13] Spring. 2014. Dostupné z: http://spring.io/

[14] Wildfly 8. 2014. Dostupné z: http://www.wildfly.org/

[15] Java API for RESTful Services (JAX-RS). 2014. Dostupné z: https://jax-rs-spec.java.net

[16] Java Architecture for XML Binding (JAXB). 2014. Dostupné z: http://jaxb.java.net/

[17] Enterprise Java Beans (EJB). 2014. Dostupné z: http://www.oracle.com/technetwork/java/javaee/ejb

[18] Contexts and Dependency Injection for Java EE (CDI). 2014. Dostupné z: http://cdi-spec.org/

[19] Google Web Toolkit (GWT). 2014. Dostupné z: http://www.gwtproject.org/

[20] Adam Tacy, J. E., Robert Hanson; Tökke, A.: *GWT in Action, Second Edition*. Manning, 2013, ISBN 9781935182849.

[21] Errai. 2014. Dostupné z: http://erraiframework.org/

[22] Vaadin. 2014. Dostupné z: https://vaadin.com/

Příloha **A**

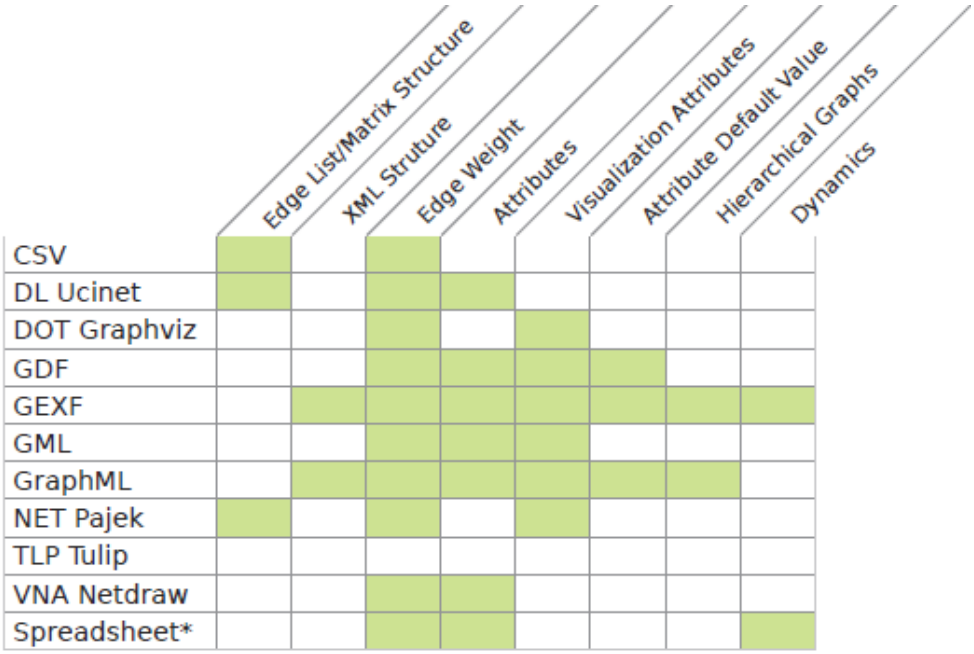
Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

Příloha B

Gephi



Obrázek B.1: Porovnání formátů vhodných pro ukládání grafů. Převzato z [1]

.

Příloha C

Náhledy uživatelského rozhraní

C.1 WebGephi Server

PrintScreens...

C.2 WebGephi Client Application

PrintScreens...

Příloha D

Instalační příručka

Příloha **E**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS