Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vizualizace rozsáhlých diagramů komponent a interakce s nimi

Plzeň 2013 Štěpán Kubásek

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů

V Plzni dne 6. srpna 2013

Štěpán Kubásek

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Lukáši Holému za odborné vedení a konzultace během mé práce.

Abstract

The goal of this paper is to examine and understand various aspects of component based application development and based on the findings implement improvements to an existing component based application.

The application focuses on visualization and manipulation with large component diagrams. Imporvements implemented into the application include advanced highlighting of connected components, handling unconnected components and most importantly, optimizing the layout of displayed components.

Obsah

1 Úvod **1**

2 Seznámení s problematikou **2**
2.1 Komponentové programování 2
2.1.1 Komponenta 2
2.1.2 Komponentové rozhraní 3
2.1.3 Kontrakty komponent 3
2.1.4 Komponentové modely a frameworky 4
2.1.5 Kompozice 6
2.2 Off-Screen techniky 9
2.2.1 Snižování vizuálního šumu ve velkých komponentových diagramech 9
2.2.2 Viewport pro komponentové diagramy 13
2.2.3 Overview plus detail 14

3 Existující aplikace **15**

4 Přidané funkce **17**
4.1 Zvýraznění propojených komponent 17
4.1.1 Návrh 17
4.1.2 Implementace 18
4.1.3 Představení a ověření 18
4.2 Vylepšení vizualizace označení 19
4.2.1 Návrh a implementace 19
4.2.2 Představení a ověření 20
4.3 Počet propojených komponent 20
4.3.1 Návrh 20
4.3.2 Implementace 21
4.3.3 Představení a ověření 21
4.4 Odebrání celé skupiny 22
4.4.1 Návrh 22

4.4.2 Implementace 22
4.4.3 Představení a ověření 23
4.5 Úložiště nepropojených komponent 24
4.5.1 Návrh a implementace 24
4.5.2 Představení a ověření 25
4.6 Layout komponent 25
4.6.1 Návrh a implementace 28
4.6.2 Představení a ověření 29
4.7 Dynamická vlastnost layoutu 30
4.7.1 Návrh a Implementace 31
4.7.2 Představení a ověření 31

5 Instalace a spuštění aplikace 33
5.1 Instalace Apache Tomcat 33
5.2 Instalace apliakce CoCA-Ex 33
5.3 Spuštění apliakce 34

6 Závěr 35

A Zdrojové soubory aplikace 38
A.1 Java balíky a třídy 38
A.2 JavaScriptové soubory 40
A.3 Soubory se styly 41

1 Úvod

Cílem této práce je implementovat aplikaci pro interakci s rozsáhlými grafy. Grafem je myšlen graf s uzly, které propojují hrany. Například graf lidí v sociální síti (kdo zná koho), komponent v aplikaci a jejich propojení, propojení letišť atd. K tomu, aby byl pro uživatele rozsáhlý graf přehledný, je nutné použít různé techniky vizualizace a interakce, které umožní uživateli se v rozsáhlých diagramech vyznat a pracovat s nimi.

Mým úkolem není tuto aplikaci vytvořit, ale vylepšit aplikaci existující. Tuto aplikaci vyvinula Bc. Jindra Pavlíková, která ji vytvořila jako svou diplomovou práci na rok 2012. Mým úkolem je implementovat do této aplikace další funkce, které zjednoduší práci jak s touto aplikací, tak s rozsáhlými komponentovými diagramy.

Navrženo bylo několik vylepšení pro stávající aplikaci, z nichž nejdůležitější, je vytvoření layoutu (rozvržení) komponent, jelikož doposud byly komponenty rozmísťovány náhodně. Implementace layoutu sníží vizuální šum, který tvoří z velké části velké množství překrývajících hran.

2 Seznámení s problematikou

V následující kapitole je popsána problematika komponentového programování a její jednotlivé součásti.

2.1 Komponentové programování

Základní myšlenka komponentového programování je ta, že aplikaci postupně dělíme na menší, samostatné a kompletní části, kterým říkáme komponenty. Implementace komponenty není vidět, místo toho je k dispozici její rozhraní (interface) přes které se komponenta používá (volají se metody interface) a má kompletně popsanou funkcionalitu (specifikaci, kontrakt), aby volající věděl, co může očekávat. [1]

Tato struktura umožňuje využívat již existující komponenty bez nutnosti znovu psát kód se stejnou funkčností, jako má využitá komponenta. Tím dochází k urychlení a zjednodušení vývoje aplikací, což vede ke snížení cen softwarů.

2.1.1 Komponenta

Komponenta je package, modul nebo namespace (závisí na použitém programovacím jazyky), který sdružuje podobnou funkcionalitu, případně třídy. Implementace komponenty (používané objekty, funkce, data) by měla zůstat skrytá, viditelný by měl být pouze interface, přes který se bude komponenta používat. Tomuto principu se říká black-boxový model a je v reálném světě téměř nedosažitelný (i ze zkompilovaného kódu se dá vždy něco získat). Třetí strana (kdokoliv, kdo komponentu používá) má k dispozici pouze viditelný interface a specifikaci komponenty.

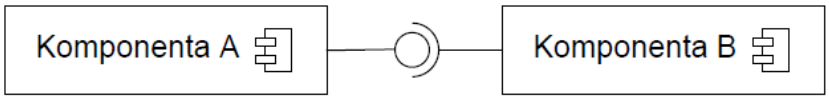
Konkrétní definice softwarové komponenty se liší v závislosti na modelu, ale obecně platí [1]:

- skrytá/neveřejná implementace funkcionality
- je používána třetí stranou (third-party)
- odpovídá komponentovému modelu

2.1.2 Komponentové rozhraní

Komponentové rozhraní, neboli interface, je ta část komponenty, která je viditelná ostatním. Rozhraní popisuje, co komponenta umí, a co potřebuje a lze skrze něj volat funkce komponenty. Díky rozhraní může být mezi dvěma komponentami navázána komunikace, nebo kontrakt, což znamená, že jedna komponenta využívá funkce druhé skrze rozhraní.

Na obrázku je znázorněno rozhraní mezi komponentami A a B. Komponenta A poskytuje vlastnosti komponentě B a naopak komponenta B vyžaduje vlastnosti komponenty A.



Obrázek 2.1: Rozhraní mezi komponentami [3]

2.1.3 Kontrakty komponent

Rozhraní lze popsat jako jednosměrné závislosti klienta na modulu, který implementuje služby tomuto klientovi. Tzn. klient má určité předpoklady o modulu a je na těchto předpokladech závislý.

Je však přesnější říci, že klient a modul jsou navzájem závislí. Klient závisí na tom, že modul poskytne službu určitým způsobem, a modul závisí na tom, že klient k němu bude přistupovat a používat jeho funkce určitým způsobem.

Kontrakty pomáhají zajistit správnou implementaci a užívání modulu.

Definují např. sémantiku rozhraní ve smyslu předpokládaného chování volajících a volaných metod.

Úrovně kontraktů a interakce:

- syntaktická (typy, pole, metody, atp.)
- sémantická (rozsahy, pre- a postconditions, invarianty)
- chování (průběh) – omezení kompozice, (vnitřní chování)

2.1.4 Komponentové modely a frameworky

V této kapitole je popsán rozdíl mezi komponentovými frameworky a modely.

Komponentový model

Komponentový model specifikuje, co je to komponenta, jak probíhá komunikace mezi komponentami, jak se komponenty skládají, jaké jsou povolené typy komponent, požadované služby, atd. [1]

Komponentový model umožňuje komponentám skrze tato pravidla vzájemně komunikovat a poskytovat si služby.

Komponentový model nastavuje následující standardy a konvence [2]:

- *Komponentové typy* - Komponentový model vyžaduje, aby komponenty implementovaly jedno, nebo více rozhraní. Komponenty potom mohou definovat jeden, nebo více komponentových typů. Různé komponentové typy mohou plnit různé role v systému a podílet se na různých schématech interakce.
- *Schémata interakce* - Určují jaké jsou použity komunikační protokoly a jak je dosaženo požadované kvality takových služeb jako např. zabezpečení nebo transakce. Mohou popisovat, jak spolu komponenty komunikují nebo jak komunikují s frameworky. Schéma interakce může být společné pro všechny komponentové typy nebo unikátní pro určitý komponentový typ.

- *Navázání zdrojů* - Proces skládání komponent znamená navazování komponent na jeden nebo více zdrojů. Zdroj je služba poskytovaná frameworkem nebo jinou komponentou v daném frameworku. Komponentový model, popisuje které zdroje jsou dostupné komponentám a jak a kdy se na ně komponenty navazují.

Framework

Komponentový framework je implementací komponentového modelu. Jeden framework může implementovat více modelů než jen jeden. Komponentový framework poskytuje množství služeb (runtime services), které obvykle zařizují běh a komunikaci komponent, případně další podpůrné služby. V mnoha ohledech jsou komponentové frameworky podobné operačním systémům, ačkoliv pracují na mnohem vyšší úrovni abstrakce. [1]

OSGi

OSGi technologie je seznam specikifací, které definují dynamický komponentový systém pro Javu. Tyto specifikace umožňují využití vývojového modelu, kde jsou aplikace dynamicky skládány z více komponent. OSGi framework umožňuje komponentám skrýt jejich implementaci před ostatními komponentami a komunikovat s nimi pomocí služeb. [4]

EJB3

EJB (Enterprise JavaBeans) je standardní komponentní architektura, sloužící pro realizaci aplikační vrstvy informačního systému. EJB komponenty jsou objekty implementované vývojářem, které zajišťují vlastní aplikační logiku systému. Komponenty EJB mají své uplatnění zejména ve tří a vícevrstvých distribuovaných aplikacích. Jedná se o součást platformy Java EE (Java Enterprise Edition). [6]

SOFA2

SOFA2 je komponentový systém využívající hierarchické uspořádání komponent. Je to přímý následník SOFA komponentového modelu, ze kterého převzal základní komponentový model a vylepšil ho. [7]

2.1.5 Kompozice

Kompozice je výraz popisující, jakým způsobem jsou systémy skládány v komponentově orientovaném programování.

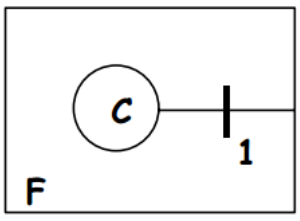
Kompoziční formy

Následující text je přeložen z [2]

Existují tři hlavní způsoby kompozice

- **Komponenta-Komponenta** - Kompozice, která umožňuje komunikaci mezi komponentami. Tyto interakce umožňují aplikační funkcionalitu a jsou klasifikovány na úrovni aplikačních kontraktů.
- **Framework-Komponenta** - Kompozice, která umožňuje komunikaci mezi frameworkem a jeho komponentami. Tyto interakce umožňují frameworku starat se o komponentové zdroje a jsou klasifikovány na úrovni systémových kontraktů.
- **Framework-Framework** - Kompozice, která umožňuje komunikaci mezi frameworky. Tyto kompozice umožňují komunikaci komponent v různých frameworcích a jsou klasifikovány jako mezioperační vazby.

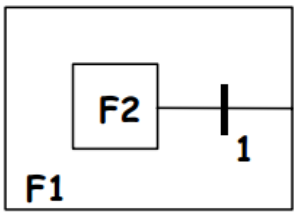
Nasazení komponenty



Obrázek 2.2: Nasazení komponenty

Komponenty musí být nasazeny do frameworku před tím, než mohou být složeny nebo spuštěny. Nasazovací kontrakt popisuje rozhraní, které komponenta musí implementovat, aby framework mohl pracovat s jejími zdroji.

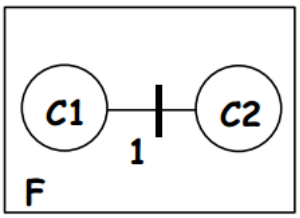
Nasazení frameworku



Obrázek 2.3: Nasazení frameworku

Framework může být nasazen do jiných frameworků. EJB s tímto částečně pracuje při nasazení EJB kontejnerů do EJB serverů. Kontrakt této vazby je analogický s kontraktem nasazení komponenty.

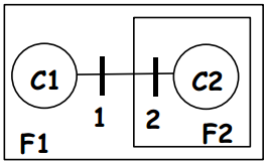
Jednoduchá kompozice



Obrázek 2.4: Jednoduchá kompozice

Komponenty nasazené ve stejném frameworku mohou být složeny. Kompoziční kontrakt vyjadřuje komponenty a specifickou funčnost aplikace. Interakční mechanismy pro podporu tohoto kontraktu jsou poskytovány frameworkem.

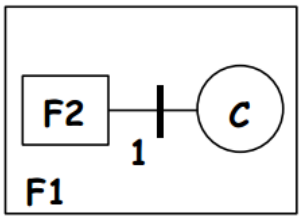
Kompozice mezi frameworky



Obrázek 2.5: Kompozice mezi frameworky

Podpora složených frameworků vyžaduje podporu skládání komponent mezi těmito frameworky, jak hierarchicky uspořádanými (viz. obrázek 2.6), tak frameworky na stejné úrovni.

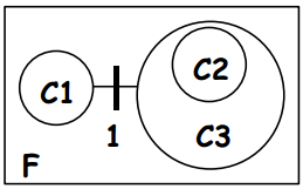
Rozšíření frameworku



Obrázek 2.6: Rozšíření frameworku

S frameworky může být zacházeno jako s komponentami, tzn. mohou být skládány s jinými komponentami. Tento typ kompozice často dovoluje parametrizaci chování frameworku pomocí plug-inů. Standardní kontrakty pro rozšíření služeb jsou stále více běžné v komerčních frameworcích.

Složení vnořených komponent



Obrázek 2.7: Složení komponent

Komponentový systém je shromáždění jedné nebo více komponent. Schopnost předvídat vlastnosti těchto komponent napovídá existenci podobné schopnosti pro vnořené komponenty. Kontrakt (1) je použit na složení C1 a C3, která obsahuje jednu nebo více komponent. Otázka, která vyvstává, je, zdali je C2 viditelné mimo C3 a zdali je samostatně nasazena.

2.2 Off-Screen techniky

V mnoha aplikacích jako 3D editory, editory obrázků a videí, aplikace poskytující různé náhledy na mapy či vizualizace softwarových komponent je poskytován omezený viewport, protože takovéto aplikace běžně pracují s velmi rozsáhlou pracovní plochou, kterou není možné zobrazit celou. Kvůli omezenému viewportu jsou uživatelé těchto aplikací nuceni pohybovat se po pracovní ploše pomocí nástrojů, které jsou umístěné mimo pracovní plochu.

Off-screen techniky dovolují uživateli se pohybovat a orientovat i po pracovní ploše, která je mimo obraz. Těchto technik existuje velké množství, ale ne všechny jsou vhodné pro všechny typy aplikací a doménových problematik. [3]

Některé z těchto technik zde budou popsány jako příklad technik vhodných pro vizualizaci komponentových systémů. Následující text je volným překladem článku [8].

2.2.1 Snižování vizuálního šumu ve velkých komponentových diagramech

Tato technika se zabývá snižováním vizuálního „šumu“, tj. tak velkého množství vizuálních informací, jako např. hran a komponent na stejném místě, že je obtížné cokoliv vyčíst. Komponenty s velkým počtem hran vytvářejí v grafu tento šum a je proto vhodné, pro zpřehlednění grafu, tyto komponenty nějakým způsobem omezit.

Tento problém je řešen tak, že jsou komponenty s velkým počtem hran přesunuty (dále odebrány) z hlavního zobrazovacího pole do pole vedlejšího, tzv. SeCo (Separated Components) panel, které se nachází vedle pole hlavního. Po odebrání komponenty jsou hrany mezi touto a ostatními komponentami odebrány také a namísto nich je vedle zbylých komponent zobrazen symbol, reprezentující odebranou komponentu. Stejně tomu je v SeCo panelu, kde je zobrazen počet komponent, se kterými je odebraná komponenta spojena. Tento způsob snižuje počet hran v grafu bez ztráty informace o propojení komponent.

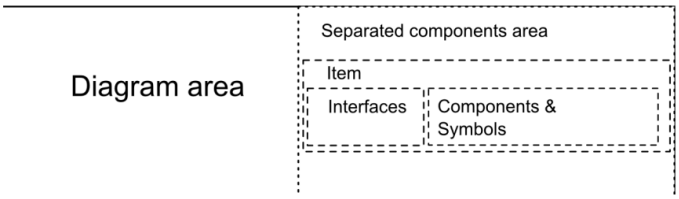
Je možné automatické i manuální vybírání komponent pro odebrání z hlavního pole. V případě automatického vybírání se odeberou komponenty s

počtem hran přesahujícím jistou mez. V manuálním vybírání uživatel přesune komponenty do SeCo panelu pomocí drag-and-drop.

Separated Components panel

Separated Components panel, nebo SeCo panel, je část aplikace, kde se ukládají komponenty odebrané z grafu. Je vhodné použít levou nebo pravou stranu aplikace, jelikož obrazovky bývají širší než vyšší a tudíž přidáním tohoto panelu nedojde k deformaci zbytku zobrazovacího pole.

Na obrázku je zobrazeno přibližné rozložení panelů v aplikaci.



Obrázek 2.8: Rozložení panelů v aplikaci

Položky

SeCo panel je vlastně seznam položek. Každá položka obsahuje jednu nebo více komponent, rozhraní a odpovídající symbol (viz. Symboly a delegáti). Komponenty uložené v SeCo panelu mají vztahy k ostatním komponentám zobrazeny na okraji mezi diagramem a SeCo panelem. Existují položky dvojího druhu.

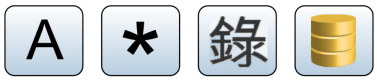
První druh obsahuje pouze jednu komponentu, rozhraní jsou připojena přímo na komponentu a symbol je zobrazen vedle ní.

Druhý druh obsahuje více komponent, které tak tvoří skupinu. V tomto případě jsou rozhraní napojena na symbol a komponenty jsou zobrazeny za symbolem.

Symboly a delegáti

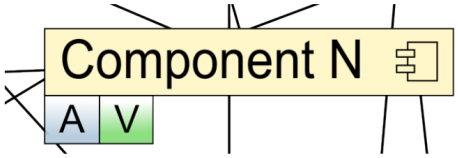
Smysl symbolů je vytvořit jednoduchý a snadno rozpoznatelný unikátní klíč, který bude přiřazen každé položce v SeCo panelu. Tyto symboly je možné použít u komponent v diagramu jako delegáty komponent, které jsou s danou komponentou propojeny, ale jsou uloženy v SeCo panelu.

Symboly by měly být dostatečně malé, aby nezabíraly mnoho místa kdekoliv budou použity. Uživatel by měl mít možnost vybrat si symbol podle jeho preference. Symbol může být písmeno, ale i jakýkoliv jiný symbol nebo ikona.



Obrázek 2.9: Příklady možných symbolů [8]

Pro udržení informace o spojení komponent po odebrání hran, jsou použity tzv. delegáti. Ty reprezentují spojení mezi komponentou, vedle které jsou zobrazeny a komponentou odebranou z diagramu. V diagramu jsou zobrazeny jako malé čtverečky vedle komponenty, obsahující daný symbol.



Obrázek 2.10: Ukázka delegátů vedle komponenty v diagramu [8]

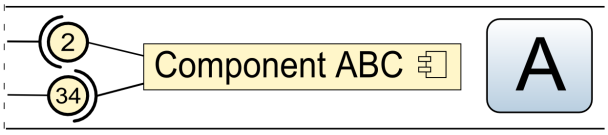
Zobrazení delegátů v diagramu může být vypnuto nebo zapnuto kliknutím na symbol v SeCo panelu. Položka indikuje viditelnost delegátů v diagramu změnou barvy pozadí.

Sloučení rozhraní

Pro každou položku v SeCo panelu jsou rozhraní sloučena do dvou skupin. Všechna rozhraní, skrze která poskytuje své vlastnosti (zobrazena jako „kulička“), a všechna rozhraní, která vyžaduje (zobrazena jako „miska“). Číslo v těchto objektech je počet komponent propojených daným způsobem s danou komponentou. Tento způsob pomáhá zmenšit prostor, který tyto

komponenty zaplňují.

Sloučená rozhraní nejsou standardně propojena hranami s ostatními komponentami v diagramu. Tato propojení se objeví pouze při manipulaci s jednou ze stran spojení.



Obrázek 2.11: Objekty sloučených rozhraní vedle komponenty [8]

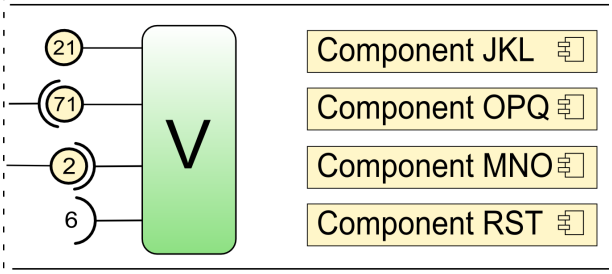
Jsou dva způsoby interakce se sloučenými rozhraními. První je jednoduché zobrazení všech hran a zvýraznění propojených komponent. Druhý je zobrazení detailů o všech rozhraních jako jména, propojení a zvýraznění komponent. Tyto akce lze vyvolat kliknutím na objekt sloučených rozhraní.

Komponentové skupiny

Skupina je jednou z položek, které je možno vytvořit v SeCo panelu. Skupiny jsou vhodné, když množina komponent tvoří určitou funkcionalitu, která je využívána velkým počtem ostatních komponent.

Všechny komponenty z takové skupiny jsou pak v diagramu zastoupeny jedním delegátem. Tímto se šetří místo v diagramu a pomáhá se vytvořit sémantické skupiny komponent, což přispívá k lepšímu porozumění celého systému. Stav, kdy jsou zobrazeni delegáti, je symbolizován u skupiny stejným způsobem, jako tomu bylo u jednotlivých komponent, aby byl dodržen nastavený standard.

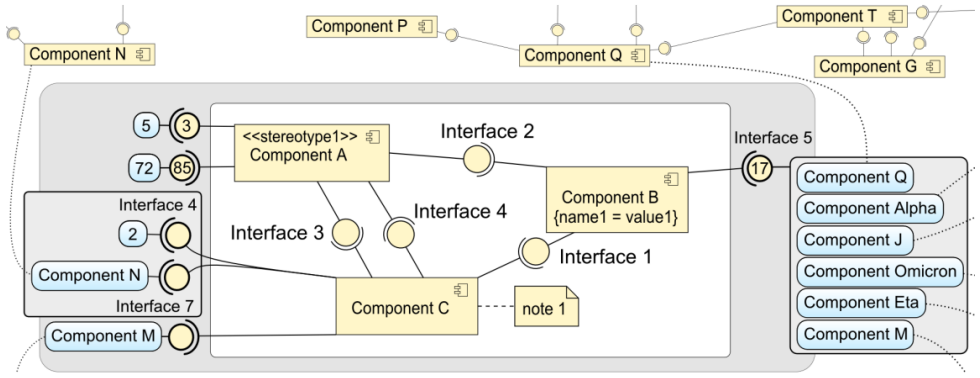
Na obrázku je skupina čtyř komponent spolu s jejich symbolem a počtem rozhraní. První dvojice udává vyžadovaná a poskytovaná rozhraní, která nejsou navázána na žádné komponenty, a druhá dvojce udává počet poskytovaných a vyžadovaných rozhraní, která jsou navázána.



Obrázek 2.12: Ukázka skupiny čtyř komponent [8]

2.2.2 Viewport pro komponentové diagramy

Tato technika ukazuje celý diagram, nepřiblížený diagram s komponentami zobrazenými bez detailů. Kromě toho ukazuje vybrané komponenty se všemi detaily a propojeními v tzv. viewportu. Ve viewportu je interaktivní okno, kde jsou vybrané komponenty zobrazeny i s hranami. Na hranách tohoto okna jsou objekty sloučeného rozhraní (počet rozhraní daného typu), která jsou ve viewportu napojena na kopii komponenty, která je na obrázku znázorněna jako obdélník s kulatými rohy, kde každý tento obdélník reprezentuje jednu, nebo více komponent. Čísla v objektech sloučených rozhraní a kopiích komponent znázorňují počet prvků sloučených v daném objektu.



Obrázek 2.13: Návrh viewportu pro komponentové diagramy [9]

Jedním z klíčových faktorů je interaktivnost viewportu, která zahrnuje manipulaci se sloučenými objekty, jako přidávání a ubírání komponent nebo upravení layoutu komponent v hlavním okně. Tento přístup by měl zjednodušit prozkoumávání velkých diagramů zobrazením kontexu několika vybraných

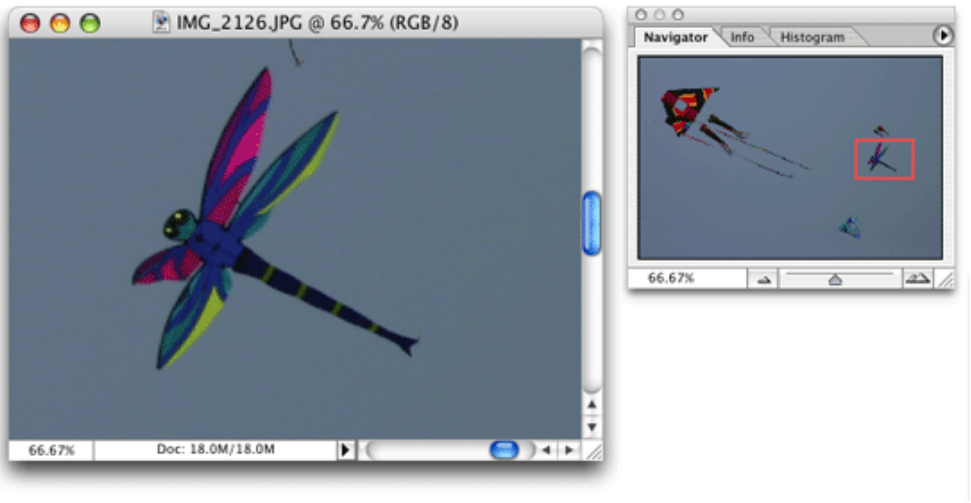
komponent ve viewportu. Kopie komponent ve viewportu by měla snížit nutnost prohledávat zbytek diagramu. [9]

2.2.3 Overview plus detail

Hlavní myšlenkou této techniky je zobrazit dvě okna, z nichž jedno obsahuje pohled na celý diagram (overview) a druhé detail diagramu. Tato technika umožňuje pracovat s každým oknem zvlášť. Akce provedené v jednom z oken se obvykle okamžitě promítnou i v okně druhém.

Overview je celkový pohled na zmenšenou pracovní plochu, ve které se nachází tvz. viewfinder, což je vyznačená část plochy overview. Tato vyznačená část je potom viditelná zblízka v druhém okně detailu. Velikost viewfinderu je libovolně nastavitelná a lze ho jakkoliv posouvat.

Na obrázku je ukázka overview a detailu. Vlevo se nachází detail s přiblíženou pracovní plochou a vpravo je overview, kde je vidět celá zmenšená pracovní plocha a červeně vyznačený viewfinder. [10]



Obrázek 2.14: Ukázka overview a detailu [10]

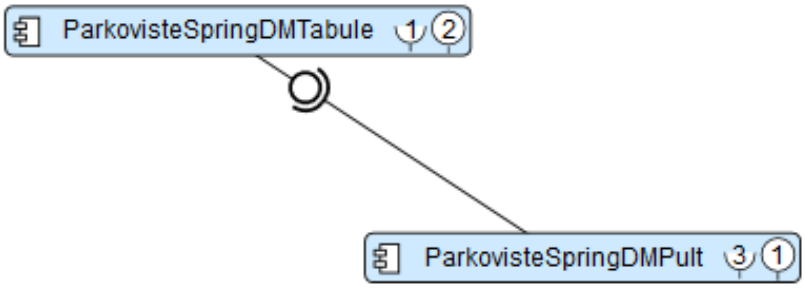
3 Existující aplikace

Původní aplikace byla vyvinuta jako diplomová práce Jindrou Pavlíkovou v roce 2012. Hlavním úkolem bylo demonstrovat vizualizaci rozsáhlých komponentových grafů a techniky používané k práci s těmito grafy.

Funkcionalita

Výsledná aplikace umí nahrát datový model komponentového grafu (např. OSGi framework) a zobrazit komponenty spolu s jejich vzájemnými vztahy do grafu. S těmito komponentami lze potom dále pracovat.

Vztahy mezi komponentami jsou znázorněny hranami označenými symbolem, který znázorňuje v jakém vztahu komponenty jsou. Je-li blíže ke komponentě kolečko, znamená to, že poskytuje své funkce komponentě na druhém konci hrany. Je-li blíže komponenty miska (nebo půlkruh), znamená to, že vyžaduje funkce komponenty na druhém konci hrany.



Obrázek 3.1: Ukázka spojení komponent

V aplikaci jsou dvě hlavní zobrazovací pole. V prvním je zobrazen hlavní diagram s komponentami a jejich vzájemnými vztahy, kde je možno s komponentami manipulovat. Druhé pole je tzv. SeCo panel (Separated Components), kam lze „odložit“ komponenty, se kterými uživatel v dané chvíli nepotřebuje pracovat.

Dalším z důležitých panelů je horní lišta s tlačítky. Tlačítka v tomto panelu ovládají většinu z funkcí na úpravu diagramu a práci s programem, např. zvětšování a zmenšování grafu, vyhledávání nebo návrat k nahrávání

komponent.

Aplikace umí zobrazit i velmi rozsáhlé grafy, které potom umí upravit tak, aby byly přehledné, např. přesunutím komponent s největším počtem hran do SeCo panelu. Tímto dojde k výraznému zpřehlednění grafu a k jednodušší práci se zbytkem komponent.

Dále je možno vyhledávat komponenty a výsledky vyhledávání zobrazit přímo v diagramu nebo seskupit více komponent do jednoho objektu, který tyto komponenty bude reprezentovat. Klikáním na komponentu v SeCo panelu je možno zvýraznit v diagramu komponenty propojené s označenou komponentou.

4 Přidané funkce

V této kapitole jsou popsány a předvedeny nově přidané funkce.

4.1 Zvýraznění propojených komponent

Jde o zvýraznění komponent, které jsou propojeny s komponentou, kterou uživatel označí. Je nutno zvýraznit vybranou komponentu a nějakým způsobem označit propojené komponenty. Rozdílně by měly být označeny komponenty, které poskytují své vlastnosti označené komponentě a ty které vyžadují vlastnosti označené komponenty. To pomůže při orientaci ve složitějších diagramech, kde je mnoho hran a mohlo by být složité odhadnout, které komponenty jsou jak propojeny.

4.1.1 Návrh

V aplikaci jsou dvě místa, kde se nachází komponenty. Zaprvé v diagramu, kde jsou znázorněny hrany mezi komponentami, a zadruhé v panelu napravo od diagramu (SeCo panel), kde je zobrazen pouze počet komponent propojených s danou komponentou. Tyto dvě možnosti je nutno zpracovat odděleně.

Zvýraznění v panelu SeCo

Při poklepání na komponentu v panelu SeCo se zvýrazní propojené komponenty jak v grafu, tak v samotném panelu. Zvýrazní se také číslo vedle označené komponenty, značící počet propojených komponent, aby bylo možno snadno zjistit, která komponenta je označena.

Zvýraznění v grafu

Při poklepání na komponentu v grafu, se zvýrazní propojené komponenty a hrany v grafu i komponenty v panelu SeCo. Označená komponenta se zvýrazní červeným okrajem.

4.1.2 Implementace

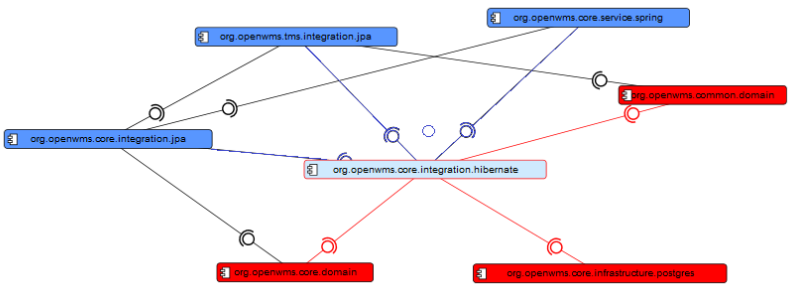
Toto označení musí fungovat zároveň s již existujícím označením hran (při poklepání na hranu se označí hrana a obě spojené komponenty). To znamená, že je nutno zrušit toto označení i při označení komponenty a naopak. V programu již existuje metoda, která se o toto stará a kterou jsem využil a doplnil o odznačování komponent.

Komponenta by se měla označit pouze při poklepání, ne při posunutí. Jelikož se metoda, zjišťující označenou komponentu, nachází v metodě starající se o posouvání komponenty, bylo nutné zajistit polohu komponenty při stisknutí tlačítka a uvolnění tlačítka a tyto dvě hodnoty porovnat. Pokud jsou stejné, znamená to, že se komponenta neposunula a tudíž ji chce uživatel označit.

Označení musí být odlišné od označení komponent z panelu SeCo, aby nedocházelo k sloučení označení. Vybral jsem proto jiné barvy jak pro požadované, tak pro poskytované komponenty.

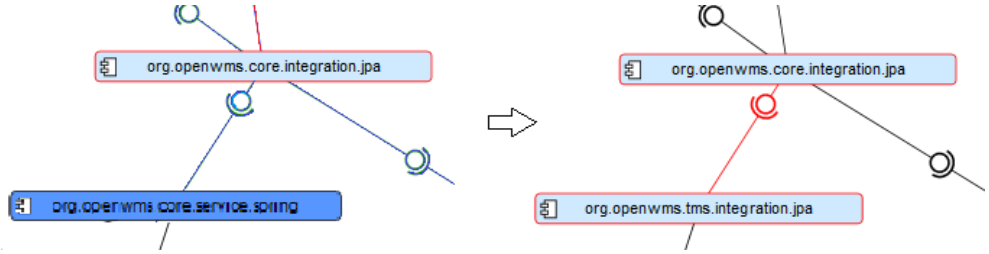
4.1.3 Představení a ověření

Na obrázku 4.1 je označená komponenta s červeným okrajem spolu s propojenými komponentami. Komponenty a příslušné hrany kterým komponenta poskytuje své vlastnosti, jsou označeny modře a ty jejichž vlastnosti komponenta využívá, jsou označeny červeně.



Obrázek 4.1: Ukázka označení propojených komponent

Na obrázku 4.2 je znázorněna změna označení komponenty na hranu. Nejprve je označena komponenta a po kliknutí na hranu se tato označí, čímž se zruší původní označení.



Obrázek 4.2: Ukázka změny označení

4.2 Vylepšení vizualizace označení

Jedna z dalších funkcí, které výrazně zlepší schopnost uživatele orientovat se v grafu, je odznačení komponent, které nejsou spojeny s označenou komponentou. Takové komponenty jsou pro uživatele v danou chvíli nedůležité a je tedy výhodné je nějakým způsobem odlišit, například zešednutím, nebo přidáním průhlednosti.

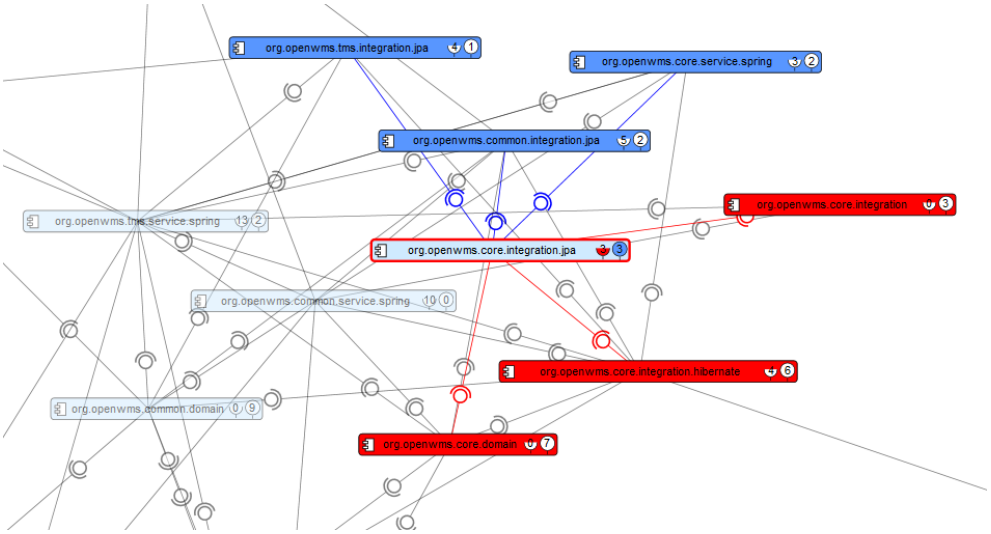
4.2.1 Návrh a implementace

V případě této aplikace je vhodné a navíc snazší implementovat průhlednost těchto komponent. Průhlednost je vhodné implementovat i pro hrany mezi těmito komponentami, jelikož tyto hrany velmi vyniknou po tom, co jsou zprůhledněny komponenty.

Bez označení jsou všechny komponenty zobrazeny, standardně. Při označení komponenty se vše nejprve zprůhlední a poté se provede požadované označení, což znamená, že všechny neoznačené komponenty a hrany jsou průhledné. Opačný postup je použit při odznačení komponenty.

4.2.2 Představení a ověření

Implementovaná funkce funguje podle očekávání. Přehlednost diagramu po označení komponenty, je mnohem vyšší.



Obrázek 4.3: Ukázka zprůhlednění diagramu po označení jedné z komponent

4.3 Počet propojených komponent

Jediný způsob, kterým bylo z grafu možno zjistit počet propojených komponent, bylo spočítat všechny hrany. Přidání objektů s počtem propojených komponent ke každé komponentě v grafu jej značně zpřehlední.

4.3.1 Návrh

Ke každé komponentě v grafu budou přidány dva objekty s čísly, jeden reprezentující počet poskytovaných a druhý počet požadovaných komponent. Jejich tvar bude odpovídat již zavedenému standardu v panelu SeCo (viz. obr. 4.4).

Obrázek 4.4: Ukázka komponent v SeCo panelu

Na každý z těchto objektů bude možno kliknout a označit tak pouze požadované nebo poskytované komponenty. Toto označování se bude rušit spolu s označováním celých komponent a hran. Tyto objekty se zvýrazní příslušnou barvou (červenou pro požadované a modrou pro poskytované) k jednoduššímu rozpoznání označené komponenty.

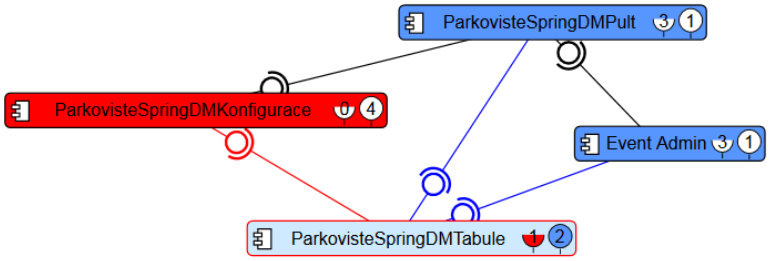
4.3.2 Implementace

Ke každé komponentě v grafu byly přidány dva nezávislé objekty s požadovaným tvarem a s počtem propojených komponent. Tyto nezávislé objekty se pohybují spolu s posouvající se komponentou a budí tak dojem, že jsou její součástí. Počet propojených komponent není ve vlastnostech komponent uložen, proto jej bylo nutno spočítat průchodem všech hran pro každou komponentu. Tyto dva objekty byly navíc moc velké pro současnou šířku komponenty, proto bylo třeba komponentu zvětšit.

Pro funkčnost těchto objektů byly vytvořeny dvě metody, jedna pro požadované a druhá pro poskytované komponenty. Tato metoda obsahuje volání metody pro odznačení všech objektů, aby fungovala zároveň s ostatními typy označení.

4.3.3 Představení a ověření

Na obrázku 4.4 je označená komponenta spolu s objekty s počty propojených komponent, které jsou příslušně zvýrazněny. Půlkruh (miska) vlevo označuje počet požadovaných komponent a kruh vpravo označuje počet komponent, kterým jsou poskytovány vlastnosti označené komponenty.



Obrázek 4.5: Ukázka objektů s počty propojených komponent

4.4 Odebrání celé skupiny

Skupina je spojení více komponent v jednom objektu a její odebrání znamená rozložení této skupiny zpět na všechny komponenty, které obsahovala. V aplikaci nebylo implementováno odebrání celé skupiny, což značně zpomalovalo práci se skupinami.

Skupiny se nacházejí v SeCo (Separated Components) panelu, napravo od hlavního zobrazovacího pole. Využívají se k přehlednějšímu ukládání právě nepoužívaných komponent nebo ke kategorizaci komponent s podobnými vlastnostmi.

4.4.1 Návrh

Je několik způsobů řešení, např. kliknutí pravým tlačítkem na skupinu, což by vyvolalo panel s možnostmi. Nejjednodušší řešení je však přidat ke každé skupině vhodně označené tlačítko, které skupinu odstraní. Nejlepší místo pro toto tlačítko je přímo v objektu skupiny (viz. obrázek v Představení a ověření).

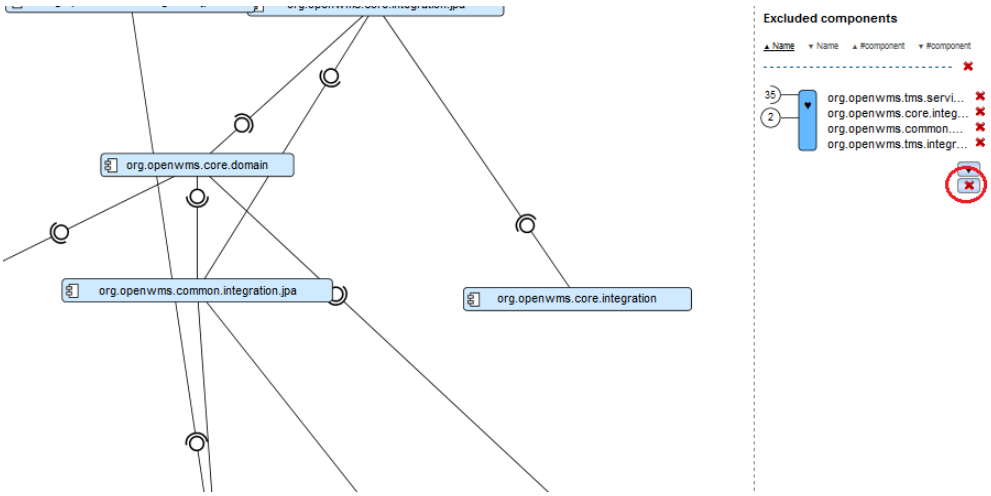
4.4.2 Implementace

U každé komponenty obsažené ve skupině již existuje tlačítko, které danou komponentu odstraní ze skupiny. Funkce nového tlačítka pouze projde

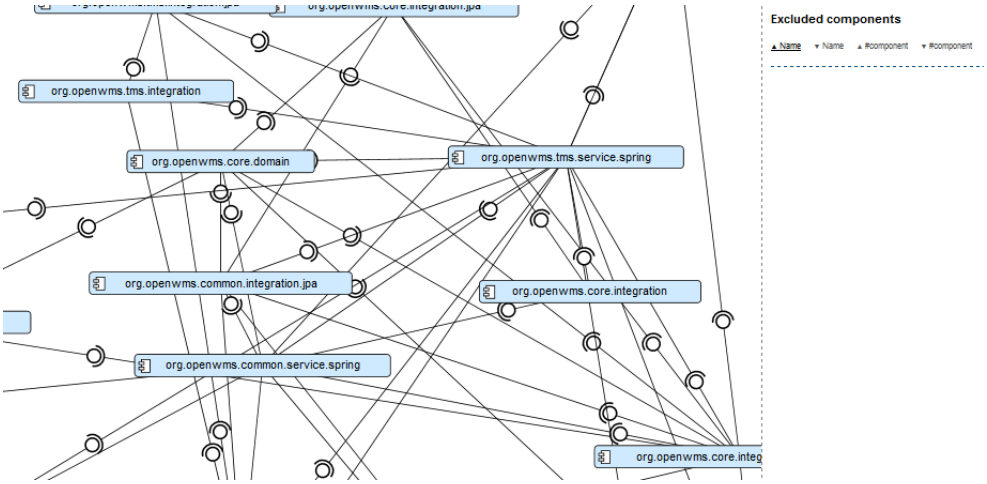
všechna tlačítka na odebrání u dané komponenty a spustí jejich funkci, čímž se odstraní všechny komponenty ve skupině.

4.4.3 Představení a ověření

Zakroužkované tlačítko na obrázku 4.6 je tlačítko odstranění celé skupiny ze SeCo panelu a přsunustí zpět do diagramu.



Obrázek 4.6: Stav diagramu před odebráním skupiny



Obrázek 4.7: Stav diagramu po odebrání skupiny

4.5 Úložiště nepropojených komponent

Nepropojené komponenty, tj. komponenty, které neposkytují vlastnosti jiným komponentám ani nevyžadují jejich vlastnosti, jsou pro vizualizaci propojení komponent nevýznamné. Vyvstává otázka, co s takovými komponentami udělat. Jedním z vhodných řešení je vytvořit úložiště pro tyto komponenty mimo hlavní zobrazovací pole. Do tohoto úložiště se nepropojené komponenty schovají a nebudou tak zobrazovány v hlavním panelu.

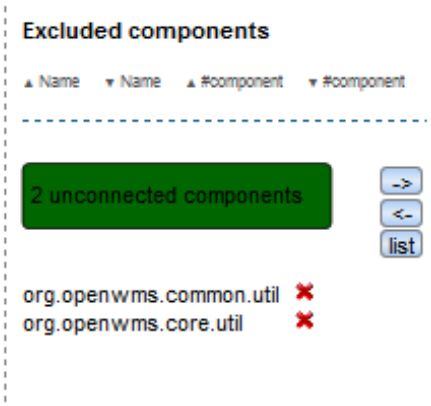
4.5.1 Návrh a implementace

V panelu odebraných komponent bude vytvořen objekt, reprezentující nepropojené komponenty. Bude obsahovat jejich počet a po rozkliknutí seznamu jejich výčet. Bude možno jedním kliknutím všechny komponenty bud'to přesunout z úložiště do diagramu nebo je v tomto objektu všechny schovat. Mezi diagramem a tímto úložným prostorem bude možno přesouvat i jednotlivé komponenty.

V pravém panelu byl tedy vytvořen objekt tvaru obyčejné komponenty uschované v tomto panelu s barvou odlišnou od ostatních komponent pro snadné rozeznání, který složí jako úložiště nepropojených komponent. Vedle objektu jsou umístěna tři tlačítka na práci s uloženými komponentami. Dvě z nich slouží k posunu všech nepropojených komponent bud' do nebo ven z úložného prostoru. Třetí tlačítko zobrazí seznam všech komponent v úložném prostoru. Vedle každé komponenty v seznamu je tlačítko, které umožňuje odebrat tuto komponentu z úložného prostoru samostatně.

4.5.2 Představení a ověření

Na obrázku je zobrazeno úložiště komponent s několika komponentami a otevřeným seznamem.



Obrázek 4.8: Úložný prostor nepropojených komponent se dvěma komponentami

Tlačítko list slouží k zobrazení a schování seznamu komponent, tlačítko se šipkou doleva slouží k přesunutí všech komponent z úložiště do diagramu a tlačítko se šipkou doprava slouží k přesunutí všech nepropojených komponent z diagramu do úložiště.

4.6 Layout komponent

Komponenty v hlavním zobrazovacím poli byly původně rozmístěny náhodně. To znamenalo velkou nepřehlednost zobrazeného grafu, jelikož komponenty spolu spojené mohly být na opačných koncích diagramu a naopak komponenty, které spolu nemají nic společného, na stejném místě.

Je proto vhodné použít algoritmus, který rozmístí komponenty pokud možno tak, aby byly v co nejméně chaotickém uspořádání. Existuje mnoho algoritmů, které se touto problematikou zabývají. Pro tento případ jsem vybral tzv. *Force-Directed Graph Layout* (Layout řízený silou).

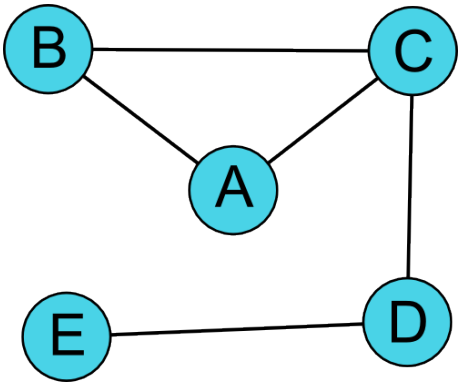
Force-Directed Graph Layout

Tento layout využívá k určení pozice komponent síly působící mezi těmito komponentami. V tomto systému se uvažují dva typy sil. Přitažlivé síly založené na Hookovu zákonu jsou použity k posouvání komponent na opačných koncích hran k sobě a zároveň odpudivé síly na základě nabitých částic (komponent) založených na Coulombovu zákonu jsou použity k odpuzování všech komponent od sebe.

Je nutno aby měly komponenty na začátku algoritmu přiřazenou polohu, např. náhodnou, jak je tomu v této aplikaci. Z těchto poloh se potom počítají síly mezi všemi komponentami.

Obrázek 4.9: Ukázka rozložení uzlů po použití Force-Directed algoritmu

Na několika dalších obrázcích je představena funkce Force-Directed algoritmu na jednoduchém grafu.

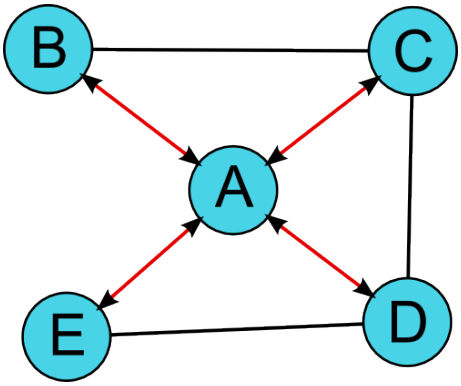


Obrázek 4.10: Počáteční stav ukázkového grafu

Odpudivá síla

Odpudivá síla je počítána pro každou z komponent v závislosti na všech ostatních. Je to proto, aby komponenty, které spolu nejsou spojeny, neskončily na stejném, nebo velmi podobném místě.

Na obrázku jsou červeně znázorněny odpudivé síly působící na uzel A.



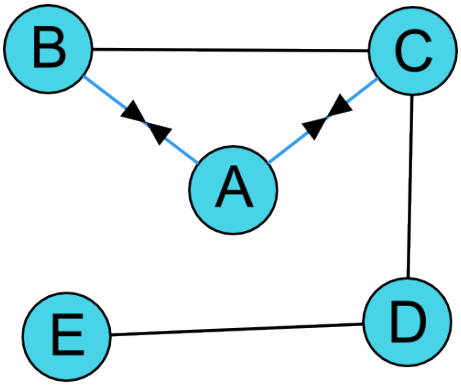
Obrázek 4.11: Odpudivé síly působící na uzel A

Přitažlivá síla

Přitažlivá síla je počítána pro každou komponentu v závislosti pouze na těch komponentách, které jsou s ní spojeny. Počítání přitažlivých sil tímto způsobem vede k tomu, že uzly, které mají málo hran (např. jednu, nebo žádnou), jsou odstrčeny mimo hlavní shluk uzlů, jelikož jsou odpuzovány ode všech uzlů a přitahovány pouze jedním nebo žádným uzlem. Tento jev funguje i opačně, tzn. při velkém počtu sousedů je uzel blízko středu shluku

uzlů.

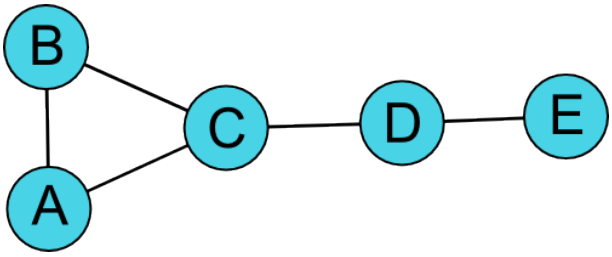
Na obrázku jsou modře znázorněny přitažlivé síly působící na uzel A.



Obrázek 4.12: Přitažlivé síly působící na uzel A

Po spočtení všech sil jsou všechny komponenty současně posunuty do nově spočtené pozice. Tento postup se opakuje v několika iteracích, které převážně závisí na velikosti grafu a rozmístění uzlů. Při dostatečném počtu iterací by výsledkem měl být přehlednější diagram.

Na obrázku je ukázkový graf po několika iteracích popsaného algoritmu.



Obrázek 4.13: Graf po aplikaci Force-Directed algoritmu

4.6.1 Návrh a implementace

Algoritmus použitý v aplikaci bude fungovat tak, jak je popsán v předešlé sekci. V aplikaci je již navržena struktura, ve které jsou uchovávány vlastnosti jednotlivých komponent. V tomto případě je užitečná hlavně pozice komponenty. Bylo vytvořeno pole, které je v každé iteraci algoritmu nulováno, a do kterého se při každé iteraci ukládají síly použité na posun komponent.

Očekávání je, že komponenty, které mají relativně mnoho sousedů, budou situovány uprostřed, komponenty s méně sousedy budou po okrajích a komponenty s jedním nebo dvěma sousedy budou mimo shluk.

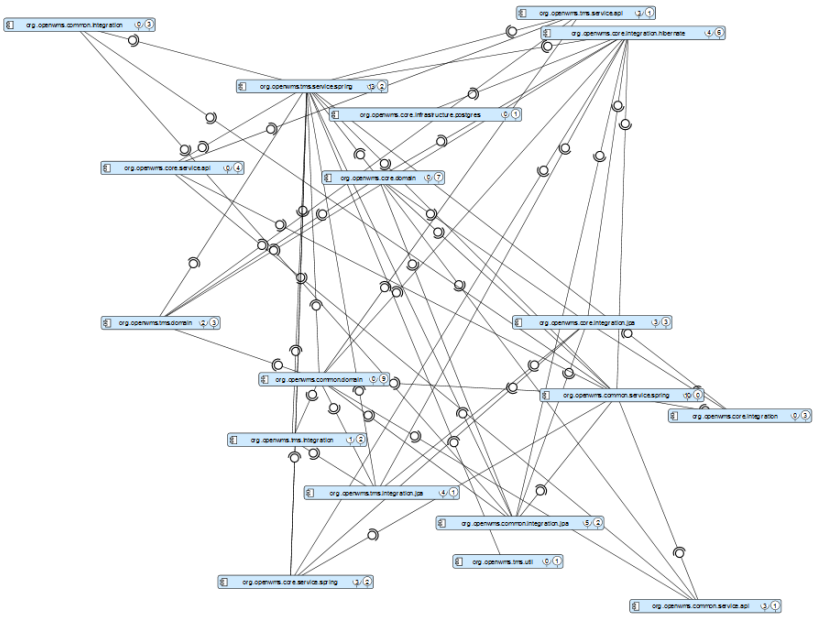
Komponenty se při kliknutí na tlačítko „Start Visualization“ při nahrávání komponent upraví tímto algoritmem a poté se zobrazí v již optimalizovaných pozicích.

4.6.2 Představení a ověření

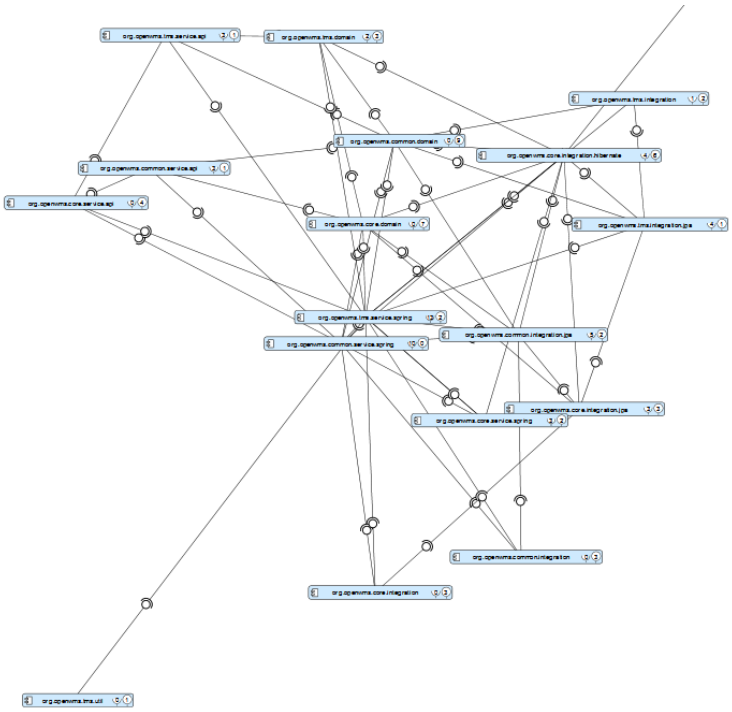
Čas nahrávání diagramu se podle očekávání zvýšil, jelikož Force-Directed algoritmus je velmi složitý a časově náročný. Výsledky jsou však dobré a diagram je o poznání přehlednější.

Diagram s náhodným rozložením je o poznání chaotičtější, než upravený diagram. Komponenty v upraveném diagramu se rozložily podle očekávání z návrhu, tj. čím více sousedů tím je komponenta blíže středu shluku.

Na následujících obrázcích je porovnání náhodného a Force-Directed rozložení na stejném diagramu.



Obrázek 4.14: Diagram s náhodně rozloženými komponentami



Obrázek 4.15: Diagram s Force-Directed rozložením

4.7 Dynamická vlastnost layoutu

Dosavadní metoda, popsaná výše, se nezabývá posunem komponent při běhu programu, pouze při vytvoření diagramu. To znamená, že pokud si uživatel upraví nahraný graf, např. odebere několik komponent z diagramu, může znovu uspořádat komponenty pouze ručně.

Funkce dynamického layoutu umožní uživateli optimalizovat pozice komponent přímo za běhu programu a to navíc v reálném čase. Uživatel může kdykoliv spustit a zastavit upravování diagramu Force-Directed layout algoritmem, který vidí pracovat v reálném čase po jednotlivých krocích.

4.7.1 Návrh a Implementace

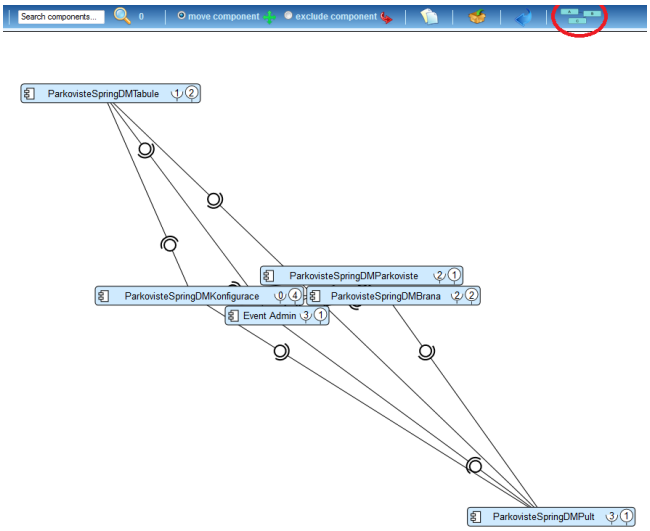
Do horního panelu bylo přidáno tlačítko na ovládání dynamického layoutu. Při kliknutí na tlačítko se spustí funkce dynamického layoutu a je zapnuta tak dlouho, dokud není tlačítko stisknuto znovu. Tlačítko změní barvu, což indikuje, že funkce dynamického layoutu je zapnuta a diagram je upravován v reálném čase.

Jediné dva rozdíly oproti algoritmu, který upravuje pozice komponent při nahrávání jsou, že uživatel může z diagramu odebrat několik komponent, které potom nesmí být uvažovány ve výpočtu pozic, a to, že je třeba po každém kroku posunout objekt komponenty, ne jenom zapsat jeho pozici, což zpomaluje výpočetní proces.

Funkce obsahující jeden krok algoritmu je po stisknutí tlačítka volána každých 10 ms a běží neustále dokud není vypnutá tlačítkem v horním panelu. To znamená, že reaguje na změny, jako např. odebrání komponenty, nebo posun komponenty myší v reálném čase.

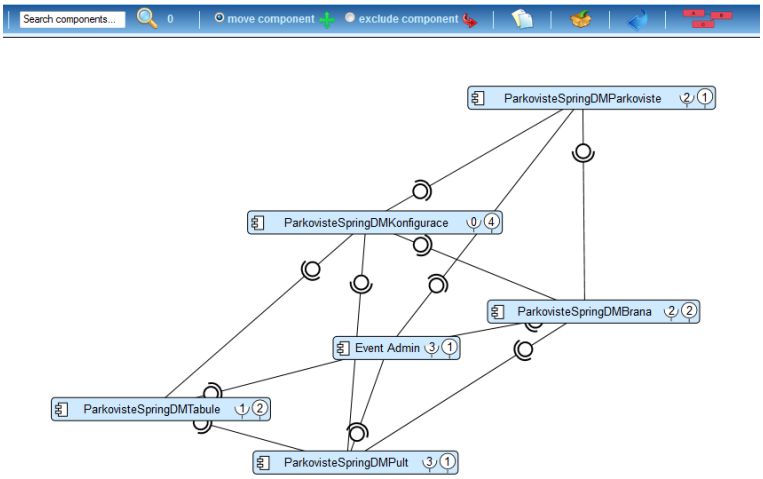
4.7.2 Představení a ověření

Na několika následujících obrázcích je demonstrována tato funkce v několika krocích.



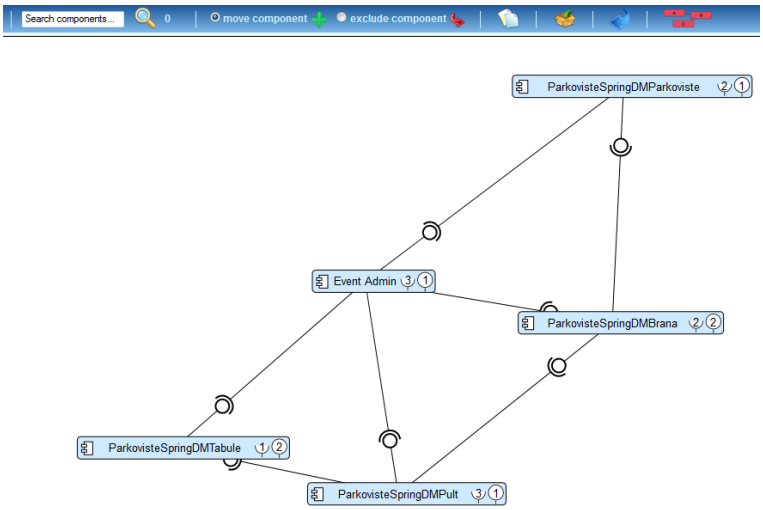
Obrázek 4.16: Neupravený diagram s vypnutým dynamickým layoutem

Zakroužkované tlačítko v obrázku nahoře je tlačítko pro spuštění layoutu. Po zapnutí dynamického layoutu se tlačítko rozsvítí červeně a diagram se začne pohybovat směrem do ustálené pozice.



Obrázek 4.17: Ustálený diagram po zapnutí dynamického layoutu

Po odebrání komponenty se diagram začne znovu pohybovat, jelikož se již nenachází v optimálním poloze. To je způsobeno odstraněním sil spojených s odebranou komponentou. Na dalším obrázku je diagram několik sekund po odebrání komponenty „ParkovisteSpringDMKonfigurace“.



Obrázek 4.18: Ustálený diagram po odebrání komponenty

5 Instalace a spuštění aplikace

V této kapitole je popsán postup pro spuštění vytvořené aplikace.

Požadavky na správný běh aplikace:

- JRE 1.6 nebo vyšší verze
- Mozilla Firefox 11.0

5.1 Instalace Apache Tomcat

Nejprve je nutno nainstalovat webový server, na kterém aplikace poběží. Aplikace je určena pro server Apache Tomcat, jehož instalační soubor se nachází na přiloženém DVD ve složce *Instalace*. Při instalaci není třeba měnit jakákoliv nastavení, je však třeba zapsat si HTTP/1.1 Connector port (standardně 8080).

5.2 Instalace apliakce CoCA-Ex

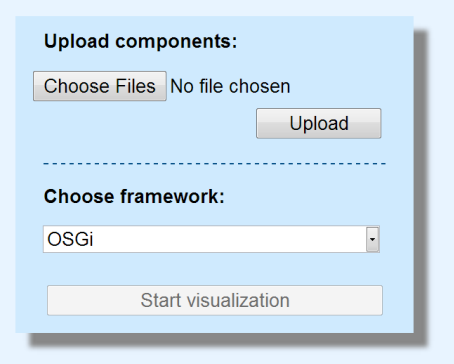
Pro instalaci aplikace stačí zkopírovat soubor „VisualizationTool.war“ z přiloženého DVD do adresáře *...\Tomcat 7.0\webapps* (defaultní adresář je „C“:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\ ).

Je třeba ještě nastavit cestu k adresáři existujícímu na disku, do kterého se budou nahrávat uložené soubory. Tato cesta je uložena v souboru *VisualizationTool/WEB-INF/configuration.properties* v adresáři *webapps* v instalaci Tomcat, jako parametr „storageLocation“. Pokud tento adresář neexistuje, je třeba spustit nebo restartovat server.

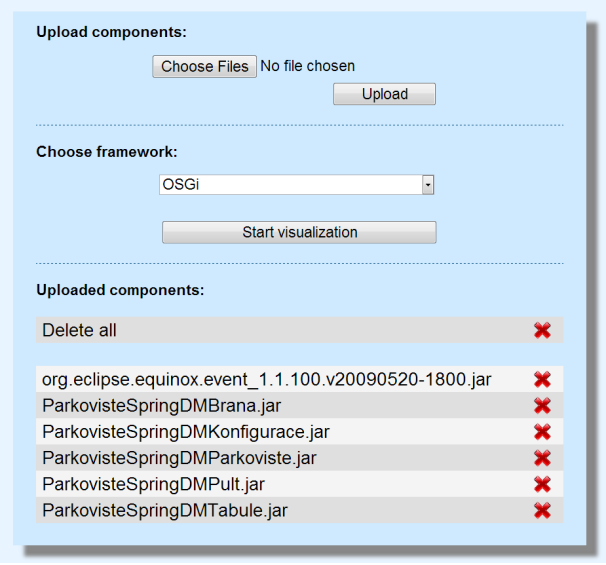
5.3 Spuštění apliakce

Ve webovém prohlížeči stačí přejít na adresu „localhost:8080/VisualizationTool/“ (podle potřeby změnit port). Je nutné aby server Tomcat byl spuštěn.

Pro vizualizaci grafu, je třeba pomocí tlačítka „procházet“ (Choose Files), vybrat testovací data z přiloženého DVD, která se nachází ve složce *Testovací data*. Tlačítkem „upload“, se data nahrají a tlačítkem „Start Visualization“, se spustí samotná vizualizace dat.



Obrázek 5.1: Spouštěcí okno aplikace



Obrázek 5.2: Spouštěcí okno s nahranými daty

6 Závěr

Cílem této práce bylo rozšířit původní aplikaci pro vizualizaci komponentových diagramů o vylepšení, která by usnadnila práci s touto aplikací.

Jedno z implementovaných vylepšení je schopnost označování propojených komponent. Po kliknutí na vybranou komponentu se zvýrazní všechny komponenty propojené s označenou komponentou. Komponenty a hrany mezi nimi se různě barevně zvýrazní podle typu spojení. Dále byly ke každé komponentě přidány objekty reprezentující počty požadovaných a poskytovaných komponent. Na tyto objekty lze kliknout a stejně jako po kliknutí na komponentu se zvýrazní komponenty propojené s označenou komponentou, tentokrát však pouze ty, které jsou buď poskytované nebo požadované (podle objektu, na který bylo kliknuto). Byla také implementována funkce, která zprůhlední všechny neoznačené komponenty. To výrazně pomáhá přehlednosti diagramu.

Další implementovaná funkce je úložiště nepropojených komponent. Nepropojené komponenty jsou ty, které nemají žádné vazby s ostatními komponentami. Tyto komponenty jsou pro vizualizaci komponentových diagramů v mnoha případech nevýznamné, proto je pro vhodné vytvořit úložiště mimo hlavní zobrazovací pole. Takové úložiště bylo vytvořeno v SeCo panelu. Mezi tímto úložištěm a diagramem lze přesouvat komponenty po jedné nebo všechny najednou. Je také možno nechat si zobrazit a skrýt list všech komponent obsažených v úložišti.

Hlavní vylepšení aplikace je implementace layoutu (rozmístění) komponent. Pro tento účel byl vybrán Force-Directed algoritmus, který komponentám přiřazuje vektory síly podle jejich pozice vůči ostatním komponentám a tím s nimi pohybuje. Byly implementovány dva typy toho algoritmu. První je statický algoritmus, který při načítání grafu nejprve náhodně rozmístí všechny komponenty a potom, ještě před zobrazením, vypočte pozice komponent v mnoha iteracích Force-Directed algoritmu. Po dokončení tohoto výpočtu jsou všechny komponenty zobrazeny již na správných místech. Druhý je dynamický algoritmus, který pracuje s pozicemi komponent za běhu aplikace. V horním panelu je tlačítko, které tuto funkci vypíná a zapíná. Po zapnutí této funkce se graf začne pohybovat směrem do optimální pozice, vypočtené Force-Directed algoritmem. Komponenty jsou posouvány v reálném čase, proto je možné měnit parametry za běhu programu. Tím je myšlen např. posun myší s komponentami nebo odebrání komponenty z grafu.

Literatura

[1] Úvod do komponent, KIV Wiki. 29. Března, 2011 (cit. Červenec, 2013)
http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage

[2] Bachmann, Felix; Bass, Len; Buhman, Charles; Comella-Dorda, Santiago; Long, Fred; Robert, John; Seacord, Robert; & Wallnau, Kurt. Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition (CMU/SEI-2000-TR-008). Software Engineering Institute, Carnegie Mellon University, 2000. (cit. Červenec, 2013)
http://www.sei.cmu.edu/library/abstracts/reports/00tr008.cfm

[3] Jindra Pavlíková, Vizualizace rozsáhlých diagramů komponent a interakce s nimi. Plzeň, 2012. Diplomová práce. ZČU, FAV, KIV.

[4] OSGi Alliance, The OSGi architecture (cit. Červenec, 2013)
http://www.osgi.org/Technology/WhatIsOSGi

[5] EJB 3.1 Specifications, 5. Listopadu, 2009 (cit. Červenec, 2013)
http://jcp.org/aboutJava/communityprocess/final/jsr318/

[6] FI WIKI, Enterprise Java Beans, 23. 4. 2013 (cit. Červenec, 2013)
http://kore.fi.muni.cz/wiki/index.php/Enterprise_JavaBeans

[7] SOFA2, 26. 3. 2013 (cit. Červenec, 2013)
http://sofa.ow2.org/index.html

[8] HOLÝ, Lukáš.; BRADA, Přemysl. Lowering Visual Clutter in Large Component Diagrams. 2011. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[9] HOLÝ, Lukáš.; BRADA, Přemysl. Lowering Visual Clutter in Large Component Diagrams. 2011. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[10] Jenifer Tidwell, Designing Interfaces, Patterns for Effective Interaction Design (cit. Červenec, 2013)

http://designinginterfaces.com/firstedition/index.php?page=About_the_Book

[11] Daniel Bureš, Vizualizace rozsáhlých diagramů. Plzeň, 2013. Diplomová práce. ZČU, FAV, KIV.

A Zdrojové soubory aplikace

Následující seznam zdrojových souborů byl převzat z práce [11].

A.1 Java balíky a třídy

- cz.zcu.kiv.offscreen.api
  - EdgeInterface.java – rozhraní definující nutný obsah každé hrany v grafu
  - GraphInterface.java – rozhraní definující nutný obsah grafu
  - VertexInterface.java – rozhraní definující nutný obsah uzlu grafu
- cz.zcu.kiv.offscreen.graph
  - EdgeImpl.java – třída reprezentující hranu, která implementuje rozhraní EdgeInterface
  - GraphImpl.java – třída reprezentující graf, který implementuje rozhraní GraphInterface
  - VertexImpl.java – třída reprezentující uzel grafu, který implementuje rozhraní VertexInterface
  - GraphExport.java – třída, která zajistí zjednodušení struktury grafu, aby mohla být data převedena do formátu JSON
- cz.zcu.kiv.offscreen.graph.creator
  - GraphMaker.java – třída, která zajišťuje vytvoření grafu na základě informací získaných z načtení komponent
- cz.zcu.kiv.offscreen.graph.loader
  - GenericComponentLoader – třída, která je oddděděna od třídy ComponentLoader a zajišťuje získání dat z načtených komponent
- cz.zcu.kiv.offscreen.loader.configuration
  - ConfigurationLoader.java – třída zajišťující získání cesty pro ukládání adresářů jednotlivých uživatelů ze souboru configuration.properties

- cz.zcu.kiv.offscreen.servlets
  - LoadGraphData.java – servlet zajišťující odeslání struktury grafu ve formátu JSON klientovi
  - Login.java – servlet, který přihlásí uživatele
  - Logout.java – servlet, který odhlásí uživatele
  - Register.java – servlet, který zaregistruje uživatele
  - SettingGraph.java – servlet, který zajistí vytvoření adresáře na základě získané session id z cookies a id diagramu
  - ShowGraph.java – servlet, starající se o zobrazení stránky aplikace, která obsahuje vytvořený graf komponent
- cz.zcu.kiv.offscreen.servlets.actions
  - DeleteAllComponents.java – servlet, který smaže veškeré nahrané komponenty na serveru, dle session id
  - DeleteComponent.java – servlet odstraní nahranou komponentu na serveru dle zadaných parametrů
  - DeleteDiagram.java – servlet, který odstraní vytvořený diagram, dle hash kódu a id diagramu
  - LoadDiagram.java – servlet, který načte uložený diagram dle hash kódu a id diagramu
  - SaveDiagram.java – servlet, který uloží diagram
  - UploadFiles.java – servlet, který se stará o nahrání komponent na server
- cz.zcu.kiv.offscreen.session
  - SessionManager – třída obstarávající odstranění nahraných komponent po vypršení session
- cz.zcu.kiv.offscreen.storage
  - FileManager.java – třída obsahující metody, které zajišťují vytváření uživatelského adresáře, ukládání a mazání komponent a diagramů
- cz.zcu.kiv.offscreen.user
  - DB.java – třída, která komunikuje s databází, přes tuto třídu se zpracovávají veškeré sql dotazy

  - Diagram.java – třída, ve které se vytvářejí, načítají a ukládají parametry diagramu, dále se zde také načítají seznamy veřejných a uživatelských diagramů
  - User.java – třída, ve které vytvářejí, načítají a ukládají uživatelé
  - Util.java – třída, ve které jsou uložené užitečné funkce využívané napříč aplikací, například funkce pro vytvoření md5 hashe.

A.2 JavaScriptové soubory

- JavaScriptové soubory jsou umístěné v adresáři js.
- diagram.js – soubor obsahující funkce potřebné pro načítání, ukládání, odstranění diagramu
- graphManager.js – reprezentuje graf, který načte data ze serveru a vytvoří graf ve formátu svg, který zobrazí na stránce
- gridMarg.js - zajišťuje vytváření známek (delegátů) a jejich řazení u komponenty
- group.js – zastupuje skupinu a obsahuje funkce pro přidávání a mazání prvků skupiny
- groupManager.js – správce vytvořených skupin komponent
- loader.js – zajišťuje zobrazení a skrytí JavaScriptového loaderu
- main.js – zajišťuje pohyb s komponentami v grafu
- mark.js – reprezentuje známku (symbol), který je přiřazován vyjmutým komponentám
- markSymbol.js – vytváří symboly (znak + barva)
- offScreenKiv.js – obsahuje implementaci off-screen techniky CoCA-Ex
- tooltips.js – registrace veškerých tooltipů v aplikaci
- util.js – obsahuje pomocné funkce
- zoom.js – obsahuje funkce pro přiblížení a oddálení diagramu komponent

- jquery-1.7.1.js – knihovna jQuery
- jquery.contextMenu.js – plugin, který slouží k vytvoření kontextového menu
- jquery.qtip.js – plugin k vytváření tooltipů
- mootools-core-1.4.1.js – plugin, který umožňuje vytvářet třídy v jazyku JavaScript
- spin.js – plugin pro vytváření loaderů

A.3 Soubory se styly

Soubory s css styly jsou umístěné v adresáři styles.

- basic.css – definované styly pro celou aplikaci
- jquery.contextMenu.css – styly používané v pluginu jquery.contextMenu.js
- jquery.qtip.css – styly doplňující plugin jquery.contextMenu.js
- tooltips.css – styly přepisující některé styly z jquery.qtip.css