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

Diplomová práce

Vizualizace mimofunkčních charakteristik (EFP)

Plzeň 2014 Jiří Loudil

Prohlášení

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

V Plzni dne 12. května 2014

Jiří Loudil

Abstract

Software development shifts towards component based technologies. Applications using these technologies can easily consist of hundreds of components and therefore their complexity makes it hard to follow their structure and brings the need for a visualization. This master's thesis copes with methods of visualization used for displaying components' extra-functional properties. These methods were implemented as an extension of an existing large diagram visualization software tool. This thesis also describes a few other new features which were incorporated into the software tool.

Abstrakt

Trend vývoje software směřuje k využití komponentových technologií. Aplikace postavené na základě těchto technologií se často sestávají ze stovek komponent, což zhoršuje jejich přehlednost a přináší potřebu vizualizace jejich struktury. Tato práce se zabývá především metodami pro vizualizaci mimofunkčních charakteristik komponent. Zmíněné metody byly implementovány jako rozšíření existujícího nástroje pro vizualizaci rozsáhlých diagramů. Práce též popisuje několik dalších rozšíření funkcionality tohoto nástroje.

Poděkování

Tímto bych chtěl poděkovat vedoucímu diplomové práce Ing. Lukáši Holému za cenné rady, připomínky a vedení práce.

Obsah

1 Úvod

Obecný trend spojený s vývojem softwarových aplikací směřuje ke stále vzrůstající jak časové tak strukturální náročnosti. Tuto náročnost utváří z části velice rychlé tempo rozšiřování oblastí nasazení informačních technologií a též nutnost přizpůsobovat náročnost obsluhy softwarových aplikací co nejširšímu okruhu uživatelů. Zmíněnému trendu zvyšování náročnosti jsou nuceni čelit vývojáři, kteří musejí svou práci odvádět efektivně a spolehlivě. Proto se nutně po celou historii informačních technologií vyvíjí i používaná paradigmata programování. Postupným vývojem přes procedurální, objektové a jiná paradigmata, která reagovala na vzrůstající náročnost požadavků, dospěl vývoj k prosazujícímu se komponentově orientovanému paradigmatu (Component-based Model). Tento model přináší efektivní a udržitelný vývoj i velice rozsáhlých softwarových aplikací za pomoci využití principu dekompozice a znovupoužitelnosti.

Při vývoji komponentově orientovaných aplikací vyvstává problém s udržením přehledu o závislostech mezi jednotlivými dekomponovanými částmi aplikace. Tento problém je převážně zapříčiněn rozsahem aplikace a také různými typy sledovaných závislostí. Jako jednu z těchto závislostí lze chápat i mimofunkční charakteristiky (Extra-functional Property – EFP) a požadavky kladené na jednotlivé dekomponované části aplikace. Pro maximální využití efektivity komponentově orientované technologie je tak nutné využívat vizualizačních nástrojů pro zobrazení struktury aplikace.

Cílem této práce bylo seznámit se s teorií mimofunkčních charakteristik, nástrojem *Extra-Functional Property Featured Compatibility Checks* (EFFCC), technikami vizualizace pro rozsáhlé diagramy a nástrojem *Complex Component Applications Explorer* (CoCA-Ex). Též navrhnout vhodnou formu vizualizace mimofunkčních charakteristik pro nástroj *CoCA-Ex*. V praktické části bylo cílem implementovat navrženou formu vizualizace společně s propojením obou výše zmíněných nástrojů a také ověření funkčnosti na ukázkové aplikaci. Na závěr byl nástroj *CoCA-Ex* rozšířen o množství další doplňkové funkcionality.

Nástroji *EFFCC* tak bude umožněno využít rozšířeného nástroje *CoCA-Ex* paralelně vyvíjeného Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni jako svou vizualizační část.

2 EFPs v komponentovém modelu

2.1 Komponentově orientované technologie

Paradigmata programovaní využívaná při vývoji software prošla řadou vývojových fází, které se vždy snažily odpovídat na potřeby efektivního vývoje aplikací. V dnešní době se především u rozsáhlých projektů prosazují technologie vývoje založené na komponentách – *Component-based software engineering* (CBSE). Tento přístup využívá spojení komponentové softwarové technologie a komponentového návrhu pro návrh komponentově orientovaných softwarových systémů. Lze tedy hovořit o komponentovém stylu návrhu aplikace.

2.1.1 Softwarová komponenta

Již z názvu je zřejmé, že základním prvkem uvedené technologie jsou *komponenty*, na základě kterých lze tvořit vlastní aplikace. Softwarovou komponentu náležící do komponentové technologie lze definovat mnoha způsoby, uvedu zde definici dle Szyperského [Szyperski et al.(2002)Szyperski, Gruntz„ Murer]:

> Softwarová komponenta je jednotka kompozice, která obsahuje pouze standardem definované rozhraní a explicitní závislosti na kontextu. Takováto komponenta může být dále nezávisle nasazována a komponována v celky.

Jako další upřesnění lze použít tato kritéria, kterým musí softwarová komponenta vyhovovat [Niekamp(2006)]:

- možnost vícenásobného použití,
- nezávislost na konkrétním kontextu,
- možnost kompozice v celky,
- využití zapouzdření,

- možnost nezávislého nasazení a verzování.

Na základě těchto specifikací lze prohlásit, že softwarová komponenta je jakýsi znovupoužitelný stavební prvek typu Black-box[1], který vyhovuje danému komponentovému modelu, využívá interakce přes své jednoznačně určené rozhraní a je nezávislý na ostatních komponentách.

2.1.2 Komponentový model

Přesnějším definováním komponent se zabývají komponentové modely. Za dobu využívání komponentově orientovaného stylu programování přinesl vývoj celou řadu komponentových modelů. Tyto modely jednoznačně definují dovolené typy komponent, jejich rozhraní, užitý komunikační protokol a podobně.

Podrobnější přehled vlastností modelů nabízí [Bachmann(2000)]:

- **Jednotná kompozice** – představuje standardizační postupy pro zajištění identifikace komponent, jejich vzájemnou synchronizaci, použití komunikačních protokolů, použití kódování, atd.
- **Dodržení kvality produktu** – je dosaženo pomocí standardizování typů komponent a veškerých využitých metod interakce.
- **Nasazení komponent a systému** – znamená nutnost definovat standardem, které zdroje (systémové či jiné komponenty) jsou komponentám k dispozici a též způsob přístupu k těmto zdrojům. Též je nutné standardizovat infrastrukturu pro vývoj komponent, jejich kompozici a nasazení systému.

V oblasti CBSE se mluví o implementaci komponentového modelu, která se označuje jako *komponentový framework*. Nejjednodušeji lze tento framework přirovnat k operačnímu systému, kde softwarové komponenty představují procesy běžící v operačním systému (dle [Bachmann(2000)]). Vybrané komponentové frameworky jsou blíže popsány v kapitole 2.1.4.

Majoritní přednosti plynoucí z popisovaného přístupu komponentově orientovaného programování jsou (dle [Bachmann(2000)]) následující:

[1] Black-box – chování entity se projevuje navenek, vnitřní jevy nejsou pozorovatelné.

- **Nezávislost součástí** – komponenty v komponentovém modelu mají definovaný standard chování, tudíž je zajištěna jejich značná nezávislost. Tuto nezávislost lze využít jak při vývoji tak i nasazení jednotlivých komponent systému a to i pro rozšiřování funkčnosti či udržování systému.
- **Rodiny součástí** – komponentové modely specifikují dané standardy a podpůrné systémy, které dovolují vývoj rodin komponent pro specifické nasazení.
- **Urychlení vývoje** – použití komponentového stylu návrhu aplikace je typicky snížena doba nutná k návrhu, vývoji a nasazení aplikace. Tato vlastnost je dosažena díky množství faktorů – především jde o: znovupoužitelnosti existujících otestovaných komponent, již předem definované klíčkové součásti architektury pomocí použitého modelu nebo unifikovaná abstrakce u návrhu jednotlivých komponent.
- **Zlepšení předvídatelnosti systému** – již při návrhu komponentových modelů a frameworků lze definovat upřednostňované vlastnosti, které budou od komponent vyžadovány. Tímto lze zvýšit míru předvídatelnosti vlastností u komponentového systému založeném na dané technologii (např. škálovatelnost nebo bezpečnost).

2.1.3 Interakce v modelu

Lze odvodit, že v komponentovém modelu dochází k řadě interakcí, jak mezi samotnými komponentami, mezi komponentami a prostředím (frameworkem) a dokonce také i mezi různými prostředími. Druhy interakcí lze dle [Bachmann(2000)] dále dělit na:

- nasazení komponenty – vložení komponenty do prostředí frameworku;
- nasazení frameworku – jiné frameworky lze vkládat do frameworku;
- jednoduchou kompozici – kompozice dvou komponent uvnitř jednoho frameworku;
- heterogenní kompozici – kompozice dvou komponent z odlišných frameworků;
- rozšíření frameworku komponentou – kompozice frameworku s komponentou uvnitř jiného frameworku;

- hierarchickou kompozici komponent – kompozice komponenty a skupiny komponent.

V této práci existuje zaměření přenesené na typ jednoduché kompozice v kontextu dvou komponent.

Jak již bylo uvedeno komponenty interagují se svým okolím prostřednictvím svého jednoznačně určeného rozhraní. Toto rozhraní určuje *vlastnosti* (features) komponent, které lze okolními komponentami využívat. Rozhraní si též lze představit, jako skupinu pojmenovaných operací, které lze volat z pozice klienta [Szyperski et al.(2002)Szyperski, Gruntz„ Murer]. Obecně lze rozhraní dělit na části *provided* a *required*. Část provided značí, že daná komponenta vlastnosti implementuje, ovšem k implementování těchto vlastností potřebuje, aby interagující komponenta implementovala část required.

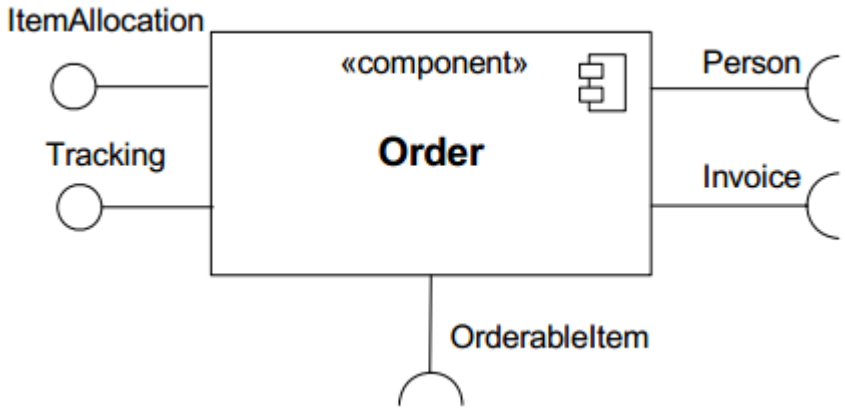
Můžeme tedy říct, že interakce komponent klade funkční a v některých případech i mimofunkční požadavky (viz dále kapitola 2.2) na jednotlivé komponenty. Tyto požadavky jsou na obou stranách interakce vyjednávány za pomoci *kontraktů* (interface contracts) [Bachmann(2000)]. Pokud tyto požadavky nejsou dodrženy, pak systém jako celek nemůže správně plnit svou funkci. Je tedy nutné ve všech interakcích komponentově orientovaného systému uspokojit veškeré požadavky interagujících komponent pro správnou funkci systému.

V této práci je dále pro vizualizaci interakce komponent využita symbolika z větší části převzatá z *UML diagramu komponent*. Ukázku z tohoto diagramu lze vidět na obrázku 2.1, který zobrazuje komponentu nazvanou *Order* se dvěmi *provided* a třemi *required* vlastnostmi.

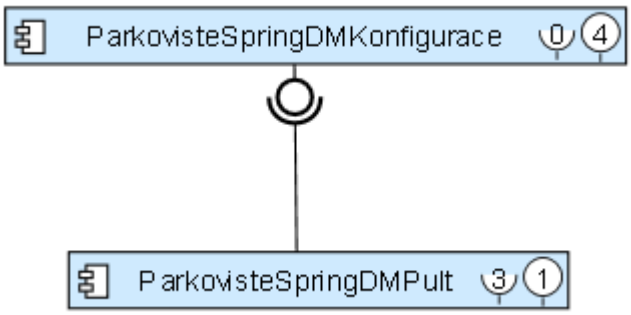
Symbolika použitá v této práci, především v nástroji *CoCA-Ex*[2], kterého se dotýkala tato práce, je vidět na obrázku 2.2. Tento obrázek znázorňuje interakci dvou pojmenovaných komponent, přičemž komponenta umístěná v jeho horní části disponuje provided a druhá required vlastností.

Podrobněji o využívaných metodách a obecně problematice vizualizace v oblasti CBSE pojednává kapitola 4.1 této práce.

[2]Podrobně v 3.2.



Obrázek 2.1: UML komponenta [UML()].



Obrázek 2.2: Interakce komponent v nástroji CoCA-Ex.

2.1.4 Vybrané frameworky

Pro reálný vývoj aplikací využívající některý komponentově orientovaný model jsou zapotřebí komponentové frameworky, které byly definované v kapitole 2.1.2. Každý framework je vystavěn nad svým příslušným komponentovým modelem (případně i nad více modely současně) a přináší implementaci jím definovaných služeb a standardů. Frameworky tak prostřednictvím svých služeb poskytují běhové a komunikační prostředí pro jednotlivé komponenty vyvíjeného systému.

Následující text se věnuje zástupcům komponentových frameworků, které využívají obě aplikace, o kterých pojednává tato práce. V tomto případě jde výhradně o frameworky, které jsou zástupci rodiny využívající nějakou formu

Java technologií [3].

Další typy rodin komponentově orientovaných frameworků jsou zmíněny v [Szyperski et al.(2002)Szyperski, Gruntz„ Murer] a to kupříkladu rodina *OMG*[4] obsahující frameworky jako *IBM WebSphere* nebo *IONA Orbix* a rodina obsahující produkty zaštítěné společností *Microsoft*[5] jako jsou *Microsoft COM* nebo *Microsoft Common Language Runtime*.

OSGi Service Platform

Open Services Gateway initiative (*OSGi*) je sada otevřených specifikací, které definují komponentový model s využitím *Java technologií* a je udržovaná mezinárodním neziskovým konsorciem *OSGi Alliance* [OSGi(c)], které bylo založeno v roce 1999. Toto konsorcium též poskytuje referenční implementace, testovací nástroje a též hraje roli certifikační autority.

Na základě OSGi standardu bylo vyvinuto množství frameworků, např.: Apache Felix[6], Eclipse Equinox[7] nebo Knopflerfish[8].

OSGi mimojiné zavádí pojmy [OSGi(b)] jako jsou:

- **Bundle** – představuje základní stavební jednotku modelu – softwarovou komponentu. Bundle představuje klasický Java JAR balík[9], který ovšem musí jasně definovat, co ze svého obsahu poskytuje okolí.
- **Manifest** – soubor obsažený v JAR balíku, který definuje informace pro běh pod OSGi frameworkem nad rámec standardního JAR manifestu.
- **Import-Package** – představuje deklaraci závislostní části (required) rozhraní bundlu [OSGi(a)].
- **Import-Package** – představuje deklaraci poskytované části (provided) rozhraní bundlu.

[3] Více na: http://java.com/en/
[4] Domovská stránka: http://www.omg.org/
[5] Domovská stránka: http://www.microsoft.com/
[6] Domovská stránka: http://felix.apache.org
[7] Domovská stránka: http://www.eclipse.org/equinox
[8] Domovská stránka: http://www.knopflerfish.org
[9] Java ARchive

Komponenty vyhovující tomuto modelu jsou podporovány oběma aplikacemi, kterými se zabývá tato práce – EFCC i CoCA-Ex.

EJB3

Enterprise Java Beans 3 (*EJB3*) představuje třetí verzi jednoho ze standardů využívající platformu Java Enterprise Edition (Java EE)[10]. Od roku 1997 byl postupně vyvíjen společnostmi IBM a Sun Microsystems, nyní je, stejně jako vlastní Java EE, spravován společností Oracle[EJB3()]. EJB lze též označit za Java EE *kontejner* (container) specializující se na běh bussines logic. Z frameworků lze jmenovat otevřený Apache OpenEJB[11].

Obdobně, jako u předchozího modelu komponenty fyzicky představují JAR balíky. Ovšem, namísto modifikovaného manifestu využívá EJB model v JAR balíku jeden nebo více souborů nazývaných *deployment deskriptory* (deployment descriptors) [EJB()].

V současnosti jsou komponenty tohoto modelu ze dvojice výše zmíněných nástrojů podporovány pouze v CoCA-Ex.

SOFA 2

SOFtware Appliances 2 (*SOFA 2*), jde o implementaci hierarchického komponentového modelu *SOFA* vyvíjeného na Katedře distribuovaných a spolehlivých systémů Matematicko-fyzikální fakulty Univerzity Karlovy v Praze [D3S()]. SOFA 2 podobně jako předchozí frameworky využívá Java technologie [SOFA2()].

Shodně s modelem EJB3 jsou tyto komponenty z obou nástrojů podporovány pouze v CoCA-Ex.

CoSi

Components Simplified (*CoSi*) je komponentový model a zároveň framework vyvíjený Katedrou informatiky a výpočetní techniky Fakulty aplikovaných

[10] Více na: http://www.oracle.com/technetwork/java/javaee/overview/index.html
[11] Domovská stránka: http://openejb.apache.org/

věd Západočeské univerzity v Plzni [CoSii()]. CoSi vychází z modelu OSGi[12], ale liší se především snahou o maximální dodržení míry Black-box přístupu ke komponentám, který není v četných ostatních případech vždy dodržen [Brada(2008)].

Obdobně jako OSGi přejímá framework CoSi využívání Java technologie a komponenty ve fyzické formě JAR balíku s manifestem vyhovujícím vlastnímu standardu.

Komponenty tohoto modelu lze zpracovat pouze nástrojem EFCC; ne však CoCA-Ex.

2.2 Úvod do EFPs

Extra-functional Property (EFP) – česky: mimofunkční charakteristika – je pojem úzce spojený s CBSE, o kterém pojednává kapitola 2.1 této práce. Dle [Jezek(2010a)] představují EFPs jednu z ne zcela přesně definovaných oblastí těchto moderních technologií, která může v důsledku limitovat praktické použití komponentových technologií. Tudíž vyvstává reálná potřeba se problematikou mimofunkčních charakteristik důkladně zabývat.

2.2.1 Definice EFPs

Možností, jak definovat pojem mimofunkční charakteristiky existuje celá řada a často je i obtížné stanovit hranici mimofunkcionality [Jezek(2010b)]. Z těchto možností se jeví jako nejpřímočařejší definovat EFPs komponenty jako rozdíl množiny všech charakteristik a množiny funkčních charakteristik komponenty.

Výše zmíněná definice ovšem neposkytuje žádný bližší popis vlastností EFPs. Je proto potřeba přesnější a použitelnější definice tak, jak ji představuje (dle [Jezek – Brada(2012)Jezek, Brada]):

> Mimofunkční charakteristiky poskytují prostředky pro posouzení kompatibility interakcí komponent, přičemž berou v úvahu:

[12] Podrobně viz kapitola 2.1.4.

- **kvalitativní charakteristiky** – jako je například reakční doba nebo spotřeba paměti komponent;
- **uživatelské požadavky** – mezi které patří cena, četnost aktualizací nebo technická podpora komponent;
- **charakteristiky chování** – které pokrývají synchronizaci, paralelní přístup nebo deadlock-free[13] chování komponent.

Ještě mírně detailnější představu o EFPs dává [Bachmann(2000)], podle něhož lze mimofunkční charakteristiky klasifikovat na:

- **Chování** – týká se výsledků a to především výsledků volání metody, které pochází ze sekvenčního řetězce volání metod [Jezek(2010b)].
- **Synchronizaci** – týká se veškerých záležitostí spojených s paralelním během. Jde především o zaručení thread-safe[14] vlastnosti komponent.
- **Kvalitu služeb** – obvykle zahrnuje technické aspekty jako je: maximální a průměrná doba odezvy, použitá strojová přesnost výpočtu, přenositelnost a podobně.

Z výše uvedeného vysvětlení pojmu mimofunkčních charakteristik lze vypozorovat závažnost důsledků, které EFPs mají na výsledný komponentově orientovaný systém, který je často složen ze značného počtu komponent nesoucích různý počet EFPs. Pokud vyjdeme z uvedených příkladů mimofunkčních charakteristik, tak může nastat modelová situace, kdy se pouhou volbou nevhodných komponent prodlouží průměrná doba odezvy systému nad předem stanovenou únosnou mez.

2.2.2 Užití EFPs v praxi

V praktické aplikaci CBSE pomocí komponentových frameworků popsaných v kapitole 2.1.4 narážíme z jejich strany na neexistenci přímé podpory EFPs. Ani velice rozšířený framework jako je framework OSGi[15] nepodporuje EFPs [Jezek – Brada(2012)Jezek, Brada].

[13] Takové komponenty, kde je zaručeno, že nenastane deadlock [Inverardi – Tivoli(2003)Inverardi, Tivoli].

[14] Takový program nebo jeho část, která neobsahuje souběh, pokud je vykonávána paralelně.

[15] Podrobně viz kapitola 2.1.4.

Existuje řada různých přístupů pro zakomponování EFPs do existujících komponentových modelů a frameworků nebo tvorba příslušně specializovaných nových. Na tomto místě lze uvést jako příklad model Robocop [Bondarev et al.(2006)Bondarev, Chaudron„ de With] nebo jazyk NoFun [Franch(1998)].

2.2.3 Generický EFP framework

V této práci byla využita metoda začlenění EFPs pomocí generického nadstavbového frameworku [Jezek – Brada(2012)Jezek, Brada], který je přizpůsobitelný pro řadu komponentových modelů a tudíž ho též lze zakomponovat do řady existujících frameworků. Jedinou podmínkou kladenou na model je existence provided/required částí rozhraní komponent, které bylo popsáno v kapitole 2.1.3. Zmíněný generický framework je vyvíjen Katedrou informatiky a výpočetní techniky Západočeské univerzity v Plzni.

Generický EFP framework rozšiřuje jednotlivé fáze vývoje komponentově orientovaných aplikací o EFPs vlastnosti. Jako fáze vývoje lze v kontextu EFPs chápat především:

- prvotní architektonický návrh aplikace spolu s definicí EFPs;
- vývoj vlastní aplikace s přiřazením již definovaných EFPs ke komponentám;
- ověření kompatibility komponent před nasazením aplikace.

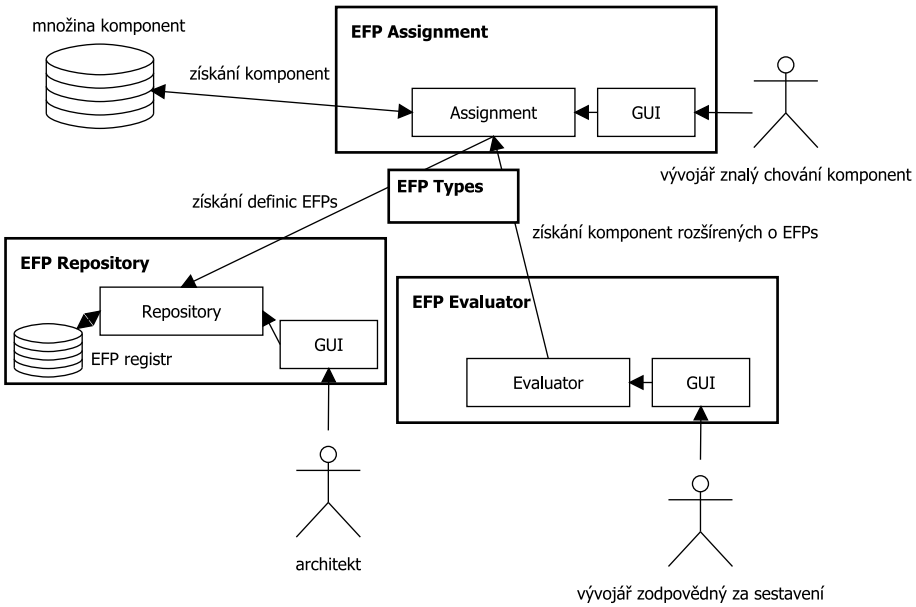
EFP framework zajišťuje rozšíření výše vyjmenovaných fází vývoje aplikace pomocí následujících činností [Jezek – Brada(2011)Jezek, Brada]:

- **definice EFPs** – Tato část poskytuje vlastní zavedení definic EFPs do komponentového modelu. Pro sdílení přístupu k uloženým definovaným EFPs využívá framework modul *EFP Repository*. Modul je využit doménovými architekty nebo návrháři ve fázi návrhu.
- **přiřazení EFPs** – Zde je využit modul *EFP Assignment* pro přiřazení definic EFPs z modulu EFP Repository k jednotlivým komponentám. Existující komponenty jsou tak rozšířeny o mimofunkční charakteristiky. Modul EFP Assignment je určen pro vývojáře při vývoji aplikace,

kteří jsou obeznámeni s vlastnostmi a chováním konkretních komponent.

- **verifikace EFPs** – Jako poslední část je definován modul *EFP Evaluator*. Tento modul má za úkol vyhodnotit vzájemnou kompatibilitu komponent rozšířených o EFPs před zahájením jejich vzájemných interakcí. Tento modul je typicky využíván vývojářem zodpovědným za sestavení komponent před nasazením aplikace.

Všechny výše zmíněné moduly EFP frameworku využívají meta-model definovaný modulem *EFP Types*. Tento modul slouží ke spojení celého frameworku v celek. Podrobnosti o EFP Types jsou popsány v [Jezek(2010a)]. Celkový přehled o architektuře poskytuje obrázek 2.3.



Obrázek 2.3: Generický EFP framework.

Tento framework lze po nasazení využít pro kontrolu vzájemné kompatibility komponent před navázáním jejich rozhraní a též při studiu nahraditelnosti jednotlivých komponent v závislosti na EFPs. V současné době[16] existuje podpora pro komponentové modely CoSi a OSGi (viz kapitola 2.1.4)

[16]Při psaní této práce – únor 2014.

2.3 Nástroj EFFCC/EFPPortal

Jedním ze sady nástrojů implementovaných v rámci podpory generického EFP frameworku popsaného v kapitole 2.2.3 této práce je i webový nástroj EFFCC/EFPPortal [Jezek et al.(2013a)Jezek, Brada„ Holy]. Tento nástroj obsahuje dvě části a to provedení ověření vzájemné kompatibility komponent označované jako *Compatibility Verification* a rozhraní pro přiřazení EFPs komponentám nazvané *Properties Assignment*. V době psaní této práce je dostupná pouze první jmenovaná funkcionalita. Jde tedy o vizualizační nástroj[17], který využívá výstupu modulu EFP Evaluator[18]. Díky vizualizaci těchto dat tak EFFCC/EFPPortal slouží jako podpora pro vývojáře zodpovědného za sestavení komponent.

Zmíněný modul zpracovává ověřovanou vstupní množinu komponent do podoby orientovaného grafu, kde jeho vrcholy jsou tvořeny: komponentami, rozkladem rozhraní komponent na jednotlivé vlastnosti a EFPs navázanými na vlastnosti. Hrany grafu tvoří obecně dva typy:

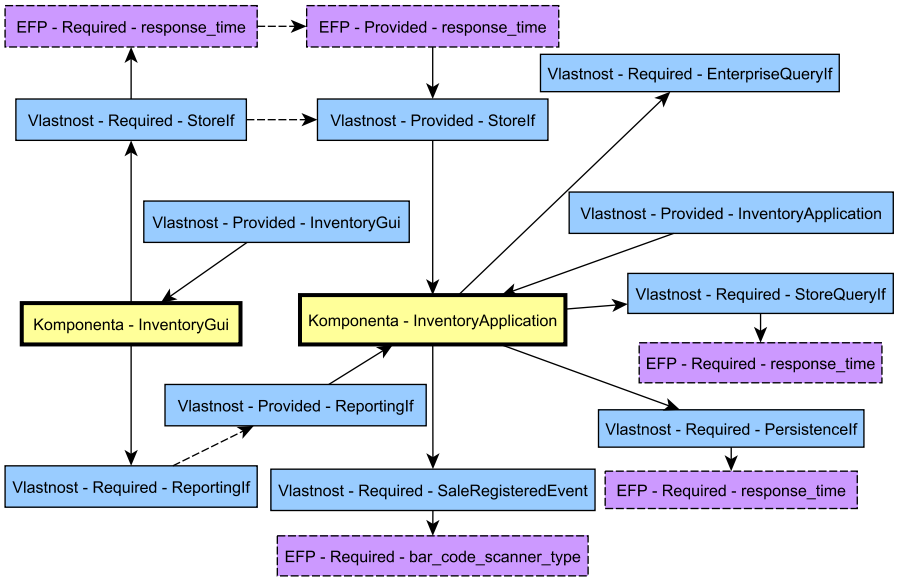
- **strukturální** – Hierarchicky spojují příslušející komponenty, vlastnosti a EFPs. Tyto hrany mají směr od EFP, přes vlastnost až ke komponentě v případě provided a v případě required je směr opačný.
- **porovnávací** – Spojují jednotlivé vlastnosti a EFPs. Směr hran tohoto typu je obecně od required k provided části.

Podrobně je celá struktura popsána v práci [Jezek – Brada(2011)Jezek, Brada]. Na obrázku 2.4 je vyobrazen příklad demonstrující podobu vyhodnocovacího grafu. Ten je použit modulem EFP Evaluator a obsahuje dvě komponenty spolu s množstvím vlastností, z nichž čtyři mají navázané EFP.

Pro vlastní vyhodnocení vzájemných interakcí mezi komponentami je použita varianta algoritmu průchodu do hloubky (Depth-first search). Při průchodu grafem jsou kontrolovány typy jednotlivých vrcholů spolu s jejich atributy a je tak vyhodnoceno jejich spojení. Jak vyplývá z definice použitého grafu, může toto spojení existovat v několika variantách, kdy procházený vrchol je:

- **komponentou**

[17] Více o vizualizaci v komponentovém modelu v kapitole 3.
[18] Implementován jako *EFP Comparator*.



Obrázek 2.4: Vyhodnocovací graf [Vlcek(2011)].

- **vlastností** – Zde je možné odhalit chybějící provided vlastnost, která by interagovala s odpovídající required částí.
- **EFP** – V tomto případě lze dále rozlišit stavy:
  - kdy není definována EFP u provided části, ačkoli na required části definována je a tudíž deklarace EFP chybí;
  - kdy hodnota EFP je definovaná na obou částech interakce a lze tak následně rozhodnout, zda-li hodnota EFP vyhovuje (je kompatibilní) či nevyhovuje (není kompatibilní).

Je tedy možné vyvodit závěr, že po skončení procesu vyhodnocení lze ve vstupní množině softwarových komponent odhalit nesrovnalosti. A to jak neuspokojený required požadavek vlastnosti, tak i nekompatibilní EFP. Tímto přístupem tedy lze řešit detekci nesrovnalostí v interakci komponent, která byla popsána v kapitole 2.1.3.

2.3.1 Využívané technologie

Nástroj EFFCC/EFPPortal je aplikace založená na Java webových technologiích, využívá především Spring Framework[19] a knihovnu jQuery[20]. Použití těchto technologií umožnilo snadnou přístupnost nástroje EFFCC/EFPPortal, protože v klientském prostředí postačí přítomný kompatibilní webový prohlížeč[21].

Spring Framework

Jde o otevřený framework využívaný pro tvorbu především webových aplikací nad Java EE[22] technologií. Spring Framework přináší zefektivnění vývoje aplikací především díky možnosti odstínit se od infrastruktury a věnovat se návrhu business logic[23]. Tento framework je rozdělen na moduly, které jsou uspořádány do následujících skupin [Spring()]:

- **Core Container** – Představuje jádro celého frameworku, které využívají ostatní moduly. Obsahuje základní funkčnost, jako je například kontext.
- **Data Access/Integration** – Tato skupina zajišťuje přístup k datům. Skládá se z modulů pro přístup k různým databázovým systémům, XML a podobně.
- **Web** – Zde jsou zahrnuty moduly pro podporu webových aplikací, jako je například podpora model-view-controller (MVC)[24] architektury.
- **Aspect Oriented Programming (AOP) + Instrumentation** – Představuje moduly, které přináší aspektově orientované programování[25].

[19]Domovská stránka: http://projects.spring.io/spring-framework/
[20]Domovská stránka: http://jquery.com/
[21]Kupříkladu lze využít databázi: http://caniuse.com/
[22]Více na: http://www.oracle.com/technetwork/java/javaee/overview/index.html
[23]Business/aplikační logika – Ta část aplikace, která je zodpovědná za proces operací nad daty.
[24]Podrobně na: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html
[25]Podrobně na: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/aop.html

- **Test** – Skupina Test obsahuje prostředky pro izolované testování částí aplikace a podporu JUnit[26] testů.

CSS

Cascading Style Sheets (CSS)[27] je jazyk pro zápis definice vzhledu a formátu dokumentů zapsaných pomocí jazyků logického vyznačování, mezi které patří například HTML[28], XHTML nebo XML. Jeho filosofií je především oddělit strukturální část dokumentu od prezentační, což jinak činí dokumenty neflexibilní a nepřehledné.

JavaScript

JavaScript (JS) je interpretovaný jazyk se syntaxí vycházející z jazyka C. Nejrozšířenější pole působnosti je jeho implementace ve webových prohlížečích [JavaScript()]. Zde je využíván jako skriptovací jazyk například pro zavedení dynamické interakce ve webových aplikacích či pro potřeby asynchronní komunikace.

jQuery

Tato otevřená *JavaScript* knihovna představuje podstatné rozšíření jeho funkcionality. Je výrazně orientována na použití ve webových aplikacích, kde přináší velké množství vlastností, které umožňují efektivnější implemetnaci různé funkcionality. Její filosofický přístup je oddělení funkcionality od struktury. Tato knihovna je, podobně jako vlastní JS, podporována všemi majoritními webovými prohlížeči[29].

Mezi hlavní přednosti, které vedou k jejímu využití patří:

- snadnější průchod HTML dokumentem;
- snadnější výběr a manipulace s DOM[30] elementy;

[26]Domovská stránka: http://junit.org/
[27]Domovská stránka: http://www.w3.org/Style/CSS/
[28]Domovská stránka: http://www.w3.org/html/
[29]Viz http://jquery.com/browser-support/
[30]Více na: http://www.w3schools.com/jsref/dom_obj_all.asp

- efektivnější obsluha událostí;
- podpora široké škály animací;
- snadnější využití asynchronní technologie *Asynchronous JavaScript and XML* (AJAX)[31].

2.3.2 Popis nástroje

EFFCC/EFPPortal je, jak již bylo zmíněno v kapitole 2.3.1, vizualizační nástroj postavený na webových technologiích. Celá nerozšířená aplikace je dělena do úvodní a čtyř funkčních stránek – výběr komponentového frameworku, výběr s následným nahráním souborů/komponent ke zpracování na server, výběr lokálního registru a vlastní vizualizace výsledků. Na každé funkční stránce je možné se pomocí navigační lišty vrátit k některému z předchozích kroků a nebo o jeden krok zpět pomocí vyhrazeného tlačítka.

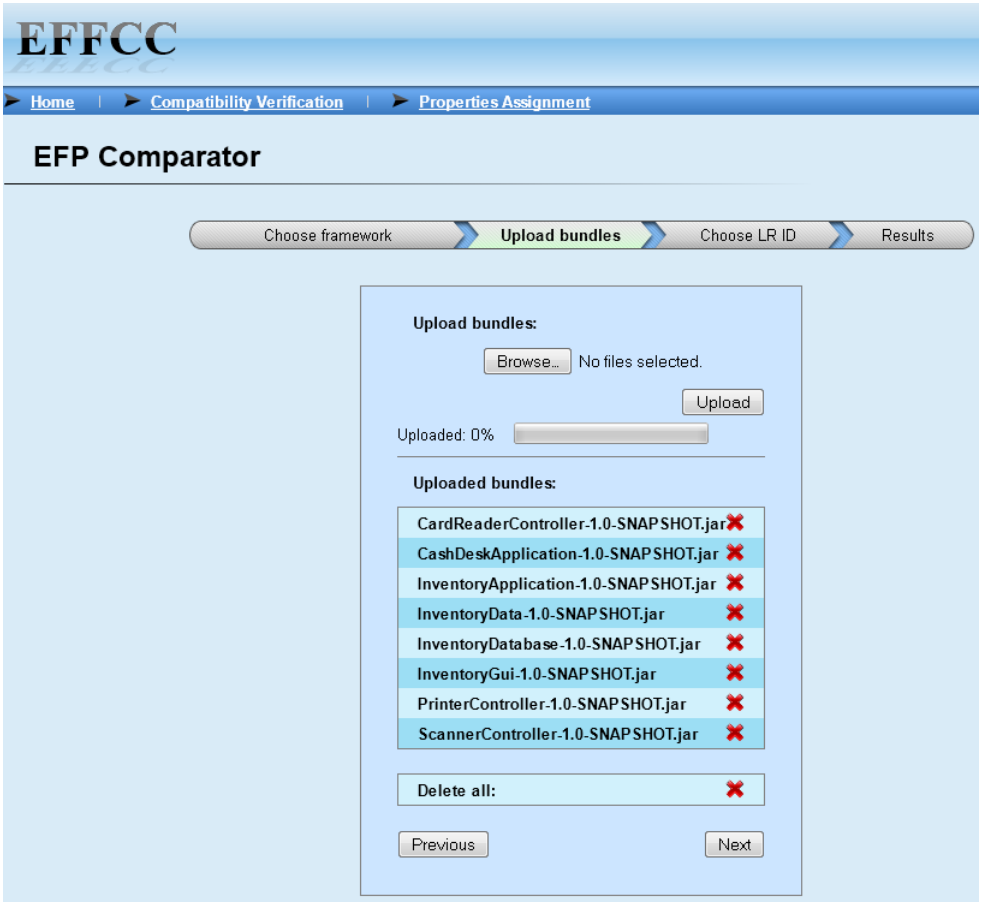
Po zadání příslušné URL aktivní nasazené aplikace je uživateli zobrazena uvítací/hlavní strana. Zde je nutné pokračovat aktivním odkazem na následující stránku s výběrem komponentového frameworku. Framework je zvolen z rozbalovací nabídky a po potvrzení je uživateli umožněno nahrání souborů určených k vyhodnocení na server. To lze provést pomocí standardního okna pro vícenásobný výběr souborů následovaného potvrzením. Již nahrané soubory lze pomocí kliknutí na symbol křížku u názvu souboru z výběru mazat. Podrobný pohled na rozhraní poskytuje obrázek 2.5.

S dokončením fáze výběru a nahrání komponent na server je uživateli dovoleno zvolit lokální EFP registr (blíže popsán v kapitole 2.2.3). Výběr je opět proveden z rozbalovací nabídky, která obsahuje ID jednotlivých dostupných lokálních registrů. Po potvrzení výběru registru je již provedeno vlastní vyhodnocení kompatibility. Toto vyhodnocení může být poměrně časově náročné. Jde především o ty případy, kdy jsou k analýze předkládány rozsáhlé aplikace čítající značný počet komponent (řádově stovky[32] nahraných souborů k vyhodnocení). O probíhajícím výpočtu na pozadí je uživatel informován pomocí animovaného prvku[33].

[31] Více na: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

[32] Typická hodnota pro rozsáhlejší komponentové aplikace – dle [Snajberk et al.(2013)Snajberk, Holy„ Brada].

[33] Jde o tzv. loading circle.



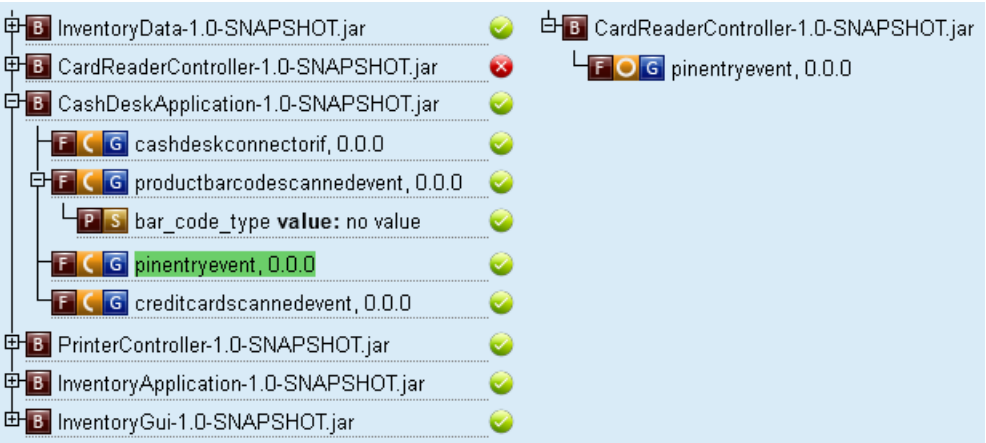
Obrázek 2.5: Výřez z rozhraní nástroje EFPportal.

Po ukončení vyhodnocení jsou pomocí vizualizace uživateli prezentovány výsledky[Jezek et al.(2013a)Jezek, Brada,, Holy]. To se děje pomocí interaktivní stromové struktury, kdy je tato interaktivita zajištěna využitím technologie JavaScript[34], která je dělena na pravou (statickou) a levou (dynamickou) část. Tato struktura prioritně zobrazuje stav vyhodnocení vzájemných interakcí, které byly popsány v kapitolách 2.1.3 a 2.3. Stav je zobrazen pomocí názorných symbolů. Grafická podoba vyhodnocení viz obrázek 2.6.

Výchozí úroveň zanoření struktury představují kořeny odpovídající jednotlivým souborům/komponentám předloženým ke zpracování. O úroveň níže ve stromové struktuře leží jednotlivé vlastnosti, které náleží zpracovaným souborům/komponentám. Nejnižší úroveň zanoření představují EFPs, které jsou navázany na vlastnosti. U každé z EFPs je kromě jména zobrazena

[34]Viz kapitola 2.3.1.

její přiřazená hodnota.



Obrázek 2.6: Hierarchická struktura vizualizace.

Výše zmíněnou levou stranu struktury vyhodnocení představují required části rozhraní, po kliknutí na některý ze zde se nacházejících prvků se zpřístupní korespondující zobrazení provided části v pravé části. Tímto způsobem je umožněn průzkum jednotlivých interakcí.

Popsané strukturované rozložení dovoluje uživateli nástroje rychlou lokalizaci vzájemných nekompatibilit s následným bližším průzkumem každé z nich.

3 Vizualizace v komponentovém modelu

Nutnost použití některé z forem vizualizace informací ve spojení s CBSE vyplývá z nutnosti udržení přehledu o struktuře aplikace.

Rozsáhlejší komponentově orientované aplikace obsahují typicky stovky až tisíce komponent s komplikovanou vnitřní strukturou, mezi kterými dochází k interakcím, jejichž počet je typicky ještě vyšší. Přičemž metody využívané k vizualizaci se stále nacházejí ve fázi vývoje [Snajberk et al.(2013)Snajberk, Holy,, Brada]. Vizualizační metody a nástroje jsou též konfrontovány s množstvím různých využívaných komponentových frameworků[1], což na ně klade požadavek jisté obecnosti pro umožnění analýzy komponent odpovídajících různým frameworkům a modelům.

Jako výhodná metoda pro získávání informací o komponentách, rozhraní a jejich interakcích se jeví postupy reverzního inženýrství (reverse-engineering).

Existující vizualizační nástroje často využívají orientovaných grafů, kde vrcholy představují komponenty a orientované hrany interakce. Při vysokých počtech komponent a vzájemných interakcí, zmíněných výše, se diagram stává těžko čitelným [Holy et al.(2012)Holy, Jezek, Snajberk,, Brada].

3.1 Techniky pro vizualizace diagramů

3.1.1 UML

Mezi nejznámější a nejpřímočařejší postupy při vizualizaci struktury nejen komponentových aplikací patří modelování pomocí diagramů grafického jazyka *Unified Modeling Language* (UML)[2] ve verzi 2.0 a vyšší. Tento jazyk disponuje řadou specifických i obecných nástrojů a modelů pro vizualizaci, dokumentování a návrh nejrůznějších informačních systémů.

Pro vizualizaci aplikací využívající komponentový model je možné využít

[1] Podrobněji v kapitole 2.1.4.

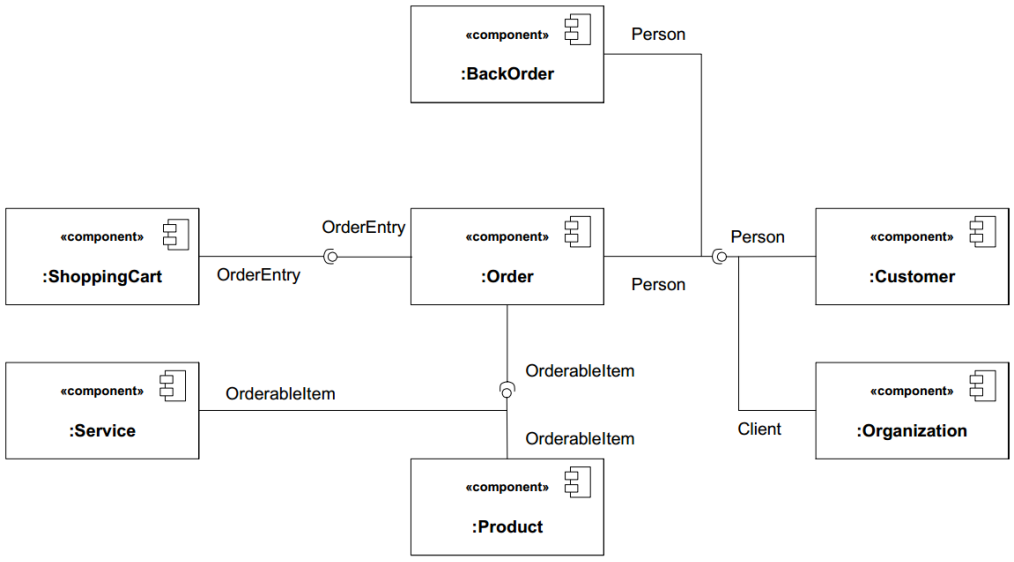
[2] Domovská stránka: http://www.omg.org/spec/UML/2.1.2/

například *UML component diagram* [UML()]. Tento statický model ovšem není zcela vhodný pro užití v rychle se vyvíjející oblasti CBSE.

Jako jeho nevýhody lze spatřovat především následující aspekty:

- Pro své zaměření na vyšší úroveň abstrakce, není schopen postihnout rozličné detaily – například odlišnosti libovolného komponentového modelu od ostatních. Tuto skutečnost je nutné dodatečně řešit přes uživatelské profily [Snajberk et al.(2013)Snajberk, Holy„ Brada].
- Překážku k jeho využití tvoří dle [Snajberk et al.(2012)Snajberk, Holy„ Brada] též nevhodné vizuální škálování, kvůli kterému je nutné často snižovat rozsah prezentovaných informací pro udržení úrovně použitelnosti.
- Prezentace veškerých informací pouze v jedné úrovni tvoří překážku ve snadné práci s tímto modelem. Není tedy například možné interaktivně zkoumat vnitřní, ve výchozím stavu skryté, detaily pouze těch komponent, které jsou předmětem zájmu.

Jako názorný příklad vzhledu a možností může sloužit velice jednoduchý UML diagram složený ze sedmi komponent zachycený na obrázku 3.1.



Obrázek 3.1: Příklad využití UML component diagramu.

3.1.2 Řešení nedostatků UML

Problémy použití jazyka UML v kontextu CBSE zmíněné v sekci 3.1.1 jsou řešeny různými přístupy a metodami, k jejichž popisu slouží tato kapitola.

Přílišná generičnost

Problém nedostatečného zaměření na jednotlivé komponentové modely je adresován například pomocí specializovaných vizualizačních nadstaveb, pracujících ovšem nad generickými analyzátory komponent, které dokáží zpracovávat komponenty náležící do odlišných modelů.

Jako zástupce těchto nástrojů lze jmenovat aplikaci ComAV (Component Application Visualizer)[3], která je vyvíjena Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Pro zajištění výše zmíněné kompatibility nástroje s různými modely využívá komponentový meta-model ENT [Snajberk – Brada(2011)Snajberk, Brada].

ComAV je aplikace založená na technologii *Eclipse Rich Client Platform* (RCP)[4], která pomocí svého generického jádra analyzuje komponenty z modelů, které jsou definovány pomocí pluginů typu *loader* a výsledky lze vizualizovat pomocí *visualize* pluginů. Tato architektura zajišťuje snadné rozšíření o podporu dalšího komponentového modelu [Brada et al.(2012)Brada, Holy„ Snajberk]. V době psaní této práce aplikace ComAV disponovala loader pluginy pro analýzu komponent založených na modelech frameworků OSGi, EJB3 a SOFA 2[5]. Lepší představu o architektuře aplikace lze získat z obrázku 3.2.

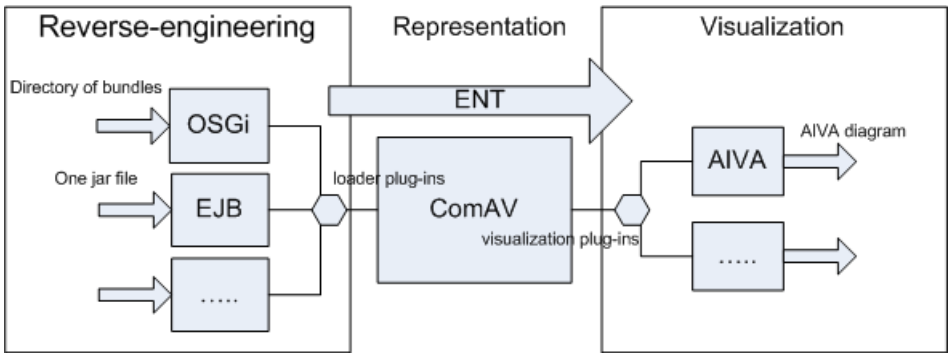
Jako vizualizační výstup této aplikace je možné použít zmíněné pluginy či jiné nástroje. Jednou z možností je použití nástroje CoCA-Ex[6] a nebo *AIVA*. Tento nástroj je vyvíjen stejným pracovištěm jako aplikace ComAV a poskytuje vysoce interaktivní přístup, který si klade za cíl odstranit nedostatky statického popisu pomocí UML. AIVA využívá jako základ vizualizace orientovaných grafů pro znázornění komponent a jejich interakcí [Snajberk et al.(2013)Snajberk, Holy„ Brada].

[3]Domovská stránka: http://www.kiv.zcu.cz/research/groups/dss/projects/dynamic-component-visualization.html

[4]Domovská stránka: http://www.eclipse.org/home/categories/rcp.php

[5]Blíže popsány v kapitole 2.1.4.

[6]Podrobněji v kapitole 3.2.



Obrázek 3.2: Architektura integrace aplikace ComAV [Snajberk – Brada(2012)Snajberk, Brada].

Nevhodné škálování a jednoúrovňovost

Problémy těchto typů lze (dle [Cockburn et al.(2008)Cockburn, Karlson„ Bederson]) v zásadě řešit využitím kombinací trojice základních technik.

- **Overview + detail** – Tato technika spočívá v poskytnutí dvou nebo více samostatných pohledů s různou úrovní detailů. Nejčastěji využívanou variantou v oblasti vizualizačních nástrojů je dvojice: detailní pohled (*detail*), který zabírá většinu prezentační plochy poskytované aplikací spolu s druhým menším pohledem (*overview*), který je odstiňuje od detailů a slouží tak k orientaci uživatele.

- **Zooming** – Zde je uživateli umožněno zvolit si úroveň detailů nad jím požadovanou částí prezentační plochy aplikace. Technika dovoluje volit detailní pohled pomocí zvětšení velikosti stávajícího zobrazovaného pohledu, nebo naopak volit méně detailní pohled pomocí zmenšení. První způsob je označován jako *zoom in* a druhý jako *zoom out*. Pomocí kombinace obou způsobů je tedy přímo kontrolována úroveň detailů.

- **Focus + context** – Jde o transformaci techniky *Overview + detail* do jediného pohledu tak, aby nebyly patrné ostré hranice mezi odlišnými pohledy. V prezentační ploše aplikace existuje oblast zájmu (*focus*), která částečně odpovídá pohledu detail, a zbytek pohledu tvořící oblast okolí (*context*), které má paralelu v pohledu overview. Nejčastěji využívanou implementací je pokřivená perspektiva pomocí *birds-eye* [Mackinlay et al.(1991)Mackinlay, Robertson„ Card] nebo *fisheye* [Sarkar – Brown(1993)Sarkar, Brown].

Existují i další nadstavbové techniky, které dále zlepšují uživatelské možnosti. Jako jednoho ze zástupců lze uvést techniky snižování vizuálního šumu (visual clutter) v diagramech [Holy et al.(2012)Holy, Jezek, Snajberk„ Brada].

Je zde využito skutečnosti, že vizualizovaná komponentová aplikace (nebo její část) velmi často obsahuje malý počet komponent s extrémně vysokým počtem interakcí. Vizualizace těchto interakcí ubírá prezentační plochu aplikace a též pro uživatele značně znepřehledňuje celý pohled. Jako řešení bylo navrhnuto (manuální nebo i automatické) přesunutí zmíněných komponent s vysokým počtem interakcí z hlavního pohledu do vyhrazeného izolovaného pohledu nazvaného *separated components area* (SeCo).

Popsaným postupem je zachována přehlednost pohledu a to bez narušení přehledu o celkové struktuře.

3.2 Nástroj CoCA-Ex

Webový nástroj Complex Component Applications Explorer (CoCA-Ex) má za úkol přinést vyšší efektivitu do procesu reverzního inženýrství komponentových aplikací. Mezi jeho přednosti vůči obdobným nástrojům, jako jsou *Eclipse Plug-in Dependency Visualization*[7], *Save-IDE*[8] nebo *SoMoX*[9] je nezávislost na komponentovém modelu nebo využití nadstavbových vizualizačních technik.

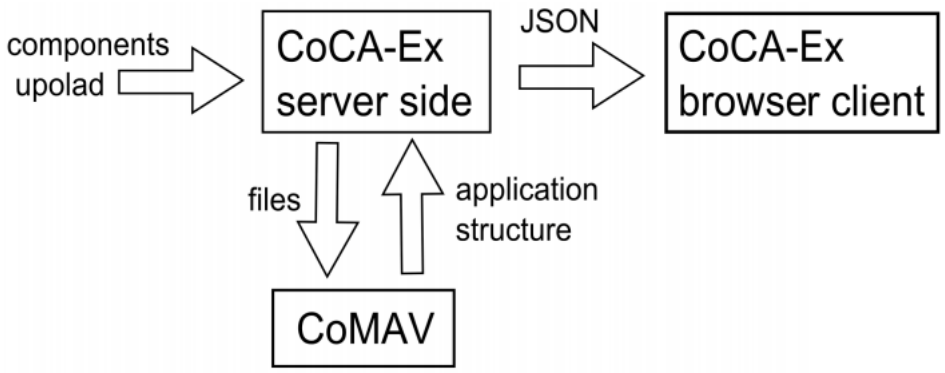
CoCA-Ex využívá platformy ComAV popsané v kapitole 3.1.2. Díky její generické povaze je tedy v současné době možné provádět vizualizaci systémů složených z komponent patřících do frameworků OSGi, EJB3 a SOFA 2 [Pavlikova(2012)].

Pro vlastní získání jednotlivých komponent z množiny předložených souborů je využita výše zmiňovaná aplikace ComAV. Po zvolení některého z podporovaných komponentových frameworků je této aplikaci předložena množina souborů ke zpracování. Výstup aplikace ComAV, který slouží k další vizualizaci, tvoří exportovaný seznam komponent spolu s jejich atributy. Mezi tyto atributy patří například části rozhraní komponent nebo jméno komponenty.

[7] Domovská stránka: http://www.eclipse.org/pde/incubator/dependency-visualization/
[8] Domovská stránka: http://sourceforge.net/projects/save-ide/
[9] SOftware MOdel eXtractor – http://www.palladio-simulator.com/tools/addons/somox/

Z exportovaného seznamu komponent je již nástrojem CoCA-Ex vytvořena grafová struktura, kde vrcholy přestavují komponenty a orientované hrany tvoří interakce mezi komponentami. Tyto interakce lze orientovat dle provided či required části (zde jsou obecně nazývány jako *exported* resp. *imported*) příslušného rozhraní. Tato transformovaná struktura slouží pro vlastní vizualizaci vstupních dat pomocí CoCA-Ex.

Celková architektura aplikace je zachycena na obrázku 3.3.



Obrázek 3.3: Architektura nástroje CoCA-Ex [Jezek et al.(2013b)Jezek, Brada, Holy„ Snajberk].

Nástroj CoCA-Ex se nadále nalézá ve stádiu vývoje a je předmětem změn a přidávání další funkcionality, kterou představuje například i tato práce.

3.2.1 Využívané technologie

Nástroj CoCA-Ex je webová aplikace založená především na Java a SVG[10] technologii. Použití webových technologií umožnilo snadnou uživatelskou přístupnost tohoto nástroje, protože aplikace neklade na klientské prostředí žádné požadavky, kromě webového prohlížeče, který musí splňovat kompatibilitu s níže zmíněnými technologiemi[11].

[10]Domovská stránka: http://www.w3.org/Graphics/SVG/
[11]Kupříkladu lze využít databázi: http://caniuse.com/

Java Servlet API

Jde o technologii náležící platformě Java EE[12], která zajišťuje zpracování požadavků vysílaných na webový server. Vlastní servlet je Java třída, která odpovídá Servlet API[13]. Podstatnou výhodou této technologie je možnost přístupu k dalším API z platformy Java (např. *JDBC*[14]) a také vysoká míra kontroly nad tvořenou aplikací.

JavaServer Pages

JavaServer Pages (JSP) představují technologickou nadstavbu nad Java Servlet, která umožňuje efektivnější tvorbu dynamických webových stránek. JSP stránky jsou příslušným kompilátorem překládány do Java Servlet.

SVG

Otevřený standard Scalable Vector Graphics (SVG)[15] představuje značkovací jazyk pro tvorbu planární vektorové grafiky. Je zde možné využívat obvyklé elementy, jako je například křivka, obdélník, elipsa, text, výplň, gradient a další. Tento standard tak umožňuje efektivní použití vektorové grafiky například v oblasti webových stránek. Podpora ze strany webových prohlížečů není v době psaní této práce úplná[16].

HTML5

Značkovací jazyk HyperText Markup Language (HTML) ve verzi 5 prochází schvalovacím řízením World Wide Web Consortium (W3C)[17]. Oproti svému předchůdci přináší celou řadu změn, jako například [HTML5()]:

- zjednodušenou definici hlavičky dokumentu, jako je `DOCTYPE`;

[12]Více na: http://www.oracle.com/technetwork/java/javaee/overview/index.html
[13]Domovská stránka: http://www.oracle.com/technetwork/java/javaee/servlet/index.html
[14]Java Database Connectivity – API definující přístup k relačním databázím.
[15]Domovská stránka: http://www.w3.org/Graphics/SVG/
[16]Dle: http://www.w3.org/Graphics/SVG/WG/wiki/Implementations
[17]http://www.w3.org/TR/html5/

- přímou podporu pro elementy standardu SVG nebo MathML;
- nové strukturální elementy, jako například: `section`, `article`, `main`, `header`, `footer` nebo `nav`;
- nové multimediální elementy, mezi které patří: `video`, `audio`, `time`, `progress`, `dialog`, atd.;
- řadu nových atributů;
- nové a pozměněné APIs a DOM elmenty.

Tyto změny tedy především cílí na snadnější podporu grafického a multimediálního obsahu. Podobně jako u výše zmíněného SVG je podpora tohoto standardu webovými prohlížeči neúplná[18].

CSS

Viz kapitola 2.3.1.

JavaScript

Viz kapitola 2.3.1.

jQuery

Viz kapitola 2.3.1.

MooTools

MooTools[19] (My Object-Oriented Tools) je otevřený JavaScript framework. Jeho přínos spočívá především v rozšíření podpory objektově orientovaného programování (OOP) v rámci JavaScriptu a též snazší manipulaci s DOM elementy. Tento framework je podporován všemi majoritními webovými prohlížeči.

[18]Blíže na: http://caniuse.com/
[19]Domovská stránka: http://mootools.net/

JSON

JavaScript Object Notation[20] (JSON) je otevřený standard pro znakově orientovaný přenos dat ve srozumitelné formě. Jeho ideovým základem je jazyk JavaScript a využívané datové struktury představují: spojové seznamy nazývané *arrays* a uspořádané dvojice název-hodnota nazývané *object*.

Většina rozšířených programovacích jazyků[21] disponuje podporou této technologie a díky tomu JSON představuje v oblasti webových aplikací jeden z nejrozšířenějších způsobů přenášení dat.

3.2.2 Popis nástroje

CoCA-Ex je webová aplikace rozdělená do dvou funkčních stránek – stránku pro nahrávání souborů/komponent a vlastní vizualizační stránku [Snajberk – Brada(2012)Snajberk, Brada]. Uživateli je dovoleno mezi nimi intuitivně navigovat.

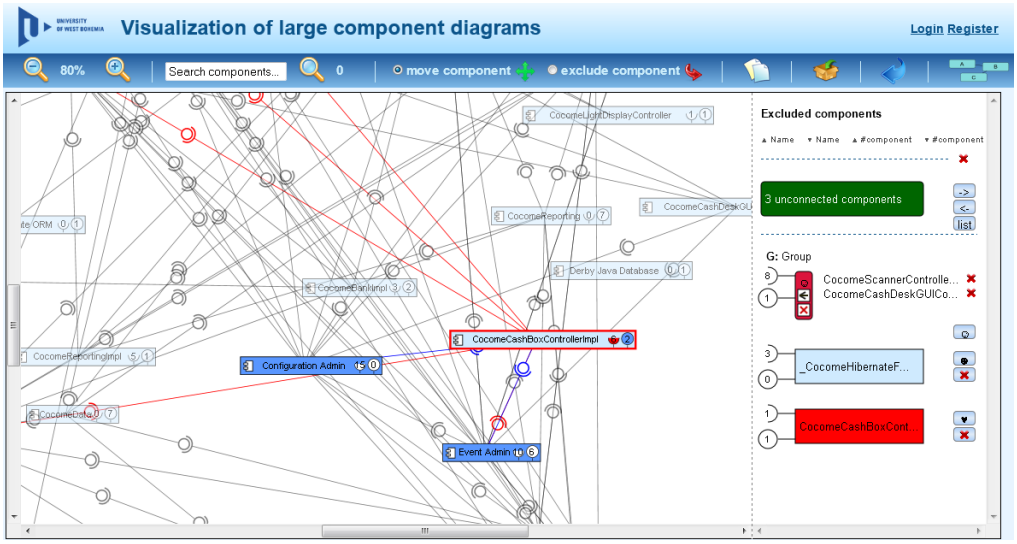
První zobrazená stránka po zadání URL uživateli umožňuje výběr a nahrání souborů pro následnou vizualizaci nebo jeho přihlášení do systému a následnou správu či vytvoření nového diagramu. Pro zavedení nové vizualizace je nejprve nutné dodat vstupní data pro aplikaci ComAV. Tudíž je nutné pomocí rolovací nabídky vybrat k jakému komponentovému frameworku vizualizované soubory přísluší. Druhým krokem je samotný výběr souborů k vizualizaci, který je umožněn pomocí standardního okna pro vícenásobný výběr souborů. Výběr souborů je zakončen potvrzovacím tlačítkem.

Po tomto potvrzení jsou předložené soubory zpracovány a uživatel je přesměrován na stránku s vlastním vizualizačním rozhraním nástroje – zde již může interaktivně pracovat s vygenerovaným diagramem.

Vlastní vizualizační rozhraní je zachyceno na obrázku 3.4. Největší díl rozhraní nástroje zabírá hlavní prezentační plocha, která poskytuje prostor pro interakci s diagramem. K hlavní ploše přiléhá postranní oblast nazývaná SeCo (blíže popsána v kapitole 3.1.2), která obsahuje vyjmuté jednotlivé komponenty či skupiny komponent. Rozhraní nástroje doplňuje nástrojová lišta situovaná do horního okraje, která umožňuje přístup k sadě akcí pomocí

[20] Domovská stránka: http://www.json.com/

[21] Například JavaScript, PHP, Java nebo platforma .NET.



Obrázek 3.4: Celkový pohled na rozhraní nástroje CoCA-Ex.

názorných ikon.

Uživateli je nad diagramem umožněna řada akcí [Jezek et al.(2013b)Jezek, Brada, Holy„ Snajberk]. Mimo jiné lze přesouvat komponenty z hlavní prezentační plochy do SeCo a zpět, lze zvýrazňovat interakce mezi komponentami, lze zjišťovat detaily o komponentách a jejich rozhraní, jednotlivé komponenty lze seskupovat do skupin a s těmi dále manipulovat, lze přeskupovat jednotlivé komponenty v diagramu, mezi komponentami lze vyhledávat fulltextovým vyhledáváním, přihlášený uživatel může ukládat diagram a podobně.

Interaktivita diagramu, potažmo celé aplikace, je zajištěna především díky využití kombinace technologií SVG a JavaScript popsaných v kapitole 3.2.1.

Další druhy interakce jsou předmětem rozšiřování funkčnosti, jak bylo naznačeno v kapitole 3.2.

4 Návrh a implementace

Tato kapitola se bude věnovat popisu vlastních provedených prací na obou výše zmíněných aplikacích – CoCA-Ex a EFFCC/EFPPortal.

Stěžejní prací bylo zakomponování podpory vizualizace EFPs (viz kapitola 2.2) v nástroji CoCA-Ex, který doposud touto možností nedisponoval. Další činností byly opravy chyb a implementace rozšíření nástroje CocCA-Ex.

4.1 Vizualizace mimofunkčních charakteristik

Požadavek v CBSE na určování kompatibility mezi komponentami, který byl nastíněn v kapitole 2.1.3, existuje též na úrovni EFPs. Z podstaty problému vyplývá, že je v této oblasti též možno uplatnit vizualizační techniky diagramů popsané v kapitole 4.1.

Pro ověření kompatibility softwarových komponent využívajících generický EFP framework[1] je využívána aplikace EFFCC/EFPPortal – konkrétně její část Compatibility Verification. Výstupem operace ověření kompatibility touto aplikací je vizualizace v podobě dynamické stromové struktury (viz též kapitola 2.3.2).

Tato výstupní struktura ovšem neposkytuje žádaný uživatelský komfort. Především však výstup neposkytuje informace týkající se struktury analyzované aplikace efektivně a v přehledné formě. Tedy informace o: jednotlivých interakcích, vlastnostech na rozhraní komponent a v neposlední řadě též stavu vyhodnocení EFPs.

Taktéž při vyšším počtu analyzovaných komponent (což je obvyklý případ, protože komponentově orientované aplikace se skládají běžně ze stovek komponent [Snajberk et al.(2013)Snajberk, Holy„ Brada]) se při použití aplikace neúměrně zvýší počet prvků této vizualizační struktury. Uživatel je tedy konfrontován s rozsáhlou strukturou, která ovšem zůstává rozdělena pouze na své dvě části – levou a pravou.

Jako řešení popsaných nedostatků ve výstupu aplikace EFFCC/EFPPor-

[1] Blíže v kapitole 2.2.3.

tal bylo navrženo použití vizualizačního nástroje CoCA-Ex, který je paralelně vyvíjený Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Lze tedy uvažovat o nástroji CoCA-Ex jako o vizualizační části, jehož zdroj dat představuje EFFCC/EFPPortal.

4.1.1 Návrh vizuální podoby

Nejprve bylo nutné zjistit typický scénář použití připravované vizualizace a jemu především přizpůsobit návrh vizuální podoby. Tudíž bylo analyzováno, že použití vizualizační části aplikace typicky představuje sled těchto činností:

- zahájení vlastní vizualizace;
- nalezení případných nekompatibilit v diagramu;
- získání bližších informací o dotyčných vlastnostech a EFPs komponent;
- ukončení vizualizace.

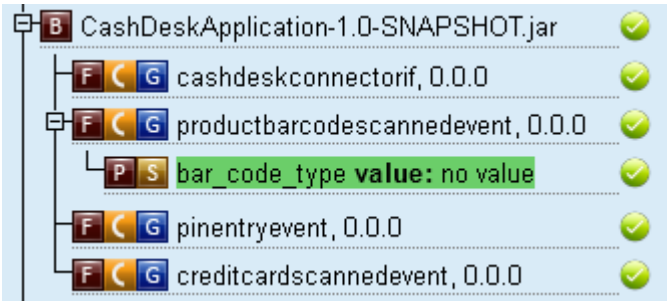
Navíc byla celková podoba vizualizace EFPs z části definována či omezena již použitými technologiemi u aplikace CoCA-Ex a z části též filosofií hierarchického zobrazení dat využívaného na straně aplikace EFFCC/EFPPortal.

Výše zmíněné hierarchické zobrazení představuje struktura: komponenta, její vlastnosti a případné EFPs. Tuto strukturu pro required stranu rozhraní u jednoho souboru/komponenty ukazuje obrázek 4.1, odpovídající strana provided i s komponentou je na obrázku 4.2.

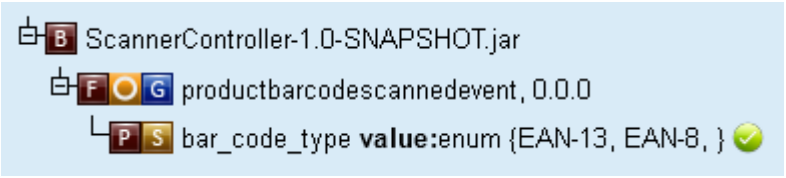
Obrázky zobrazují vztah komponenty `CashDeskApplication-1.0-SNAPSHOT` a `ScannerController-1.0-SNAPSHOT`, kdy první z nich vyžaduje svou částí required vlastnost `productbarcodescannedevent` s EFP nazvanou `bar_code_type` a obojí poskytuje svou částí provided druhá zmíněná komponenta.

4.1.2 Mapování interakcí do diagramu

Jelikož výše zmíněné rozhraní a interakce mezi dvěmi komponentami definují vztah, lze tuto dvojici vizualizovat jako orientované spojení v diagramu apli-



Obrázek 4.1: Hierarchická struktura na required straně zobrazení EFPs.



Obrázek 4.2: Hierarchická struktura na provided straně zobrazení EFPs.

kace CoCA-Ex. Toto spojení spojuje komponenty znázorněné v diagramu a je orientované ve směru daném příslušností provided a required částí ke komponentám. K tomuto spojení dále náleží i všechny jeho dané atributy, jako jsou vlastnosti a EFPs. Výslednou podobu vizualizace možného vztahu mezi komponentami je možné vidět na již uvedeném obrázku 2.2.

4.1.3 Zobrazení detailní informace

Nástroj CoCA-Ex již obsahoval možnost zobrazovat data náležící k různým prvkům aplikačního rozhraní ve formě výpisu podrobností. Tuto možnost poskytuje jQuery plugin *qTip*[2], který umožňuje přiřazení a následné zobrazení grafického prvku zvaného *tooltip*[3].

Alternativou by bylo například použití *jQuery UI*[4], avšak aplikace již využívala dříve zmíněný qTip a bylo tedy výhodnější ponechat stávající technologii kvůli konzistenci využití těchto prvků napříč aplikací. Ovšem, při dalším použití tohoto pluginu bylo zjištěno, že tento plugin bude vyžadovat provedení aktualizace. Poslední dostupná verze – qTip `2.2.0` – poté již vykazovala

[2]Domovská stránka: http://craigsworks.com/projects/qtip/
[3]Definice prvku dostupná na http://www.techterms.com/definition/tooltip
[4]Domovská stránka: https://jqueryui.com/

lepší chování například při dynamické změně velikosti zobrazované informace nebo při aktivaci poblíž okraje zobrazení.

4.1.4 Hierarchické zobrazení

Jako základ pro zobrazení EFPs tedy posloužil jQuery plugin qTip, který ovšem bylo nutné rozšířit o podporu hierarchické grafické struktury. Tato struktura je nutná pro udržení strukturované povahy vizualizovaných dat: komponenta, její vlastnosti a případné EFPs. Obdobně, jako v případě tooltipu, tak i u aplikace EFFCC/EFPPortal bylo výhodné využít některý z jQuery pluginů.

Pro hierarchické zobrazení je výhodné využít některý ze *stromových* (treeview) pluginů. Tyto pluginy umožňují používat grafické hierarchické struktury, které dovolují uživateli postupně expandovat jednotlivé úrovně zanoření a docilují tak úsporného zobrazení mezi zobrazovanými informacemi. Mezi stromovými pluginy existují hlavní zástupci: *jsTree*[5] a *jQuery Treeview*[6].

Při použití druhého zmíněného pluginu jsem ovšem narazil na problém s vykreslováním uvnitř výše zmíněného tooltip prvku qTip. Grafické prvky tohoto pluginu se vinou nekompatibility rozpadaly a problikávaly. Avšak ani použití prvního jmenovaného pluginu – jsTree se neobešlo bez problémů. Pro jeho chod je vyžadována verze knihovny jQuery `1.8.3` a vyšší, nástroj CoCA-Ex disponoval pouze verzí `1.7.2`. Bylo tedy v aplikaci nutné provést upgrade knihovny na verzi vyžadovanou pluginem jsTree.

Pořadí ve stromové struktuře je dodržené, aby byla usnadněna orientace uživatele a nebyly zanášeny zbytečné rozdíly mezi oběma aplikacemi. Proto byla zvolena následující struktura:

- v první úrovni zanoření se nalézá seznam vlastností;
- druhá úroveň obsahuje případně navázané EFPs;
- v nejvíce zanořené jsou podrobné informace příslušející k EFP.

Pokud k vlastnosti přísluší jedna nebo více EFPs, tak je u pojmenování vlastnosti umístěn symbol, který označuje stav vyhodnocení všech přísluš-

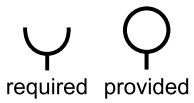
[5]Domovská stránka: http://www.jstree.com/
[6]Domovská stránka: http://jquery.bassistance.de/treeview/demo/

šejících EFPs. V tomto případě platí, že jediná nevyhovující EFP poruší vzájemnou kompatibilitu a tomu odpovídají i symboly. Je zřejmé, že tyto symboly se též vyskytují i na úrovni jednotlivých EFPs.

I na tomto místě využitá symbolika je pro usnadnění orientace převzata z aplikace EFFCC/EFPPortal. Zmíněný symbol může rozlišovat stavy (ty jsou blíže popsány v kapitole 2.3), kdy je výsledek analýzy EFPs v pořádku tj. komponenty jsou kompatibilní – zelený *tick* symbol nebo naopak pokud nejsou komponenty kompatibilní nebo EFP chybí – červený *cross*.

Ke každé ze zobrazovaných EFPs je nutné poskytnout detaily, které uživatel nástroje dále využije. Pro zobrazení podrobností o jednotlivých EFPs na úrovni nejhlubšího zanoření byla zvolena forma výpisu do jedné řádky. Zde se tak vyskytují informace:

- název datového typu zobrazované EFP, který je definován generickým EFP frameworkem[7];
- hodnota EFP na straně required s příslušným symbolem (viz obrázek 4.3);
- hodnota EFP na straně provided též s odpovídajícím symbolem (viz obrázek 4.3).



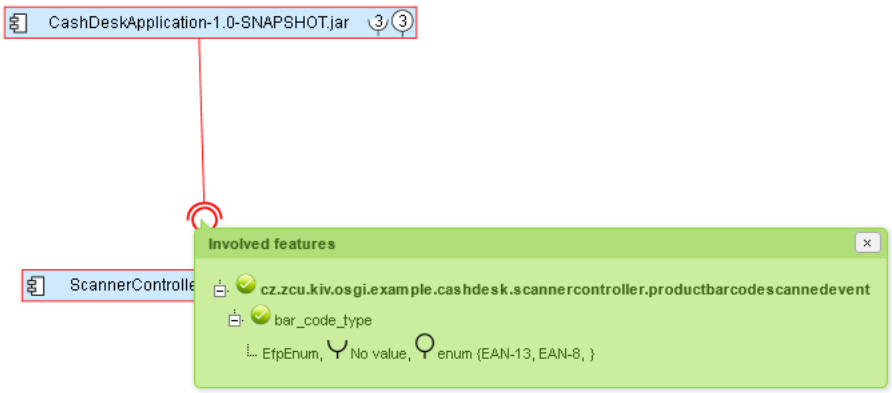
Obrázek 4.3: Použité provided a required symboly.

Pokud hodnota EFP není na některé ze stran interakce definována je tato informace oznámena. Příklad plně expandovaného hierarchického zobrazení implementovaného pomocí knihovny jsTree, které obsahuje jednu vlastnost a na ní navázanou jednu EFP, je uveden na obrázku 4.4.

4.1.5 Vizualizace stavů na spojeních diagramu

Jak vyplývá z případu užití, uvedeného na počátku kapitoly 4.1.1, je důležitá prvotní orientace uživatele nástroje ve stavech vyhodnocení mezi jednotli-

[7]Viz kapitola 2.2.3

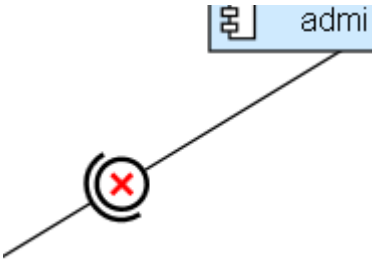


Obrázek 4.4: Expandované hierarchické rozhraní pomocí knihovny jsTree.

vými komponentami. Proto bylo vhodné zvolit reprezentaci těchto stavů již na úrovni spojení diagramu, jaké je na obrázku 2.2.

Stav nekompatibility

Nejdůležitější informací je indikace existence nesrovnalostí na daném spojení (viz kapitola 2.3). Pro takovéto spojení byly uvažovány různé podoby, jako například: „zuby“ v symbolu spojení, přeškrtnutí napříč nebo křížek v kruhu symbolu spojení. Nakonec byla zvolena poslední zmíněná reprezentace a celé spojení v diagramu je tak znázorněno jako kruh obsahující červený kříž – podoba znázorněna na obrázku 4.5. Tento symbol také koresponduje s podobou užitou při škálování velikostí symbolů[8].



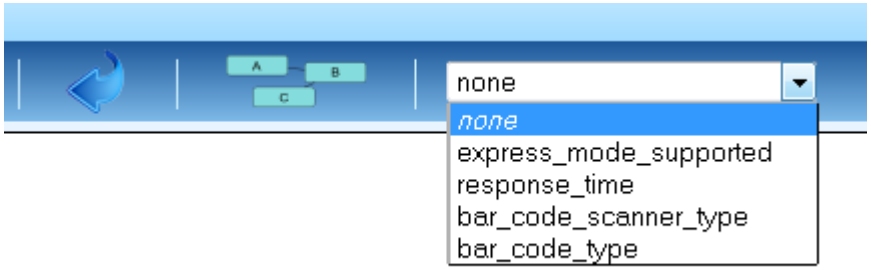
Obrázek 4.5: Symbol nekompatibilního spojení.

[8]Viz kapitola 4.1.5

Škálování velikosti symbolů spojení

Dalším vizualizačním prostředkem pro zobrazení stavu EFPs je zobrazení relativní velikosti symbolu spojení[9] v závislosti na hodnotě EFP.

Pro udržení míry přehlednosti v diagramu je uživateli nástroje pro tento druh vizualizace dovolen výběr pouze jedné konkrétní požadované EFP ze všech EFPs, které jsou definovány v analyzovaných interakcích komponent. Tento výběr názvu byl realizován pomocí rolovací nabídky v liště, která je zobrazena na obrázku 4.6, kdy po výběru názvu EFP okamžitě proběhne aktualizace vzhledu diagramu.



Obrázek 4.6: Rolovací nabídka s výběrem názvů EFPs.

Na straně serveru je umožněno definovat minimální a maximální velikost (v pixelech) pro zmíněnou relativní velikost symbolu pomocí nastavení proměnných `minInterfaceDiameter` a `maxInterfaceDiameter` v souboru `WEB-INF/web.xml`. První jmenovaná proměnná je využita jako výchozí. To znamená, že je využita pro velikost u počátečního zobrazení symbolů a též zobrazení bez zvolení EFP. Vlastní velikost obou částí (provided a required) symbolu je následně odvozena od hodnot, kterou nabývají části zvolené EFP v konkrétním spojení diagramu ve vztahu k ostatním hodnotám této EFP. Konkrétně jde o lineární závislost velikosti symbolů pro provided („lízátko“) a required („mistička“) na příslušející hodnotě EFP.

Vyskytují se zde různá specifika jako případ, kdy jedno spojení v diagramu obsahuje více různých vlastností, kterým ovšem náleží EFP shodného názvu. Jedno spojení tak může obsahovat EFP, které nabývají více hodnot díky příslušnosti k různým vlastnostem spojení.

Pro škálování symbolů v tomto případě bylo možné zvolit z různých vyjá-

[9] Viz symboly na obrázku 4.3.

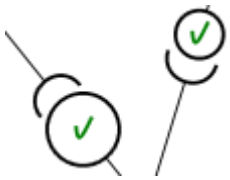
dření hodnot, například pomocí matematických funkcí: *min()*, *max()*, *sum()* nebo *avg()*. Jako nejuniverzálnější se jevil aritmetický průměr – avg() – a proto byl implementován jako způsob reprezentace hodnot při vícenásobném výskytu.

Dalším problémem byla různorodost datových typů, kterých mohou být hodnoty vlastních EFPs. Tuto různorodost vnáší již definice generického EFP frameworku[10]. Z tohoto důvodu byla zatím provedena implementace pouze pro datový typ `EfpNumber`, který představuje paralelu k datovému typu `double` jazyka Java.

Symboly spojení v diagramu mohou po zvolení konkrétní EFP nabývat různých stavů, které závisí na stavu spojení. Tyto stavy lze dělit dle několika hledisek:

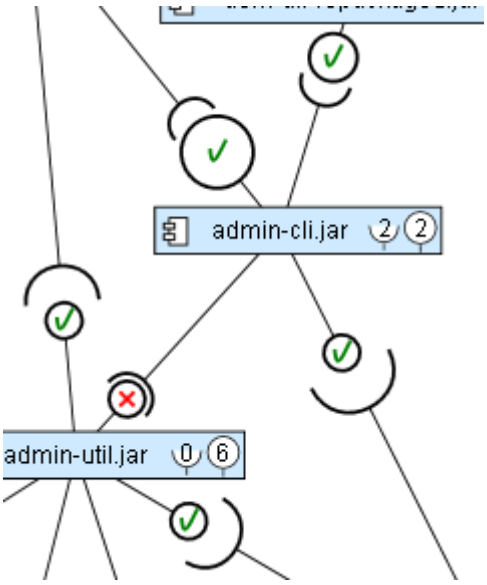
- **neobsahuje zvolenou EFP** – Obě části symbolu zůstávají ve výchozí velikosti definované proměnnou `minInterfaceDiameter` zmíněnou výše.
  - **spojení je kompatibilní** – Tvar symbolu se neliší od standardního, který lze vidět na obrázku 2.2.
  - **spojení není kompatibilní** – Symbol je doplněn o křížek[11], zachycené na obrázku 4.5.
- **obsahuje zvolenou EFP s podporovaným datovým typem** – Obě části symbolu jsou velikostně škálovány vzhledem k hodnotě EFP.
  - **spojení je kompatibilní** – Ke škálování je zaveden nový symbol „tick“, který je zachycen spolu se škálovanými symboly na obrázku 4.7.
  - **spojení není kompatibilní** – Spolu se škálováním je, obdobně jako v předchozím případě, použit symbol křížku.
- **obsahuje zvolenou EFP s nepodporovaným datovým typem** – Chování odpovídá případu, kdy spojení neobsahuje zvolenou EFP.
- **obsahuje zvolenou EFP a minimálně jedna strana spojení nemá definovanou hodnotu EFP** – Chování též odpovídá scénáři beze zvolené EFP.

[10] Viz kapitola 2.2.3 a příslušné odkazy.
[11] Blíže o funkcionalitě viz kapitola 4.1.5.



Obrázek 4.7: Relativní velikosti symbolů kompatibilního spojení.

Oba výše zmíněné symboly – křížek i „tick“ – byly implementovány tak, aby dodržovaly svou určenou orientaci a nenásledovaly tak měnící se orientaci symbolu spojení, ke kterému přísluší. Též se ukázalo jako nutné vyřešit stav, kdy značný rozdíl velikostí obou částí symbolu, zapříčiněný extrémními hodnotami EFP, znemožňoval vizuální kontrolu velikosti překryvem jedné z částí symbolu. Jako méně efektivní řešení se ukázalo být využití mírné průhlednosti částí symbolu, která tak zabránila překryvu. Tento způsob nevyhovoval jednak pro zaměření symboliky nástroje na styl odpovídající UML[12] a též působil rušivě. Proto byla tato varianta zavržena a bylo přistoupeno k využití změny vzdálenosti středů obou symbolů. Výsledná podoba, která ilustruje chování pro extrémní velikosti se nachází na obrázku 4.8.



Obrázek 4.8: Využití změny vzdálenosti pro zamezení překryvu symbolů.

[12]Domovská stránka: http://www.omg.org/spec/UML/2.1.2/

4.2 Integrace aplikací EFPPortal a CoCA-Ex

Pro zajištění funkcionality vizualizace analýzy vzájemné kompatibility EFPs bylo nutné provést systémovou integraci aplikace EFFCC/EFPPortal a nástroje CoCA-Ex. To znamená zajistit, aby byla data získaná první zmíněnou aplikací přístupná ve druhé.

4.2.1 Analýza zdroje dat

V kapitole 4.1.2 bylo popsáno, jak je možné mapovat výstupní hierarchickou strukturu aplikace na spojení diagramu v nástroji CoCA-Ex. Nejprve je tedy nutné získat vlastní data, která využívá aplikace EFPPortal jako výstupní zobrazení analýzy.

Tato data bylo nejvhodnější získat ještě před následujícími transformacemi na straně aplikace EFFCC/EFPPortal, a proto je představuje struktura `List<EfpEvalResult> efpResults` ze třídy kontroléru `EFPComparatorController`, která se nachází v metodě:

```
String stepResult(final String jSessionId, final String
    framework, final String localRegister, final Model
    model)
```

Zmíněná struktura je v závěrečném kroku vyhodnocení kompatibility komponent naplněna daty, které tvoří výstup z analýzy komparačním modulem (blíže popsán v kapitole 2.3) ze třídy `EfpComparator`. Konkrétně se jedná o návratovou hodnotu metody:

```
List<EfpEvalResult> evaluate(List<String> components,
    String assignmentModul, Integer lrID)
```

Tato struktura tak obsahuje data, která bude nutno přenést do aplikace CoCA-Ex.

4.2.2 Výběr přenosových technologií

Vlastní jednosměrnou komunikaci mezi webovými aplikacemi bylo možné implementovat s využitím různých přístupů a technologií. Zaslání dat vizualizačnímu nástroji tak bylo například možné realizovat pomocí:

- předání parametru v HTTP dotazu metodou *POST*[13];
- zápisem do databáze cílové aplikace;
- využitím univerzální komunikační technologie *Network Sockets*[14].

Z těchto možností bylo vybráno předání dat jako parametr HTTP POST dotazu díky tomu, že jde o snadno realizovatelnou metodu, která nevyžadovala žádné rozsáhlé úpravy.

Dále bylo nutné podrobit analýze výchozí formát přenášených dat. Zde se nabízely možnosti využití technologií jako je XML[15] nebo JSON. Obě tyto technologie lze využít pro serializaci objektů představujících strukturu naplněnou daty k přenosu. Jako finální možnost byl zvolen formát JSON především z důvodu již existující podpory na straně nástroje CoCA-Ex (viz kapitola 3.2.1) a též díky výchozímu doporučení zadavatele.

4.2.3 Formát přenášených dat

Po zvolení technologií bylo nutné definovat i objektový návrh, jehož prvky budou dále reprezentovat odesílaná data a přímo tak ovlivňovat formát přenášených dat.

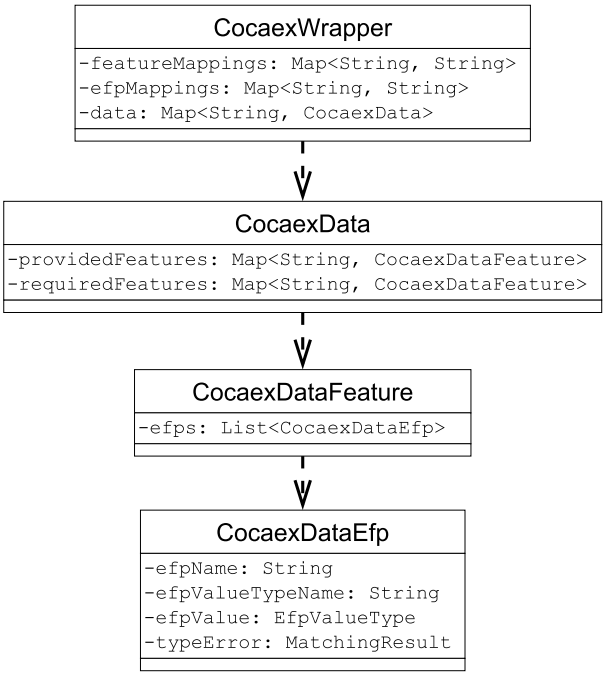
Na straně aplikace EFPPortal byly navrženy objekty, které byly posléze serializovány do datového formátu JSON. Tato část práce probíhala v součinnosti s oborovým projektem kolegy Hung Duong Manh[16].

Po analýze objektu výchozí datové struktury `List<EfpEvalResult> efpResults`, zmíněné v kapitole 4.2.1, bylo nutné navrhnout vlastní strukturu. Tato potřeba vyplynula z velkého množství informací nepotřebných pro další vizualizaci, které zmíněná výchozí struktura obsahovala. Tyto informace by pouze zvětšovaly objem přenášených dat mezi aplikacemi.

Proto byla navržena vlastní datová struktura pro přenášená data, kterou zachycuje diagram tříd na obrázku 4.9. Tento diagram popisuje konečnou podobu struktury, kdy byla doplněna třída `CocaexWrapper`, jež má atributy

[13]Podrobněji na: http://www.jmarshall.com/easy/http/#postmethod
[14]Podrobněji na: http://docs.oracle.com/javase/tutorial/networking/sockets/
[15]Domovská stránka: http://www.w3.org/XML/
[16]E-mail: zerx@students.zcu.cz

`efpMappings` a `featureMappings`, které fungují jako slovníky přiřazující řetězci názvu EFP a nebo vlastnosti jejich jedinečný číselný identifikátor. Tento identifikátor figuruje na místech původního výskytu daného jména v atributu `data` téže třídy a slouží tak pro efektivní zkrácení dlouhých názvů, jež se pro pojmenování EFPs a vlastností používají. K použití těchto slovníků muselo být přistoupeno pro zmenšení objemu přenášených dat, který byl bez jejich použití neakceptovatelný.



Obrázek 4.9: Diagram tříd – Navržená datová struktura pro přenášená data.

4.2.4 Příprava dat k přenosu

Datová struktura představovaná objektem `CocaexWrapper` byla po své inicializaci daty postoupena k serializaci. K tomuto úkonu byla využita knihovna *Google Gson*[17]. Data serializovaná tímto způsobem ovšem vykazovala přílišný objem. Proto bylo nutné ještě zmenšit tento objem provedením bezeztrátové komprese serializovaných dat.

Pro kompresi byla použita metoda *gzip*[18], která je již implementována

[17]Domovská stránka projektu: https://code.google.com/p/google-gson/
[18]Podrobná specifikace na: http://tools.ietf.org/html/rfc1952

v knihovně *Apache Commons IO*[19]. Po dokončení komprese bylo ještě nutné provést kódování komprimovaných binárních dat do znakové podoby. K tomuto účelu posloužilo kódování *Base64*[20], které se v obdobných případech využívá. Konkrétně byla využita knihovna *Apache Commons Codec*[21].

Po aplikování výše zmíněných úprav podoby odesílaných dat – nahrazení jmen číselnými identifikátory a použití komprese – bylo dosaženo významné redukce jejich velikosti vůči velikosti před těmito úpravami. Tato redukce dosahovala až poměru přibližně 30:1[22].

Takto připravená data byla vložena do POST formuláře s neviditelným textovým polem pojmenovaným `data` na poslední funkční stránce, která je ovládána kontrolérem `EFPComparatorController` (představený v kapitole 4.2.1) a představuje poslední krok – zobrazení výsledků. Až po provedení těchto kroků byla data připravena na vložení jako parametr HTTP POST dotazu s následovným přijetím aplikací CoCA-Ex. Adresa tohoto dotazu je určena pomocí URL vedoucí na běžící instanci aplikace CoCA-Ex, konkrétně na její stránku `EfpImport`. Zmíněnou URL je možné konfigurovat pomocí proměnné `cocaexUrl` v souboru `WEB-INF/applicationContext.xml`. Odeslání proběhne po kliknutí na tlačítko umístěné na stránce s krokem zobrazení výsledků pod vizualizační strukturou (viz obrázek 2.6).

4.2.5 Přijmutí dat na straně CoCA-Ex

Postup při přijmutí dat na straně nástroje CoCA-Ex svými kroky odpovídá postupu popsanému v kapitole 4.2.4 provedeném v opačném pořadí.

Jde tedy o postupné přijení dat, jejich dekódování ze znakového kódování Base64 do binárního formátu, gzip dekomprimování a zakončené deserializací. Pro příjem dat byl do nástroje CoCA-Ex přidán servlet `EfpImport`, u něhož je pro vlastní příjem využita metoda:

```
void doPost(HttpServletRequest request,
   HttpServletResponse response)
```

Pro provedení dekódování a dekomprimování byly využity metody třídy

[19]Domovská stránka: http://commons.apache.org/proper/commons-io/
[20]Podrobná specifikace na: https://tools.ietf.org/html/rfc3548
[21]Domovská stránka: http://commons.apache.org/proper/commons-codec/
[22]Na testovacích datech *glassfish* – ~3MB přenásených dat oproti ~100kB.

`JsonTransformer`, které používaly stejné knihovny, jako na straně aplikace EFPPortal, které byly popsané též v kapitole 4.2.4. K provedení deserializace přijatého obalovacího objektu `CocaexWrapper` (celá struktura viz obrázek 4.9) bylo nutné nutné vytvořit nový deserializér. Ten byl implementován pomocí třídy `WrapperDeserializer`.

Po získání deserializovaného objektu `CocaexWrapper` bylo nutné transformovat přijatá data do formátu, který nástroj využívá k vlastnímu zobrazení diagramu. Tuto transformaci mají na starost metody třídy `EfpGraphTransfomer`, jejichž volání vrací řetězec ve formátu JSON, který je možné použít jako vstupní data pro zobrazení. Tyto metody inicializují data v instancích tříd `GraphInterface`, `EdgeEfp` a `VertexEfp`. Dokončení inicializace je následováno serializací objektu `GraphExport`, po které jsou získána data ve formátu JSON sloužící pro zobrazení diagramu.

4.2.6 Zpracování přijatých dat

Přijatá data transformovaná do podoby JSON využívané pro zobrazení jsou ve formě session servletem `EfpImport` uchována a dále je tímto servletem provedeno přesměrování na stránku `index.jsp`. Po přistoupení na tuto stránku je, podobně jako u obvyklého použití nástroje CoCA-Ex s daty poskytnutými nástrojem ComAV, zahájen proces tvoření diagramu pomocí JavaScript funkce:

```
function loadGraphData(diagram_id, diagram_hash, with_efps
    , efp_settings)
```

Tato funkce získá data ve formátu JSON jako odpověď na požadavek vyslaný na servlet `LoadGraphData`. Zdroj dat tohoto servletu v tomto případě tvoří zmíněná session, do které byla data předtím uložena servletem `EfpImport`. Po ukončení přenosu dat ze serveru na stranu klienta je na základě přijatých dat zahájen vlastní proces tvorby SVG struktury diagramu. Zmíněný proces je se liší od standardního, který využívá nástroje ComAV, především použitím tříd specifických pro EFP strukturu diagramu[23] a též zpřístupněním řady jinak deaktivovaných prvků (věnuje se jim kapitola 4.1).

Nejprve jsou inicializovány struktury pro přístup ke spojením diagramu na základě jména konkrétní EFP. Tyto struktury se nacházejí v `efps.js`.

[23] Jde především o třídy: `EdgeEfp` a `VertexEfp`.

Tento přístup umožní rychlou funkci zvýrazňování u výběru konkretní EFP blíže popsané v kapitole 4.1.5. Též je při tvorbě SVG struktury diagramu vyhodnocena EFP nekompatibilita jednotlivých spojení a je tak v těchto případech vytvořena symbolika popsaná v kapitole 4.1.5.

Následně po vytvoření struktury diagramu je potřeba provést inicializaci struktur pro zobrazení detailních informací o EFPs na jednotlivých spojeních diagramu. Tomuto účelu slouží funkce z `tooltips.js`, které provedou změny ve standardním zobrazení pro účely EFP diagramu tak, jak bylo popsáno v kapitolách 4.1.3 a 4.1.4. Především se jedná o správnou inicializaci jQuery pluginů qTip a jsTree[24]. Převážná část konstrukce zobrazované struktury detailních informací je vykonána funkcemi:

```
function getTooltipEdge(id)
function configurationEdgeTooltip(selectorString)
```

Po provedení inicializace již lze se zobrazeným diagramem pracovat. Celkovou představu o komunikaci v aplikaci probíhající mezi klientem a serverem při přijmu dat z aplikace EFFCC/EFPPortal a jejich následného zobrazení pomáhá utvořit obrázek 4.10.

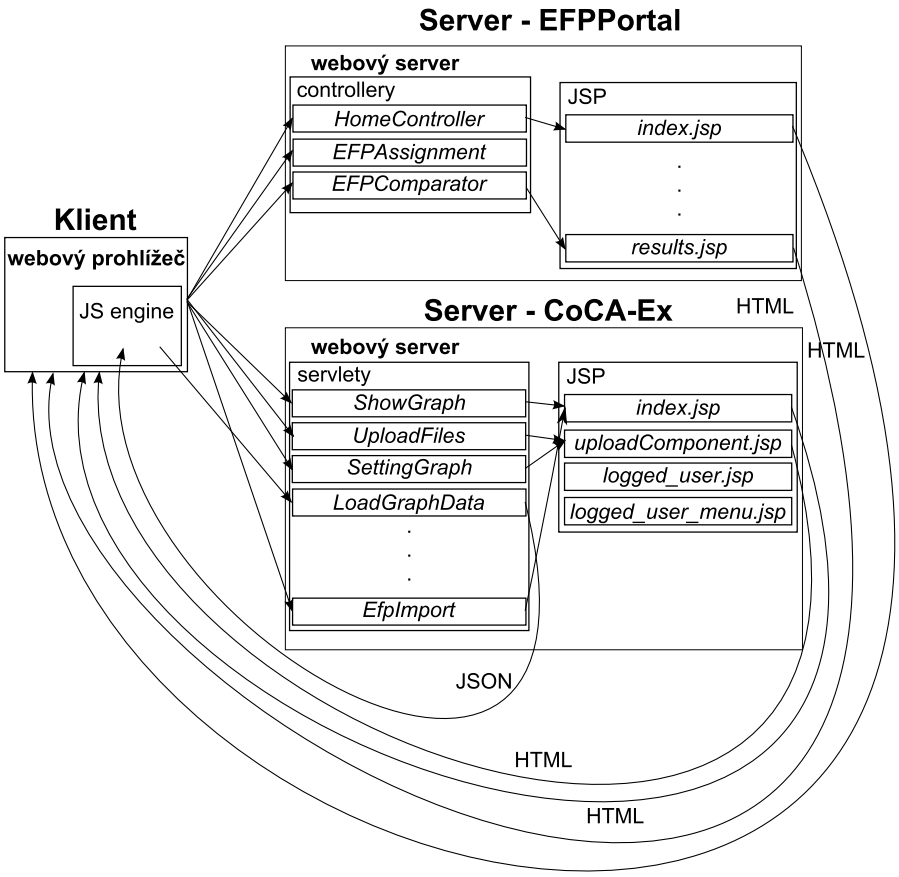
4.3 Zavedení pojmenovaných skupin komponent

V souladu s přístupem vizualizace rozsáhlých diagramů (např. [Holy(2012)]) je výhodné seskupovat vizualizované komponenty do skupin. Do jednotlivých skupin lze zařazovat ty komponenty, které společně tvoří nějakou funkčnost. Tento přístup podobně jako další přináší zpřehlednění diagramu a též do vizualizace vnáší sémantiku.

Z tohoto důvodu jsou i v nástroji CoCA-Ex implementovány skupiny komponent. Uživatel zde má možnost vytvářet tyto skupiny nebo je též přesouvat mezi hlavní aplikační plochou a SeCo (podrobně v kapitole 3.2.2).

Ve stávající implementaci ovšem nebyl bezezbytku využit potenciál sémantiky, který implementace skupin přináší. Běžný případ užití totiž počítá pro diagram o řádově stovkách komponent s poměrně velkým množstvím skupin. Zde je, jak bylo zmíněno výše, v každé typicky seskupena jedna funkč-

[24]Viz též kapitoly 4.1.3 a 4.1.4

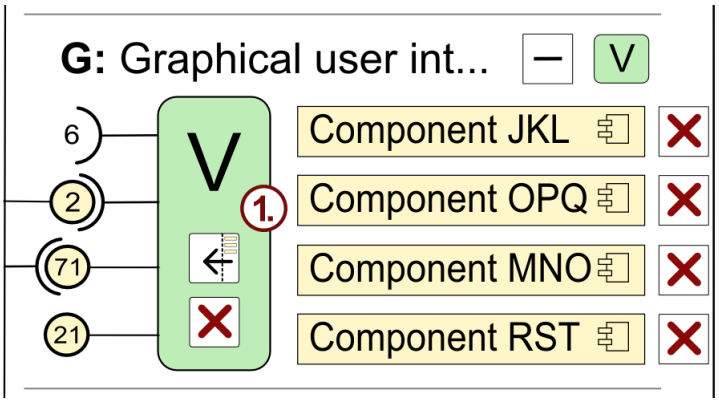


Obrázek 4.10: Schéma komunikace klient-server.

nost. Pro udržení sémantiky je tak vhodné zavést pro skupiny pojmenování a umožnit tak jejich snazší rozlišení. Tento přístup byl naznačen v návrhovém dokumentu pro nástroj CoCA-Ex a je zobrazen na obrázku 4.11.

Pro pojmenování skupiny byla zaveden atribut `label` ve třídě `Group` nabývající výchozí hodnoty *Group*. Dále bylo potřeba modifikovat SVG struktury, které zobrazují symbol skupiny jak na hlavní aplikační ploše tak i v SeCo. Těmto strukturám byl přidán prvek nadpisu, který dodržuje vzhled a umístění dle designového návrhu (viz obrázek 4.11) a zobrazuje hodnotu zmíněného atributu s názvem skupiny. Počet znaků tohoto názvu byl též v souladu s návrhem omezen.

Dále bylo nutné vyřešit otázku změny hodnoty pojmenování skupiny. Jako nejvýhodnější alternativa se jevil mechanismus kliknutí na zobrazený název skupiny, který by se následně změnil tento zobrazené pojmenování na vstupní



Obrázek 4.11: Návrh designu pojmenované skupiny v SeCo.

pole, do kterého by bylo možné zadat nové pojmenování. Potvrzení nového pojmenování by bylo realizováno přesunutím kurzoru (focus) mimo oblast názvu. Tato možnost se ovšem následně ukázala jako nerealizovatelná pro oba typy skupin zároveň – v hlavní aplikační ploše i SeCo. V hlavní aplikační ploše totiž nebylo možné pomocí JavaScriptu manipulovat s obsahem SVG elementu `<text>`, který zde musí tvořit název skupiny. Tento problém se vyskytoval i v aktuální verzi prohlížeče *Firefox*[25]. Nebylo tedy možné v tomto případě nahradit název skupiny za vstupní pole tak, jako je to možné u názvu skupiny v SeCo, která může být tvořena standardními HTML elementy (v tomto případě `<span>`).

Nesoulad mezi změnou pojmenování mezi oběma skupinami by pouze zaváděl nekonzistenci a znepříjemňoval užívaní nástroje. Proto byl zvolen alternativní přístup zadávání údajů a to pomocí dialogu. Konkrétně byla zvolena implementace jQuery pluginem *Dialog*, který patří do značně populárního balíku pluginů *jQuery UI*[26].

Dialog pro změnu pojmenování skupiny definovaný v `index.jsp` se objeví po kliknutí na aktuální název skupiny díky zaregistrování příslušné události na grafické elementy názvů skupin a to pomocí volání `dialog("open")`. Jako výchozí pojmenování je ve vstupním poli dialogu předem nastaveno stávající pojmenování. Celkové rozvržení dialogu je zachyceno na obrázku 4.12.

Pro další interakci s dialogem slouží tato tři tlačítka:

[25]Firefox je cílovým prohlížečem pro tuto aplikaci. Domovská stránka: http://www.mozilla.org/cs/firefox/new/

[26]Domovská stránka: https://jqueryui.com/

Obrázek 4.12: Dialog pro změnu pojmenování skupiny nazvané *DB*.

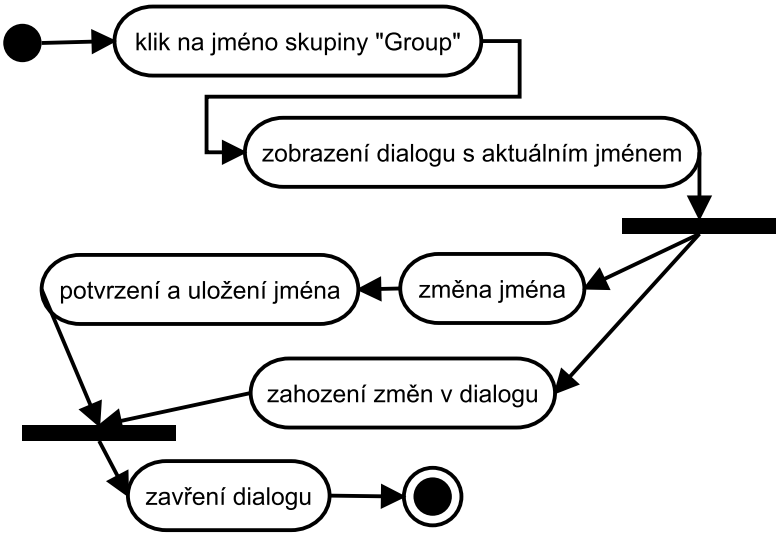
- **Clear** – Vymaže stávající obsah vstupního pole.
- **Storno** – Zavře dialog a zachová původní pojmenování skupiny. Stejnou funkcionalitu zastává klasický symbol křížku v pravém horním rohu.
- **OK** – Zavře dialog a uloží změny v pojmenování skupiny pomocí volání funkce `saveDialogValue(callingParent)`.

Pro zvýšení uživatelského komfortu je po aktivaci dialogu kurzor nastaven do jeho vstupního pole, výše zmíněnému potvrzovacímu tlačítku OK je přiřazena obvyklá klávesová zkratka *Enter* a stejně tak tlačítku Storno je přiřazena klávesa *Esc*.

Celý proces změny názvu skupiny popisuje diagram aktivit na obrázku 4.13.

4.4 Přidání ukázkového diagramu

Nástroj CoCA-Ex se stále nachází ve fázi vývoje, a proto vyvstala potřeba pro rychlou demonstraci dosažených výsledků a pro praktický a efektivní debugging klientské části aplikace s využitím reálných dat. Původně nástroj k uvedeným účelům nedisponoval žádným dedikovaným prostředkem a musel



Obrázek 4.13: Diagram aktivit – Změna pojmenování skupiny.

tak využívat základního postupu nahrání komponent, popsaném v kapitole 2.3.2.

Tento postup byl shledán jako nevyhovující pro svou nedostatečnou rychlost a též nutností mít vždy k dispozici připravenou množinu souborů. Z výše zmíněných důvodů byla navržena implementace rozšíření nástroje o funkcionalitu umožňující rychlé zobrazení předem připraveného ukázkového diagramu.

Pro co nejrychlejší možný přístup k ukázkovému diagramu byl na něho přidán odkaz již na úvodní stránce v jejím levém sloupci rozvržení. Tento sloupec tedy obsahuje jak odkaz na ukázkový diagram, tak může též obsahovat odkazy na diagramy uložené uživateli nástroje.

Uložení ukázkového diagramu bylo implementováno pomocí souboru uloženého na serveru. Tento soubor obsahuje data ve formátu JSON[27], která odpovídají datové struktuře, kterou používá klient k zobrazení diagramu a je mu poskytována serverem. Tento formát se jevil jako nejvýhodnější díky již existující podpoře a tudíž snadné implementaci.

Soubor s ukázkovými daty je umístěn v adresáři `WEB-INF` a nese jméno `demo_diagram_JSON_1.dat`. Číslo na konci názvu souboru slouží jako jeho identifikátor, je tak usnadněno rozšíření ukázkových diagramů v aplikaci

[27]Podrobnosti zmíněny např. v kapitole 3.2.1.

Obrázek 4.14: Umístění odkazu na diagram `Parkoviste` ve frameworku OSGi.

o další zástupce. Zmíněný soubor s ukázkovými daty byl získán uložením přenášených dat ve formátu JSON v nástroji CoCA-Ex při vizualizaci množiny komponent zvané *Parkoviště* – jde tedy o data použitelná k testování.

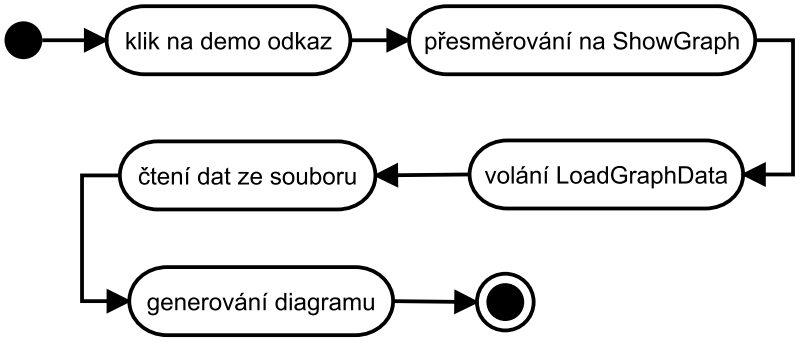
Po kliknutí na odkaz s ukázkovým diagramem proběhne přesměrování na servlet `ShowGraph` a předání parametrů o použitém komponentovém frameworku spolu s identifikátorem souboru s daty. Předané parametry jsou poté uloženy do session.

Uložení umožní, že při následném volání servletu `LoadGraphData` je namísto volání metod třídy `GenericComponentLoader` – což představuje zpracování dat poskytnutých nástrojem ComAV (popsaný v kapitole 3.1.2) – inicializuje instanci třídy `DemoDiagramLoader` zavolá metodu:

```
String readDemoJSONFromFile(InputStream in)
```

Tato metoda vrací data jako řetězec ve formátu JSON, který byl načten z předem definovaného souboru obsahujícího ukázkový diagram. Tento proces je zachycen na obrázku 4.15.

Čili je v tomto případě změněn zdroj dat ve formátu JSON na straně serveru a veškeré následné kroky algoritmu konstrukce diagramu na straně klienta proběhne standardně. Uživateli nástroje tedy pro zobrazení ukázkového diagramu stačí pouhé následování odkazu z hlavní strany, který je zvýrazněný na obrázku 4.14, což přináší zmíněné zlepšení použitelnosti nástroje.

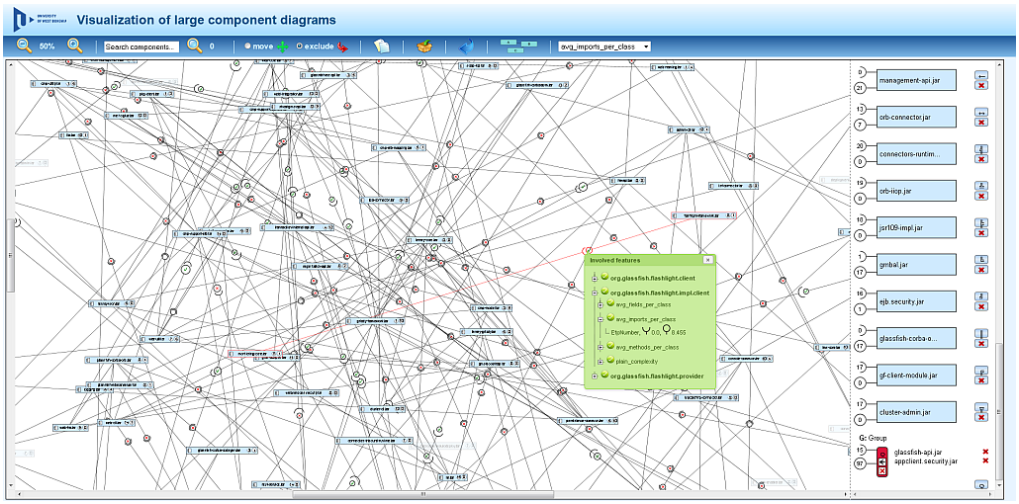


Obrázek 4.15: Diagram aktivit – Sestavení ukázkového diagramu.

5 Ověření a demonstrace funkčnosti

Pro ověření funkčnosti bylo provedené testování obou aplikací nad testovacími daty *glassfish-with-metrics* (rozsáhlá data s 233 komponentami), *efp-PortalBundle* (omezená demonstrační data s 8 komponentami). Též byly k testování využity podmnožiny těchto dat.

K ověřování byl využit internetový prohlížeč Firefox 28[1]. Prohlížeč Google Chrome verze `33.0.1750.117 m`[2] vykazoval nestabilní chování s náhodnými chybami při vizualizaci rozsáhlejších dat, a proto nebylo možné jej použít pro ověření. Ostatní rozšířené internetové prohlížeče nesplňují požadavky na podporu technologií, které obě testované aplikace využívají[3].



Obrázek 5.1: Celkový pohled na rozhraní aplikace.

[1]Domovská stránka: http://www.mozilla.org/cs/firefox/new/
[2]Domovská stránka: http://www.google.com/chrome/
[3]Viz kapitoly 3.2.1 a 2.3.1

6 Závěr

Cílem této diplomové práce bylo seznámení se s technikami vizualizace rozsáhlých diagramů a s příslušnými nástroji, které jsou vyvíjeny Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Déle navrhnout vhodnou formu vizualizace mimofunkčních charakteristik softwarových komponent v nástroji CoCA-Ex, implementovat tuto zvolenou formu vizualizace pro data zpracovaná nástrojem EFCC a v neposlední řadě také rozšířit nástroj CoCA-Ex o další funkcionalitu.

Nejdříve byly v kapitolách 2 a 3 této práce stručně popsány možnosti mimofunkčních charakteristik v komponentově orientovaných softwarových technologiích spolu s nástroji CoCA-Ex a EFFCC. Poté následoval návrh podoby vizualizovaných dat, který byl podřízen tomu, aby ideově navazoval na již implementované uživatelské rozhraní obou nástrojů. Zmíněnému návrhu výsledné podoby se věnuje kapitola 4.1 této práce. Kapitoly 4.2 až 4.4 se zabývají popisem implementace navržené formy vizualizace a dalších rozšíření funkčnosti nástroje. V kapitole 5 je obsaženo závěrečné ověření funkčnosti aplikace.

Vypracování práce bylo splněno v souladu se zadáním a výsledná podoba nástroje CoCA-Ex tak může díky množství nově implementovaných rozšíření poskytnout širší možnost uplatnění a zároveň též uživatelsky přívětivější rozhraní.

Seznam obrázků

Seznam zkratek

CBSE	Component-Based Software Engineering – Technologie vývoje software založená na komponentách.
CoCA-Ex	Complex Component Applications Explorer – Komplexní softwarový nástroj pro vizualizaci rozsáhlých diagramů. Vyvíjený Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni.
ComAV	Component Application Visualizer – Softwarový vizualizační nástroj. Vyvíjený Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni.
CoSi	Components Simplified – Komponentový model vyvíjený Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni.
CSS	Cascading Style Sheets – Jazyk pro zápis formátu a vzhledu dokumentů.
EFFCC	Extra Functional Property Featured Compatibility Checks – Sada nástrojů umožnujících rozšíření existujících komponentových modelů o kontrolu kompatibility EFPs. Jde o nástroje vyvíjené Katedrou informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni.
EFP	Extra-Functional Property – Mimofunkční charakteristika komponenty. Jde o charakteristiky blíže popisující chování, synchronizaci, kvalitu služeb nebo další podobné kvality komponenty.
EJB3	Enterprise Java Beans 3 – Třetí verze standardů využívající platformy Java Enterprise Edition.

HTML5	HyperText Markup Language 5 – Pátá verze značkovacího jazyka používaného pro tvorbu webového obsahu.
HTTP	Hypertext Transfer Protocol – Protokol pracující na aplikační vrstvě modelu TCP/IP. Slouží pro přenos hypertextových dokumentů.
JAR	Java Archive – Souborový formát využívaný technologiemi Java.
JS	JavaScript – Interpretovaný jazyk se syntaxí vycházející z jazyka C.
JSON	JavaScript Object Notation – Datový formát objektů jazyka JavaScript využívaný v řadě dalších jazyků.
OSGi	Open Service Gateway initiative – Sada otevřených specifikací, které definují komponentový model.
SeCo	Separated Components – Vyhrazená část prostředí nástroje CoCA-Ex. Slouží pro vydělení částí diagramu za účelem zlepšení přehlednosti.
SOFA 2	Software Appliances 2 – Implementace hierarchického komponentového modelu. Vyvíjeno na Katedře distribuovaných a spolehlivých systému Matematicko-fyzikální fakulty Univerzity Karlovy v Praze.
SVG	Scalable Vector Graphics – Značkovací jazyk pro tvorbu planární vektorové grafiky.
UML	Unified Modeling Language – Univerzální modelovací jazyk využívaný při návrhu a dokumentaci software.
URL	Uniform Resource Locator – Řetězec znaků, který identifikuje umístění zdrojů.
XML	Extensible Markup Language – Rozšířený jazyk logického vyznačování.

Literatura

[Bachmann(2000)] BACHMANN, F. Volume II: Technical Concepts of Component-Based Software Engineering. Technical report, Carnegie Mellon, Software Engineering Institute, 2000.

[Bondarev et al.(2006)Bondarev, Chaudron„ de With] BONDAREV, E. – CHAUDRON, M. R. – WITH, P. H. Compositional Performance Analysis of Component-Based Systems on Heterogeneous Multiprocessor Platforms. Technical report, Eindhoven Univ. of Technology, 2006.

[Brada(2008)] BRADA, P. The CoSi Component Model: Reviving the Black-Box Nature of Components. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2008.

[Brada et al.(2012)Brada, Holy„ Snajberk] BRADA, P. – HOLY, L. – SNAJBERK, J. Visual clutter reduction for UML component diagrams: A tool presentation. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2012.

[Cockburn et al.(2008)Cockburn, Karlson„ Bederson] COCKBURN, A. – KARLSON, A. – BEDERSON, B. B. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*. 2008, 41.

[CoSii()] CoSii. DSS: The CoSi Component Model. [online, navštíveno 10. 2. 2014], 2010. Dostupné z: http://www.kiv.zcu.cz/research/groups/dss/projects/cosi.html.

[D3S()] D3S. Projects @ D3S - Department of Distributed and Dependable Systems. [online, navštíveno 10. 2. 2014], 2010. Dostupné z: http://d3s.mff.cuni.cz/projects/components_and_services/sofa.

[EJB()] EJB. Overview of Assembly and Deployment (Sun GlassFish Enterprise Server 2.1 Application Deployment Guide). [online, navštíveno 10. 2. 2014], 2010. Dostupné z: http://docs.oracle.com/cd/E19879-01/820-4337/beacu/index.html.

[EJB3()] EJB3. Enterprise JavaBeans Technology. [online, navštíveno 10. 2. 2014], 2014. Dostupné z: http://www.oracle.com/technetwork/java/javaee/ejb/index.html.

[Franch(1998)] FRANCH, X. Systematic Formulation of Non-Functional Characteristics of Software. Technical report, Dept. Llenguatges i Sistemes Informatics Universitat Politecnica de Catalunya, 1998.

[Holy(2012)] HOLY, L. *Large component diagrams visualization*. PhD thesis, Department of Computer Science and Engineering University of West Bohemia, 2012.

[Holy et al.(2012)Holy, Jezek, Snajberk,, Brada] HOLY, L. et al. Lowering Visual Clutter in Large Component Diagrams. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2012.

[HTML5()] HTML5. Differences from HTML4. [online, navštíveno 3.3. 2014], 2014. Dostupné z: http://www.w3.org/TR/html5-diff/#mathml-svg.

[Inverardi – Tivoli(2003)Inverardi, Tivoli] INVERARDI, P. – TIVOLI, M. Deadlock-free software architectures for COM/DCOM Applications. *The Journal of Systems and Software*. 2003, 65.

[JavaScript()] JavaScript. JavaScript. [online, navštíveno 19. 2. 2014], 2014. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[Jezek(2010a)] JEZEK, K. A Complex Meta-model for Extra-functional Properties Concerning Common Data Types Their Comparing and Binding. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2010a.

[Jezek(2010b)] JEZEK, K. *Component Compatibility in Terms of Extra-functional Properties*. PhD thesis, Department of Computer Science and Engineering University of West Bohemia, 2010b.

[Jezek – Brada(2012)Jezek, Brada] JEZEK, K. – BRADA, P. Formalisation of a Generic Extra-functional Properties Framework. Technical report,

Department of Computer Science and Engineering University of West Bohemia, 2012.

[Jezek – Brada(2011)Jezek, Brada] JEZEK, K. – BRADA, P. Correct Matching of Components with Extra-functional Properties – A Framework Applicable to a Variety of Component Models. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2011.

[Jezek et al.(2013a)Jezek, Brada„ Holy] JEZEK, K. – BRADA, P. – HOLY, L. Static Component Compatibility Visualisation for Various Component Models. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2013a.

[Jezek et al.(2013b)Jezek, Brada, Holy„ Snajberk] JEZEK, K. et al. A Visualization Tool for Reverse-engineering of Complex Component Applications. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2013b.

[Mackinlay et al.(1991)Mackinlay, Robertson„ Card] MACKINLAY, J. D. – ROBERTSON, G. G. – CARD, S. K. The Perspective Wall: Detail and Context Smoothly Integrated. Technical report, Xerox Palo Alto Research Center, 1991.

[Niekamp(2006)] NIEKAMP, R. Software Component Architecture. Technical report, Institute for Scientific Computing TU Braunschweig, 2006.

[OSGi(a)] OSGi. OSGi Core Release 5, 2012a.

[OSGi(b)] OSGi. OSGi Alliance | Technology / The OSGi Architecture. [online, navštíveno 10. 2. 2014], 2014b. Dostupné z: http://www.osgi.org/Technology/WhatIsOSGi.

[OSGi(c)] OSGi. OSGi Alliance | About / HomePage. [online, navštíveno 9. 2. 2014], 2014c. Dostupné z: http://www.osgi.org/About/HomePage.

[Pavlikova(2012)] PAVLIKOVA, J. Vizualizace rozsáhlých diagramů komponent a interakce s nimi – nepublikováno. Master's thesis, Department of Computer Science and Engineering University of West Bohemia, 2012.

[Sarkar – Brown(1993)Sarkar, Brown] SARKAR, M. – BROWN, M. H. Graphical Fisheye Views. Technical report, Department of Computer Science of Brown University, 1993.

[Snajberk – Brada(2012)Snajberk, Brada] SNAJBERK, J. – BRADA, P. ComAV – A Component Application Visualization Tool - Use of Reverse Engineering and Interactivity in Visualization for Component Software Comprehension. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2012.

[Snajberk – Brada(2011)Snajberk, Brada] SNAJBERK, J. – BRADA, P. ENT: A Generic Meta-Model for the Description of Component-Based Applications. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2011.

[Snajberk et al.(2012)Snajberk, Holy„ Brada] SNAJBERK, J. – HOLY, L. – BRADA, P. AIVA vs UML: Comparison of Component Application Visualizations in a Case-Study. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2012.

[Snajberk et al.(2013)Snajberk, Holy„ Brada] SNAJBERK, J. – HOLY, L. – BRADA, P. Visualization of Component-Based Applications Structure Using AIVA. Technical report, Department of Computer Science and Engineering University of West Bohemia, 2013.

[SOFA2()] SOFA2. SOFA 2 Documentation. [online, navštíveno 10. 2. 2014], 2011. Dostupné z: http://sofa.ow2.org/docs/index.html.

[Spring()] Spring. 1. Introduction to Spring Framework. [online, navštíveno 19. 2. 2014], 2013. Dostupné z: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html.

[Szyperski et al.(2002)Szyperski, Gruntz„ Murer] SZYPERSKI, C. – GRUNTZ, D. – MURER, S. *Component Software: Beyond Object-Oriented Programming*. Pearson Education, 2nd edition, 2002. ISBN 978-0321753021.

[UML()] UML. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. [online, navštíveno 9. 2. 2014], 2007. Dostupné z: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF.

[Vlcek(2011)] VLCEK, L. Bakalářská práce – Vyhodnocování mimofunkčních charakteristik na komponentách – nepublikováno., 2011.

A Seznam zdrojových souborů

Tato kapitola obsahuje výpis veškerých zdrojových souborů obou softwarových aplikací ve stavu po implementaci rozšíření funkčnosti, kterých se týkala tato práce.

A.1 CoCA-Ex

Java třídy

- **cz.zcu.kiv.offscreen.api** – balík obsahující rozhraní struktur zobrazovaného diagramu
  - **EdgeInterface.java** – rozhraní definující spojení ve struktuře diagramu
  - **GraphInterface.java** – rozhraní definující strukturu diagramu
  - **VertexInterface.java** – rozhraní definující vrchol ve struktuře diagramu
- **cz.zcu.kiv.offscreen.data.efpportal** – balík obsahující definice datových tříd využívaných na datech z aplikace EFPPortal
  - **CocaexData.java** – třída obsahující přijatá data se spojeními z aplikace EFPPortal
  - **CocaexDataEfp.java** – třída obsahující vlastní přijaté hodnoty EFPs, které lze přiřadit k vlastnostem
  - **CocaexDataFeature.java** – třída obsahující přijaté vlastnosti spojení
  - **CocaexWrapper.java** – obalovací třída, která obaluje přijatá data a příslušející mapování jmen vlastností a EFPs
- **cz.zcu.kiv.offscreen.efp.utils** – balík obsahující obslužné třídy pro manipulaci s EFP daty
  - **JsonTransformer.java** – třída obsahující metody pro dekomprimaci, dekódování a deserializaci přijatých dat

  - **WrapperDeserializer.java** – třída obsahující definici deserializeru

- **cz.zcu.kiv.offscreen.graph** – balík obsahující implementace struktur zobrazovaného diagramu
  - **EdgeEfp.java** – třída definující spojení ve struktuře diagramu s EFPs
  - **EdgeImpl.java** – třída definující spojení ve struktuře diagramu
  - **GraphExport.java** – třída definující strukturu určenou pro serializaci diagramu do formátu JSON
  - **GraphImpl.java** – třída definující strukturu diagramu
  - **VertexEfpl.java** – třída definující vrchol ve struktuře diagramu s EFPs
  - **VertexImpl.java** – třída definující vrchol ve struktuře diagramu
- **cz.zcu.kiv.offscreen.graph.creator** – balík obsahující třídy mající za úkol vygenerování či transformaci struktury diagramu
  - **EfpGraphTransfomer.java** – třída obsahující metody pro transformaci dat z přijatých ve tvaru EFPPortal na interní reprezentaci
  - **GraphGenerator.java** – třída obsahující metody pro generování diagramu dle parametrů
  - **GraphMaker.java** – třída obsahující metody pro transformaci dat ze tvaru využívaného nástrojem ComAV na interní reprezentaci
- **cz.zcu.kiv.offscreen.graph.efp** – balík obsahující definice datových tříd využívaných pro diagramy s EFPs
  - **EfpComparison.java** – třída obsahující vlastní přijatá vyhodnocená spojení s EFPs, které lze přiřadit k vlastnostem
  - **EfpFeature.java** – třída obsahující přijaté vlastnosti spojení
  - **EfpGraphicSettings.java** – třída obsahující data umožňující zavedení nastavení vzhledu diagramu přes externí proměnné
- **cz.zcu.kiv.offscreen.graph.loader** – balík obsahující třídy mající za úkol získávání dat k vizualizaci diagramů bez EFPs
  - **DemoDiagramLoader.java** – třída obsahující metody pro načtení dat určených pro zobrazení ukázkových diagramů

  - **GenericComponentLoader.java** – třída obsahující metody pro využití nástroje ComAV

- **cz.zcu.kiv.offscreen.loader.configuration** – balík obsahující třídu pro správu nastavení
  - **ConfigurationLoader.java** – třída obsahující metody pro správu nastavení aplikace

- **cz.zcu.kiv.offscreen.servlets** – balík obsahující definice hlavních servletů aplikace
  - **EfpImport.java** – servlet zajišťující přijetí dat v požadavku z aplikace EFFCC/EFPPortal
  - **LoadGraphData.java** – servlet zajišťující odeslání dat ve formátu JSON k zobrazení
  - **Login.java** – servlet zajišťující přihlášení uživatele do aplikace
  - **Logout.java** – servlet zajišťující odhlášení uživatele z aplikace
  - **Register.java** – servlet zajišťující registraci nového uživatele aplikace
  - **SettingGraph.java** – servlet zajišťující nastavení prostředí aplikace
  - **ShowGraph.java** – servlet zajišťující zobrazení diagramu

- **cz.zcu.kiv.offscreen.servlets.actions** – balík obsahující definice jednoúčelových servletů aplikace
  - **DeleteAllComponents.java** – servlet zajišťující smazání celého seznamu komponent k nahrání
  - **DeleteComponent.java** – servlet zajišťující smazání komponenty ze seznamu komponent k nahrání
  - **DeleteDiagram.java** – servlet zajišťující smazání uloženého diagramu
  - **LoadDiagram.java** – servlet zajišťující načtení diagramu
  - **SaveDiagram.java** – servlet zajišťující uložení diagramu
  - **UploadFiles.java** – servlet zajišťující nahrání souborů na server

- **cz.zcu.kiv.offscreen.session** – balík obsahující třídu pro správu sessions v aplikaci

  - **SessionManager.java** – třída obsahující metody pro manipulaci a generování sessions

- **cz.zcu.kiv.offscreen.storage** – balík obsahující třídu pro správu souborů
  - **FileManager.java** – třída obsahující metody pro nahrávání a manipulaci se soubory v úložišti
- **cz.zcu.kiv.offscreen.user** – balík obsahující třídy pro správu uživatelů aplikace a interakcí s databází
  - **DB.java** – třída obsahující metody pro přístup k databázi
  - **Diagram.java** – třída obsahující metody pro přístup k uloženému diagramu v rámci databáze
  - **User.java** – třída obsahující metody pro přístup k datům uživatelů aplikace v rámci databáze
  - **Util.java** – třída obsahující podpůrné metody pro práci s databází

JavaServer Pages soubory

- **index.jsp** – soubor sloužící pro zobrazení hlavní aplikační plochy
- **logged_user_menu.jsp** – soubor sloužící pro zobrazení nabídkového sloupce na úvodní stránce aplikace
- **logged_user.jsp** – soubor sloužící pro zobrazení registračního a přihlašovacího formuláře
- **uploadComponent.jsp** – soubor sloužící jako úvodní stránka aplikace

JavaScript soubory

- **diagram.js** – soubor obsahující funkcionalitu ukládání a načítání diagramu
- **dialog.js** – soubor obsahující definici a nastavení dialogu z pluginu jQuery UI Dialog
- **efps.js** – soubor obsahující funkcionalitu škálování zobrazených spojení v diagramu s EFPs

- **graphManager.js** – soubor obsahující funkcionalitu veškeré obsluhy zobrazení diagramu počínaje jeho generováním
- **gridMark.js** – soubor obsahující veškerou funkcionalitu obsluhy delegátů
- **group.js** – soubor obsahující definici skupin komponent v diagramu
- **groupManager.js** – soubor obsahující funkcionalitu skupin komponent v diagramu
- **hash.js** – soubor obsahující definici ADT hash tabulka
- **jquery.contextMenu.js** – soubor obsahující jQuery Context Menu plugin
- **jquery.jstree.js** – soubor obsahující jQuery jsTree plugin
- **jquery.qtip.js** – soubor obsahující jQuery qTip plugin ve verzi `2.2.0`
- **jquery-1.8.3.js** – soubor obsahující vlastní knihovnu jQuery ve verzi `1.8.3`
- **jquery-ui-1.10.3.custom.js** – soubor obsahující jQuery UI plugin ve verzi `1.10.3`
- **loader.js** – soubor obsahující nastavení a definici jQuery Loader pluginu
- **main.js** – soubor obsahující hlavní část interaktivní funkcionality aplikace na straně klienta
- **mark.js** – soubor obsahující definici značky zobrazení v SeCo oblasti
- **markSymbol.js** – soubor obsahující funkcionalitu značky zobrazení v SeCo oblasti
- **mootools-core-1.4.1.js** – soubor obsahující plugin MooTools ve verzi `1.4.1`
- **offScreenKiv.js** – soubor obsahující hlavní část oddělené funkcionality aplikace
- **spin.js** – soubor obsahující nastavení a definici jQuery Loader pluginu
- **tooltips.js** – soubor obsahující nastaveni a definici jQuery qTip pluginu

- **user.js** – soubor obsahující funkcionalitu odeslání registračního adresáře
- **util.js** – soubor obsahující nejrůznější jednoúčelovou funkcionalitu
- **zoom.js** – soubor obsahující funkcionalitu a nastaveni pro funkci přibližování a oddalování hlavní aplikační plochy

CSS styly

- **jquery-ui/smoothness** – adresář obsahující styly pro jQuery UI
  - **jquery-ui-1.10.3.custom.css** – soubor stylů pro jQuery UI
- **jstree/themes/classic** – adresář obsahující styly pro jQuery jsTree plugin
  - **style.css** – soubor stylů pro jQuery jsTree plugin
- **basic.css** – soubor základních stylů užitých napříč celou aplikací
- **dialog.css** – soubor dodatečných definic stylů užitých pluginem jQuery UI Dialog
- **jquery.contextMenu.css** – soubor stylů užitých pluginem jQuery Context Menu
- **jquery.qtip.css** – soubor stylů užitých pluginem jQuery qTip
- **tooltip.css** – soubor dodatečných definic stylů užitých pluginem jQuery qTip
- **tree.css** – soubor dodatečných definic stylů užitých pluginem jQuery jsTree

A.2 EFFCC/EFPPortal

Java třídy

- **cz.zcu.kiv.efps.efpportal.beans** – balík obsahující třídu pro reprezentaci nahrávaných souborů

  - **UploadBundleBean.java** – bean zajišťující reprezentaci nahrávaných souboru

- **cz.zcu.kiv.efps.efpportal.controllers** – balík obsahující hlavní kontroléry aplikace
  - **EFPAssignmentController.java** – kontrolér obsluhující akce pro EFP Assigment část aplikace
  - **EFPComparatorController.java** – hlavní kontrolér obsluhující akce pro EFP Comparator část aplikace
  - **HomeController.java** – kontrolér obsluhující úvodní stránku aplikace

- **cz.zcu.kiv.efps.efpportal.data.entity** – balík obsahující definice datových tříd
  - **BundleData.java** – třída reprezentující data nahraného souboru
  - **CocaexData.java** – třída obsahující data se spojeními určená pro aplikaci CoCA-Ex
  - **CocaexDataEfp.java** – třída obsahující vlastní hodnoty EFPs určené pro aplikaci CoCA-Ex
  - **CocaexDataFeature.java** – třída obsahující vlastnosti spojení určené pro aplikaci CoCA-Ex
  - **CocaexWrapper.java** – obalovací třída, která obaluje data a příslušející mapování jmen vlastností i EFPs a je určená pro aplikaci CoCA-Ex
  - **EFPData.java** – třída obsahující definice dat příslušejících k jednotlivým EFP
  - **FeatureData.java** – třída obsahující definice dat příslušejících k jednotlivým vlastnostem spojení

- **cz.zcu.kiv.efps.efpportal.service** – balík obsahující třídy pro přípravu a transformaci dat v aplikaci
  - **BundleLRService.java** – rozhraní definující metody pro získání dat pro lokální registr
  - **BundleLRServiceImpl.java** – třída implementující metody pro získání dat pro lokální registr
  - **CocaexDataCreator.java** – rozhraní definující metody pro transfomraci dat do formátu pro přenos do nástroje CoCA-Ex

  - **CocaexDataCreatorImpl.java** – třída implementující metody pro transfomraci dat do formátu pro přenos do nástroje CoCA-Ex
  - **DataSetCreato.java** – třída obsahující metody pro transfomraci dat pro vizualizaci
  - **SessionHandler.java** – třída obsahující metody pro správu session
- **cz.zcu.kiv.efps.efpportal.service.storage** – balík obsahující třídy pro správu serverového úložiště
  - **StorageConfiguration.java** – třída obsahující metody pro konfiguraci úložiště
  - **StorageManager.java** – rozhraní definující metody pro manipulaci se soubory v úložišti
  - **StorageManagerImpl.java** – třída implementující metody pro manipulaci se soubory v úložišti
- **cz.zcu.kiv.efps.efpportal.utils** – balík obsahující třídu pro výběr komponentového frameworku
  - **FrameworkType.java** – třída obsahující metody pro výběr a nastavení komponentového frameworku
- **cz.zcu.kiv.efps.efpportal.validators** – balík obsahující třídu pro testování nahraných souborů
  - **UploadedBundlesValidator.java** – třída obsahující metody pro kontrolu a testy nahraných souborů

JavaServer Pages soubory

- **choose_lr.jsp** – soubor sloužící pro zobrazení stránky s výběrem lokálního registru
- **index.jsp** – soubor sloužící jako úvodní stránka aplikace
- **results.jsp** – soubor sloužící pro zobrazení výsledkové stránky s výsledkem porovnání
- **upload_bundles.jsp** – soubor sloužící pro umožnění nahrání souborů na server

JavaScript soubory

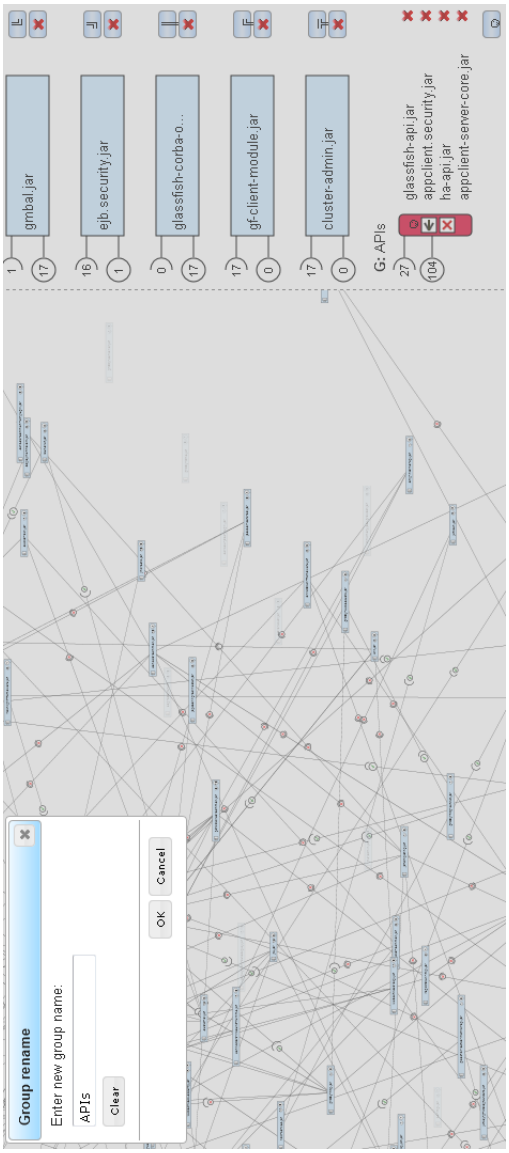
- **efpcomparator**
  - **ChooseLRActions.js** – soubor obsahující funkcionalitu výběru lokálního registru porovnání
  - **ChooseLREvents.js** – soubor obsahující definice přiřazení funkcionality lokálního registru porovnání
  - **ResultActions.js** – soubor obsahující definice zobrazení vizualizační struktury porovnání
  - **ResultEvents.js** – soubor obsahující definice přiřazení funkcionality struktury porovnání
  - **UploadBundleActions.js** – soubor obsahující funkcionalitu pro odesílání souborů na server
  - **UploadBundleActions.js** – soubor obsahující funkcionalitu pro odesílání souborů na server
  - **UploadBundleEvents.js** – soubor obsahující definice přiřazení funkcionality odesílání souborů na server
  - **Wizard.js** – soubor obsahující funkcionalitu jednotlivých kroků průvodce mezi stránkami aplikace
  - **WizardManager.js** – soubor obsahující funkcionalitu přechodu mezi kroky průvodce mezi stránkami aplikace
- **efpcomparator/tree**
  - **jquery.treeview.js** – soubor obsahující plugin jQuery jsTree
- **CookieHandler.js** – soubor obsahující funkcionalitu obsluhy cookies
- **jquery.cookie.js** – soubor obsahující jQuery Cookie plugin
- **jquery.form.js** – soubor obsahující jQuery Form plugin
- **jquery.qtip.min.js** – soubor obsahující jQuery qTip plugin
- **jquery-1.7.1.js** – soubor obsahující vlastní knihovnu jQuery ve verzi `1.7.1`
- **jquery-ui-1.10.3.custom.js** – soubor obsahující jQuery UI plugin ve verzi `1.10.3`

- **loader.js** – soubor obsahující nastavení a definici jQuery Loader pluginu
- **spin.js** – soubor obsahující nastavení a definici jQuery Loader pluginu

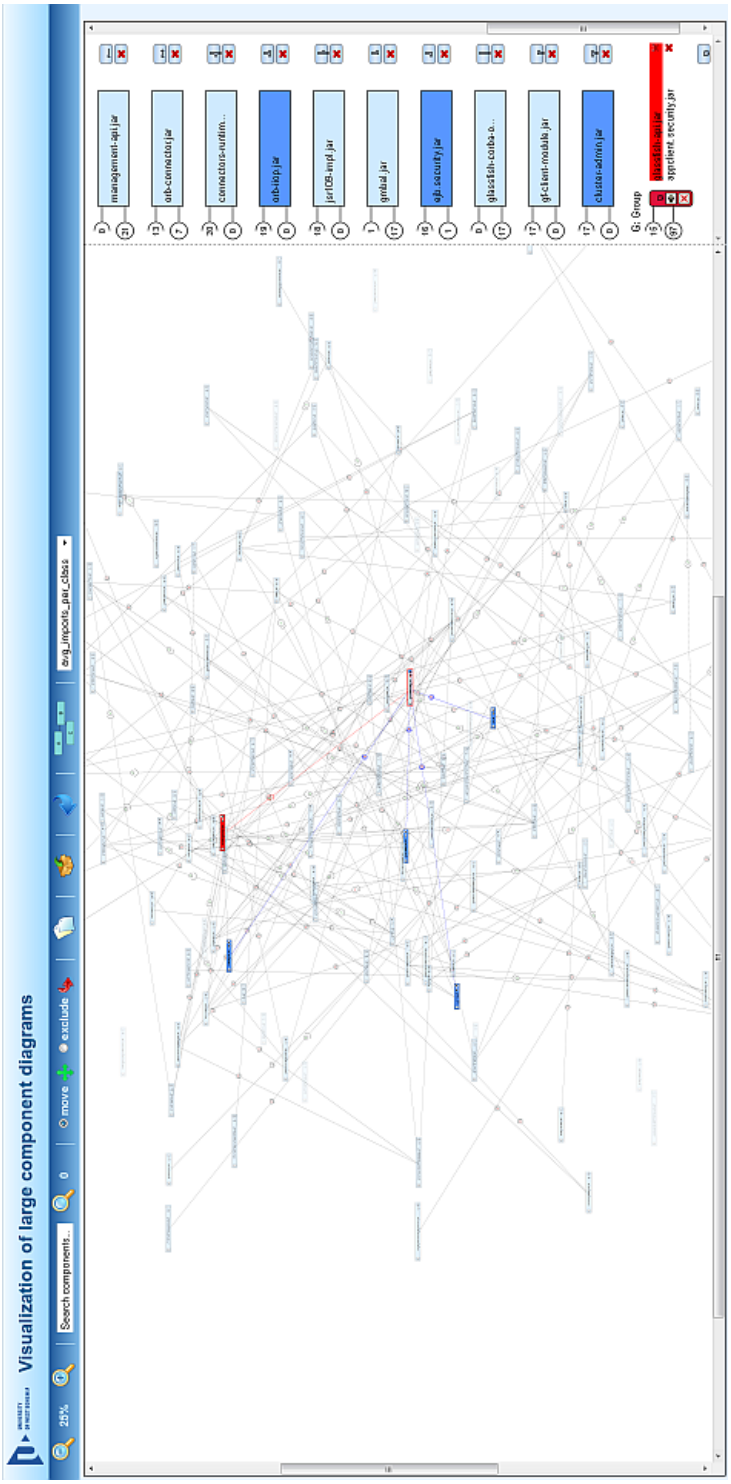
CSS styly

- **efp_comparator**
  - **jquery.treeview.css** – soubor obsahující styly pluginu jQuery jsTree
  - **main.less** – soubor obsahující styly pro vyhodnocovací stránku aplikace
- **common.less** – souborstylů užitých napříč celou aplikací ve všech oknech průvodce
- **dialog.css** – soubor dodatečných definic stylů užitých pluginem jQuery UI Dialog
- **elements.less** – soubor stylů užitých pro definici oblých rohů a gradientů v aplikaci
- **jquery.qtip.min.css** – soubor stylů užitých pluginem jQuery qTip
- **main.css** – soubor základních stylů užitých napříč celou aplikací
- **main.less** – soubor základních stylů užitých napříč celou aplikací

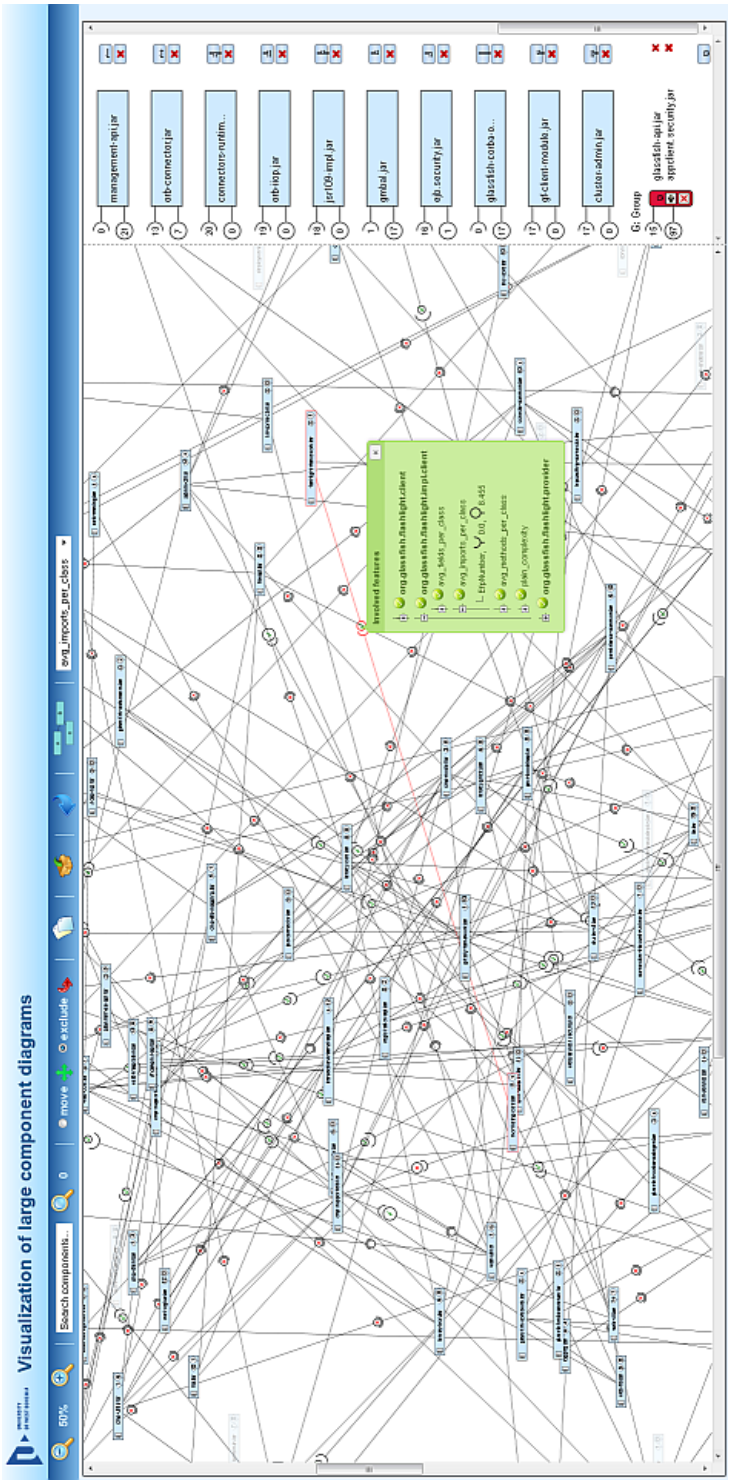
B Ukázky rozhraní aplikace



Obrázek B.1: Pohled na aplikaci při přejmenování skupiny.



Obrázek B.2: Celkový pohled na rozhraní aplikace.



Obrázek B.3: Celkový pohled na rozhraní aplikace – detail.