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

Diplomová práce

Vizualizace rozsáhlých diagramů komponent a interakce s nimi

Plzeň 2012 Jindra Pavlíková

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2012

Jindra Pavlíková

Abstract

This master's thesis deals with the visualization of large component diagrams and the use of the off-screen visualization techniques for improving their usability. The purpose of this thesis is to design and create a tool integrated with ComAV application that will demonstrate usage of selected off-screen techniques in visualization of component diagrams. The tool is in form of the HTML5 application. Functionality of the created tool is demonstrated on real applications consisting of hundreds of components. This work also describes the principles of component-based application design and capabilities of visualization using off-screen techniques in general.

Obsah

Seznam obrázků

Poděkování

Tímto bych chtěla poděkovat vedoucímu mé diplomové práce, Ing. Lukášovi Holému, za cenné rady a připomínky v průběhu zpracování této práce. Dále bych ráda poděkovala celé své rodině za podporu a pomoc během celého mého studia.

1 Úvod

V posledních letech bylo stále více vývojářů nuceno čelit vzrůstající náročnosti vývoje aplikací. Tento nárůst složitosti návrhu je dán neustálým rozšiřováním působnosti informačních technologií a nutností neustále se přizpůsobovat konkurenčním tlakům či zvyšujícím se nárokům na komfort ze strany uživatelů. Reakcí na tento vývoj byl postupný přechod na komponentově orientovaný vývoj aplikací, jež umožňuje vývojářům vytvářet aplikace, které jsou i přes svoji složitost snadno rozšiřitelné a není problém jejich dlouhodobá udržitelnost. Aby bylo možné plně využít výhody, jež tento přístup přináší, vznikla potřeba vývoje nástrojů, které by umožňovaly zobrazení závislostí mezi komponentami. Tato potřeba je tím silnější, čím větší je rozsah komponentové aplikace.

Předmětem této diplomové práce je vytvoření nástroje, jehož účelem by bylo naplnění potřeby nastíněné výše. Tento nástroj by tedy měl umožnit jednodušší analýzu vztahů mezi komponentami. Vzájemné vztahy těchto komponent by byly zobrazeny ve formě diagramů libovolného rozsahu, s jejichž částmi by bylo možné interagovat způsobem, jež uživateli umožní snazší orientaci v takto složitých strukturách. Pro dosažení tohoto cíle a zvýšení uživatelského komfortu bylo využito tzv. off-screen technik vizualizace a navigace popsaných v teoretické části této práce. Tento nástroj by měl umožnit uživateli snadnou práci a orientaci v rozsáhlých diagramech komponent a měl by být rozšířením aplikace ComAV, která je vyvíjena na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni.

Pro plné pochopení významu vizualizace vzájemných vztahů mezi komponentami bude v úvodu této práce podrobněji popsán vývoj komponentových aplikací. Čtenář tak bude seznámen se základními pojmy související s touto problematikou. V dalším textu je poskytnut popis aplikace ComAV, jehož některé části byly při realizaci vytvářeného řešení této práce využity. Poté tato práce definuje pojem off-screen technika, poskytuje jejich stručný přehled, seznamuje s technikami specifickými pro vizualizaci komponentových aplikacích a zdůvodňuje jejich výběr pro realizaci ve vytvářeném vizualizačním nástroji.

Praktická část je zaměřena na volbu technologií vhodných pro implementaci vizualizačního nástroje. Zmiňuje se o dostupných knihovnách pro práci s grafy a rozebírá možnost implementace pomocí webových technologií. Je zde popsán návrh nástroje od uživatelského rozhraní přes jednotlivé off-screen techniky včetně jejich rozšíření až po návrh architektury. Dále se zabývá jednotlivými aspekty implementace zvolených technik. V závěru této práce jsou demonstrovány implementované funkce hotového řešení.

Nástroj bude používán na vědeckém pracovišti Katedry informatiky a výpočetní techniky, kde bude jejím pracovníkům sloužit k výzkumné činnosti v oblasti komponentově ori-

entovaného vývoje aplikací. Přínosem tohoto nástroje je výrazné rozšíření možností práce a vizualizace rozsáhlých diagramů komponent použitím na katedře vyvinutých off-screen technik.

2 Komponentové programování

V současné době se při vývoji softwaru klade důraz na znovupoužitelnost, což znamená, že některé části softwaru bude možné opětovně využít v jiném. Důležitá je také přizpůsobitelnost softwaru na jiné platformy (např. MacOs či Unix).

Základní myšlenkou komponentově orientovaného softwarového inženýrství (CBSE[1]) je rozdělit funkcionalitu aplikace na menší funkční části (komponenty). Tyto části se mohou stát stavebními jednotkami pro jiné aplikace, které budou vyžadovat stejnou funkčnost. Z toho vyplývá, že jedním z hlavních přínosů komponentového programování je znovupoužitelnost. Rozdělení aplikace na komponenty by mělo urychlit vývoj, popřípadě snížit náklady, protože se nebudou muset opětovně vyrábět části se stejnou funkcionalitou [1].

2.1 Komponenta

Komponenty můžeme vnímat jako menší funkční celky, které poskytují množinu služeb prostřednictvím rozhraní a jsou schopné přes tato rozhraní mezi sebou komunikovat. Definic, které se zabývají pojmem softwarová komponenta, je více [2].

Jednou z nich je definice podle Clemense Szyperského [3], která říká, že softwarová komponenta je jednotka skládající se ze smluvně stanovených rozhraní a kontextových závislostí. Softwarová komponenta může být nasazena nezávisle a je předmětem kompozice třetích stran.

Podle Clemese Szyperského a Davida Messerschmitta [4] by komponenta měla splňovat pět následujících kritérií:

- znovupoužitelnost
- nahraditelnost – komponenta by měla být kontextově nezávislá
- možnost skládat se s jinými komponentami
- zapouzdření – o vnitřku komponenty není nic známo, komponenta poskytuje navenek pouze rozhraní[2]

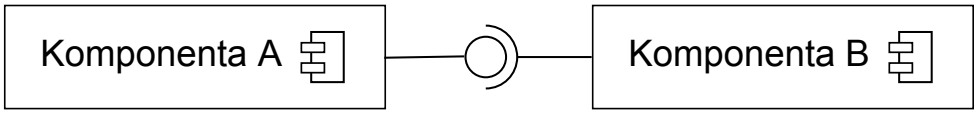
[1] Component-based software engineering

[2] Tomuto principu se říká black-boxový model.

- jednotka nezávislého nasazení a verzování

Každá komponenta obsahuje kromě seznamu poskytovaných služeb (vlastností), také seznam vlastností, které potřebuje, aby mohla být správně propojena s ostatními komponentami. Pokud komponentě nebudou poskytnuty vyžadované vlastnosti, nebude komponenta fungovat. Poskytované vlastnosti budou v této práci označovány jako *provided* a vyžadované jako *required*.

Ukázka 2.1 znázorňuje propojení komponenty A s komponentou B. Hrana mezi komponentami značí, že došlo ke spojení těchto dvou komponent. Kolečko na hraně mezi komponentami symbolizuje, že komponenta A poskytuje danou vlastnost komponentě B a symbol mističky naopak, že komponenta B vyžaduje tuto vlastnost od komponenty A.



Obrázek 2.1: Spojení komponent

2.2 Komponentové modely a frameworky

V podkapitole 2.1, jsme se seznámili s pojmem softwarová komponenta. S komponentami můžeme však začít reálně pracovat pouze v případě, budou-li umístěny do nějakého komponentového modelu. Komponentový model [3] specifikuje pravidla, která musí být splněna při skládání komponent, způsob komunikace mezi jednotlivými komponentami, požadované služby, konstrukci komponenty samotné, atd.

Komponentový framework [5] je implementace služeb, které podporuje nebo vyžaduje komponentový model. Lze si jej představit jako malý operační systém. Komponenty jsou pro framework to samé, co procesy pro operační systém. Komponentový framework řídí zdroje sdílené komponentami a poskytuje mechanismy, které mezi nimi umožňují vzájemnou komunikaci.

Ačkoliv komponentových modelů je celá řada, například OSGi, CoSi, EJB 3, SOFA 2 nebo CORBA Component Model, tak v následující části budou popsány pouze komponentové modely OSGi, EJB 3 a SOFA 2. A to z toho důvodu, že jsou podporovány aplikací COMAV.

2.2.1 OSGi

OSGi[3] Alliance[4] je nezisková společnost, která byla založena v roce 1999. Tato organizace se zabývá vytvářením otevřených specifikací z oblasti modulárního skládání softwaru postaveného na Java technologiích.

Jak uvádí [6], tak OSGi je platforma, která umožňuje nasazení a správu služeb. Jádrem této platformy je OSGi framework, který poskytuje běhové prostředí pro nasazení a provoz komponentových aplikací. Komponenty, ze kterých jsou aplikace složeny se v tomto frameworku nazývají bundly. Mezi nejznámější implementace jádra OSGi frameworku patří:

- Eclipse Equinox[5]
- Apache Felix[6]
- Knopflerfish[7]

OSGi bundle [7] je softwarová komponenta obsahující Javovské třídy a jiné pomocné zdroje, které společně poskytují uživatelům nějakou funkcionalitu. OSGi bundle je běžný JAR archiv, který obsahuje:

- zdroje potřebné pro svůj běh – přeložené Javovské třídy, obrázky, pomocné soubory, atd.
- manifest – textový soubor, který popisuje obsah JAR archivu a obsahuje důležité informace pro úspěšné nasazení a spuštění OSGi bundlu

2.2.2 EJB

Enterprise JavaBeans (EJB) je komponentová architektura na straně serveru pro Java platformu, která je v současné době vyvíjena společností Oracle[8]. Cílem EJB je oddělit business logiku aplikace od prezentační a perzistentní vrstvy a také zajistit kompatibilitu mezi produkty různých výrobců.

[3] Open Services Gateway initiative
[4] http://www.osgi.org/About/HomePage
[5] www.eclipse.org/equinox
[6] http://felix.apache.org
[7] http://www.knopflerfish.org/
[8] Dříve vyvíjeno firmou Sun Microsystems

EJB komponenta je stejně jako OSGi bundle distribuována v JAR archivu. Uvnitř JAR archivu jsou třídy dané komponenty, manifest soubor a soubor *ejb-jar.xml*. Tento soubor se nazývá *deployment descriptor* a obsahuje informace o komponentě. V *ejb-jar.xml* je možné upravovat některé vlastnosti komponenty, jako jsou například práva pro přístup ke komponentě, jejím metodám nebo částečně měnit její funkčnost. Pro spuštění komponent je nutné je umístit do EJB kontejneru, kde jsou vzájemně spárovány a řízeny, jak uvádí Pavel Stuna ve své diplomové práci [8]. Enterprise Java Bean architektura poskytuje 3 druhy komponent:

- Entity Beans – poskytují objektový pohled na data (entity) z databáze
- Session Beans – obsahují logiku aplikace
- Message-Driven Beans – reagují na události

2.2.3 SOFA 2

SOFA 2[9] [9] je komponentový systém s hierarchickým komponentovým modelem, který je vyvíjen na Karlově univerzitě v Praze. SOFA 2 není pouze nástroj sloužící k modelování komponent, ale poskytuje plnohodnotný framework podporující všechny stavy životního cyklu aplikace od vývoje až po její nasazení. Tento systém je přímým následníkem komponentového modelu SOFA[10]. Komponenta v SOFA 2 je zapouzdřený objekt, který s ostatními komponentami komunikuje pouze prostřednictvím poskytovaných či vyžadovaných rozhraních. Od svého předchůdce převzal jádro komponentového modelu, které je vylepšené a rozšířené o vlastnosti:

- popis komponentového modelu pomocí meta-modelu,
- dynamické změny konfigurace architektury komponentové aplikace,
- skládání komponent a jejich ověřování,
- podporu vícenásobné komunikace,
- zavádění aspektů do komponent,
- oddělení business logiky komponenty od její řídící části a poskytnutí možnosti jednoduché rozšiřitelnosti pomocí aspektů,
- podpora vývoje komponenty a jejího verzování.

[9] SOFtware Appliances
[10] http://sofa.ow2.org/sofa1/

3 Vizualizace komponentových aplikací

Tato kapitola popisuje důvody vizualizace komponentových aplikací a jejich současné možnosti. Dále se zabývá potřebami vizualizace a také způsobem získávání informací podstatných pro vizualizaci. V návaznosti na tyto fakta bude popsána aplikace ComAV[1], která byla částečně využita pro splnění cílů této práce.

Vizualizace softwarových aplikací a aplikací založených na komponentově orientovaném vývoji hraje důležitou roli ve snaze pochopit tyto softwarové systémy. Jak z pohledu nového uživatele začínajícího s těmito systémy pracovat, tak i z pohledu stále se zvyšující komplexnosti těchto systémů.

Na tomto vývoji je založeno mnoho aplikací, které pracují s komplexní strukturou tvořenou základními jednotkami (komponentami)[2]. Vzhledem k tomuto faktu je zajímavé, že pro komponentově orientované systémy není na výběr mnoho vizualizačních nástrojů [10]. Vizualizace je dostupná jen pro některé komponentové modely a poskytuje pouze jednoduché a statické komponentové diagramy UML 2.0.

V případě vizualizace komponentových aplikací je potřeba zobrazovat detailní informace o komponentách nikoli jen názvy komponent a spojení mezi nimi. Komponentové modely však nemají jednotnou reprezentaci komponent, která by o nich poskytla podrobné informace. Sjednocenou reprezentací by se zajistila lepší čitelnost diagramů různých komponentových modelů.

Informace o komponentách je možné získat pomocí reverzního inženýrství. Existují nástroje, které dokážou zrekonstruovat a vizualizovat strukturu komponentových aplikací, ale jsou úzce svázány s konkrétními komponentovými modely.

3.1 ComAV

ComAV [10] je univerzální platforma pro vizualizaci a reverzní inženýrství komponentově orientovaných aplikací, a proto ho lze využít pro libovolnou komponentovou aplikaci.

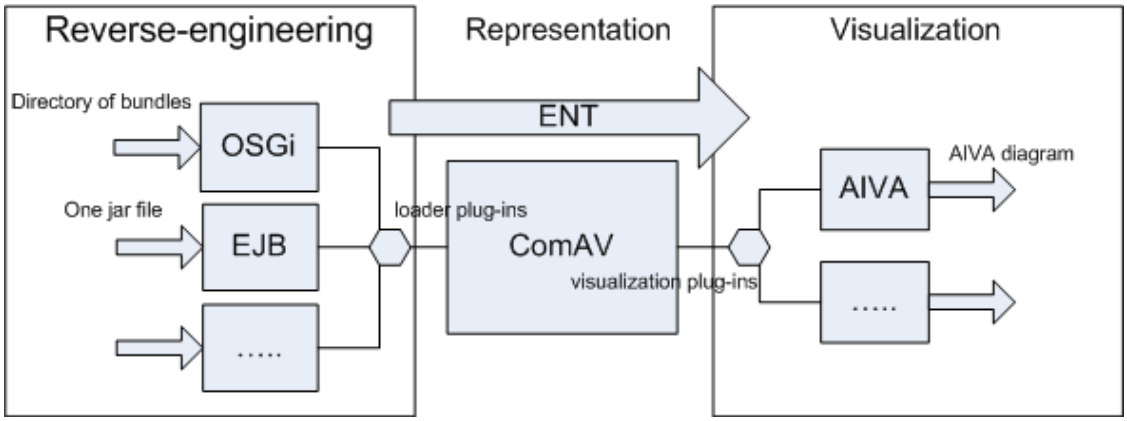
[1] Component Application Visualization

[2] Komponenty svému okolí poskytují funkcionalitu prostřednictvím rozhraní, ale neposkytují vlastní implementaci těchto funkcí (black-box model), jak je uvedeno v kapitole 2.

Tato platforma je velmi flexibilní a poskytuje jednoduchý způsob pro možné rozšíření jednotlivých částí. Je vyvíjena na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni. ComAV je RCP aplikace založená na Eclipse IDE a je napsána v jazyce Java. Jeho architektura je postavená na základě pluginů.

3.1.1 Architektura

Architektura ComAVu je členěna do tří hlavních vrstev, které je možné vidět na obrázku 3.1.



Obrázek 3.1: Architektura ComAVu (převzato z [10])

Díky této architektuře jsou komponentový model a vizualizace na sobě nezávislé a lze je použít v libovolných kombinacích.

Comav umožňuje přidávání jednotlivých pluginů. Tyto pluginy mohou být dvojího druhu:

- loadery,
- vizualizace.

Loadery jsou pluginy ve vrstvě *Reverse-engineering* sloužící k načítání informací z komponent. Tyto pluginy poskytují snadný mechanismus pro rozšíření o podporu načítání komponent z dalších dosud nepodporovaných komponentových frameworků.

Vizualizační pluginy ve vrstvě *Visualization* slouží k zobrazování informací o komponentách různými způsoby. Jako v předchozím případě i zde je možné rozšířit funkcionalitu o nové styly zobrazení pouhým přidáním pluginu.

ComAV díky pluginům aktuálně podporuje načítání komponentových modelů OSGi, EJB a SOFA 2.

ComAV umí pomocí reverzního inženýrství načíst informace z komponentové aplikace napsané pro libovolný komponentový model, který však přímo nepodporuje. Místo toho používá jednotnou datovou strukturu jako formát pro výměnu dat, který umožňuje udržovat informace jak o komponentovém modelu tak i o aplikaci.

Výměnný formát je výstupem vrstvy *Reverse-engineering*, který je ukládán ComAVem do formátu XML, a může být později použit jako vstup pro vrstvu *Visualization*. ComAV jako výměnný formát používá pokročilý meta-model (ENT)[3].

Uživatelské rozhraní Comavu poskytuje pohled na projekt a skládá se z několika částí:

- console, která slouží k informování uživatele o tom, co Comav dělá,
- editor, kam vizualizační vrstva respektive vizualizační pluginy zobrazují své výsledky,
- menu, kam mohou být přidávány pluginy z vrstvy Reverse-engineering pro načítání nových komponentových modelů.

[3]ENT meta-model je obecný model definovaný strukturami komponentových modelů a komponentově orientovaných aplikací. Více informací je uvedeno v práci [11]

4 Off-screen techniky

V této kapitole budou obecně popsány off-screen techniky a jejich využití. Dále budou přiblíženy základní přístupy off-screen technik. Poté se práce zaměří na techniky, které mají uplatnění ve vizualizaci komponentových aplikací. Následně budou některé z nich pro tuto práci vybrány.

V mnoha aplikacích jako 3D editory, editory obrázků a videí, aplikace poskytující různé náhledy na mapy, myšlenkové 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. Z tohoto důvodu, zde budou popsány takové techniky, které se hodí pro vizualizaci komponentových aplikací a následně z nich budou některé vybrány. Vybrané techniky budou v rámci této práce implementovány.

4.1 Technika Overview & Detail

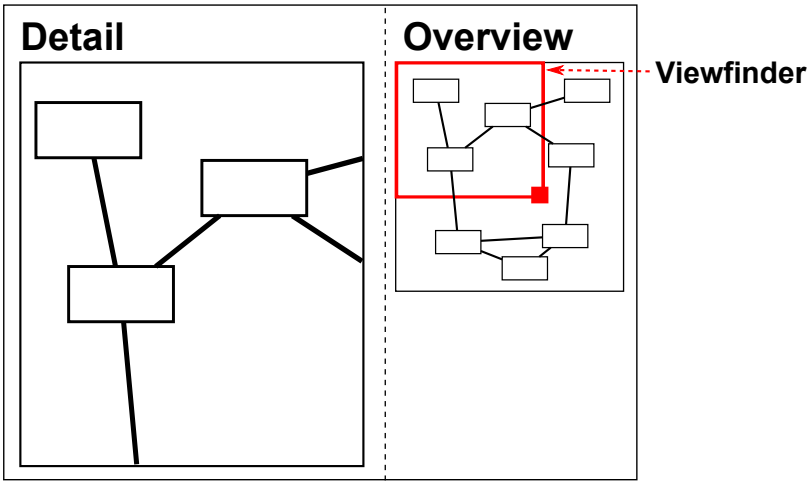
Tato technika se vyznačuje současným zobrazením pohledů přehledu (angl. overview) a detailu, jak uvádí práce[12]. Oba tyto pohledy jsou umístěny v samostatných prezentačních oblastech. Díky fyzickému rozdělení obrazu na dva pohledy může uživatel pracovat s každým pohledem zvlášť. Přesto se akce provedené v jednom pohledu většinou okamžitě promítnou do pohledu druhého a naopak.

Existuje mnoho variant techniky overview & detail, které se využívají ve standardních desktopových prostředích a výzkumných systémech. Zahrnují mnoho důležitých vlastností jako jsou:

- procentuální zvětšování v oblastech overview & detail,
- relativní velikost a pozice pohledů,
- mechanismus na ovládání navigace,

- propojení mezi pohledy overview a detail.

Overview je zmenšený pohled celé pracovní plochy, který je umístěn v dané oblasti. Tento pohled obsahuje tzv. hledáček (angl. viewfinder) [13], což je vyznačená oblast v overview, která je zobrazena na obrázku 4.1. V detailu je pak vidět přiblížený pohled oblasti, kterou obklopuje hledáček. Velikost hledáčku může být libovolně nastavitelná pomocí malé ikonky nacházející se v jednom z jeho rohů a tím i úroveň přiblížení detailu.



Obrázek 4.1: Ukázka overview & detail

Jakákoli změna pozice v detailním zobrazení se projeví uvnitř overview přesunutím hledáčku nad oblast, která je zobrazena v detailu. Uživatel se může pohybovat po pracovní ploše přesouváním (angl. dragging) hledáčku uvnitř overview, což se projeví pohybem detailu na oblast, kterou hledáček vyznačuje. Jeho přesun nemusí být vždy pevně spřažen s detailem, tzn., že se pohyb hledáčku neprojeví na detailu okamžitě, ale například až po uvolnění tlačítka myši. Toto chování je závislé na konkrétní implementaci varianty této techniky.

Nepatrné narušení vazby dovoluje uživatelům prozkoumávat overview, aniž by se to jakkoli projevilo v okně s detailem. Tímto se sníží výpočetní nároky, jelikož se změny neprojevují okamžitě. Tento princip je vhodný pro rozsáhlé dokumenty a pro aplikace zobrazující stovky až tisíce prvků, kde při změně polohy v overview je výpočetně náročné překreslení aktuálně zobrazované plochy v detailu.

Pro tuto techniku nejsou definována obecná pravidla, jak reagovat na změny. Standardem se stává, že se lze pohybovat v náhledu aplikace, aniž by se projevila změna v detailu, zatímco manipulace s detailem je okamžitě promítnuta do overview. Technika DragMag [13] volí jako hlavní oblast overview a menší oblast detail, což se pro některé úlohy může hodit.

Běžnou praktikou v aplikacích je poskytnout uživateli možnost volby zobrazení či skrytí overview, protože nemusí být vždy užitečný a tudíž by na obrazovce pouze zabíral místo.

4.1.1 Posuvníky

Posuvníky (angl. scrollbars)[12] jsou známou technikou, která se používá v grafických uživatelských rozhraních. Každý posuvník umožňuje pohyb pouze v jednom směru. Posuvníky mohou být horizontální nebo vertikální. Uvnitř scrollovací lišty je tzv. *thumbnail* nebo také *knob*, jehož velikost v poměru k velikosti scrollovací lišty udává, jaká část dokumentu je viditelná vzhledem k jeho celkové velikosti.

Posuvníky se většinou využívají v oblasti detailu, kde slouží k pohybu po pracovní ploše. Implementace posuvníků je taková, že jsou vždy zobrazeny mimo pracovní plochu nikoli nad ní.

4.2 Zoom

Druhá základní technika je zoom [12], která zahrnuje jak pohled na okolí (angl. context), tak i na konkrétní oblast (angl. focus). Je založena na přibližování, které dočasně vyvolá oddělení mezi zmíněnými pohledy. Přibližováním (angl. zoom in) se zaměřuje pohled na konkrétnější oblast a oddalováním (angl. zoom out) na okolí.

Zoomování se používá častěji, než technika focus & context viz níže. Tato technika lze kombinovat s již zmíněnou technikou overview & detail, ale tato část se zaměřuje pouze na zoomování nikoli na různé kombinace.

Mnoho aplikací obsahuje funkci zvětšování, především aplikace pro práci s textem a grafikou. Funkce zvětšení je většinou dostupná v nástrojové liště nebo v menu. V aplikacích jsou často předem definovány jednotlivé úrovně přiblížení, mezi kterými může uživatel přecházet vždy o jednu úroveň nahoru nebo dolů pomocí tlačítek v uživatelském rozhraní nebo klávesových zkratek, které bývají nastaveny na Ctrl + "+", Ctrl + "-", Ctrl + kolečko myši. Úroveň zvětšení, která je často udávána v procentech, lze také nastavit na libovolnou předdefinovanou úroveň nebo určit svou vlastní. Přecházení mezi jednotlivými úrovněmi lze realizovat plynule nebo okamžitými změnami.

S použitím techniky zoomování přichází několik designových problémů. Jedním z nich je pohyb po přiblížené ploše, neboť s rostoucí úrovní přiblížení roste vzdálenost mezi jednotlivými body dokumentu. Dalším problémem zoomování je ztráta orientace v přiblíženém dokumentu. Například velkým přiblížením mapy v dané oblasti se ztrácí přehled v loka-

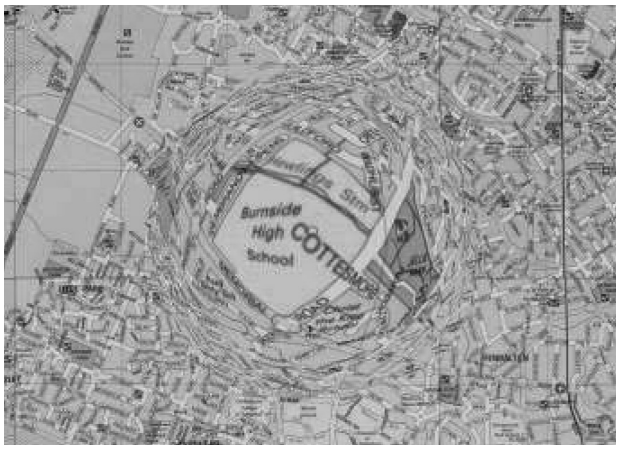
litě. To lze řešit kombinací této techniky s technikou overview & detail tak, že při velkém přiblížení je možné pro pohyb využít hledáček a také se díky němu zlepší orientace, neboť zobrazuje pohled detailu na pracovní plochu v kontextu.

Jak již bylo zmíněno, tak zoomování může být realizováno diskrétní změnou mezi jednotlivými úrovněmi přiblížení. Touto změnou dojde okamžitě k přechodu mezi pohledem z jedné úrovně do pohledu jiné úrovně, přičemž může dojít k dezorientaci v dané oblasti. Tento problém je možné řešit pomocí animací, které mohou napomoci uživateli zlepšit orientaci.

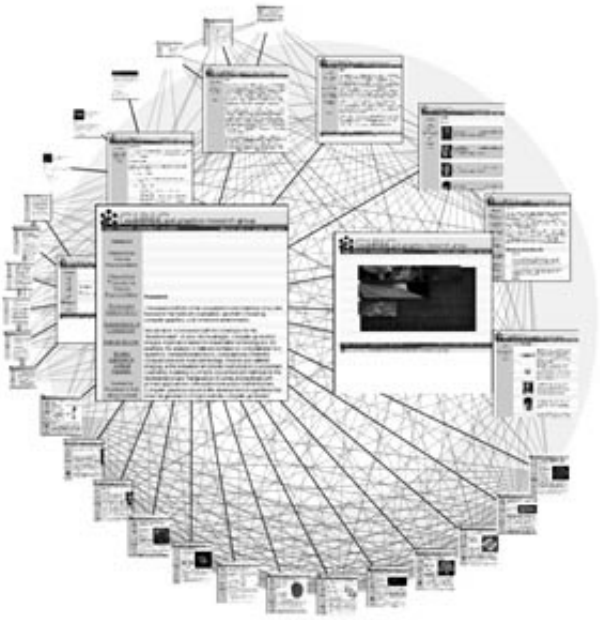
4.3 Focus & Context

Předchozí techniky oddělovaly okolí (angl. context) a konkrétní oblast (angl. focus) do dvou pohledů. Technika *Focus & Context* [12, 13] je třetí základní off-screen technikou, která spojuje focus & context do jediného pohledu. Focus je zobrazen detailně a je začleněn do kontextu.

Focus je oblast, o kterou se uživatel zajímá a je zobrazena detailně. Kolem této oblasti se nachází její okolí. S rostoucí vzdáleností kolem této oblasti se snižuje úroveň detailu a zároveň se tím zvětšuje pohled na její okolí. Snižování úrovně detailu může být realizováno tak, že s rostoucí vzdáleností od oblasti focusu dochází k postupnému oddalování pohledu (viz obrázek 4.2) nebo snižování detailu popisovaných informací (viz obrázek 4.3). Focus & context může být například podobný rybímu oku (angl. fisheye), které je blíže popsáno v [14].



Obrázek 4.2: Oddálení oblasti mimo focus



Obrázek 4.3: Snížení detailu informace v oblasti mimo focus (převzato z http://www.visualcomplexity.com)

4.4 Viewport pro komponentové diagramy

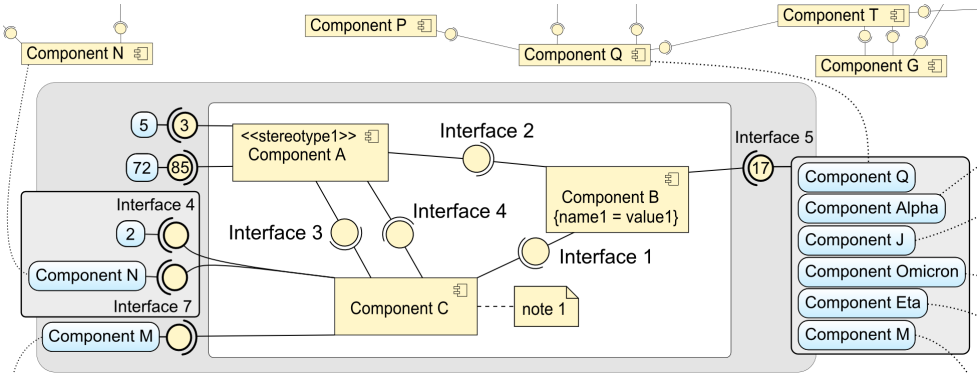
Myšlenka viewportu pro komponentové diagramy vznikla na Katedře informatiky a výpočetní techniky Západočeské univerzity na základě faktu existence mnoha komponentových aplikací, které se skládají ze stovek až tisíců komponent, které není možné vizualizovat pomocí komponentových UML diagramů tak, aby byly přehledné a zároveň poskytovaly podrobné informace o zobrazovaných komponentách. Popis této techniky je částečný překlad článku [15].

U takto rozsáhlých diagramů je potom obtížné vyhledávat závislosti mezi jednotlivými komponentami. To je způsobeno tím, že při zobrazení celého diagramu najednou a současně velké hustotě zobrazených prvků mohou být jednotlivé prvky těžko rozpoznatelné.

Tato technika zobrazuje náhled na celý graf bez jakýchkoli detailů. Detailní informace o vybraných komponentách jsou zahrnuty v oblasti viewportu. V rámečku kolem viewportu jsou zobrazeny všechny závislosti vybraných komponent s ostatními, které leží mimo viewport. Závislosti každé komponenty z viewportu s okolím jsou svázány do množin udávajících počet poskytovaných a vyžadovaných rozhraní.

Tato rozhraní jsou spojena pod zástupnou komponentu, která je na znázorněna obdélníkem s kulatými rohy. Každý obdélník může reprezentovat jednu nebo více komponent.

Rozvržení celého viewportu včetně jeho okolí je znázorněno na obrázku 4.4. Více informací o této technice je v dokumentu [15].



Obrázek 4.4: Viewport pro komponentové diagramy (převzato z [15])

4.5 COmplex Component Applications EXploration

Technika CoCA-Ex [16] se zabývá snižováním vizuálního šumu. Rovněž jako předchozí technika je vyvíjena na Katedře informatiky a výpočetní techniky Západočeské univerzity. Je založena na principu odebírání prvků, které způsobují vizuální šum (špatnou přehlednost). Přičemž zachovává informace o jejich vzájemném propojení.

Pro zobrazení vyjmutých komponent s velkým počtem spojení využívá izolovanou komponentovou oblast. Pro každou komponentu v této oblasti jsou všechna její rozhraní seskupena do dvou zástupných skupin místo toho, aby byla zobrazena všechna rozhraní zvlášť. Tyto zástupné skupiny reprezentují poskytovaná a vyžadovaná rozhraní a umožňují zobrazit detailní informace na vyžádání.

Hlavní myšlenka této techniky je, že může být použita v podobných případech pro snížení počtu uzlů a hran v grafu. Dále budou popsány detailní vlastnosti této techniky. Popis detailních vlastností je převzat z [16].

4.5.1 Komponentová oblast

Tato oblast je část aplikačního okna, kterou je vhodné umístit na jeho levou či pravou stranu. Dnešní obrazovky jsou širokoúhlé, a proto je toto použití výhodné, neboť nedochází k tak výrazné deformaci zbývajícího prostoru, než v případě umístění této části nahoru či dolů. Rozmístění jednotlivých částí aplikace je ukázáno na obrázku 4.5.

Obrázek 4.5: Rozmístění prvků

4.5.2 Položky

Komponentová oblast obsahuje seznam položek, které se skládají z vyjmuté komponenty, jejích vyžadovaných a poskytovaných rozhraní a zástupného symbolu. U komponent umístěných v této oblasti jsou zobrazena spojení s ostatními komponentami v diagramu na hraně mezi oblastí diagramu a komponentovou oblastí.

Položky v komponentové oblasti jsou dvojího druhu. První druh položky obsahuje pouze jednu komponentu, jejíž rozhraní jsou přímo spojena se zobrazovanou komponentou. Součástí této položky je také zástupný symbol, který je umístěn za komponentou viz obrázek 4.8.

Druhá reprezentace položky se skládá z více komponent, které tvoří skupinu. V tomto případě jsou rozhraní přímo spojena se symbolem, za kterým je seznam obsažených komponent. Ukázka skupiny je uvedena v obrázku 4.9.

4.5.3 Symboly a delegáti

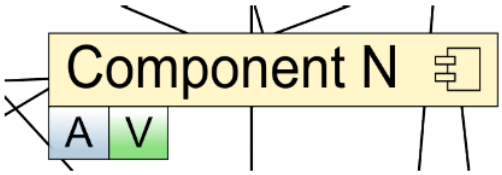
Smyslem symbolů je vytvořit jasný a jednoduše rozpoznatelný klíč, který jednoznačně identifikuje konkrétní komponentu vyjmutou do komponentové oblasti. Tyto symboly je možné využít jako delegáty pro komponenty sousedící s vyjmutou komponentou, které jsou stále zobrazeny v oblasti diagramu.

Symboly by měly být dostatečně malé, aby ušetřily místo kdekoli, kde budou použity. Zástupné symboly mohou být v podobě Symboly by měly být dostatečně malé, aby ušetřily místo kdekoli, kde budou použity. Zástupné symboly mohou být v podobě znaků nebo obrázků. Příklady těchto symbolů jsou znázorněny na obrázku. Příklady těchto symbolů jsou znázorněny na obrázku 4.6.

Jak již bylo zmíněno, delegáti jsou zobrazováni u komponent sousedících s komponentou vyjmutou do komponentové oblasti. Delegáti v tomto případě zastupují odebrané

Obrázek 4.6: Příklady symbolů (převzato z [16])

hrany dané komponenty. V diagramové oblasti jsou u komponent zobrazovány jako malé obdélníky, které nesou zástupný symbol odpovídající jedné komponentě z komponentové oblasti (viz obrázek 4.7).



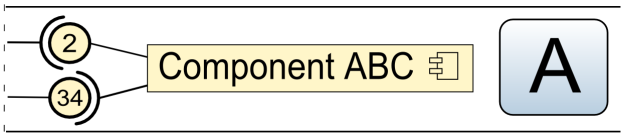
Obrázek 4.7: Delegáti u komponent v oblasti diagramu (převzato z [16])

Po kliknutí na symbol u vyjmuté komponenty budou zobrazeni delegáti u všech sousedících komponent. U vyjmuté komponenty může být tento stav zachycen několika způsoby, jako jsou například podbarvení položky, podbarvení zástupného symbolu, checkbox aj.

4.5.4 Sloučená rozhraní

Každá komponenta v komponentové oblasti obsahuje dvě množiny sloučených rozhraní. První množina reprezentuje všechna rozhraní, která poskytuje, a druhá ty, které vyžaduje. Poskytovaná rozhraní jsou značena symbolem „lízátka“ a vyžadovaná rozhraní symbolem „mističky“.

Uvnitř sloučeného rozhraní jsou zobrazena čísla, která udávají počet, s kolika komponentami je daná komponenta spojená přes poskytovaná a vyžadovaná rozhraní. Takto zobrazená položka je na obrázku 4.8.



Obrázek 4.8: Sloučená rozhraní (převzato z [16])

Sloučená rozhraní nejsou standardně spojená hranami s komponentami uvnitř diagramové oblasti. Hrany se objeví až po manipulaci s komponentami, které mají vazby se sloučeným rozhraním. Existují dva způsoby interakce se sloučenými rozhraními. První z nich je jednoduché zobrazení hran a zvýraznění sousedících komponent po najetí či kliknutí myši na sloučené rozhraní. Druhý ze způsobů je zobrazení detailů o všech rozhraních, zobrazení hran a zvýraznění sousedících komponent.

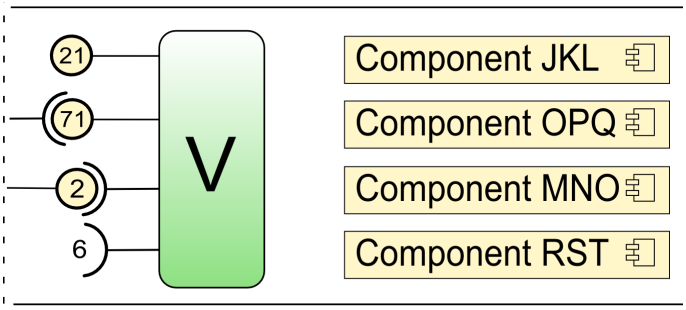
4.5.5 Skupiny

Skupina, která již byla zmíněna výše, je jednou z možných položek komponentové oblasti. Skupiny jsou vhodné pro případ, kdy množina komponent tvoří určitou funkcionalitu, která je využívána velkým počtem ostatních komponent.

Všechny komponenty ve skupině jsou zastoupeny jedním symbolem, jež je využíván jako delegát u komponent sousedících s touto množinou. Stav, kdy jsou zobrazeni delegáti je symbolizován u skupiny stejným způsobem, jako tomu bylo u jednotlivých komponent z důvodu jednotnosti.

Slučování komponent do skupin umožňuje vytvářet sémantické celky, což přispívá k lepšímu porozumění celého systému. Vytváření skupiny také zlepšuje orientaci v diagramové oblasti díky odebrání skupiny komponent, které mohou být silně provázány se zbytkem systému. Společně s komponentami jsou z diagramu odebrány všechny jejich vazby.

Na obrázku 4.9 je vidět čtveřice sloučených rozhraní. První dvojce udává vyžadovaná a poskytovaná rozhraní, která nejsou navázána na žádné komponenty a druhá dvojce říká počet poskytovaných a vyžadovaných rozhraních, která jsou navázána.



Obrázek 4.9: Skupina komponent reprezentované symbolem skupiny (převzato z [16])

4.6 Závěr

V této kapitole byly obecně popsány off-screen techniky a jejich využití. Dále byly zmíněny tři základní přístupy těchto technik a některé techniky zaměřené na zjednodušování zobrazovaného diagramu pomocí sofistikovaného odebírání prvků z něj. Myšlenky těchto technik vznikly na Katedře informatiky a výpočetní techniky Západočeské univerzitě a stále jsou rozvíjeny přidáváním nových vlastností a úpravou stávajících.

Pro tuto práci byla zvolena technika COmplex Component Applications EXploration (CoCA-Ex) v kombinaci s technikou přibližování (angl. zoom). Tato technika byla vybrána, protože se přímo zaměřuje na sofistikovanou manipulaci s diagramem komponentových aplikací a na zjednodušení detailních informací bez jejich ztráty.

Implementace této techniky se zoomem, se může od původní myšlenky mírně odlišovat, neboť je zmíněná technika stále ve fázi vývoje, který částečně probíhal s touto prací paralelně, a také díky vlastním záměrným změnám za účelem zlepšení.

5 Výběr technologií pro rozšíření ComAV

V předchozích kapitolách 3, 4 byly popsány obecné principy off-screen technik společně s technikou COmplex Component Applications EXploration (CoCA-Ex) a také byla představena aplikace ComAV.

Cílem této práce je propojit aplikaci ComAV s nástrojem pro vizualizaci rozsáhlých diagramů komponent, který bude obsahovat implementaci některých technik popsaných výše. K vytvoření tohoto nástroje je možné využít již existující grafové frameworky, které by mohly v různých ohledech implementaci usnadnit, ale na druhou stranu se mohou objevit problémy s jejich rozšířením o konkrétní off-screen techniky.

Předmětem této kapitoly bude tedy výběr technologií či grafových frameworků, pomocí kterých bude tento nástroj vytvořen.

5.1 Kritéria výběru

Hlavním kritériem pro výběr technologií, které budou použity, byl požadavek na propojení aplikace ComAV, jež je vyvíjen v programovacím jazyce Java. Z čehož vyplývá, že jedno z kritérií určující výběr technologie je požadavek na platformu Java.

S ohledem na cílové využití vizualizačního nástroje (zobrazování velkého množství komponent) vyplývá také požadavek na rostoucí počet zobrazovaných interaktivních prvků, jejichž množství má dopad na výkon.

V případě využití již existujících grafových frameworků, vzniká další důležité kritérium a to přizpůsobitelnost frameworků na konkrétní požadavky plynoucí z off-screen technik. Ostatními sledovanými vlastnostmi u grafových frameworků jsou dostupné layouty, jejich samotná podpora, která se týká dokumentace, komunity a návodů s postupným představením všech funkcí, jež by mohly být při vývoji vizualizačního nástroje využity.

5.2 Grafové frameworky

Na základě výše zmíněných kritérií, byl tedy výběr grafových frameworků mírně omezen. Při hledání grafových frameworků, které jsou postavené na platformě Java, jich i přesto bylo nalezeno velké množství. Z tohoto důvodu byly pro jejich bližší zkoumání vybrány pouze některé z nich a to konkrétně JGraph[1], ZEST[2] a JUNG[3].

Všechny tyto grafové frameworky umožňují vytvořit orientované i neorientované grafy a provádět nad nimi různé operace typu prohledávání grafu podle standardních algoritmů (prohledávání do hloubky, do šířky, atd.). Frameworky také poskytují reprezentaci grafu pomocí grafického uživatelského rozhraní, které je založeno na rozdílných grafických knihovnách (Java Swing, SWT). Toto rozhraní uživateli umožňuje interakci s daným grafem, jehož uzly je možné přesouvat po pracovní ploše a tím tak zlepšit jeho přehlednost. Funkcionalit, které zlepšují orientaci v zobrazeném grafu a práci s ním, je v těchto frameworcích mnoho.

Jednou z těchto významných funkcionalit jsou layouty, které zprostředkovávají rozmisťování uzlů grafu na pracovní plochu ještě před jeho samotným zobrazením. Jednotlivé frameworky poskytují různé algoritmy layoutů. Více informací o layoutech a zmíněných frameworcích je uvedeno v práci [17].

Tyto frameworky však nejsou zcela obecné a některé jejich funkcionality jsou zaměřené na danou množinu využití. Z čehož vyplývá, že by bylo nutné některé funkce přizpůsobit potřebám vybraných technik. Analýzou a vyhodnocením jednotlivých frameworků se zabývá práce [17], ze které vyplývá, že framework JUNG nejvíce vyhovuje daným kritériím.

5.3 Webová aplikace

Jak již bylo zmíněno v kapitole 3, je možné ComAV rozšířit pomocí vizualizačních pluginů, ale také lze využít jeho načítací (Reverse-engineering) pluginy do vlastní aplikace. Tento poznatek vedl k myšlence vytvořit vizualizační nástroj buď jako samostatnou desktopovou či webovou aplikaci.

Vytvoření nástroje pro vizualizaci rozsáhlých diagramů komponent jako webové aplikace s sebou přináší několik výhod a nevýhod. Hlavní výhodou je, že aplikace bude dostupná kdykoli a odkudkoli. Nevýhodou pak je, že veškerá základní funkcionalita, kterou výše zmíněné frameworky poskytují, bude muset být naimplementována znovu.

[1] http://www.jgraph.com/
[2] http://www.eclipse.org/gef/zest/
[3] http://jung.sourceforge.net/index.html/

Poslední zmíněný fakt však s sebou přináší i výhodu, že funkcionalita může být již od začátku přizpůsobena požadavkům vybraných technik, které vizualizační nástroj musí splňovat. Nebude nutné zasahovat do již naimplementovaných funkcí a vynakládat tak úsilí na jejich přizpůsobení, když není zaručeno, že dané grafové frameworky tento zásah umožní. Protože tento fakt je velmi závažný, byla zvolena možnost vytvořit vizualizační nástroj bez použití zmíněných grafových frameworků. Tímto se jako jediné vhodné řešení jeví vytvořit vizualizační nástroj jako webovou aplikaci.

Z toho vyplývá, že je potřeba zvolit vhodné webové technologie, které umožní realizovat napojení na ComAV a implementovat vybrané techniky. Tím je myšlena technologie, jak na straně serveru (angl. back-end), tak i na straně klienta (angl. front-end).

5.3.1 Webové technologie

Na straně serveru je možné využít *Java Servlety* nebo nadstavbové frameworky typu *Spring MVC*. Pro tuto práci byly zvoleny Java Servlety, protože implementace vizualizačního nástroje bude na straně serveru vyžadovat pouze propojení s ComAVem a načtení dat potřebných pro prezentaci vybraných technik. Nadstavbové frameworky poskytují velké množství modulů a funkcionalit, které by však v tomto případě nebyly žádným přínosem.

Na straně klienta bylo pro realizaci vizualizačního nástroje vybráno více technologií, které dohromady poskytnou požadovanou funkcionalitu. Mezi tyto technologie patří i HTML5. Důvodem vybrání nové verze HTML, ačkoli ještě zdaleka není podporována všemi prohlížeči, bylo umožnění zlepšení uživatelského komfortu a to díky nové možnosti nahrávat více souborů najednou.

Tato vlastnost je pro vizualizační nástroj stěžejní, protože pro zobrazování diagramů komponent je potřeba jejich nahrání na server, který je dále zpracuje. Pro uživatele aplikace by bylo neúnosné nahrávat na server každou komponentu zvlášť. Výběr HTML5 přináší mnoho dalších výhod, které byly také využity. Více informací o využití jeho vlastností je v kapitole 7.

Pro samostatnou prezentaci diagramů komponent byla zvolena nová možnost a to zobrazování diagramu pomocí SVG, které může být díky HTML5 jeho součástí, což znamená, že bude možné přistupovat k jednotlivým prvkům SVG pomocí JavaScriptu. Z čehož vyplývá, že další vybranou technologií je JavaScript, který zajistí veškerou aplikační logiku v součinnosti s javascriptovými frameworky, jež s ním usnadňují práci.

Javascriptových frameworků je na výběr velké množství (Dojo, Vaadin, ExtJS, SmartJS, jQuery, atd.). V této práci byl však vybrán framework jQuery vzhledem k jeho dobré podpoře, velké komunitě a dostupnosti mnoha rozšiřujících pluginů plynoucí z délky jeho existence, jak je uvedeno v práci [18].

Pro design a rozložení prvků bylo použito CSS3. Na straně klienta tedy byly vybrány technologie HTML5, CSS3, SVG, JavaScript a jQuery včetně některých jeho pluginů.

6 Návrh aplikace

Předmětem této kapitoly bude návrh grafického uživatelského rozhraní pro vizualizační nástroj. Uživatelské rozhraní je obohaceno o off-screen techniky, jež jsou oproti jejich původnímu návrhu rozšířeny takovým způsobem, aby výrazně napomohly orientaci uživatele ve zpracovávaných datech.

Dalším z řešených témat v této kapitole je návrh vzájemného propojení celé architektury aplikace jako celku, kde jsou obecně popsány jednotlivé elementy aplikace. Následuje podrobnější popis těchto celků. Dále je tedy zmíněn způsob realizace přístupu uživatelů k aplikaci, vzájemná komunikace klientské a serverové části, struktura dat reprezentující zpracovávaná data a v poslední řadě návrh propojení řešené aplikace s nástrojem ComAV, jež umožňuje získání těchto dat.

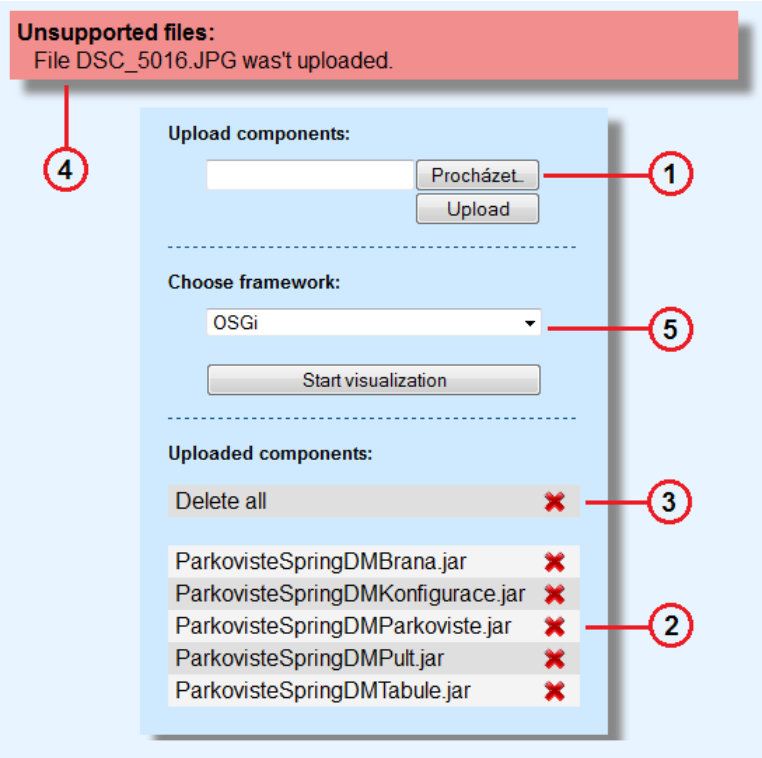
6.1 Uživatelské rozhraní

Uživatelské rozhraní aplikace je navrženo do dvou HTML stránek. Na první stránce (viz obrázek 6.1) je umístěn formulář (1), pomocí kterého uživatel vybere komponenty pro následnou vizualizaci a nahraje je na server.

Výše zmíněný formulář umožňuje vícero možností výběru vizualizovaných komponent. Potom co zvolíme adresář obsahující komponenty aplikace, máme k dispozici několik variant výběru, můžeme buď vybrat libovolné množství komponent, nahrát je na server a tím skončit nebo pokračovat dále výběrem jiného adresáře s dalšími komponentami. Poté co je uživatel spokojen s výběrem komponent, může přistoupit k jejich vizualizaci. Tato implementace je uživatelsky přívětivá, neboť si může přesně určit komponenty, jež budou zobrazeny.

Na této stránce po nahrání komponent na server budou jména nahraných komponent zobrazena v seznamu s možností smazat jednotlivé komponenty (2). Z důvodu uživatelského komfortu byla dodána možnost odstranit všechny komponenty najednou (3). Kvůli lepší přehlednosti v seznamu zobrazených komponent bylo použito jiné podbarvení sudých a lichých řádků.

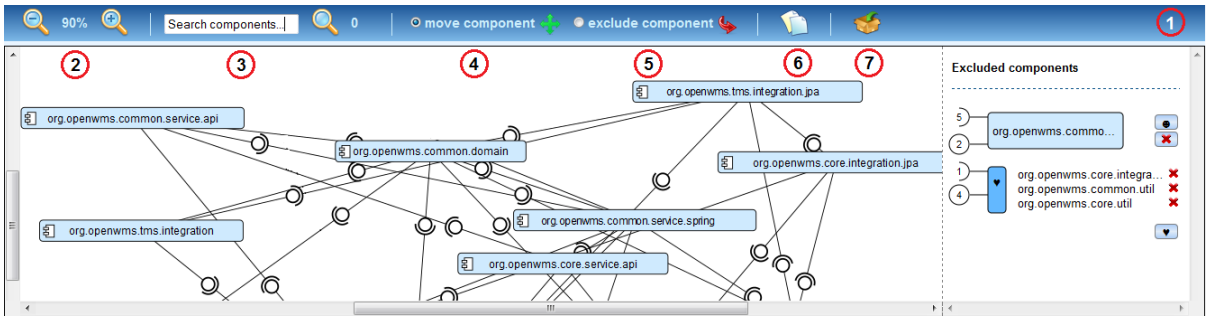
Pokud při nahrávání komponent dojde k nějaké chybě například nahrání neočekávaného typu souboru, je uživatel informován prostřednictvím chybového hlášení (4), které je umístěno v horní části obrazovky.



Obrázek 6.1: Úvodní obrazovka aplikace

Dále je na této stránce umístěn formulář (5), kde je možné vybrat typ komponent (OSGi, EJB nebo SOFA2) nahrávaných na server, které mají být vizualizovány. Po potvrzení tohoto formuláře bude zobrazena druhá stránka, kde bude vykreslen diagram zvolených komponent.

Grafické uživatelské rozhraní druhé stránky bylo navrženo tak (viz obrázek 6.2), aby zachovalo původní rozmístění prvků off-screen techniky CoCA-Ex popsané v podkapitole 4.5). Toto rozhraní bylo rozšířeno o nástrojovou lištu (1) s ovládací prvky umožňující kontrolu zobrazení diagramu komponent.



Obrázek 6.2: Hlavní obrazovka aplikace

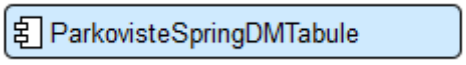
Tato lišta konkrétně obsahuje nástroje pro:

- přibližování a oddalování zobrazeného diagramu komponent (2),
- vyhledání komponenty obsahující hledaný řetězec (3),
- přesouvání komponent v oblasti diagramu (4),
- odstranění komponent z oblasti diagramu do komponentové oblasti (5),
- odstranění nejvíce zatížených komponent (pozn. komponenty, které mají nejvíc vazeb na okolí) do komponentové oblasti samostatně (6),
- odstranění nejvíce zatížených komponent do komponentové oblasti sloučených do jedné skupiny (7).

Při zobrazování aplikací s velkým počtem komponent (řádů stovek), které jsou navíc mezi sebou hustě propojeny mohou vznikat problémy s orientací a přehledností. Tyto problémy je možné vyřešit využitím techniky, která zajistí zpřehlednění grafu takovým způsobem, že z grafu budou vyjmuty uzly s nejvyšším stupněm. Cílem tedy je, aby se odebralo z grafu co nejvíce uzlů, které mají nejvyšší počet hran způsobující špatnou orientaci při manipulaci s uzly a grafem obecně.

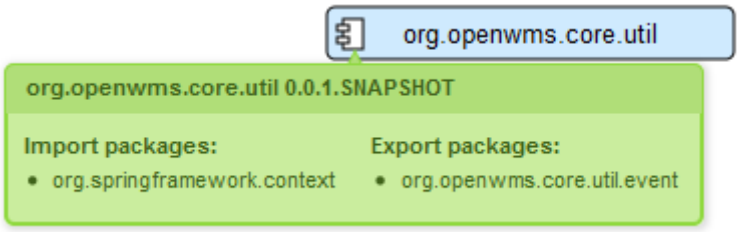
Otázkou však zůstává, kolik uzlů s nejvyšším zatížením je vhodné odebrat tak, aby došlo ke zmiňovanému zpřehlednění. Z grafu bude odebírána procentuální část nejvíce zatížených uzlů. Určení počtu procent odebraných uzlů, aby došlo ke zlepšení je velmi subjektivní a nelze tedy najít optimální řešení. Při vývoji této techniky a několika experimentů s různými aplikacemi v řádu desítek až stovek komponent, se osvědčilo vyjímat patnáct procent nejvíce zatížených uzlů. Tato konstanta byla určena pro demonstraci myšlenky nad rámec zadání implementované funkce, která obecně poskytuje možnost pro další rozšíření.

Pro vykreslení diagramu komponent je využito SVG. Stejně tak je tomu i u komponent a skupin v komponentové oblasti. Komponenta uvnitř diagramu je vyobrazena jako obdélník obsahující symbol komponenty a její název (viz obrázek 6.3).



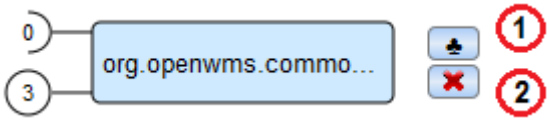
Obrázek 6.3: Návrh komponenty

Komponenta v této oblasti je oproti návrhu off-screen techniky CoCA-Ex rozšířena o možnost zobrazit po kliknutí na symbol komponenty detailní informace o poskytovaných a požadovaných rozhraní (viz obrázek 6.4). Pro zobrazení těchto informací jsou využity popisky (*angl. tooltips*), vytvářené pluginem pro jQuery jménem *qtip2*.



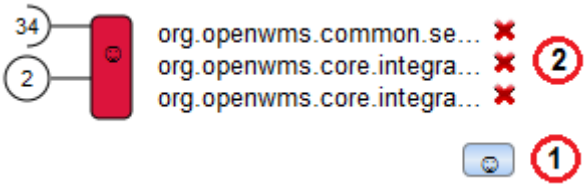
Obrázek 6.4: Tooltip s detailními informacemi o komponentě

Položka v komponentové oblasti reprezentující samostatnou komponentu (viz obrázek 6.5) byla od původního návrhu mírně změněna. Obsahuje navíc dvě tlačítka, jedno pro zobrazení delegátů u sousedních komponent (1) a druhé pro vrácení komponenty do oblasti diagramu (2).



Obrázek 6.5: Samostatná komponenta z komponentové oblasti

Položka reprezentující skupinu, zobrazena na obrázku 6.6, je také rozšířena o tlačítko (1), které zobrazí delegáty u sousedních komponent. Navíc byly dodány tlačítka pro odebrání jednotlivých komponent ze skupiny (2), což umožňuje uživateli skupinu upravovat podle svých potřeb. Čísla uvedená v rozhraních udávají celkový počet rozhraní, která skupina poskytuje či vyžaduje.

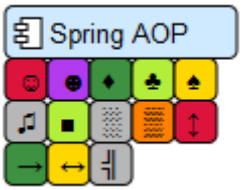


Obrázek 6.6: Návrh skupiny

Výhoda plynoucí z použití principu delegátů je popsána v části 4.5.3. U této techniky bylo potřeba navrhnout řešení, jak naložit s velkým množstvím zobrazených delegátů u jedné komponenty. V této práci bylo navrženo řešení, které je bude vkládat pod komponentu do mřížky postupně po řádcích.

Tento způsob se zdál jako nejvhodnější možné řešení, neboť zobrazení delegáti jsou stále v kontextu dané komponenty a tudíž se zachová přehlednost i při velkém počtu zobrazených delegátů. Například oproti řešení, kde by se delegáti zobrazovali pod komponentou

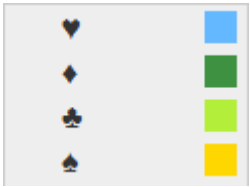
stále v jednom řádku, tak by od určitého množství zobrazených delegátů nemuselo být zřejmé s jakou komponentou jsou spojeny. Komponenta se zobrazenými delegáty v mřížce je na obrázku 6.7.



Obrázek 6.7: Zobrazení delegátů

Pokud se v komponentové oblasti nachází alespoň jeden prvek, je možné stisknutím pravého tlačítka myši nad komponentou v diagramové oblasti vyvolat kontextové menu. Toto menu obsahuje symboly a barvy přiřazené vyjmutým komponentám.

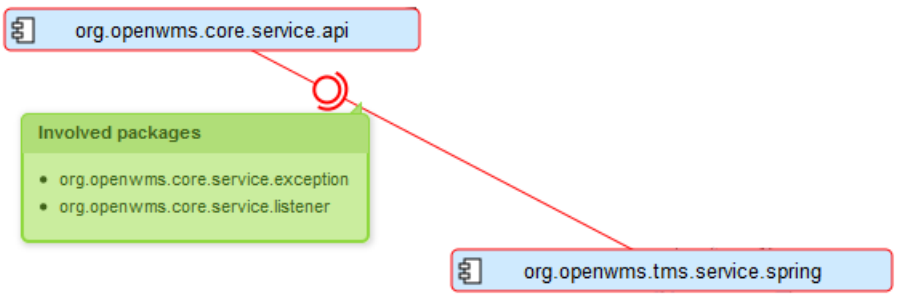
Po zvolení libovolného symbolu z nabídky je komponenta, nad kterou bylo kontextové menu vyvoláno, přiřazena do skupiny označené stejným symbolem, neexistuje-li, je tato skupina vytvořena. Kontextové menu je realizováno pomocí pluginu *JQuery Context Menu Plugin*. Ukázka kontextového menu je na obrázku 6.8.



Obrázek 6.8: Kontextové menu

Implementovaná off-screen technika CoCA-Ex byla rozšířena o zvýrazňování hran a komponent na ni napojených, což vede k lepší orientaci uživatele v diagramu komponent. Zvýrazněná hrana dovede uživatele ke komponentám, které jsou vzájemně propojené, ale jsou mimo oblast viewportu (oblast, která je pro uživatele viditelná). Po kliknutí na hranu, konkrétně na symbol "lízátka"a "mističky", byl navíc přidán *tooltip*, který nese informaci o tom, přes jaká rozhraní jsou komponenty propojeny. Toto rozšíření umožní uživateli lépe pochopit vizualizovanou komponentovou aplikaci. Ukázka zvýrazněné hrany je na obrázku 6.9.

Původní myšlenka off-screen techniky CoCA-Ex při zobrazování sousedů vyjmuté komponenty byla zvýraznit pozadí položky. Tento způsob však neumožní rozpoznat, zda jsou zobrazeni sousedi, které od vyjmuté komponenty vyžadují nějaká rozhraní, či naopak zda jí nějaká poskytují.

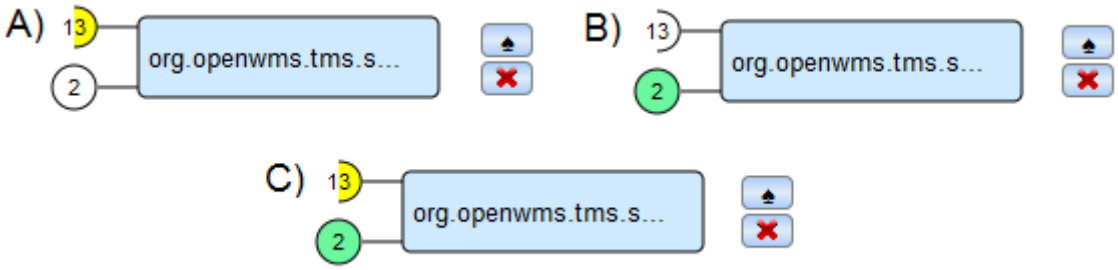


Obrázek 6.9: Zvýraznění hrany

Z tohoto důvodu bylo navrženo, že sousedé poskytující dané komponentě nějaká rozhraní, budou zvýrazněni žlutě včetně pozadí symbolu "mističky". Sousedé, kteří od dané komponenty rozhraní vyžadují, budou obarveni zeleně včetně symbolu kolečka u vyjmuté komponenty. Tento způsob zvýrazňování zajistí uživateli lepší představu o vztazích komponent v dané komponentové aplikaci. Při zvýrazňování sousedů mohou nastat tři stavy:

- zvýrazněni budou pouze sousedé požadující rozhraní od vyjmuté komponenty (A)
- zvýrazněni budou pouze sousedé poskytující rozhraní dané komponentě (B)
- zvýrazněni budou všichni sousedé (C)

Tyto stavy jsou znázorněny na obrázku 6.10. Princip zvýrazňování byl využit i při implementaci skupin.



Obrázek 6.10: Možnosti zvýraznění rozhraní komponenty

6.2 Architektura

Nároky kladené na aplikaci vymezily postup při navrhování architektury určitým směrem. K těmto nárokům se řadí například požadavek na to, aby uživatel mohl aplikaci začít využívat, co nejdříve a nebyl zdržován zbytečným registračním a přihlašovacím procesem. Dalším požadavkem byla co nejefektivnějším způsobem vyřešená komunikace klient-server a neméně důležité bylo spolehlivé propojení navrhované aplikace s nástrojem ComAV využitého k získávání dat určených k vizualizaci.

6.2.1 Identifikace uživatele

Vzhledem k tomu, že tato aplikace je webová a může k ní přistupovat více uživatelů najednou, kteří chtějí vizualizovat svá data, je nutné jednotlivé uživatele od sebe jednoznačně rozlišit. Uživatele lze rozlišit několika způsoby. Jedním z nich je, pro každého vytvořit samostatný uživatelský účet prostřednictvím registračního formuláře, který by obsahoval například email (kvůli jednoznačné identifikaci) a heslo (pro ověření uživatele). Tento způsob by obnášel vytvoření databáze, ve které by se vytvořené účty uchovávaly.

Dalším možným řešením je vytvořit na serveru pro každého uživatele adresář, který by nesl jméno podle přiděleného *session ID*. Do tohoto adresáře by byly nahrávány všechny komponenty, se kterými by chtěl uživatel pracovat. Každý uživatel dostane své jedinečné *session ID* přiděleno při vstupu na stránku s aplikací.

ID je ukládáno do *cookie* s nastavením vlajky na *http only*, což znamená, že tato položka v *cookie* nelze žádným způsobem změnit na klientovi. Uživatel má *session* s daným ID přidělenou tak dlouho, dokud session nevyprší. Délka trvání *session* se nastavuje v konfiguračním souboru *web.xml*. Nastavení délky trvání *session* v konfiguračním souboru je na následující ukázce.

```
<session-config>
   <session-timeout>120</session-timeout>
</session-config>
```

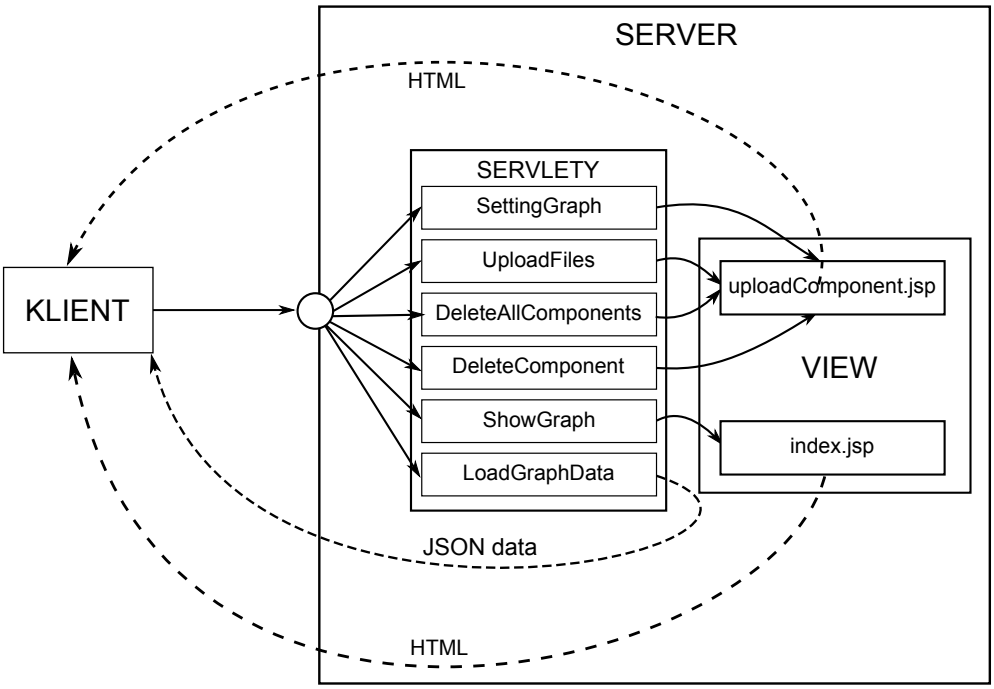
Aby nedocházelo k zaplňování serveru daty, byl by adresář s nahranými soubory po vypršení *session* smazán.

Při implementaci této aplikace byl pro jednoznačnou identifikaci uživatele využit druhý způsob, protože není nutné vytvářet databázi a také není uživatel zbytečně obtěžován přihlašováním.

6.2.2 Komunikace mezi serverem a klientem

Klient odešle požadavek na server, kde se podle konfiguračního souboru *web.xml* rozhodne, jaký servlet bude na požadavek reagovat. Servlet požadavek zpracuje a předá řízení *view*, které se skládá z jednotlivých JSP stránek vytvářející HTML stránky, jež jsou následně odesílány klientovi jako odpověď.

Konkrétní realizace návrhu architektury je zobrazena na obrázku 6.11. Prvním přístupem uživatele k aplikaci přijde požadavek na server, který je předán servletu *SettingGraph*. Následně je požadavek předán *view*, které vygeneruje úvodní stránku aplikace a tato stránka jež je obsažena v odpovědi, se odešle klientovi.



Obrázek 6.11: Schéma komunikace mezi serverem a klientem

Uživatel na této stránce může provádět akce, které se promítnou do servletů *UploadFiles*, *DeleteAllComponents* a *DeleteComponent*. Tyto servlety vedou řízení do *view*, které vytváří stejnou stránku jako tomu bylo u prvního přístupu uživatele.

Poslední dva servlety uvedení na obrázku zpracovávají akce týkající se zobrazení diagramu komponent. Konkrétně servlet *ShowGraph* zajišťuje zobrazení hlavní stránky aplikace. Po zobrazení hlavní stránky aplikace klient požádá server o data. Na tento požadavek reaguje servlet *LoadGraphData*, který zajistí načtení komponent pomocí pluginů ComAVu a vytvoření datové struktury s informacemi potřebnými k zobrazení diagramu komponent. Následně získaná data odešle klientovi ve formátu *JSON*.

6.2.3 Zdrojová data (JSON)

Server vrací data klientovi ve formátu JSON. Data, která jsou odesílána, obsahují informace o jednotlivých komponentách a jejich spojení. Konkrétně jsou odesílány seznamy hran a uzlů. Jeden záznam hrany nese informaci o jednom spojení a obsahuje:

- id hrany
- symbolické jméno komponenty odkud hrana vede,
- symbolické jméno komponenty, do které hrana vede,
- seznam názvů rozhraních, přes které se komponenty spojily.

Jedna položka seznamu s uzly zahrnuje detailní informace o jedné komponentě. Položka seznamu reprezentující jednu komponentu tedy obsahuje:

- id uzlu,
- seznam jmen exportovaných balíčků, což jsou rozhraní která komponenta poskytuje
- seznam jmen importovaných balíčků, což jsou rozhraní která komponenta vyžaduje
- jméno komponenty,
- symbolické jméno

Následuje ukázka dat ve formátu *JSON*, která jsou serverem odesílána klientovi. Klient tato data zpracuje pomocí JavaScriptu.

```
{"edges":[{"from":"ParkovisteSpringDMKonfigurace",
"id":1,"packageConnections":["cz.zcu.kiv.cosi.parkoviste.konfigurace"],
"to":"ParkovisteSpringDMTabule"}],

"vertices":[
{"exportedPackages": ["cz.zcu.kiv.cosi.parkoviste.konfigurace"],
"id":1,"importedPackages":[],
"name":"ParkovisteSpringDMKonfigurace",
"symbolicName":"ParkovisteSpringDMKonfigurace"},
{"exportedPackages":[],
"id":2, "importedPackages":["org.osgi.service.event",
"cz.zcu.kiv.cosi.parkoviste.konfigurace"],
"name":"ParkovisteSpringDMTabule",
"symbolicName":"ParkovisteSpringDMTabule"}]}
```

Po obdržení dat obsahujících informace o načtených komponentách od serveru, jsou tato data využita k sestavení objektů reprezentující jednotlivá spojení mezi komponentami (hrany) a jednotlivé komponenty (uzly). Po sestavení grafu z obdržených dat budou zaregistrovány všechny události jak na uzly tak i na hrany grafu. Události budou zaregistrovány i na ovládací prvky v nástrojové liště.

6.2.4 Propojení s ComAVem

Vizualizační nástroj, který je v tomto případě realizován jako webová aplikace, lze s ComAVem propojit několika způsoby. První z možností propojení je přístup využívající rozšiřitelnosti nástroje ComAV o komponentu simulující webový prohlížeč, jež by umožňovala zobrazení vizualizačních dat vytvořených navrhovanou aplikací. V tomto případě by u dané aplikace nebylo nutné řešit samostatné načítání zpracovávaných dat. Tato výhoda je ale výrazně zastíněna nemožností jednoduše využít celý rozsah webových technologií a jejich nejnovějších verzí (například JavaScript a jeho pluginy, HTML5).

Dalším možným řešením by bylo rozšířit ComAV o modul, který by umožňoval export jím získaných dat do standardního formátu. Možné formáty, jež by mohly dostát požadavkům, jsou například XML, JSON, YAML. Vizualizační nástroj by dále musel vyřešit jakým způsobem takto vytvořená data načíst a následně zobrazit.

Poslední možnost, jež byla současně v této práci aplikována, spočívá ve využití pluginů, jež jsou v rámci ComAV použity k načítání dat. Jejich výstupem je datová struktura, která obsahuje všechna data potřebná pro vizualizaci. Úkolem vizualizačního nástroje je následně takto připravená data vhodným způsobem převést do vlastních datových struktur a tyto struktury zpracovat do požadovaného grafického výstupu.

Prvním krokem k docílení úspěšného propojení bylo využití vlastní třídy (*GenericComponentLoader*), která je rozšířená o funkcionalitu třída *ComponentLoader*. Tato třída, jenž je součástí nástroje ComAV, využívá pluginy pro načítání komponent různých typů. Metoda *load(URI[] uri)* je modifikovaná takovým způsobem, aby aplikace zachovávala univerzálnost vůči typům komponent.

7 Implementace zvolených technik

V této kapitole bude přiblížena implementace jednotlivých technik, jež byly vybrány jako vhodné pro zlepšení orientace ve vizualizaci rozsáhlých diagramů komponent. Tyto techniky byly v kapitole 6 popsány a byl zhodnocen jejich přínos. Na základě tohoto zhodnocení došlo k případným úpravám ve specifikaci daných technik.

7.1 Struktura prezentovaných dat

Diagram vizualizovaných komponent je vykreslen pomocí jazyka SVG. Implementace využívá vlastnost HTML5 umožňující vložit SVG přímo do HTML kódu. SVG kód je generován dynamicky pomocí JavaScriptu.

Hlavní část zobrazení je rozdělena na dva sloupce, které jsou reprezentované elementy s ID *viewport* a *rightPanel*. *Viewport* obsahuje SVG kód diagramu komponent a rightPanel je tvořen seznamem vyjmutých komponent, jejichž zobrazení jsou tvořena pomocí samostatných SVG bloků.

Viewport obsahuje jediný <svg> element, který je tvořen hlavní skupinou (element <g> s ID *graph*), která dále obsahuje sekvenci skupin představujících jednotlivé hrany následovanou sekvencí skupin uzlů. Popisovaná struktura je uvedena na následující ukázce.

```
<svg id="svg1" xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns="http://www.w3.org/2000/svg">
        <g id="graph" >
                <g id="e1" class="edge">...</g>
                <g id="e2" class="edge">...</g>
                ...
                <g id="vertex1" class="vertex colorNormal">...</g>
                <g id="vertex2" class="vertex colorNormal">...</g>
                ...
    </g>
</svg>
```

7.2 Hrany

Hrany reprezentují vazby mezi komponentami a jsou tvořeny čárou a symboly kolečka a mističky v odpovídající orientaci. Každá hrana má přiděleno ID podle vzoru *„e“ + ID hrany* (například *e1*). Pozice koncových bodů hran jsou nastavovány pomocí grafických transformací. Obousměrné vazby jsou zobrazovány jako dvě samostatné hrany. Reprezentace hrany pomocí SVG je znázorněna na následující ukázce.

```
<g id="e1" class="edge" transform="translate(113.5, 13)">
        <line y2="173" x2="521" y1="768" x1="861">
        <g class="lollipop" transform="rotate(240,776,619)
          translate(776,619)" data-edgeid="1">
             <path class="SamplePath" d="M0,-12 C16,-12 16,12 0,12">
            <circle r="8" cy="0" cx="0">
        </g>
</g>
```

7.3 Uzly

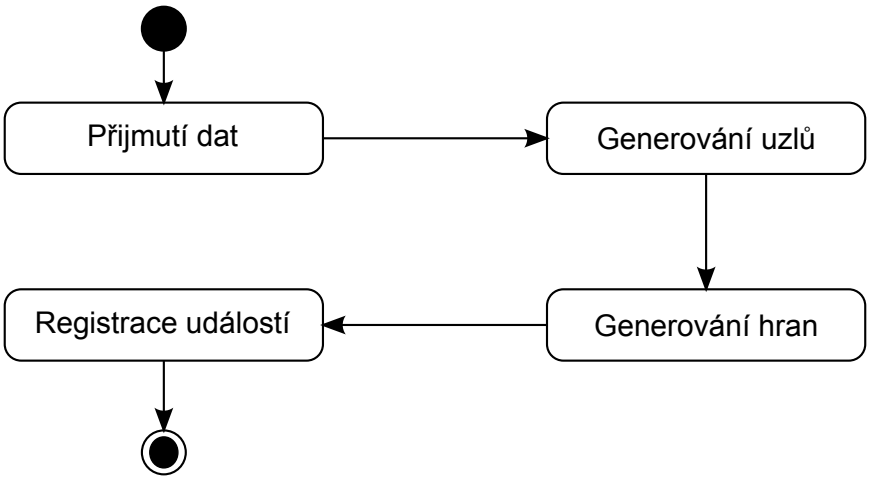
Uzly reprezentující jednotlivé komponenty jsou opět tvořeny základními geometrickými tvary. Při vytváření reprezentace komponenty není možné určit výslednou velikost vykresleného textu, v tomto případě názvu komponenty, a není tedy možné nastavit správnou velikost obalujícímu obdélníku (element <rect>).

Tento problém je řešen nastavením šířky obdélníku na základě počtu znaků názvu vynásobeného konstantou průměrné šířky znaku použitému fontu a přičtení rezervní konstantní hodnoty. Pokud by bylo třeba změnit font, bylo by nutné změnit i tyto konstanty. Pozice je opět nastavována pomocí grafických transformacích. Každý uzel má též přiřazeno ID, které je v tomto případě tvořeno podle vzoru *„vertex“ + ID uzlu* (například *vertex1*). Na následující ukázce je zobrazena část SVG, které definuje jeden uzel v grafu.

```
<g id="vertex1" class="vertex colorNormal"
  transform="translate(521,173)">
        <rect ry="4" rx="4" height="26" width="123">
        <text y="13" x="22">Event Admin</text>
        <g class="interface" transform="translate(8,6)"
          data-vertexid="1">
                <rect y="0" x="0" height="15" width="10">
                <rect y="3" x="-3" height="3" width="6">
                <rect y="9" x="-3" height="3" width="6">
        </g>
</g>
```

7.4 Sestavení diagramu komponent

Generování SVG kódu grafu probíhá jako součást zpracování dat přijatých ze serveru. Přijatá datová struktura (formát těchto dat je popsán v podkapitole 6.2) je převedena na JavaScriptový objekt, který slouží jako podklad pro tvorbu grafu a jeho manipulaci. Vytvoření diagramu komponent probíhá ve čtyřech fázích. Tyto fáze jsou zachyceny na obrázku 7.1.



Obrázek 7.1: Diagram aktivit – sestavení diagramu komponent

Po přijetí dat ze serveru dochází k vytvoření objektů, které reprezentují jednotlivé uzly. Současně s vytvářením těchto objektů dochází k sestavovaní SVG kódu pro jednotlivé uzly zastupující komponenty v diagramu. SVG kód je ukládán do *string bufferu*.

Po vygenerování všech objektů uzlu dojde k vytvoření objektů, jež zastupují hrany.

Stejně tak, jako tomu bylo v předchozím případě dochází k sestavení SVG kódu jednotlivých hran. SVG kód je též ukládán do *string bufferu*.

Když jsou vygenerovány všechny uzly a hrany včetně SVG kódu, je tento kód přidán do struktury HTML, konkrétně pod s ID viewport. Sestavení SVG reprezentující hrany a uzly je popsáno výše. Vkládané SVG je mnohonásobně větší než tento , proto mu byly pomocí stylů přidány posuvníky, které zajistí pohyb po celém SVG.

V případě, že jsou již jednotlivé elementy SVG vloženy do HTML, dojde k zaregistrování všech událostí na jednotlivé uzly a hrany. Registrování událostí není možné provádět dříve, nežli jsou všechny elementy vložené do struktury HTML.

7.5 Pohyb s uzly

Každý uzel má zaregistrovanou událost na stisknutí levého tlačítka myši (*mousedown*), která je spřažena s obslužnou funkcí *vertexMousedownHandler(event)*. Tato funkce zajistí nastavení vlajky, která značí, že je stisknuté tlačítko, a zaregistruje uzlu nové události reagující na pohyb myši a na puštění tlačítka. I k těmto událostem jsou přiřazeny obslužné funkce, které budou vykonávány po jejím vyvolání.

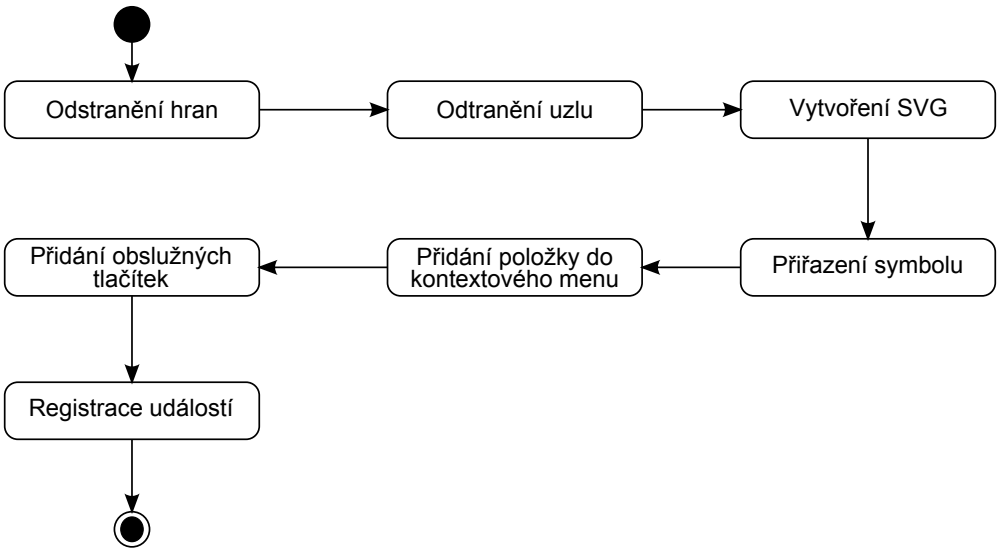
Pokud je tlačítko stále stisknuté a dochází k pohybu myši, je volána obslužná funkce *viewportMousemoveHandler(event*), která na základě předchozí pozice myši a pozice aktuální vypočte, o kolik a jakým směrem bude uzel posunut. Důležité je, že s uzlem jsou zároveň přesouvány i všechny hrany na něj navázané. Nedochází tedy jen k přepočítávání pozice uzlu, ale dochází také k přepočtení pozice jednotlivých hran a úhlu natočení symbolů "lízátka"a "mističky".

Pohyb uzlu je ukončen po puštění tlačítka myši, kdy dojde vyvolání události (*mouseup*), jejíž obslužná funkce *viewportEndDraggingHandler(event)* zajistí nastavení vlajky symbolizující, že je stisknuto tlačítko, do původního stavu. Také dojde k odregistrování události na *mousemove* a to z toho důvodu, aby nedocházelo ke zbytečnému porovnávání, zda je stisknuto tlačítko myši.

7.6 Odebírání uzlů z oblasti diagramu

Pokud je zvolen mód pro odebírání uzlů z oblasti diagramu do komponentové oblasti, která je zde označována jako pravý panel, má každý uzel zaregistrovánu událost na kliknutí, jež je svázána s obslužnou funkcí *detailHandler(event)*.

Po vyvolání této události bude vykonáno několik kroků, které zajistí odstranění uzlu z diagramové oblasti a jeho následné zobrazení v pravém panelu. Tyto kroky jsou zachyceny na obrázku 7.2.



Obrázek 7.2: Diagram aktivit – odebírání uzlů z oblasti diagramu

První krok, který musí být vykonán, je odstranění hran z oblasti diagramu. Hrany nejsou však smazány fyzicky, ale jsou pouze skryty pomocí frameworku jQuery a jeho funkce *hide()*, která zajistí, že daným elementům bude nastaven CSS styl (*style="display: none;"*).

Tento způsob byl zvolen primárně kvůli lepší uživatelské orientaci, protože při smazání odebraného prvku z DOMu by došlo ke ztrátě informace o jeho poslední pozici. Tato pozice by se musela uchovávat kvůli opětovnému zobrazení hrany v diagramu. Dalším důvodem k tomuto řešení přispěl fakt snížení výkonu způsobený mazáním a opětovným přidáním do DOMu HTML. Tento princip nastavování stylu, který element skryje, byl využit i pro odstranění uzlu.

Po skrytí uzlů a hran dojde k vytvoření samostatného elementu <svg>, který zastupuje vyjmutý uzel. SVG kód je opět ukládán do string bufferu, jehož obsah je poté vložen do struktury HTML, konkrétně do elementu nesoucí ID *rightPanel*. Ukázka SVG elementu reprezentující vyjmutý uzel ve struktuře HTML je na následující ukázce.

```
<svg id="d2" version="1.1" xmlns="http://www.w3.org/2000/svg">
  <g id="component2" class="colorNormal vertex"
   transform="translate(60,10)">
     <g id="detailVertex2" class="detail" data-vertexid="2">
        <rect x="-10" y="0" width="150" height="40" rx="4" ry="4">
        <text x="-5" y="20">ParkovisteSpringD...</text>
     </g>
     <g id="required2" class="whiteColor">
        <line stroke="black" y2="5" x2="-30" y1="5" x1="-10">
        <path stroke="black" d="M-42,-5 C-27,-5 -27,15 -42,15">
        <text id="lolipA" y="5" x="-42">2</text>
      </g>
     <g id="provided2" class="whiteColor">
        <line stroke="black" y2="35" x2="-30" y1="35" x1="-10">
        <circle stroke="black" r="11" cy="35" cx="-42">
        <text id="lolipB" y="35" x="-42">2</text>
     </g>
  </g>
</svg>
```

Jak je patrné z předchozí ukázky, SVG každé vyjmuté komponenty se skládá z obdélníku obsahující část názvu komponenty a symbolu pro rozhraní („mistička“), která jsou komponentou vyžadována, a symbolu rozhraní („lízátko“), která jsou komponentou poskytována.

Po vytvoření a vložení SVG do struktury HTML dojde k přiřazení symbolu danému uzlu. Tento symbol je využit k prezentaci vyjmutého uzlu v oblasti diagramu. Princip přiřazování symbolů daným uzlům je popsán níže.

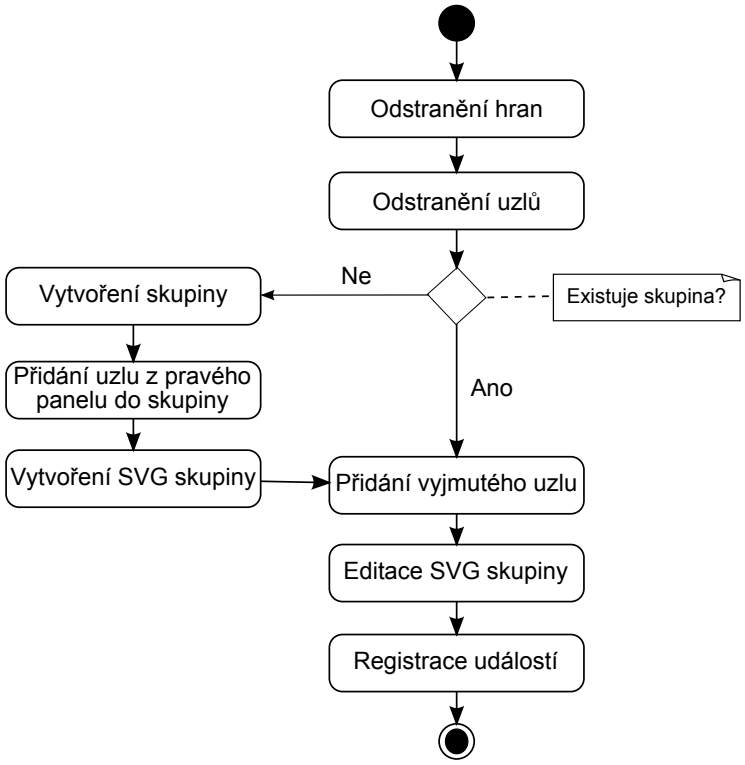
Další nutný krok při odebírání uzlu z oblasti diagramu je vytvořit položku v kontextovém menu, která reprezentuje odebraný uzel. Kontextové menu, jak již bylo zmíněno v kapitole 6, je využíváno k přidávání prvků do skupin.

Ke každé vyjmuté komponentě jsou přiřazena dvě tlačítka, která budou zajišťovat zobrazování delegátů u sousedících komponent a vrácení dané komponenty zpět do oblasti diagramu. Vytvořené SVG a dvě tlačítka jsou umístěna do elementu , které je tvořeno podle vzoru *„c“ + ID uzlu*.

Posledním krokem procesu odebrání uzlu z oblasti diagramu je registrace událostí na elementy, které byly vloženy do struktury HTML.

7.7 Vytváření skupin a přidávání prvků do skupin

Vytvoření skupiny či přidání prvku do již existující skupiny je realizováno po kliknutí na položku v kontextovém menu. Uzel, nad kterým bylo kontextové menu vyvoláno, bude přidán do skupiny. Pokud skupina ještě není vytvořena, dojde k jejímu vytvoření a prvek do ní bude následně přidán. Posloupnost kroků nutných pro vytvoření skupiny či přidání prvku do skupiny je znázorněna diagramem, který je na obrázku 7.3.



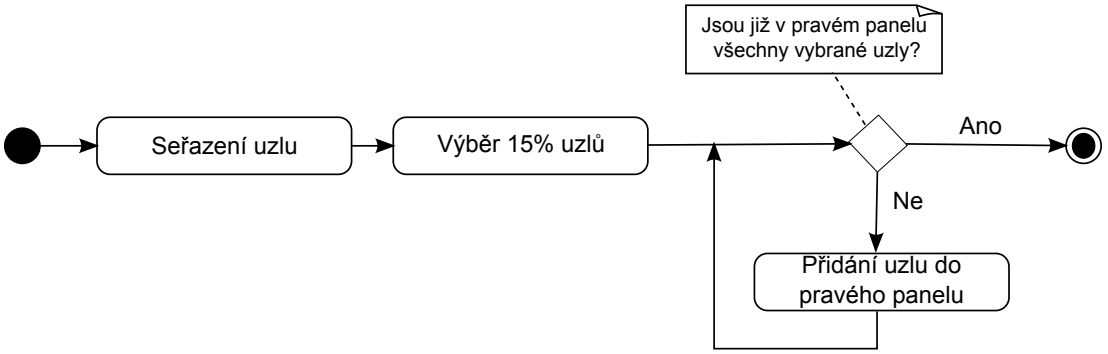
Obrázek 7.3: Diagram aktivit – vytváření skupin a přidávání prvků do skupin

Stejně jako tomu bylo u odebírání uzlu z oblasti diagramu, musí být skryty SVG elementy reprezentující uzel a hrany z něj vedoucí. Skrytí elementů funguje na stejném principu, který je popsán výše. Po odebrání prvku z grafu je testováno, zda skupina, do které má být uzel přidán, již existuje. Pokud skupina existuje, dojde k přidání prvku do skupiny a k následné editaci SVG, které reprezentuje skupinu. V případě, že skupina ještě není vytvořena, je nutné skupinu vytvořit, přidat do ní prvek, který je vyjmutý v pravém panelu a následně sestavit SVG, které bude tuto skupinu vizuálně zastupovat. Správu nad skupinami zajišťuje třída *GroupManager*.

Skupina bude zastupována symbolem a barvou prvku, který byl již vyjmutý. Po vytvoření skupiny do ní bude přidán uzel, nad kterým bylo vyvoláno kontextové menu. Dále musí dojít k aktualizaci velikosti SVG reprezentující skupinu. Po přidání prvku do skupiny a vykreslení SVG je nutné zaregistrovat události nově přidanému prvku.

7.8 Hromadný přesun uzlů – zobrazeny jednotlivě

Pro hromadný přesun uzlů z oblasti diagramu je nutné zajistit několik kroků. Tyto kroky jsou zachyceny diagramem na obrázku 7.4. V tomto případě budou uzly v pravém panelu zobrazeny samostatně.



Obrázek 7.4: Diagram aktivit – hromadný přesun uzlů – zobrazeny jednotlivě

První provedený krok zajistí seřazení všech uzlů podle počtu hran na něj navázaných. Když jsou uzly seřazené, dojde k vybrání patnácti procent nejvíce zatížených uzlů. Poté jsou tyto uzly procházené, postupně vyjímané z oblasti diagramu a přidávány do pravého panelu. Princip přidávání prvků do pravého panelu je popsán v podkapitole 7.6.

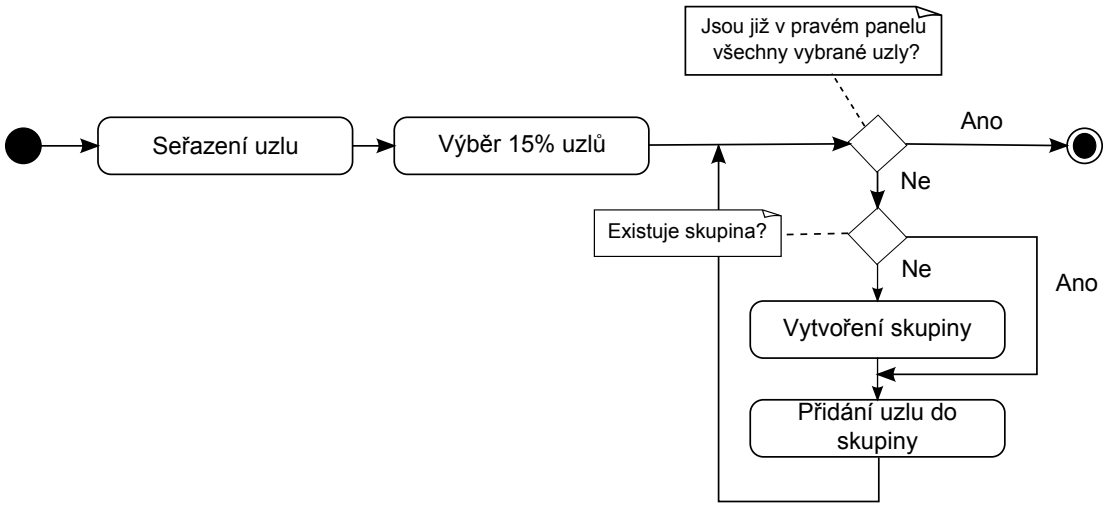
7.9 Hromadný přesun uzlů – zobrazeny skupinově

Tento princip je velmi podobný jako v předchozím případě. Jediný rozdíl je, že uzly nebudou zobrazovány v pravém panelu samostatně, ale budou sloučeny do jedné skupiny. Posloupnost jednotlivých kroků je naznačena diagramem na obrázku 7.5.

V tomto případě je také nutné uzly seřadit podle počtu hran, které jsou na uzly navázané, a vybrat z nich patnáct procent uzlů, jež budou odebrány. Při prvním průchodu uzlů dojde k vytvoření skupiny, do které budou následně jednotlivé uzly přidávány. Postup vytváření skupin je popsán v podkapitole 7.7.

7.10 Přidělování symbolů

Každý symbol, který je přiřazován vyjmutému uzlu, se skládá z jednoho znaku a barvy (dále jen symbol). O přidělování symbolů, které jsou jedinečné, jednotlivým kom-



Obrázek 7.5: Diagram aktivit – hromadný přesun uzlů – zobrazeny skupinově

ponentám se stará třída *MarkSymbol*.

V případě, že je komponenta, jež má přidělený symbol, odebrána z pravého panelu, je tento symbol uložen do zásobníku. Při přiřazování symbolu další komponentě je nejprve zkontrolováno, zda zásobník obsahuje nějaký symbol. Pokud zásobník symbol obsahuje, tak je této komponentě přiřazen, v opačném případě jí je přidělen symbol nový.

7.11 Zoomování

V grafickém uživatelském rozhraní jsou dvě tlačítka umožňující zvětšování či zmenšování. Na tyto tlačítka jsou zaregistrovány události na kliknutí. Klikáním na tlačítko se symbolem *lupy s mínusem* bude volána funkce *zoomOut()*, která zajišťuje zmenšování oblasti diagramu i jednotlivých uzlů a hran. Při klikání na tlačítko se symbolem *lupy s plusem* bude volána funkce *zoomIn()*, jež umožňuje zvětšování.

Pro zoomování jsou nadefinované stavy, kterých může být dosaženo. Obraz lze zmenšit nejméně na 10% a zvětšit na 500%. Po dosažení těchto hranic dojde k deaktivaci tlačítek, aby uživatel byl upozorněn, že je již na jedné z krajních hranic. Jednotlivé hodnoty procent, kterých může být při zoomování dosaženo jsou 10, 25, 40, 50, 60, 70, 80, 90, 100, 125, 150, 200, 300, 400, 500. Ke zmenšení nebo zvětšení elementů SVG, byla využita možnost škálování prvků přímo z SVG. Toto bylo zajištěno funkcí *scale(hodnota zvětšení)*, jejíž parametr určuje koeficient zvětšení či zmenšení.

7.12 Vyhledávání

K vyhledávání komponent dojde v případě, že uživatel zadá hledaný podřetězec a stiskne tlačítko se symbolem lupy. Kliknutím na tlačítko bude vyvolána událost, jež je svázána s funkcí *search()*. Při vyhledávání je nejprve ověřeno, zda vyhledávací pole neobsahuje prázdný řetězec. Hodnotu si z vyhledávacího pole získáme funkcí *val()*. Jestliže je hodnota prázdný řetězec, vyhledávání nebude uskutečněno a bude vytvořen informující popisek (angl. tooltip) pomocí *qTip2*. Následující ukázka přibližuje jeho využití.

```
\$("#searchText").qtip({
    content: {
       text: "Please fill out this field."
    },
    style: {
       classes: "ui-tooltip-green ui-tooltip-rounded ui-tooltip-shadow"
    }
});
```

Jestliže byl řetězec zadán, budou procházeny všechny uzly zobrazené uvnitř diagramové oblasti a hledán tento podřetězec v názvu komponenty. V případě, že název komponenty bude obsahovat hledaný řetězec, bude uzel reprezentující komponentu zvýrazněn oranžovou barvou. Po dokončení prohledávání bude na stránce zobrazen počet nalezených komponent.

7.13 Zobrazování delegátů

Delegáti jsou zobrazováni u sousedů dané komponenty po vyvolání události na kliknutí nad tlačítkem se symbolem. Tato událost je svázána s funkcí *displayIconEvent()*. Po vyvolání této události dojde nejprve k podbarvení tlačítka, na které bylo kliknuto. Elementu reprezentující tlačítko bude nastavena CSS třída, která zajistí jeho podbarvení. Způsob nastavení stylu k elementu prostřednictvím funkce *attr()* je na ukázce.

```
\$('#id').attr('class', 'název CSS třídy');
```

Zobrazování delegátů u komponent s sebou přináší několik problémů. Jedním z nich je zobrazování více delegátů u jedné komponenty a jejich následné odebírání, což způsobuje vznik děr. Dalším problémem je organizace delegátů u komponent, jež mají krátký název,

protože nemají dostatek místa pro jejich zobrazování. Tyto problémy jsou vyřešeny tak, že delegáti jsou skládáni do mřížky.

V této implementaci je mřížka v y-ové souřadnici neomezená, ale v x-ové je omezena na zobrazení pěti delegátů. Skládání delegátů do mřížky umožňuje vyplňování děr, které vzniknou při odebrání některých delegátů, a také zobrazovat jejich libovolný počet. Princip je tedy takový, že po odebrání jakéhokoli delegáta, dojde k přeorganizování mřížky a zaplnění vzniklého místa. Nový delegát bude zařazen vždy na konec mřížky. Delegáti jsou v SVG reprezentováni skupinou <g>, která se skládá z elementů <rect> a <text>.

Přidávání, odebírání a reorganizaci delegátů v mřížce zajišťuje třída *GridMarks*, jejíž instanci obsahuje každý uzel.

7.14 Zvýrazňování sousedů

Na symboly „lízátko“ a „mistička“, jsou zaregistrovány události na kliknutí. Kliknutím na symbol "lízátka"bude vyvolána obslužná funkce *highlightProvidedEvent(event)*, která zajistí zvýraznění všech sousedů dané komponenty, které od ní požadují nějaká rozhraní.

Komponentám, jež sousedí s danou komponentou, bude pomocí funkce *attr()* nastavena CSS třída, která zajistí přebarvení uzlu reprezentující komponentu na zelenou barvu. Tato barva bude nastavena stejným způsobem i elementu SVG, který zastupuje symbol „lízátka“. Použití funkce *attr()* je popsáno výše.

Dále dojde k přeregistrování událostí na kliknutí, aby opětovným kliknutím mohlo dojít ke zrušení zvýraznění. Zrušení zvýraznění funguje na stejném principu jako u zvýrazňování. Kliknutím na symbol „mističky“ bude vyvolána obslužná funkce, v tomto případě však *highlightRequiredEvent(event)*, která zajistí zvýraznění všech sousedů dané komponenty, které jí nějaká rozhraní poskytují. Princip zvýrazňování je stejný jako v předchozím případě. Pouze sousedící komponenty a symbol „mističky“ budou označeny žlutě.

Zvýrazňování sousedů skupiny je realizováno stejným způsobem jako u zvýrazňování sousedů jednotlivých komponent. V tomto případě je zde malý rozdíl. Je nutné zobrazit sousedy všech komponent nacházejících se v dané skupině.

7.15 Zvýrazňování hran

Zvýrazňování hran je založeno na principu změny CSS stylu po kliknutí na symboly „lízátka“ a „mističky“. Společně se zvýrazněním hrany dochází také ke zvýraznění uzlů, jež jsou hranou propojeny. Změna CSS stylů je zajištěna funkcí *css()*, jež je součástí frameworku jQuery. Následující ukázka demonstruje použití funkce *css()*. Styly jsou nastavované přímo SVG elementů.

```
\$("#id").css("stroke", "red");
```

Zvýraznění je zrušeno po opětovném kliknutí na symboly „lízátka“ a „mističky“. Dojde opět ke změně stylů pomocí *css()*.

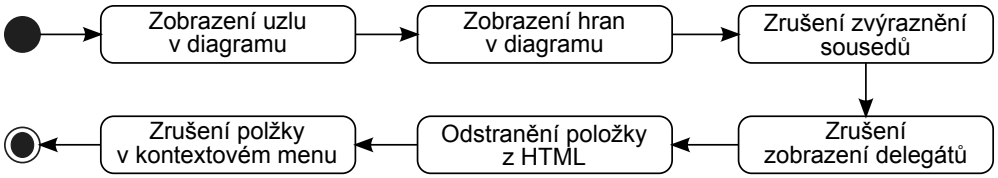
Při kliknutí na hranu nedojde pouze k zvýraznění hran a komponent hranou spojených, ale bude zobrazen popisek informující, přes která rozhraní jsou komponenty spojené. Tento popisek je realizován pomocí pluginu *qTip2*, který umožňuje popisky přidávat i na elementy SVG. Příklad použití *qTip2* je uveden výše.

7.16 Vracení komponent do oblasti diagramu

Komponenty zobrazené v pravém panelu mohou být do oblasti diagramu vráceny dvěma způsoby. Záleží na tom, zda odebíraný prvek je v pravém panelu zobrazen samostatně, nebo zda je součástí skupiny.

7.16.1 Odebrání samostatné komponenty z pravého panelu

Odebrání samostatné komponenty v pravém panelu a její zobrazení v oblasti diagramu vyžaduje provedení několika kroků. Jednotlivé kroky jsou zachyceny na diagramu (viz obrázek 7.6).



Obrázek 7.6: Diagram aktivit – odebrání samostatné komponenty z pravého panelu

V tomto případě dojde nejprve k zobrazení uzlu. K jeho zobrazení je využita funkce *show()*, která je součástí frameworku jQuery. Tato funkce pouze zruší předchozí nastavení stylu (*display:none*), který zajistil skrytí komponenty. Její použití je znázorněno na ukázce.

```
\$("#id").show();
```

Tato funkce je použita i pro zobrazení hran. Dalším krokem je zrušení zvýraznění sousedů komponenty. Zvýraznění je zrušeno nastavením jiné CSS třídy. Postup, jak změnit CSS třídu je uveden výše. Pokud byli zobrazeni delegáti u sousedů této komponenty, musí také dojít k jejich odebrání. O odebrání delegátů se stará třída *GridMarks*. Po vykonání všech těchto kroků je , který obaloval všechny prvky položky v pravém panelu, odstraněn ze struktury HTML. Pro smazání elementu je použita funkce *remove()*, která odstraní i všechny prvky v něm obsažené. Použité této funkce je demonstrováno na následující ukázce.

```
\$("id").remove();
```

Posledním krokem je odstranění symbolu a barvy reprezentující prvek v kontextovém menu. Posloupností těchto kroků je zajištěno, že vše bude ve stejném stavu, jako tomu bylo před vyjmutím komponenty.

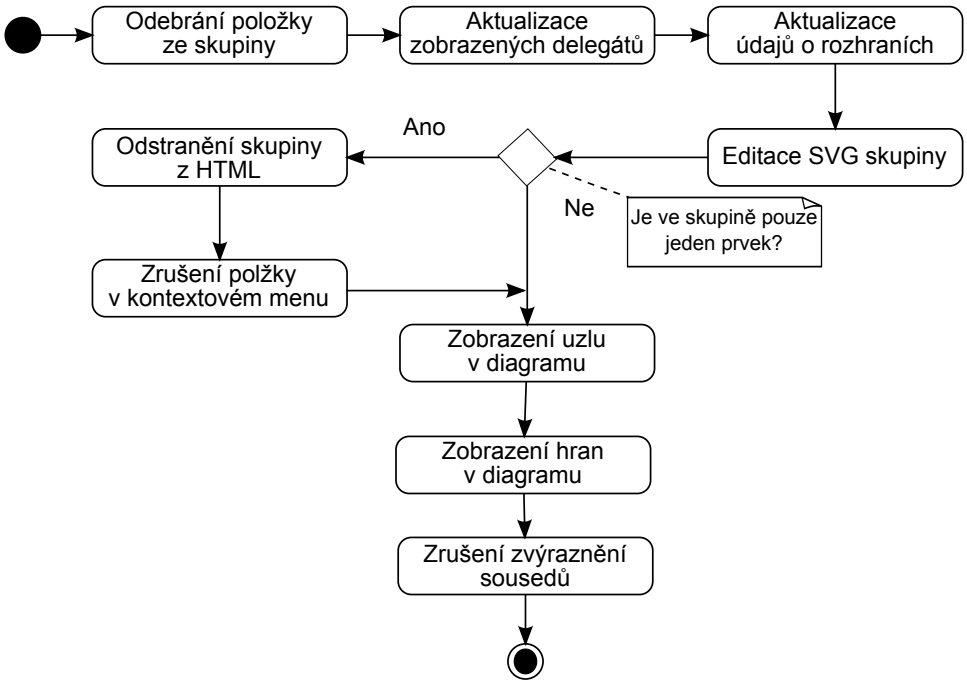
7.16.2 Odebrání komponenty ze skupiny

Odebrání komponenty ze skupiny vyžaduje, stejně tak jako tomu bylo v předchozím případě, vykonání posloupnosti několika kroků. Diagram na obrázku 7.7 zachycuje kroky potřebné pro odebrání komponenty ze skupiny. Tyto kroky zajistí, že vše bude jako před přidáním prvku do skupiny.

V tomto případě dojde nejprve k odebrání komponenty ze seznamu, který je zobrazen vedle SVG obrázku. K odebrání je opět využita funkce *remove()*. Dále je nutné aktualizovat zobrazené delegáty, což znamená, že u komponent, jež sousedí s odebíranou komponentou, budou delegáti smazáni. Nutná je také aktualizace počtu poskytovaných a vyžadovaných rozhraní. Po odebrání položky se seznamu musí být také editována velikost SVG skupiny.

Pokud dochází k odebrání posledního prvku ve skupině, musí být skupina odstraněna ze struktury HTML a také z nabídky kontextového menu.

Poslední nutné kroky pro dokončení odebrání komponenty ze skupiny jsou zobrazení uzlů a hran pomocí funkce *show()* a také zrušení zvýraznění u komponent sousedící



Obrázek 7.7: Diagram aktivit – odebrání komponenty ze skupiny

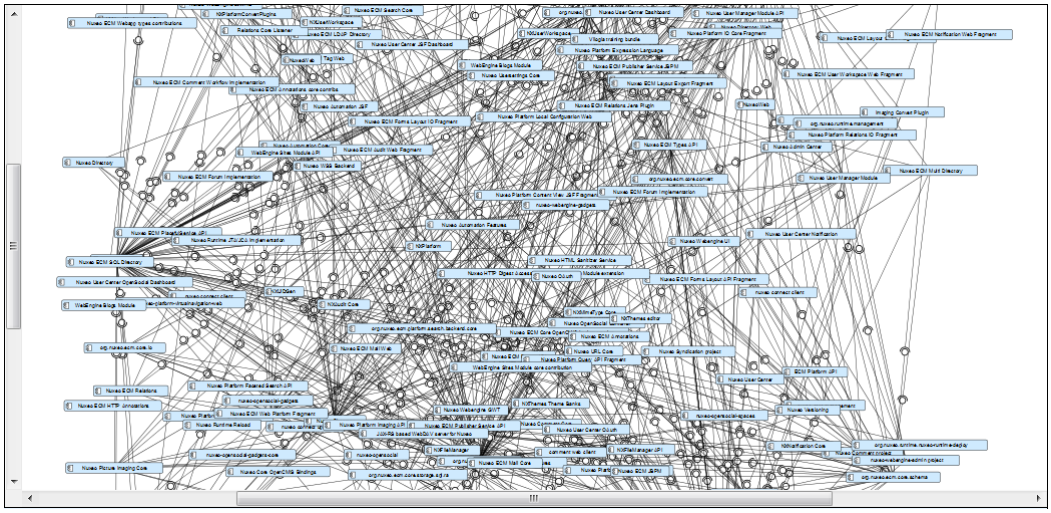
s odebíranou komponentou. Zvýraznění u sousedících komponent je realizováno stejným způsobem, jako tomu bylo u odebírání samostatné komponenty.

8 Demonstrace možností vizualizačního nástroje

Tato kapitola je věnována testování a demonstraci funkčnosti vizualizačního nástroje. Tento vizualizační nástroj byl testován ve webovém prohlížeči Firefox verze 11.0 a za použití testovacích komponentových aplikací Nuxeo, Openwms, Eclipse aj.. Tyto aplikace kromě Eclipse jsou součástí přiloženého CD a nachází se v adresáři Testovací data. Komponentová aplikace Nuxeo se skládá z 203 komponent a aplikace Openwms z 65 komponent.

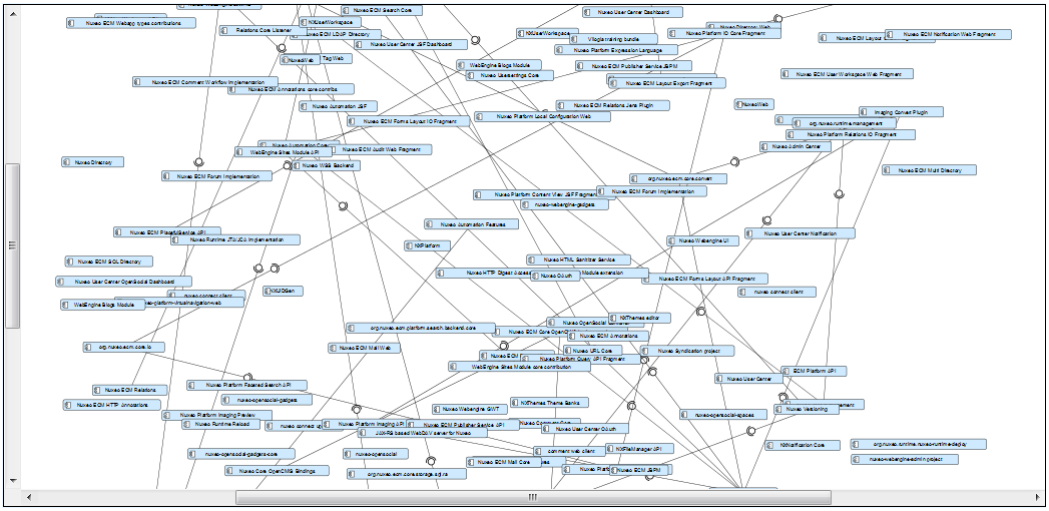
Takto rozsáhlá množina prvků tvořící danou aplikaci umožnila důkladné otestování nástrojů, jež byly výstupem této práce a jejichž cílem je umožnit zvýšení přehlednosti a orientace v diagramech tvořených na sobě závislými komponentami. Diagramy tvořené z dat takového rozsahu si však kladou relativně vysoké nároky na výkon.

Toto technické omezení bylo částečně obejito možností odebrání 15% komponent, jež jsou spojeny s velkým množstvím dalších komponent. Tato technika se v tomto vizualizačním nástroji velmi osvědčila. Po odebrání těchto uzlů, došlo k výraznému zpřehlednění vizualizovaných dat. Obrázek 8.1 ukazuje stav okna vizualizace před aplikací zmíněné techniky. Další obrázek 8.2 již zachycuje stav stejného okna po nasazení techniky zmírňující nároky na výkon.



Obrázek 8.1: Stav okna před odebrání prvků

Demonstrační ukázky funkčnosti zbývajících technik vizualizačního nástroje byly vytvořeny při vizualizaci aplikace Openwms. Tyto ukázky byly pro jejich značný rozsah pře-



Obrázek 8.2: Stav okna po odebrání prvků

sunuty do přílohy B.

Tento vizualizační nástroj byl také testován na aplikaci *Eclipse for RCP and RAP Developers* verze *Indigo Release*, id buildu 20110615-0604. Při testování nástroje na aplikaci Eclipse byla zjištěna chyba, jež se projevovala zobrazováním diagramu bez jakýchkoli spojení mezi komponentami. Aplikace Eclipse totiž obsahuje komponentu (*org.eclipse.jdt.launching_3.6.1.v20110803_r371.jar*), jejíž velikost je nulová. Pro správné fungování je nutné ze seznamu nahraných souborů tuto komponentu odstranit.

9 Závěr

Cílem této práce bylo vytvoření nástroje pro vizualizaci rozsáhlých diagramů komponent a umožnění uživateli interakci s prvky, kterými jsou tyto diagramy tvořeny. Zároveň uživateli usnadnit orientaci a proniknutí do komplexnosti zobrazovaných komponentových aplikací využitím off-screen technik. Dalším požadavkem na vytvářený nástroj bylo jeho propojení s aplikací ComAV, jež je vyvíjena na Katedře informatiky a výpočetní techniky Západočeské univerzity.

V průběhu příprav teoretické části této práce bylo rozhodnuto, že nástroj bude implementován jako webová aplikace. Hlavní výhodou tohoto řešení je snadná dostupnost a odstranění nutnosti distribuce nových verzí mezi všechny uživatele, jež jsou tímto oproštěni od potřeby stahovat nové verze aplikace. Prostředkem k dosažení požadovaných vlastností vyvíjeného nástroje bylo účelné využití off-screen technik, jež byly navrhnuty pro usnadnění práce s rozsáhlými diagramy komponent. Využité techniky byly zvoleny na základě jejich slučitelnosti s požadavky kladenými na tento nástroj. Rozbor zmiňovaných technik a zdůvodnění jejich výběru je součástí této práce.

Výsledkem této práce je nástroj, jež umožňuje vizualizaci rozsáhlých diagramů komponent. Zároveň poskytuje sadu technik, z nichž některé byly navrženy na Katedře informatiky a výpočetní techniky. Tyto techniky zlepšují přehlednost odstraňování jednotlivých prvků z diagramu do zvlášť vymezené oblasti mimo samotný diagram.

Literatura

[1] KIV. *Úvod do komponent.* Dostupné na: http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage.

[2] BROY, Manfred. et al. *What characterizes a (software) component?* Software—Concepts and Tools 19 (1998) 49–56.

[3] SZYPERSKI, Clemens. et al. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[4] NIEKAMP, Rainer. *Software Component Architecture.* Institute for Scientific Computing. TU Braunschweig, 2006. Dostupné na: http://congress.cimne.upc.es/cfsi/frontal/doc/ppt/11.pdf

[5] BACHMANN, Felix. et al. *Volume II: Technical concepts of component-based software engineering.* 2000. Carnegie Mellon University, Software Engineering Institute.

[6] VALENTA, Lukáš; BRADA, Přemysl. *Modelování existujících OSGi komponent.* Plzeň, 2006. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[7] The OSGi Alliance. *OSGi Service Platform Core Specification, Release 4.* Duben 2007, dostupná na http://www.osgi.org/.

[8] STUNA, Pavel. *Ověřování nahraditelnosti EJB komponent.* Plzeň, 2005. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[9] ČERNÝ, Ondřej; HOŠEK Petr; PAPEŽ Michal a REMEŠ Václav. *SOFA 2 Component System: User's Guide.* Dostupné na: http://sofa.ow2.org/docs/pdf/users_guide.pdf.

[10] ŠNAJBERK, Jaroslav; BRADA, Přemysl. *COMAV – A component application vizualization tool, Use of Reverse Engineering and Interactivity in Visualization for Component Software Comprehension.* Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[11] ŠNAJBERK, Jaroslav; BRADA, Přemysl. *ENT: A Generic Meta-Model for the Description of Component-Based Applications.*2011. Electronic Notes in Theoretical Computer Science, 279(2):59 – 73. Proceedings of the 8th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA).

[12] COCKBURN, A.; KARLSON, A.; BEDERSON, B.B. *A review of overview+detail, zooming, and focus+context interfaces.* ACM Computing Surveys 41 (2009) 2:1 - 2:31.

[13] IRANI, P.; GUTWIN, C.; PARTRIDGE, G. A NEZHADASL, M. *Techniques for Interacting with Off-Screen Content.* 2007. In Proceedings of INTERACT 2007. 4663:234-249, Springer Berlin / Heidelberg.September 07. Dostupné na: http://www.springerlink.com/content/j202g44683748810/.

[14] SARKAR, M.; BROWN, M. *Graphical fisheye views of graphs.* In SIGCHI, pages 83–91, 1992.

[15] HOLÝ, Lukáš.; BRADA, Přemysl. *Viewport for Component Diagrams.* Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Graph Drawing 2011: 443-444.

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

[17] PAVLÍKOVÁ, Jindra. *Frameworky vhodné pro off-screen vizualizaci rozsáhlých diagramů.* Nepublikováno. Plzeň, 2011. Oborový projekt. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

[18] KUNEŠ, Daniel. *Webové technologie.* Nepublikováno. Plzeň, 2011. Oborový projekt. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky.

Příloha A – uživatelská dokumentace

Nasazení aplikace

Pro spuštění této aplikace je nutné nainstalovat *Apache Tomcat*, který je dostupný na přiloženém CD v adresáři Instalace nebo na webové adrese *http://tomcat.apache.org/download-70.cgi*. Dále je předpokládána následující konfigurace:

- JRE 1.6 nebo vyšší verze
- Windows Vista nebo Windows 7
- Firefox 11.0

Po nainstalování *Apache Tomcat* je potřeba nahrát soubor *VisualizationTool.war*, který je umístěn též na přiloženém CD v adresáři *Vizualizační nástroj*, pod adresář *../Apache Software Foundation/Apache Tomcat 7.0.22/webapps*. K vizualizačnímu nástroji lze pak přistupovat z webového prohlížeče na adrese *localhost:8080/VisualizationTool*.

Nastavení cesty k úložišti se provede editací parametru *storageLocation* v konfiguračním souboru *WEB-INF/configuration.properties*. Výchozím nastavením je adresář *D:/Uloziste*. V případě, že adresář neexistuje, bude na disku vytvořen.

Ovládání aplikace

V této podkapitole bude podrobně popsáno, jak ovládat tento vizualizační nástroj. Nejprve budou upřesněny termíny, které budou v následujícím textu používány:

- **úvodní obrazovka** – obrazovka možností nahrání komponent na server a výběrem typu komponent

- **hlavní obrazovka** – obrazovka se zobrazeným diagramem komponent
  - **nástrojová lišta** – lišta, která se rozpíná v horní části obrazovky a která poskytuje nástroje pro ovládání některých funkcionalit vizualizačního nástroje
  - **oblast diagramu** – oblast, ve které je zobrazen diagram komponent
  - **pravý panel** – panel s popisem *Excluded component*, ve kterém jsou zobrazovány vyjmuté komponenty

Ovládání úvodní obrazovky

Po spuštění aplikace je zobrazena úvodní obrazovka, kde jsou zobrazeny dva formuláře. První formulář slouží k nahrávání komponent. Kliknutím na tlačítko *Procházet* bude vyvoláno okno, prostřednictvím kterého bude možné vybrat komponenty pro následnou vizualizaci. Když budou komponenty vybrány, je nutné kliknout na tlačítko *Upload*. Po této akci budou zobrazeny v přehledném výpisu jména nahraných komponent. Vedle každé komponenty je umístěna ikona křížku. Kliknutím na tuto ikonu bude daná komponenta odstraněna. Stisknutím tlačítka vedle popisu *Delete all*, dojde ke smazání všech nahraných komponent. Po nahrání komponent na server je možné přidávat další komponenty stejným způsobem, jak bylo popsáno výše.

Další krok nutný pro vizualizaci je výběr typu komponent, který je zprostředkován druhým formulářem nacházejícím se na stránce . Na výběr je OSGi, EJB nebo SOFA2. Pokud jsou komponenty nahrané a vybrán jejich typ, je možné kliknout na tlačítko *Start visualization*, které zajistí zobrazení stránky s diagramem komponent.

Ovládání hlavní obrazovky

V této části budou popsány všechny nástroje a možnosti, které vizualizační nástroj uživateli nabízí.

Možnosti nástrojové lišty

Na stránce s diagramem komponent je umístěna nástrojová lišta, která obsahuje několik nástrojů.

Zoomování

První nástroj v nástrojové liště umožní přibližování a oddalování diagramu. Kliknutím na ikonu *lupy s mínusem* dojde ke zmenšení obrazu a kliknutím na ikonu *lupy s plusem*

dojde naopak k jeho zvětšení. Diagram lze zmenšit až na deset procent z původní velikosti a zvětšit až na pět set procent. Informace o aktuálním zvětšení se nachází mezi ikonou *lupy s mínusem* a *lupy s plusem*.

Vyhledávání komponent

Další nástroj umožňuje vyhledávat komponenty, které jsou zobrazené v oblasti diagramu. Jméno nebo část jména je nutné vyplnit do políčka, ve kterém je napsáno *Search component...*, a následně stisknout ikonu nesoucí obrázek lupy. Po kliknutí na tuto ikonu budou oranžově zvýrazněny komponenty, které obsahují hledaný řetězec. Protože diagram není vidět celý, je vedle ikony *lupy* informace o počtu nalezených komponent. Kliknutím na toto číslo dojde ke zrušení zvýraznění u nalezených komponent. Pokud ve vyhledávacím poli nebude nic napsáno a přesto bude kliknuto na ikonu *lupy*, bude zobrazena žádost o vyplnění tohoto pole *Please fill out this field*.

Výběr ovládacího módu

Vpravo vedle vyhledávání je umístěn nástroj, který umožňuje vybrat si ze dvou módů. Po výběru prvního z nich (*move component*) je možné pohybovat s komponentami v diagramu. Druhý (*exclude component*) umožní jejich vyjmutí do panelu s popisem *Excluded components*.

Pohyb komponent

Pokud je vybrán mode *move component*, který umožňuje pohybovat s komponentami, stačí vybrat komponentu, se kterou chceme pohybovat, stisknout a stále držet levé tlačítko myši. Komponenta bude obarvena žlutě. Pro vykonání pohybu stačí hýbat s myší. Vybraná komponenta bude sledovat její pohyb. Pohyb bude ukončen po uvolnění tlačítka myši.

Odstraňování komponent

Výběrem druhého módu *exclude component* je možné odebírat komponenty z diagramu. Odebrání komponenty se provede jedním kliknutím na danou komponentu. Komponenta po kliknutí z diagramu zmizí a objeví se v pravém panelu (*Excluded components*).

Hromadné odstraňování komponent Předposlední nástroj z nástrojové lišty umožňuje odebrání patnácti procent nejvíce zatížených uzlů. Kliknutím na ikonu dvou listů papíru budou komponenty vyjmuty z diagramu a přesunuty do pravého panelu (*Excluded components*). Komponenty se v panelu zobrazí samostatně.

Kliknutím na poslední nástroj v liště budou odebrány komponenty z největším zatížením stejně jako u předchozího způsobu. V tomto případě nebudou zobrazeny komponenty samostatně, ale budou součástí jedné skupiny.

Možnosti v oblasti diagramu

V oblasti diagramu je možné provádět různé akce.

Zobrazení informací o komponentě

Kliknutím na ikonu komponenty budou zobrazeny detailní informace o rozhraních, které poskytuje nebo naopak vyžaduje. Detailní informace zmizí odjetím z ikony komponenty.

Zvýraznění hrany

Dále je možné zvýraznit hranu mezi dvěma komponentami a to kliknutím na symbol „lízátka“ a „mističky“. Hrana společně s oběma komponentami bude zvýrazněna červeně a navíc budou zobrazeny informace o spojení těchto komponent. Zvýraznění se zruší opětovným kliknutím na symbol „lízátka“ a „mističky“.

Kontextové menu

Nad každou komponentou v diagramu lze vyvolat stisknutím pravého tlačítka myši kontextové menu. Vybráním jedné položky (symbol + barva) z kontextového menu bude komponenta, nad kterou bylo kontextové menu vyvoláno, přiřazena do skupiny, jež je tímto symbolem a barvou reprezentována. Komponenta z oblasti diagramu zmizí a objeví se u dané skupiny.

Možnosti v pravém panelu

V pravém panelu se zobrazují komponenty samostatně nebo jsou sloučené do skupin. Pokud je komponenta zobrazena samostatně, lze nad ní provádět následující akce.

Zvýrazňování sousedů

Po kliknutím na symbol „mističky“ obsahující číslo se v oblasti diagramu žlutě zvýrazní všechny komponenty, které dané komponentě poskytují nějaká rozhraní. Opětovným kliknutím na stejný symbol dojde k zrušení jejich zvýraznění.

Kliknutím na symbol „lízátka“ je možné zobrazit komponent z oblasti diagramu, které od dané komponenty nějaká rozhraní vyžadují. Komponenty jsou označené zeleně. Kliknutím na tentýž symbol se zvýraznění zruší.

Stisknutím levého tlačítka nad obdélníkem obsahující jméno komponenty je možné

zvýraznit všechny sousedící komponenty nebo naopak všechny odznačit. Tento postup je stejný i při zobrazování sousedů skupin.

Zobrazení delegátů

U každé samostatné komponenty i u skupin je tlačítko se symbolem. Kliknutím na toto tlačítko se zobrazí delegáti u sousedících komponent, které jsou zobrazené v oblasti diagramu. Delegáti budou skryti po kliknutí na stejné tlačítko.

Vrácení komponenty do oblasti diagramu

Pro vrácení komponenty z pravého panelu do oblasti diagramu slouží tlačítko s obrázkem křížku.

Příloha B – ukázky programu

V této příloze jsou k dispozici sejmuté obrazovky aplikace demonstrující její funkce. Pro vytvoření této demonstrace byla použita testovací data umístěná na přiloženém CD v adresáři *Testovací data/short openwms*.

Základní pohled na aplikaci

Na obrázku B.1 je znázorněn základní pohled na aplikaci, kde zobrazen výřez diagramu komponent.

Stav aplikace před pohybem

Obrázek B.2 zachycuje stav aplikace před pohybem s komponentou. Komponenta, se kterou bude pohybováno je zvýrazněna žlutě.

Stav aplikace po pohybu

Obrázek B.3 znázorňuje stav aplikace po dokončení pohybu s komponentou.

Stav aplikace po zmenšení diagramu

Na obrázku B.4 je zobrazena obrazovka aplikace po aplikování 80% oddálení na diagram komponent.

Stav aplikace po zvětšení diagramu

Obrázek B.5 zachycuje stav aplikace po aplikování 150% přiblížení na diagram komponent.

Stav aplikace po přesunutí komponenty do komponentové oblasti

Obrázek B.6 znázorňuje stav aplikace po odebrání komponenty z oblasti diagramu do komponentové oblasti.

Stav aplikace před přidáním komponenty do skupiny

Na obrázku B.7 je zachycen stav aplikace před vyjmutím komponenty z oblasti diagramu a přidání této komponenty do skupiny, která je reprezentována symbolem z kontextového menu.

Stav aplikace po přidání komponenty do skupiny

Obrázek B.8 znázorňuje stav aplikace po přidání komponenty do skupiny.

Stav aplikace po zobrazení delegátů

Obrázek B.9 zachycuje stav aplikace po zvýraznění všech sousedů vyjmuté komponenty. Komponenty, které dané komponentě poskytují nějaká rozhraní, jsou zvýrazněni žlutě a komponenty, které od vyjmuté komponenty nějaká rozhraní vyžadují, jsou zvýrazněni zeleně.

Stav aplikace po zobrazení delegátů

Obrázek B.10 znázorňuje stav aplikace po zobrazení delegátů u všech komponent sousedících s vyjmutou komponentou do komponentové oblasti.

Stav aplikace po vyhledání zadaného řetězce

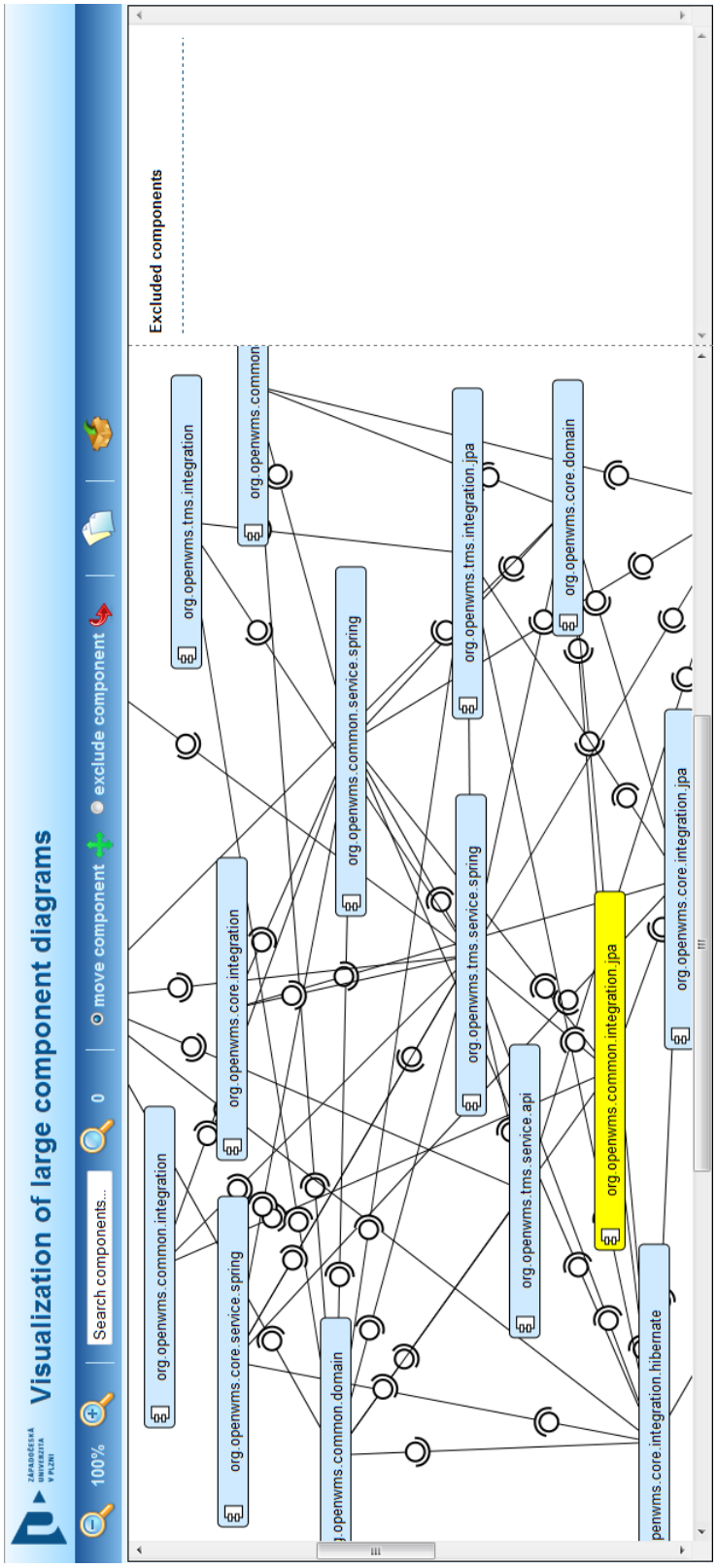
Ukázka vzhledu oblasti diagramu po vyhledání zadaného řetězce "core"je na obrázku B.11.

Stav aplikace po zobrazení aplikace Nuxeo

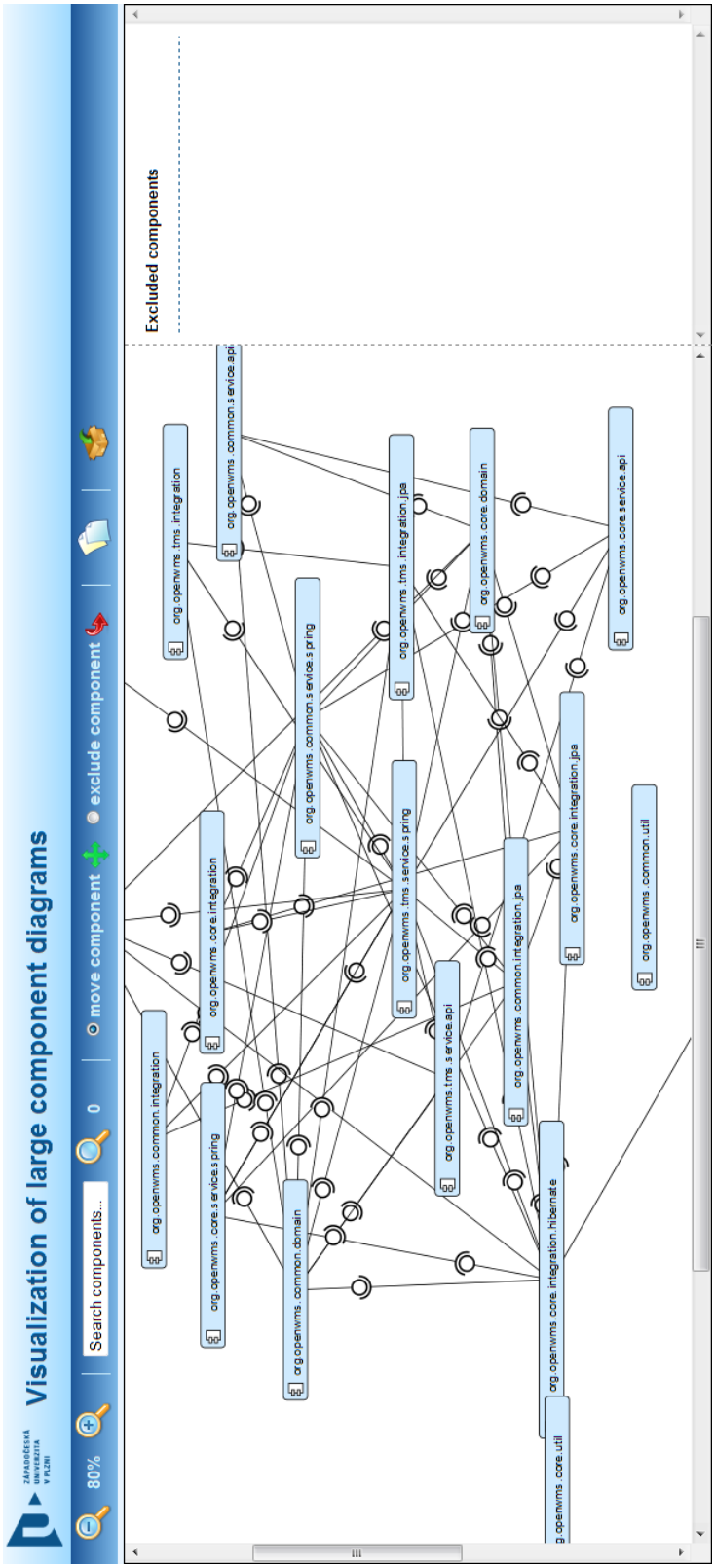
Na obrázku B.12 je vyobrazena obrazovka aplikace s vizualizovanou aplikací Nuxeo, která se skládá z 203 komponent. Obrázek také zachycuje zobrazení skupiny, samostatně vyjmuté komponenty a vyhledání komponenty obsahující řetězec ECM search.

Obrázek B.1: Základní pohled na aplikaci

Obrázek B.2: Stav aplikace před pohybem

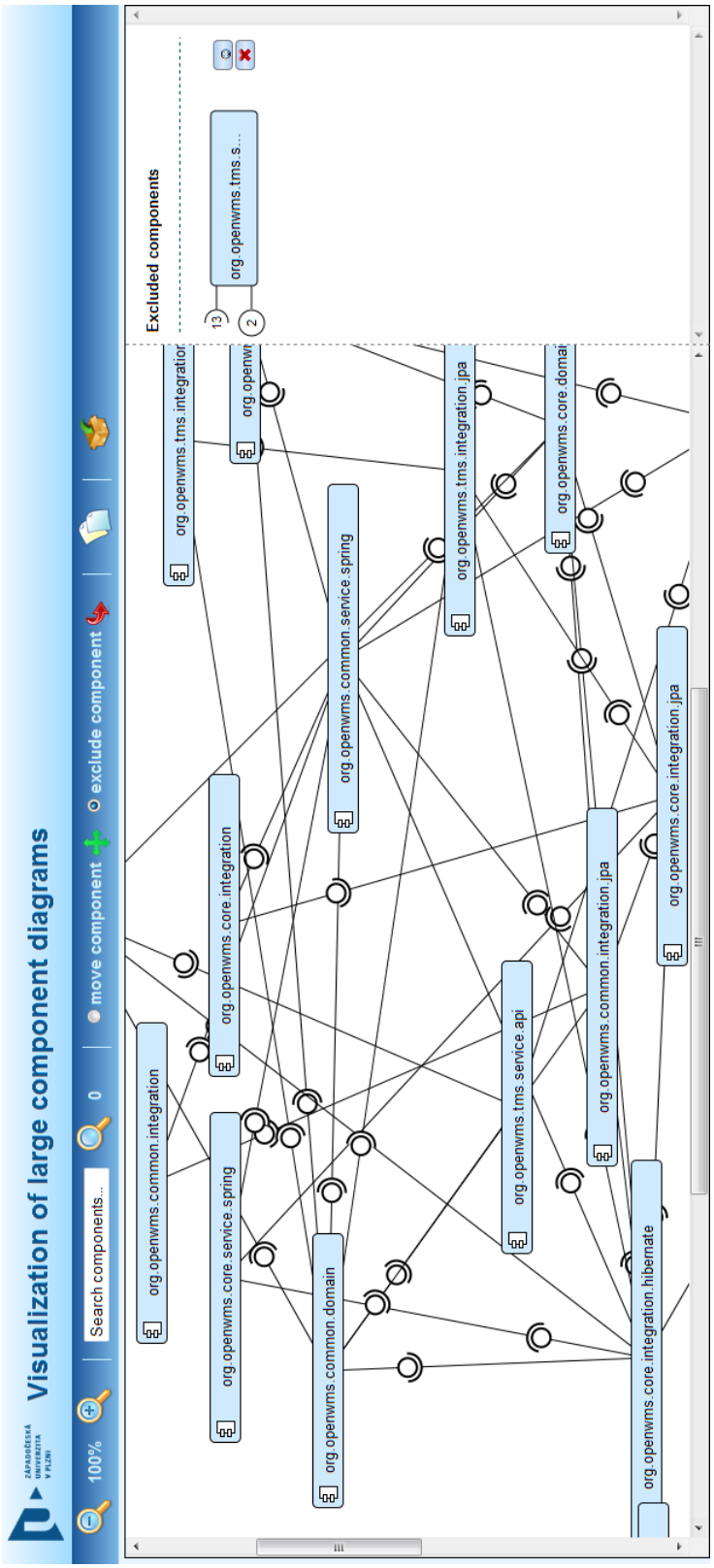


Obrázek B.3: Stav aplikace po pohybu

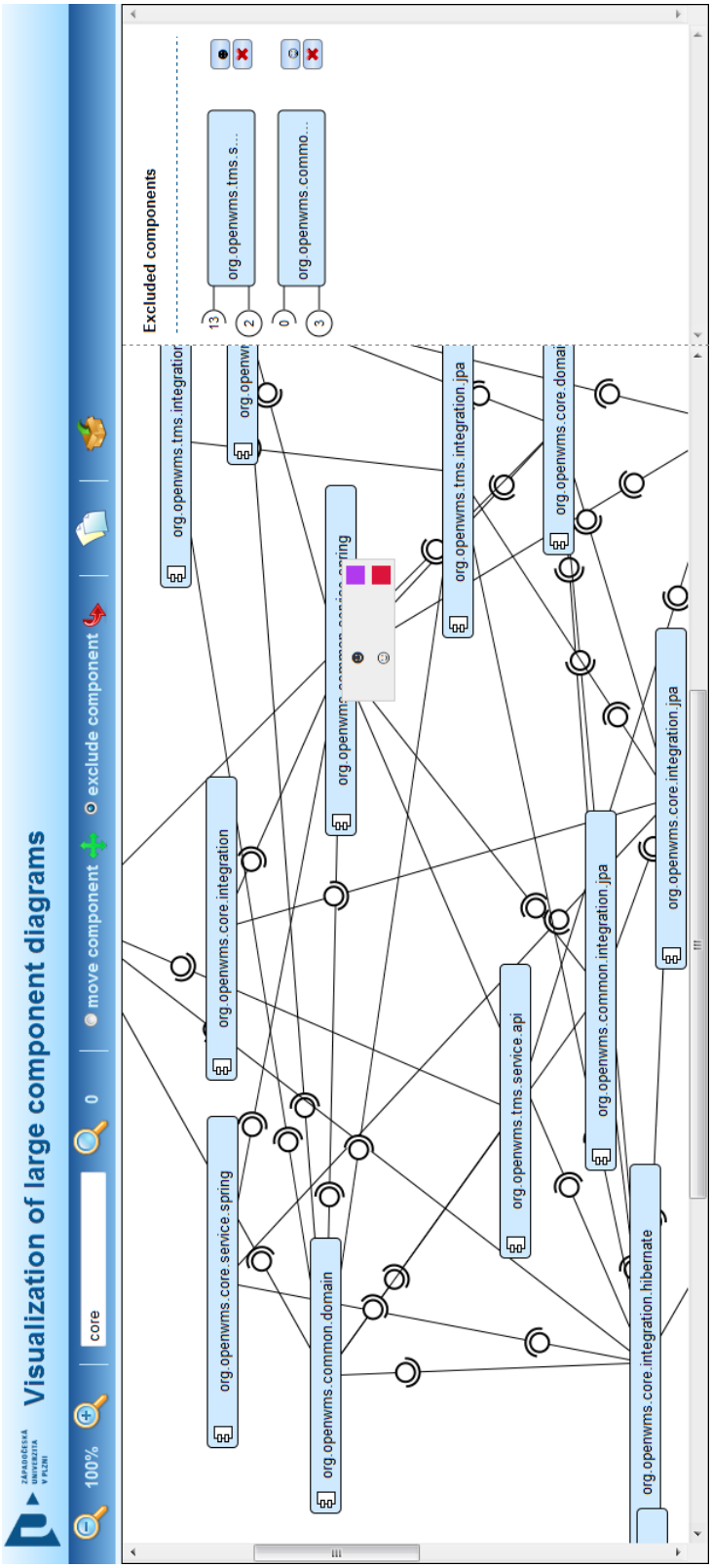


Obrázek B.4: Stav aplikace po zmenšení diagramu

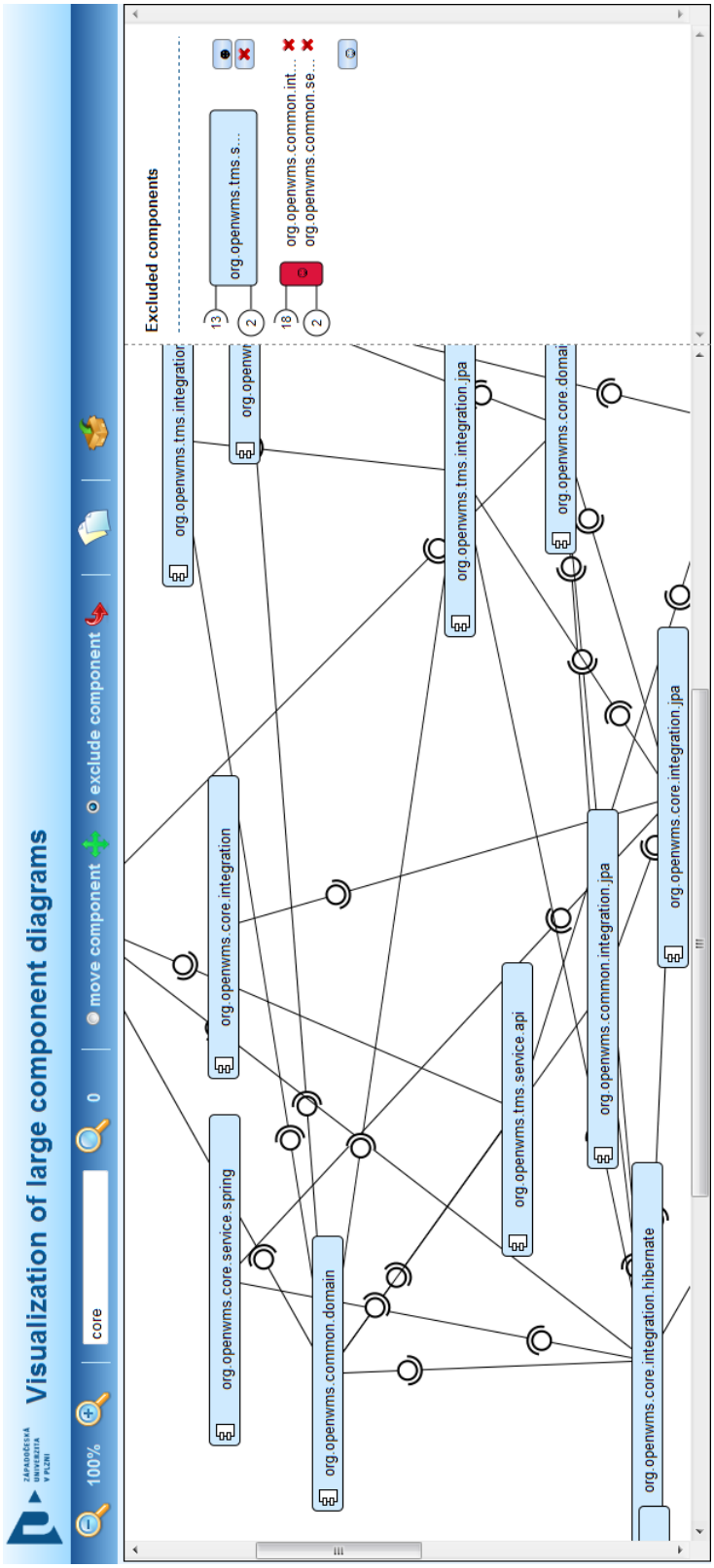
Obrázek B.5: Stav aplikace po zvětšení diagramu



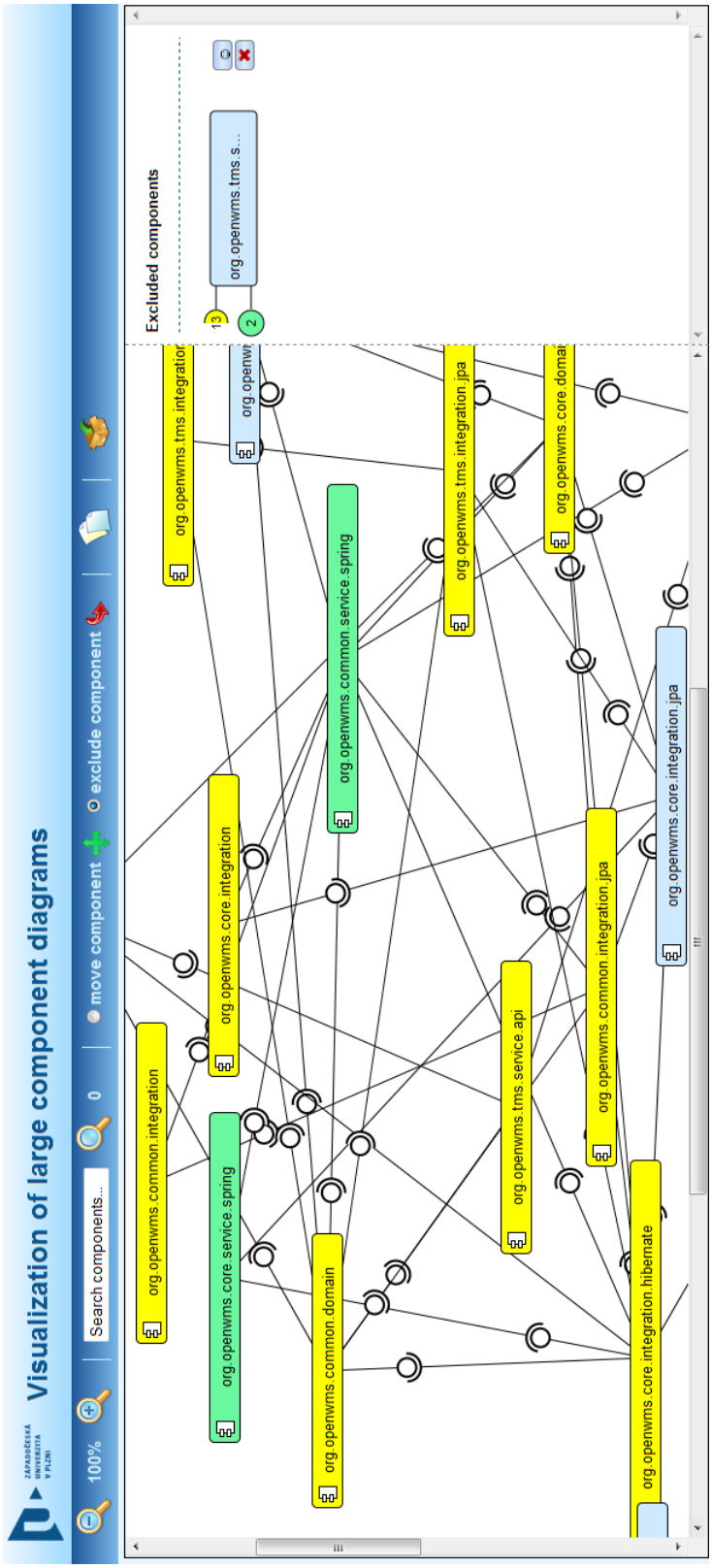
Obrázek B.6: Stav aplikace po přesunutí komponenty do komponentové oblasti



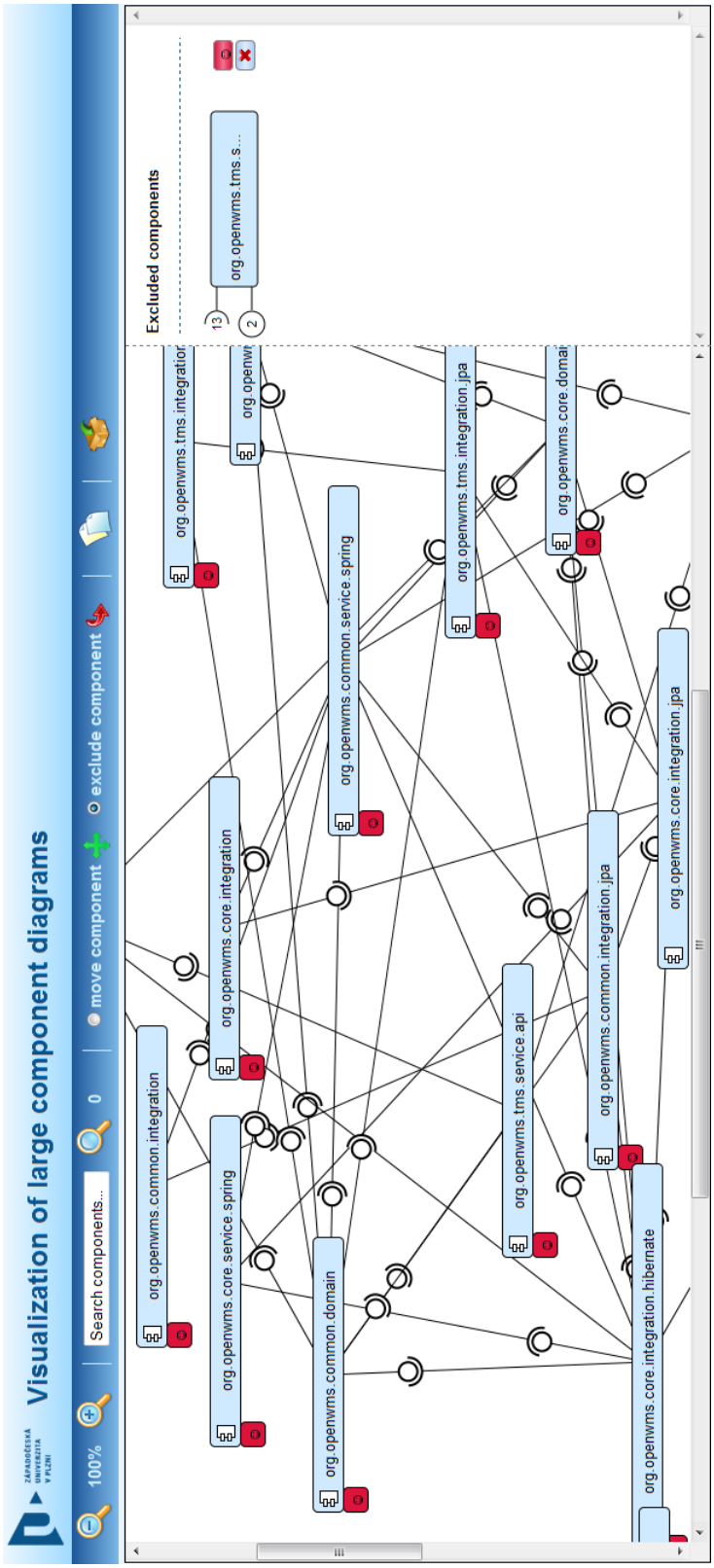
Obrázek B.7: Stav aplikace před přidáním komponenty do skupiny



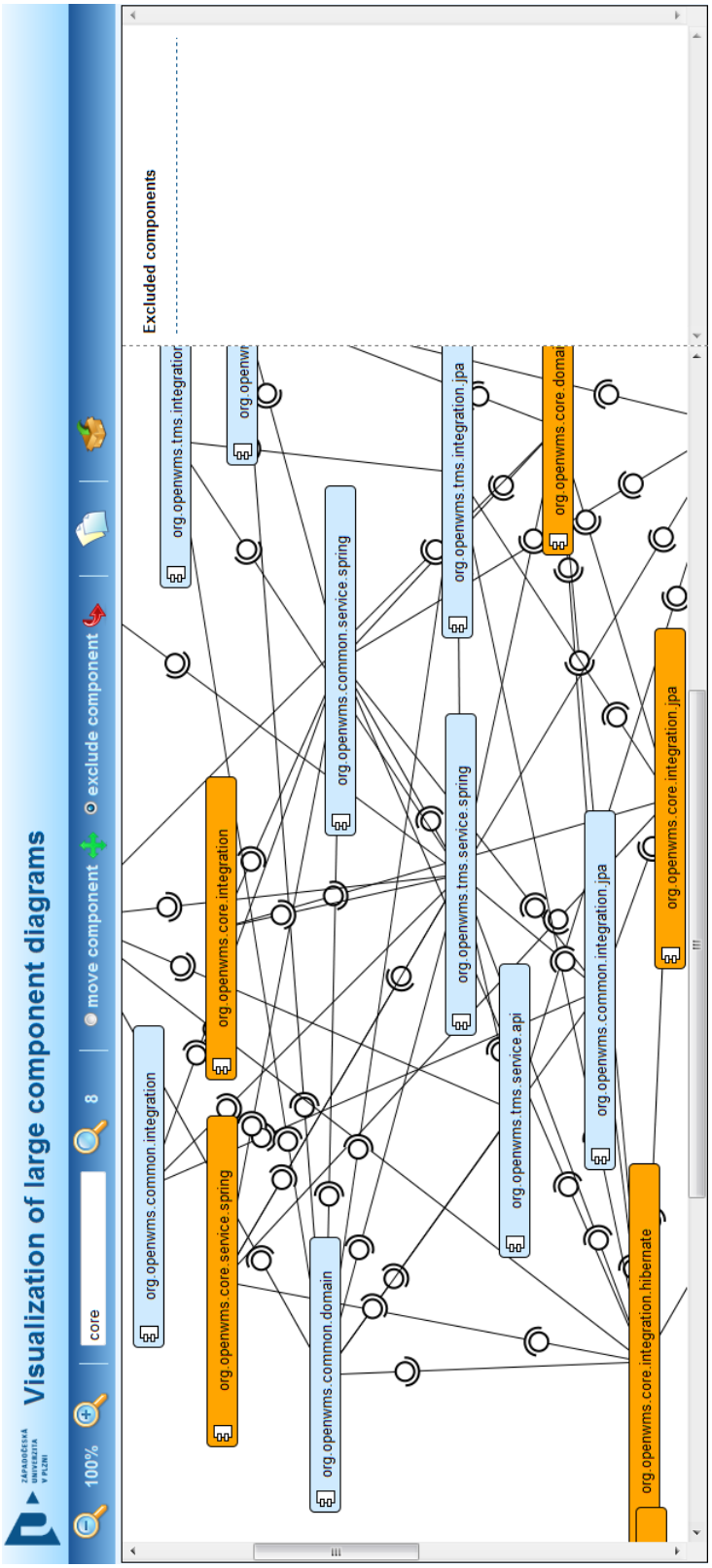
Obrázek B.8: Stav aplikace po přidáním komponenty do skupiny



Obrázek B.9: Stav aplikace po zvýraznění sousedů vyjmuté komponenty



Obrázek B.10: Stav aplikace po zobrazení delegátů



Obrázek B.11: Stav aplikace po vyhledání zadaného řetězce

Obrázek B.12: Stav aplikace po zobrazení aplikace Nuxeo

Příloha C – Struktura zdrojových souborů aplikace

V této části je zobrazen podrobný rozpis logického členění zdrojových souborů aplikace na základě jejich zaměření do jednotlivých balíčků. Z tohoto pravidla jsou vyjmuty pouze soubory JavaScriptu a CSS, jež jsou rozděleny do adresářů pouze na základě typové příslušnosti.

Java balíky a třídy

- cz.zcu.kiv.offscreen.api
  - EdgeInterface.java – rozhraní definující, co musí každá hrana obsahovat
  - GraphInterface.java – rozhraní definující, z čeho se musí graf skládat
  - VertexInterface.java - rozhraní definující, co musí každý uzel grafu obsahovat
- cz.zcu.kiv.offscreen.graph
  - EdgeImpl.java – třída reprezentující hranu, která implementuje rozhraní EdgeInterface
  - GraphImpl.java – třída reprezentující graf implementující GraphInterface
  - VertexImpl.java – třída reprezentující uzel grafu, která implementuje 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 načtených komponent
- cz.zcu.kiv.offscreen.graph.loader

  - GenericComponentLoader – třída, která dědí od ComponentLoader a zajišťuje vydolování 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
  - SettingGraph.java – servlet, který zajistí vytvoření adresáře na základě získaného session id z cookies
  - ShowGraph.java – servlet obstarávající zobrazení hlavní stránky aplikace a to stránky, kde bude zobrazený graf komponent

- cz.zcu.kiv.offscreen.servlets.action
  - DeleteAllComponents.java – servlet, který zajistí smazání všech nahraných komponent na server
  - DeleteComponent.java – servlet obstarává smazání dané komponenty uložené na serveru
  - UploadFiles.java – servlet, který se stará o nahrávání komponent na server

- cz.zcu.kiv.offscreen.session
  - SessionManager.java – 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...

Soubory JavaScriptu

- graphManager.js – reprezentuje graf, který sestaví z dat obdržených od serveru
- gridMark.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 – obstarává vytvořené skupiny
- loader.js – zajišťuje zobrazení a vypnutí 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 – stará se o všechny tooltipy.
- util.js – obsahuje pomocné funkce
- zoom.js – zajišťuje přibližování a oddalování diagramu komponent
- jquery-1.7.1.js – jquery
- jquery.contextMenu.js – plugin, který slouží k vytvoření kontextového menu
- jquery.qtip.js – plugin na tvorbu tooltipů
- mootools-core-1.4.1.js – plugin, který umožňuje vytvářet třídy v javascriptu
- spin.js – plugin pro vytváření loaderů

Soubory stylů

- basis.css – styly pro aplikaci a pro SVG elementy
- jquery.contextMenu.css – styly doplňující plugin jquery.contextMenu.js
- jquery.qtip.css – styly doplňující plugin jquery.contextMenu.js
- tooltips.css – styly přepisující některé styly z jquery.qtip.css