VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KLIENTSKÁ APLIKACE PRO SLUŽBU VYHLEDÁVÁNÍ A ŘEŠENÍ SHODY ZDROJOVÝCH KÓDŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR KNAPEK
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KLIENTSKÁ APLIKACE PRO SLUŽBU VYHLEDÁVÁNÍ A ŘEŠENÍ SHODY ZDROJOVÝCH KÓDŮ

CLIENT FOR THE BLACKDUCK PROTEX SERVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR KNAPEK
AUTHOR

VEDOUCÍ PRÁCE Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Plagiátorství a shody ve zdrojových kódech jsou v dnešní době narůstajícím problémem. Cílem této práce je zhodnocení služby Protex™ od společnosti Black Duck® Software, Inc. a jejího uživatelského rozhraní. Na základě tohoto zhodnocení pak vytvořit nového, alternativního klienta usnadňujícího práci a nabízejícího automatizaci. Při vývoji klienta byl použit programovací jazyk Java™ a jeho technologie pro vytváření grafického rozhraní knihovny Swing a vícevláknový přístup.

Abstract

Plagiarism and matching source codes are an increasing problem. The objective of this thesis is to evaluate Black Duck® Software, Inc. service Protex™ and it's user interface. The next objective is to create a new, alternative client based on this evaluation, which will ease work process and offer automatization methods. Swing graphical library and multithreaded approach of Java™ programming language have been used in the implementation of a new client.

Klíčová slova

licence, shoda zdrojových souborů, hledání shod, plagiátorství, automatizace, grafické rozhraní, vícevláknový přístup, Java, Protex, Black Duck, serializace

Keywords

licence, source code match, searching for matches, plagiatorism, automatization, graphical user interface, multithreaded approach, Java, Protex, Black Duck, serialization

Citace

Petr Knapek: Klientská aplikace pro službu vyhledávání a řešení shody zdrojových kódů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Klientská aplikace pro službu vyhledávání a řešení shody zdrojových kódů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. a pana Ing. Vladimíra Vaňka. Uvedl jsem všechny zdroje, ze kterých jsem čerpal.

. . . . . .
Petr Knapek
19. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu své práce Ing. Radku Kočímu, Ph.D. za vstřícnot a ochotu, stejně jako za cenné odborné rady při konzultacích.

Rovněž bych rád poděkoval Ing. Vladimíru Vaňkovi, jakožto zadavateli práce, za náměty a připomínky k vývoji aplikace, a také za sdílení jeho znalostí služby Protex a zkušeností s ní.

Obsah

1 Úvod **4**

2 Shody ve zdrojovém kódu **6**
2.1 Shoda zdrojových kódů 6
2.2 Příčiny vzniku shod 6
2.2.1 Dodržení standardů 6
2.2.2 Generování kódu 6
2.2.3 Hledání řešení 7
2.2.4 Průmyslová krádež 7
2.2.5 Nalezení shodného řešení 7
2.3 Důsledky 7

3 Black Duck® Protex™ **8**
3.1 Projekty a komponenty 8
3.2 Analýza 9
3.3 Identifikace 9
3.4 Databáze komponent 9
3.5 Seznam komponent se shodami 10
3.6 Logická struktura komponent 10
3.7 Uživatelské rozhraní 12

4 Analýza současného řešení **13**
4.1 Typický pracovní postup 13
4.2 Implementace 13
4.2.1 Serverová část 13
4.2.2 Rozhraní v prohlížeči 14
4.3 Ovládání 14
4.4 Zhodnocení 14
4.4.1 Pomalé rozhraní 14
4.4.2 Velké množství shod 15
4.4.3 Absence automatizace/hromadných řešení 15

5 Požadavky na nový systém **16**
5.1 Hlavní idea 16
5.2 Odezva 16
5.3 Ovládání 16
5.4 Automatizace 17
5.5 Uživatelské rozhraní 17

5.5.1 Grafické . . . 17
5.5.2 Textové . . . 17

6 Návrh nového systému **18**
6.1 Struktura aplikace . . . 18
6.2 Protex SDK . . . 20
6.3 Jazyk implementace . . . 20
6.4 Soubor API tříd . . . 20
6.4.1 Důležité třídy vracené API metodami . . . 20
6.4.2 Třída `ProjectApi` . . . 21
6.4.3 Třída `CodeTreeApi` . . . 21
6.4.4 Třída `DiscoveryApi` . . . 21
6.4.5 Třída `FileComparisonApi` . . . 21
6.4.6 Třída `ComponentApi` . . . 22
6.4.7 Třída `IdentificationApi` . . . 22
6.4.8 Třída `BomApi` . . . 22
6.5 Stahování a ukládání dat . . . 22
6.5.1 Serializace . . . 22
6.5.2 Serverová volání – přihlašování a stahování dat . . . 23
6.5.3 Serverová volání – nahrávání identifikací na server . . . 24
6.5.4 Doba trvání volání . . . 25
6.5.5 Nároky a omezení . . . 25
6.6 Vícevláknový přístup . . . 26
6.6.1 Způsob dělby práce, velikost vláken . . . 26
6.6.2 Počet souběžných vláken . . . 26
6.7 Filtry . . . 27
6.8 Tvorba grafického rozhraní a NetBeans . . . 27
6.8.1 Souborový strom . . . 27
6.9 Chybějící funkcionalita, chyby . . . 28
6.9.1 Zdrojové kódy třetí strany . . . 28
6.9.2 Absence informací o úsecích shody pro některé soubory . . . 28
6.10 Výpisy a hlášení . . . 29
6.10.1 Logovací soubor . . . 29
6.11 Testování . . . 29
6.12 Správa verzí . . . 30

7 Popis nového rozhraní, jeho ovládání a přínosu **31**
7.1 Režimy spuštění . . . 31
7.2 Grafické rozhraní . . . 31
7.2.1 Hlavní okno . . . 32
7.3 Příkazová řádka . . . 34

8 Závěr **35**

A Obsah CD **37**

B Manuál **38**
B.1 Prerekvizity . . . 38
B.2 Instalace a spuštění . . . 38

Seznam obrázků

3.1 Zjednodušená struktura dvou komponent spojených shodou. 11
3.2 Ukázka starého rozhraní. Vlevo se nachází složky a soubory projektu, uprostřed shody, dole je pak část pro identifikaci. Obrázek je ze zdroje [4]. . . . 12

6.1 Diagram důležitých částí aplikace, jejich propojení s okolím a mezi sebou. . 19

7.1 Hlavní okno grafického rozhraní aplikace rozdělené na významné části. . . . 32
7.2 Menu položky pro nahrávání a mazání lokálně vytvořených identifikací. . . 33

???

Kapitola 1

Úvod

V době snadno dostupného internetu a vytváření velkých softwarových projektů mnoha nadnárodními korporacemi se stává problém shod ve zdrojových kódech čím dál tím častější. Je v zájmu společností vyhnout se případnému nařčení z plagiátorství, které by mohlo poškodit jejich dobré jméno a vést k finančním sankcím, pokud vydávají cizí technologie či postupy za vlastní. Z tohoto důvodu jsou na trhu k dispozici služby pro kontrolu zdrojových souborů softwarových projektů, které hledají potencionální hrozby.

Při své odborné praxi ve společnosti Freescale Semiconductor, Inc. jsem se setkal se službou Protex™ (dále jen jako Protex nebo „služba“) od společnosti Black Duck Software, Inc.® (dále jen jako Black Duck) využívanou k hledání potencionálních shod vedoucími pracovníky vývojových týmů (dále jako *uživatelé*) – cílovými uživateli jsou tedy odborníci. Ti jsou s prací se službou, která je po nich vyžadována vedením společnosti, nespokojeni, a to především kvůli ovládání aplikace. Prozkoumávání možností, jak zjednodušit a zefektivnit pracovní postup a ušetřit čas, dalo důvod ke vzniku této práce.

Cílem je využít souboru knihoven služby Protex k vytvoření nového uživatelského rozhraní schopného i automatizačních úloh. Nový systém je určen pouze pro uživatele původního rozhraní, je tedy určen jako nástroj pro programátory.

K výběru tohoto tématu mě vedla možnost prohloubení si vědomostí o programovacím jazyce Java a jeho moderních technologiích a přístupech, stejně jako získání znalostí o licencích a řízení vývoje větších softwarových produktů v korporacích. Lákala mne i spolupráce s množstvím lidí z oboru žijících v různých částech světa.

V kapitole 2 je čtenář uveden do problematiky shod ve zdrojových souborech a je mu vysvětleno, jakými způsoby se shody mohou vyskytnout a proč je třeba je aktivně vyhledávat a řešit.

Výstupem této práce je i aplikace, která funguge jako klient ke službě Protex. V kapitole 3 jsou detailněji rozebrány složky služby důležité pro tuto práci a procesy, ke kterým v nich dochází.

Kapitola 4 se věnuje zhodnocení služby Protex a způsobu práce s ní. Obsahuje informace o prvcích služby, které mohou způsobovat potíže, a zhodnocení problémů, ke kterým dochází.

Před začátkem vývoje nových aplikací obvykle vzniká soubor specifikací a požadavků, které má vyvíjený systém splňovat. Ty jsou neformálně popsány v kapitole 5 tak, jak byly předány vedením a uživali, kteří mají s novým systémem pracovat.

Implementaci výsledné aplikace je věnována kapitola 6, kde je popsána její struktura a detailněji vysvětleny jednotlivé komponenty nového systému, vztahy a komunikace mezi nimi. Také jsou zde rozebrány použité technologie, knihovny a postupy.

V kapitole 7 se nalézá popis uživatelského rozhraní nového systému s výčtem jeho přínosů oproti původnímu. Také je zde popsáno základní ovládání aplikace.

V závěrečné kapitole je obsaženo shrnutí práce a nově vytvořené aplikace, zhodnocení její úspěšnosti a přínosu. Také jsou zde uvedeny informace o dalším směřování projektu a vývoje.

Kapitola 2

Shody ve zdrojovém kódu

Tato kapitola obsahuje uvedení do problematiky vzniku a vyhledávání shod ve zdrojových kódech, která se stala základem pro celou práci. Jsou zde popsány příčiny, kvůli kterým může k takovéto shodě dojít, stejně jako důvody, proč je v zájmu větších společností jim předejít, či je svým zákazníkům odůvodnit.

2.1 Shoda zdrojových kódů

Definice 1. Shoda ve zdrojovém kódu (dále také jen *shoda*) je propojení sémanticky či syntakticky totožných úseků právě dvou zdrojových souborů, z nichž každý patří k jinému softwarovému produktu.

2.2 Příčiny vzniku shod

K takovémuto jevu může dojít v přímém důsledky celé řady vlivů, z nichž ovšem ne všechny mohou být nežádoucí.

2.2.1 Dodržení standardů

V oboru informatiky vzniklo za dobu jeho existence mnoho standardů. Tyto vznikaly nejen kvůli sjednocení a vzájemné kompatibilitě produktů, ale i jako norma určité kvality softwaru.

Pokud je dodržen standard či norma (také říkáme, že nějaký úsek kódu v programovacím jazyce je „implementací standardu“), lze garantovat jisté vlastnosti a zaručit se za korektní chování v krizových situacích. A právě tohle může být zákazníkem požadováno.

Mezi nejznámější standardy řadíme například POSIX [13].

Nevýhodou je, samozřejmě, to, že implementací funkcionality stejným způsobem, jako ostatní vývojáři využívající stejný standard, vznikne programový kód, který je velice podobný, či dokonce stejný.

2.2.2 Generování kódu

U velkých projektů firem zaměřujících se na vývoj hardware či software přímo pro hardware (například ovladačů periferních zařízení) mnoha variant nebývá výjimkou generování úseků programového kódu nebo jistých konfiguračních souborů – jinými slovy program tvoří a ukládá další program podle předem nadefinovaných pravidel.

Pokud se jedná o generování kódu pro hardware, který se v mnoha věcech neliší (má například podobnou instrukční sadu), a nebo dokonce pro platformu, která je k dispozici více společnostem, dojde ke shodám v kódu ovladačů periferních zařízení a jejich nastavení.

2.2.3 Hledání řešení

V době mohutného rozšíření internetu nebývá výjimkou situace, kdy vývojář řeší problém, o kterém si je jist, že ho už řešil (a nejspíš až ke zdárnému konci) někdo před ním – tedy začne hledat odpovědi na webu. Uživatel si zjednodušší práci, ovšem ztíží ji obchodnímu a právnímu oddělení.

Pokud najde řešení, které potřebuje, a použije ho, může dojít k několika důsledkům:

- Nalezený úsek kódu je pouze něčím nechráněným nápadem[1] nebo se nachází v oficiální příručce programovacího jazyka či cílového hardware.
- Nalezený úsek kódu je chráněn nějakou licencí, ale je volně vystaven na internetu.
- Nalezený úsek kódu se nachází v projektu chráněném licencí od konkurenční nebo jiné firmy a zaměstnanec jej okopíruje aniž by to věděl.

Při kterékoliv z těchto eventualit je pravděpodobné, že bude nalezena shoda ve zdrojovém kódu jiné společnosti.

2.2.4 Průmyslová krádež

K této možnosti by nemělo docházet, ovšem vyloučit ji nelze. V tomto případě by nalezení shody ve zdrojovém kódu znamenalo její odhalení.

2.2.5 Nalezení shodného řešení

Tato možnost se může zdát nepravděpodobnou, ovšem například při vývoji malých operačních systémů (například MQX™, viz. [1]) či ovladačů pro hardware je možné nalézt stejné řešení jako konkurence, ač zcela nezávisle na ní. Důvodem bývá snaha o minimalismus.

2.3 Důsledky

Nalezení shody ve zdrojovém kódu bývá považováno za problém. Ve zvláštních případech však lze této skutečnosti využít i ve prospěch projektu. Tímto prospěchem je myšleno odkázání se na implementaci nějakého standardu (viz. kapitola 2.2.1), který může být pro zákazníka zajímavý, či jím přímo vyžadován.

Avšak nevýhody plynoucí z možného výskytu shody kvůli jedné z příčin, která je již méně příznivá, mohou být natolik zásadní, že je potřeba shody aktivně vyhledávat a zhodnocovat, aby nedošlo k problémům s licencí - nelze jako vlastní prodávat projekty s open source či GPL licencovaným zdrojovým kódem. Vzniká tedy snaha o odhalení možné „kontaminace projektu společnosti open source kódem“. Nalezení cizího kódu v komerčním software může znamenat špatnou publicitu a finanční postihy – například žaloba společnosti FSF na společnost Cisco[2].

[1] Běžné bývá používání webových stránek jako http://stackoverflow.com/
[2] http://www.internetnews.com/storage/article.php/3790721/Cisco+Hit+With+Open+Source+Lawsuit.htm

Kapitola 3

Black Duck® Protex™

Služba Protex [3] společnosti Black Duck je určená pro snadné vyhledávání, analýzu a identifikaci shod zdrojových kódů. Tato služba je určená spíše pro větší společnosti a k jejímu používání je potřeba mít vlastní server pro ukládání kopie centrální databáze.

Protex má svá omezení – například porovnává spíše jen syntaxi a kód neanalyzuje (tedy nedokáže nalézt sémanticky shodnou funkcionalitu implementovanou v odlišeném programovacím jazyce) – a není zdarma. Jeho výhodou je velikost databáze projektů, se kterými může uživatel porovnat ty své. Společnost Black Duck pracuje velmi intenzivně na skenování projektů z univerzit či webových zdrojů – časté aktualizace.

V této kapitole budou rozebrány aktualizace, jednotlivé komponenty a procesy, ze kterých se služba skládá.

3.1 Projekty a komponenty

Definice 2. Komponenta (dále jen *komponenta* nebo *projekt*) je uživatelem nebo správcem databáze služby vytvořená reprezentce projektu nebo produktu, či jejich vydané verze, se kterou jsou svázány zdrojové soubory a informace o jejich shodách s jinými komponentami.

Komponenta je základní stavební jednotkou celé služby a zároveň je na nejvyšší úrovni abstrakce. Pokud mluvíme o shodách, tak shody se vytváří mezi soubory jen když tomu zadá vztah komponent.

Základními druhy komponent jsou:

- Projekt – vytvořen a spravován uživateli. Reprezentuje projekt, pro který chceme nalézt shody. Existuje pouze na serveru společnosti, které jej vytvořila.
- Lokální komponenta – vytvořena uživatelem. Existuje pouze jako podčlen v rámci uživatelského projektu. Slouží převážně k zpřehlednění práce.
- Custom komponenta – uživatelem vytvořený projekt, který je třeba uchovat delší dobu a hodí se mít možnost porovnávat k němu jiné uživatelské projekty. Je přístupná jen ze serveru společnosti, která ho vlastní. Po schválení je možnost distribuovat ji i dále.
- Standardní komponenta – vyvořeny v rámci celé služby, přístupné všem.

3.2 Analýza

Definice 3. Analýza (dále jako *analýza*) komponenty je uživatelem spouštěný proces vyhledávání shod mezi ní a všemi ostatními komponentami dostupnými v databázi.

Analýzu je třeba opětovně spustit nad projektem kvůli rozšiřování knowledge base, kdy se mohou objevit nové shody.

3.3 Identifikace

Definice 4. Identifikace je způsob řešení shod zdrojových kódů. Identifikací označíme úsek souboru se shodou jako originální k uživatelskému projektu či ke komponentě, na kterou se shoda odkazuje.

Práce s identifikacemi – jejich vytváření nebo mazání – je základem a smyslem celé služby. Identifikace s sebou nese informaci o tom, jak je či není kód jiné komponenty využíván – úroveň nebo způsob užití („Usage Level“).

Pokud identifikujeme úsek souboru, všechny ostatní shody týkajícího se stejného úseku či jeho podmnožiny jsou vyřazeny z aktivních, jelikož již byl výsledek označen.

Identifikaci lze provést na několika úrovních (od nejvyšší úrovně abstrakce):

- Rapid ID – služba vezme v potaz nastavení této funkce již při analýze projektu, tedy v době, kdy shody teprve vznikají. Tímto je umožněno automaticky identifikovat některé shody podle toho, k jaké komponentě patří, například podle porovnání licencí. V praxi se tento způsob používá, ale neumožní identifikovat velké množství shod.

- Po komponentách – uživatel může označit všechny shody jedné komponenty jednotným způsobem hned po otevření projektu. Tento způsob je pravděpodobně nejčistší vzhledem ke službě, avšak je zde vysoké riziko identifikace nesprávné komponenty pro úsek některého ze souborů (zdrojový kód může patřit k jiné komponentě, avšak příslušná shoda bude označena za neaktivní), a proto se v praxi příliš nevyužívá.

- Deklarace souborů nebo složek – tímto způsobem uživatel nadeaklaruje, že případně i celý podstrom projektu může být originální k projektu či celý opsaný z některé koponenty. Avšak tato možnost znamená ignoraci všech v budoucnu potencionálně vzniklých shod, které však mohou být relevantní a způsobovat potíže.

- Řešení shod po jedné – nejpoužívanější a nejspolehlivější, nicméně časově nejnáročnější způsob. Často vyžadován vedením společností pro většinu shod.

3.4 Databáze komponent

Služba je založena na porovnávání komponent a projektů v databázi, která je uložena na všech lokálních serverech zákazníků. V dokumentacích a příručkách služby lze nalézt také pod označením „*Knowledge base*“. Nad touto databází probíhají aktualizace, obvykle v intervalech jednoho měsíce. Ty pochází z centrálního serveru. Každý zákazník služby má k dispozici svou vlastní databázi na svém vlastním serveru, architektura je tedy do jisté míry i decentralizovaná.

Hlavním smyslem existence tohoto prvku je minimalizace počtu přenosů dat po síti při provádění analýzy a identifikace, což výrazně přispívá k rychlosti procesu. Při každé

aktualizaci knowledge base je však většinou třeba oba tyto procesy zopakovat, jelikož se mohou objevit nové shody (změny ve stávajících komponentách a přidání nových).

V případě vlastního projektu lze v době jeho vytvoření určit, zda má být přidán do databáze. Tímto umožňují společnosti i ostatním zákazníkům služby kontrolu svých vlastních projektů vůči jejich produktům.

Hlavní složkou knowledge base jsou však komponenty přidané samotným Black Duckem, které jsou reprezentací open source projektů, známých standardů a projektů pod GPL licencí – viz. [9].

3.5 Seznam komponent se shodami

Při práci s projektem ve službě lze zobrazit seznam všech komponent, se kterými byla nalezena shoda. Nese označení „*Bill of materials*“.

Tento seznam umožňuje uživateli jistou míru abstrakce nad projektem, jelikož se nemusí zabývat pouze atomárními jednotkami jako jsou jednotlivé shody a soubory, nýbrž pracovat s celou komponentou najednou (popsáno v kapitole 3.3). Tento způsob je však nespolehlivý, protože může dojít k přehlédnutí důležitého nálezu.

3.6 Logická struktura komponent

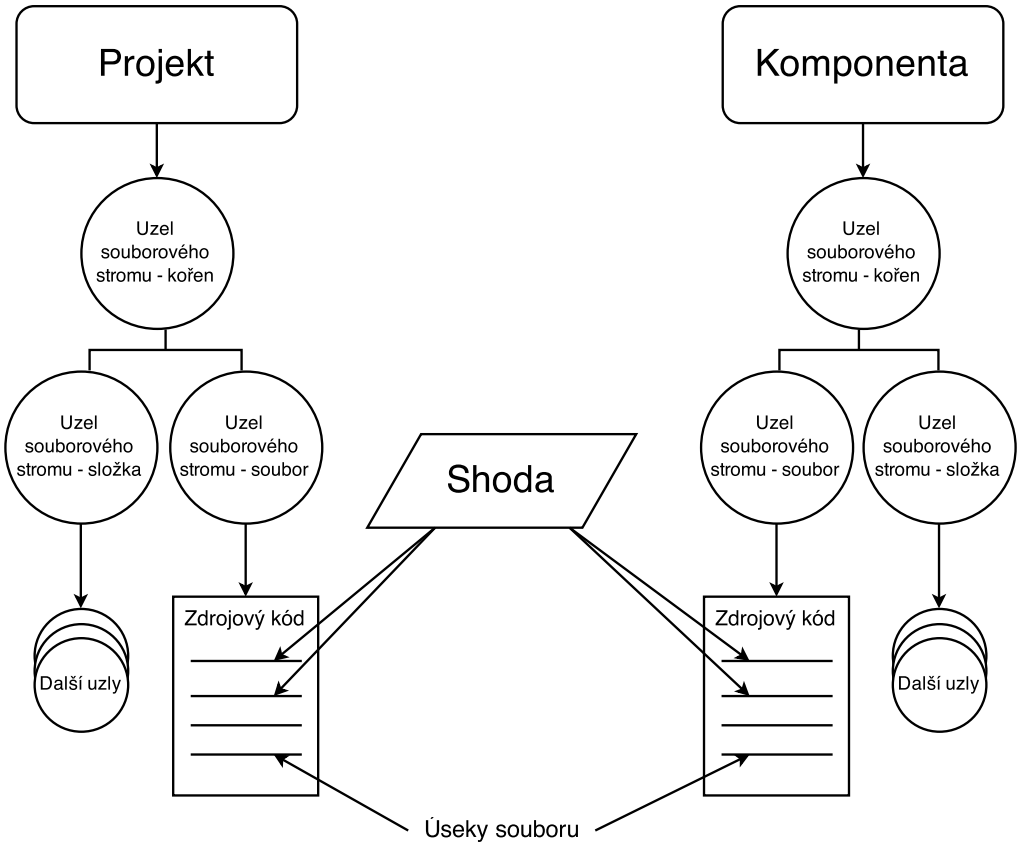
Obrázek 3.1 popisuje složení uživatelských projektů a komponent v databázi, jejich propojení shodami.

Projekt a komponenta jsou základní jednotky celé služby. Mají jméno, identifikační jméno, číslo verze a několik dalších méně podstatných vlastností. Jsou vydávané pod nějakou licencí. V zásadě jediným rozdílem mezi komponentou a projektem je v tomto ohledu přístup k nim – projekty jsou spravovány uživateli a mohou se měnit.

Komponenta je svázána se stromem adresářů a souborů patřícím jí, které jsou reprezentovány pomocí uzlů a unikátním atributem každého je cesta. Ta je vždy jen relativní ke komponentě, ke které patří.

V případě uzlů reprezentujících zdrojové soubory (projekt může obsahovat i soubory dokumentací nebo například obrázky) je k dispozici i možnost na jejich stáhnutí. To však lze pouze u souborů uživatelských projektů, ne komponent z databáze (nelze zpřístupnit majetek někoho jiného „jen tak“.

Shoda ve zdrojových kódech pak ukazuje na dva soubory různých komponent, které mají sémanticky shodné nějaké své úseky. Úsek souboru (také označován jako „Snippet“) není sám o sobě žádným objektem. Atomickou jednotkou zdrojového souboru je řádek a úsek je vlastně jen dvojice ukazatelů uložených ve shodě – ukazatel na první a poslední řádek úseku. Shoda se může týkat i více úseků jednoho souboru, které mohou být nespojité. Výsledkem je dvojice seznamů úseků – jeden pro soubor projektu a druhý pro komponentu, se kterou se shoduje. Oba tyto seznamy mají shodný počet vzájemně si odpovídajících položek.



Obrázek 3.1: Zjednodušená struktura dvou komponent spojených shodou.

3.7 Uživatelské rozhraní

Pro přístup k funkcím služby je dostupné webové rozhraní. Lze si v něm sice stáhnout soubor knihoven implementovaných v programovacím jazyce Java, které jsou určeny a vybaveny pro veškerou komunikaci (*Protex SDK*, jemuž se dále budu věnovat v kapitole o implementaci – kapitola 6.2), ovšem nejsou k dispozici žádné prostředky, které by s jejich pomocí mohly webové rozhraní nahradit.

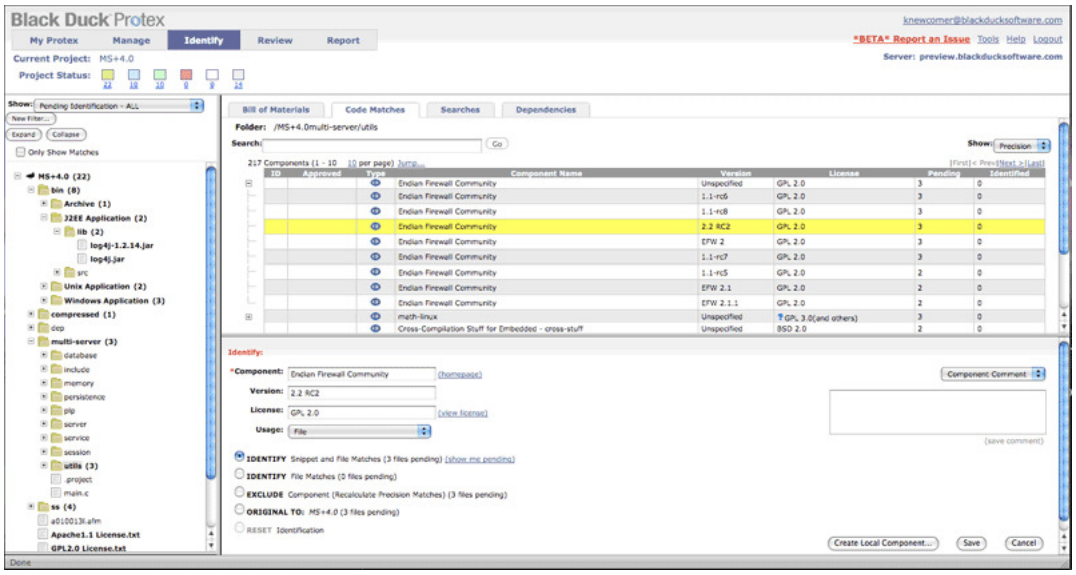
Webové rozhraní se skládá ze čtyř hlavních obrazovek:

1. Přihlašovací obrazovka – určena pouze pro přihlášení uživatele.

2. Hlavní obrazovka pro správu a analýzu projektů – obsahuje seznam projektů dostupných uživateli a při výběru jednoho z nich i možnosti pro správu jako spuštění analýzy nebo úpravu jeho vlastností. Také jsou zde k dispozici grafy a tabulky pro znázornění různých statistických dat o projektu, jako například podíl souborů s nalezenou shodou na celkovém počtu apod.

3. Původní hlavní obrazovka – méně detailní a méně přehledná forma předešlé obrazovky. Nyní již spíše přežitek starších verzí služby.

4. Obrazovka identifikace (viz. obrázek 3.2)– zde se nalézá veškerá funkcionalita spojená s identifikací. Lze si prohlížet souborový strom projektu, jednotlivé soubory, jejich obsah a shody, které obsahuje. Při zvolení shody je možné ji blíže prozkoumat a identifikovat.



Obrázek 3.2: Ukázka starého rozhraní. Vlevo se nachází složky a soubory projektu, uprostřed shody, dole je pak část pro identifikaci. Obrázek je ze zdroje [4].

Kapitola 4

Analýza současného řešení

V této kapitole bude rozebrán současný proces identifikace dodržován pracovníky zodpovědnými za projekty analyzované službou Protex, stejně jako rozhraní jimi používané.

4.1 Typický pracovní postup

Využít službu Protex je třeba v době, kdy se blíží datum vydání nové verze nějakého produktu. Příslušný pracovník jej tehdy musí ověřit přibližně takovýmto způsobem:

1. Umístění zdrojových souborů projektu na server služby.
2. Vytvoření projektu (neboli komponenty) v systému služby.
3. Spuštění analýzy.
4. Manuální prozkoumání všech shod v souborech projektu.
5. Identifikace všech shod.

Je nutné celý proces provést tak, aby se vyřešily veškeré shody čekající na identifikaci. A to v době mezi dokončením a vydáním, obvykle tedy není příliš mnoho času.

4.2 Implementace

Práce na identifikaci je ovlivněna implementací částí systému služby, konkrétně se jedná o server, kde je služba umístěna, a o klienta reprezentovaného webovým rozhraním v prohlížeči (popsáno v kapitole 3.7).

4.2.1 Serverová část

Server se stará o veškerou funkcionalitu celé služby kromě zobrazování dat uživateli a ovládání akcí uživatelů. Tímto je myšleno uchovávání a správa dat projektů a databázi komponent, analýza a identifikace. Ačkoliv aktualizace služby probíhají na úrovni mnoha serverů, pro celou společnost je k dispozici pouze jeden server na jednom místě.

Protex pro svou funkčnost vyžaduje výkonnou hardwarovou sestavu a dostatečně rychlé připojení k síti, avšak potencionálně může dojít k jeho přetížení při souběžném využívání více uživateli najednou. Pravděpodobnost tohoto zahlcení vzrůstá, pokud se v kratším období chystá vypuštění nových verzí více produktových řad.

Při práci z velké fyzické vzdálenosti od serveru může dojít k opoždění jeho odpovědí na požadavky o zobrazení dat vyžádaných uživatelem až v řádech stovek milisekund na každý požadavek či příkaz.

4.2.2 Rozhraní v prohlížeči

Hlavní ovládací rozhraní je implementováno v programovacím jazyce JavaScript. Jelikož obstarává veškerou komunikaci se serverem a zobrazení dat, potencionálním problémem je „zamrznutí“ do doby, než server odpoví a data z této odpovědi jsou správně roztříděna a zobrazena. Tohle je samozřejmě ovlivnitelné množstvím dat, které je třeba zobrazit.

Dalším problémem je paměťová náročnost při zobrazení přílišného objemu informací. Jistým faktorem je zde i odlišnost webových prohlížečů, z nichž některé se s JavaScriptem obecně vypořádávají hůře.

4.3 Ovládání

Ovládání webového rozhraní uživateli je realizovatelné pouze v internetových prohlížečích, k navigaci je využita výhradně počítačová myš. Nejsou podporovány žádné klávesové zkratky.

Soubory či složky, jejichž shody chceme zobrazit, nelze vybírat ve větším množství najednou, vždy maximálně jeden element. To samé platí i pro vybírání shod. Existuje jediná výjimka – pokud je více shod svázáno s komponentami jednoho produktu, které jsou označeny stejnou licencí a liší se pouze verzí produktu, jsou tyto shody zobrazeny jako jedna shoda s možností rozkliknutí pro zobrazení všech. Jako jednu lze tyto shody i identifikovat (ač je to jen zamaskovaná identifikace jedné shody a prohlášení ostatních za nadále irelevantní kvůli tomu, že jsou ve stejném úseku souboru – popsáno v kapitole 4).

Samotné tělo webové stránky je složeno z několika rámců, které mají uživatelem uzpůsobitelnou velikost pro přizpůsobení si šířky prvků rozhraní dle potřeb. Několik textových polí vyžaduje vstup uživatele přes klávesnici, nejsou to ale důležité položky. Zbytek vstupních prvků jsou tlačítka či pole s možností výběru několika možností.

4.4 Zhodnocení

V podkapitole o implementaci služby (viz. 4.2) bylo zmíněno několik faktorů, které mohou potencionálně způsobovat potíže při práci. K některým z nich opravdu dochází, avšak díky poměrně výkonnému serveru s dobrým připojením se většina z nich nemá možnost projevit. Následující hodnocení vyplývá z hlášení a odpovědí uživatelů služby na mé otázky.

Uživatelé jsou obvykle nespokojeni, že dochází k mnoha problémům a zdržením, a to především kvůli ovládání webového rozhraní služby a vzdálenosti od ní. Dále jsou popsány nejpodstatnější nedostatky, které uživatelé zmiňují. Ty jim přináší problémy v podobě velké časové náročnosti procesu, ten jim brání v jiných povinnostech. Za připomenutí stojí fakt, že tito pracovníci jsou obvykle vedoucí týmů, tedy ti nejzaneprázdněnější, a proto mají k takovéto práci negativní přístup.

4.4.1 Pomalé rozhraní

Stížnosti uživatelů jsou často směřovány na rozhraní v prohlížeči. Uživateli je hodnoceno jako pomalé, s dlouhým časem odezvy. Hlavními důvody jsou:

- Implementace není příliš rychlá při práci s větším množstvím dat. Mezi důsledky patří pomalé a dlouhé načítání při kliknutí na některý prvek. Při tom začne stránka dokonce automaticky měnit rozměry jednotlivých rámců, přestože si to uživatel nežádal a mnohdy je to pro něj nežádoucí. Tato změna je pomalá. Jsou vidět jisté rozdíly při spuštění v různých prohlížečích, avšak i v „těch rychlejších“ není výsledek zdaleka dostatečný. Lze hovořit i o několika sekundách po každém kliknutí.

- Se vzdáleností roste i doba odezvy serveru, a to rapidně. Z opačného konce planety, než je server umístěn, dosahuje úroveň zpoždění i stovky milisekund či sekund (ověřeno ping zprávami uživatelů v Číně).

- Obtížnost, zdlouhavost – provedení některých operací není ihned jasné a vyžaduje důkladnější prozkoumání rozhraní. Mnoho akcí vyžaduje více kliknutí, než je uživatelům příjemné.

4.4.2 Velké množství shod

Službou generované množství potenciálně škodlivých shod je obrovské. Lze hovořit i o projektech o několika tisících souborů se shodami u více než 50% z nich. U některých jsou shody v jednotkách, u většiny okolo desítky, v mnoha případech i v řádu několika desítek.

Shody jsou generovány i mezi komponentami stejné společnosti. Takovéto chování je pochopitelné v případě, pokud jsou vydávané pod jinou licencí, ovšem ze zkušeností a stížností uživatelů vyplývá, že řeší shody mezi komponentami licence stejné či velice podobné. Vlastně je takových mnohdy více, než všech ostatních shod. (TODO:obrázek projektu s hodně shodami)

4.4.3 Absence automatizace/hromadných řešení

Ačkoliv lze při identifikaci deklarovat soubor jako patřící (originální) řešenému projektu (popsáno v kapitole 4), tato možnost v důsledku zabrání zobrazení případných nových shod, které budou díky tomuto ignorovány. Uživatelé by spíše uvítali možnost vyřešit všechny shody hromadně a být upozorněni na další, které se objeví.

Samotná identifikace více shod jednoho souboru či několika souborů najednou není možná – přítomny jsou jen akce pro identifikaci po jednom souboru. Jak bylo ukázáno v předešlé kapitole, počet shod se může pohybovat v tisících. Řešení jednotlivě zabere nepřípustné množství času. Identifikace po celých komponentách často nepřichází v úvahu, jelikož uživatel mnohdy musí projít všechny soubory, aby si byl zcela jist, že nedošlo k pochybení.

Kapitola 5

Požadavky na nový systém

Existence souboru knihoven (Protex SDK [2], později bude blíže rozebrán v kapitole 6.2) otevírá možnost vytvoření nového klienta (rozhraní) pro komunikaci se službou. Tato kapitola detailněji rozebere požadavky na podobu, způsob fungování a ovládání nového systému, který má rozšířit stávající a přinést nové způsoby práce se službou Protex.

Tyto požadavky zprvu vznikaly spíše jako nápady uživatelů, ale díky jejich počtu došlo k sepsání ucelenější specifikace, které se tato práce věnuje. Specifikace nejsou popsány nikterak formálně, jedná se spíše o několik zásadních bodů, kterým bylo rozhodnuto se věnovat. Následující kapitola se vám je pokusí přiblížit.

5.1 Hlavní idea

Plánem bylo prozkoumání možnosti vytvoření aplikace, která by se chovala jako náhradní a doplňující rozhraní či klient pro službu Protex. Implementační možnosti nebyly zpočátkuu příliš zřejmé, jednalo se spíše o výzkum, jak ulehčit práci, šetřit čas a obejít nedostatky popsané v kapitole 4.4.

Nejedná se ani tak o vytvoření kompletní náhrady, jako spíš pracovního nástroje, který by byl použitelný v jednodušších případech co nejrychleji a nejefektivněji, zbylé složité faktory by se ve výsledku musely řešit v původním rozhraní.

5.2 Odezva

Jeden ze tří hlavních předmětů výzkumu je řešení odezvy jak uživatelského rozhraní služby, tak i mezi klientem a serverem. Nový systém má reagovat na každou akci co nejrychleji, nejlépe okamžitě, a nebo pracovat v takovém režimu, aby uživatel nemusel být přítomen celému běhu – tedy jej pouze spustit a po ukončení zkontrolovat výsledky.

Dalším řešením je i pokusit se eliminovat zpoždění způsobené komunikací se serverem jeho vynecháním, tedy prací zcela, nebo alespoň částečně, offline. Pravděpodobně tedy data stáhnout, případně také uložit, následně s nimi pracovat. Po dokončení práce výsledky zaslat serveru.

5.3 Ovládání

Ovládání nového systému má být co nejpodobnější původnímu, ovšem snažší a rychlejší. Mnoho věcí není třeba vůbec zobrazit, častěji volené akce z rozhraní mají být okamžité.

Důležitým faktorem má být integrace klávesových zkratek, a tedy možnost celý nový systém ovládat klávesnicí.

Dalším nápadem je prozkoumání možnosti spouštění z příkazové řádky, některé úlohy se mají vykonávat bez účasti uživatele pouhým nakonfigurováním a odstartováním skriptu.

5.4 Automatizace

Mnoho službou vygenerovaných shod uživatel řeší téměř okamžitě, jelikož již při letmém pohledu na některý z atributů ví, že nebude relevantní či nebude vyžadovat důkladnější prozkoumání. Kdyby bylo možné najít souvislosti mezi takovýmito shodami nebo vymezit nějaká pravidla pro automatickou identifikaci, a tyto posléze aplikovat získáváním informací ze serveru služby, mohly by být některé shody řešeny automatizovaně.

Tato automatize procesu identifikace by měla fungovat dvojím způsobem. První způsob by měl být reprezentován jako množina filtrů, která by zobrazila pouze soubory či shody, které splňují předem vymezené podmínky, a ty následně řešit postupně, avšak rychleji díky předpokladu společného atributu. Druhý způsob má pak fungoval obdobně, avšak identifikace by se aplikovaly zcela samostatně bez účasti uživatele, který by jen vybral podmínky, za kterých by k takovéto akci došlo.

5.5 Uživatelské rozhraní

Jelikož jsou známy (z uživatelského pohledu) nedostatky současného rozhraní, lze rozhraní uzpůsobit jinak.

5.5.1 Grafické

Pokud by bylo možné vytvořit grafické rozhraní, nezáleželo na tom, zda by bylo využíváno ve webovém prohlížeči, či by bylo implementováno jako zcela samostatná aplikace.

Mnohem podstatnější požadavky byly:

- Rychlost – co nejkratší odezva na každou akci uživatele.
- Podobnost – vzhledem k původnímu grafickému rozhraní. Tímto je myšlena především lokalizace informací ve stejných částech obrazovky a provádění akcí totožným nebo podobným způsobem.
- Jednoduchost – vynechání informací, které většinou nejsou potřeba k rozhodování. Eventuelně by měla být k dispozici možnost takové informace získat a zobrazit dodatečně.
- Odezva – při vykonávání delších akcí musí být v každém okamžiku jasné, co se právě děje, rozhraní nesmí „jen tak zamrznout“.

5.5.2 Textové

Součástí aplikace mají být i skripty, které budou spouštěny pouze z příkazové řádky a budou plnit výhradně automatizační úlohy popsané v kapitole 5.4. Tyto skripty mají být co nejjednodušší. Tím rozumíme minimum zbytečných argumentů a pokud možno nulová interakce s uživatelem po jejich spuštění.

Kapitola 6

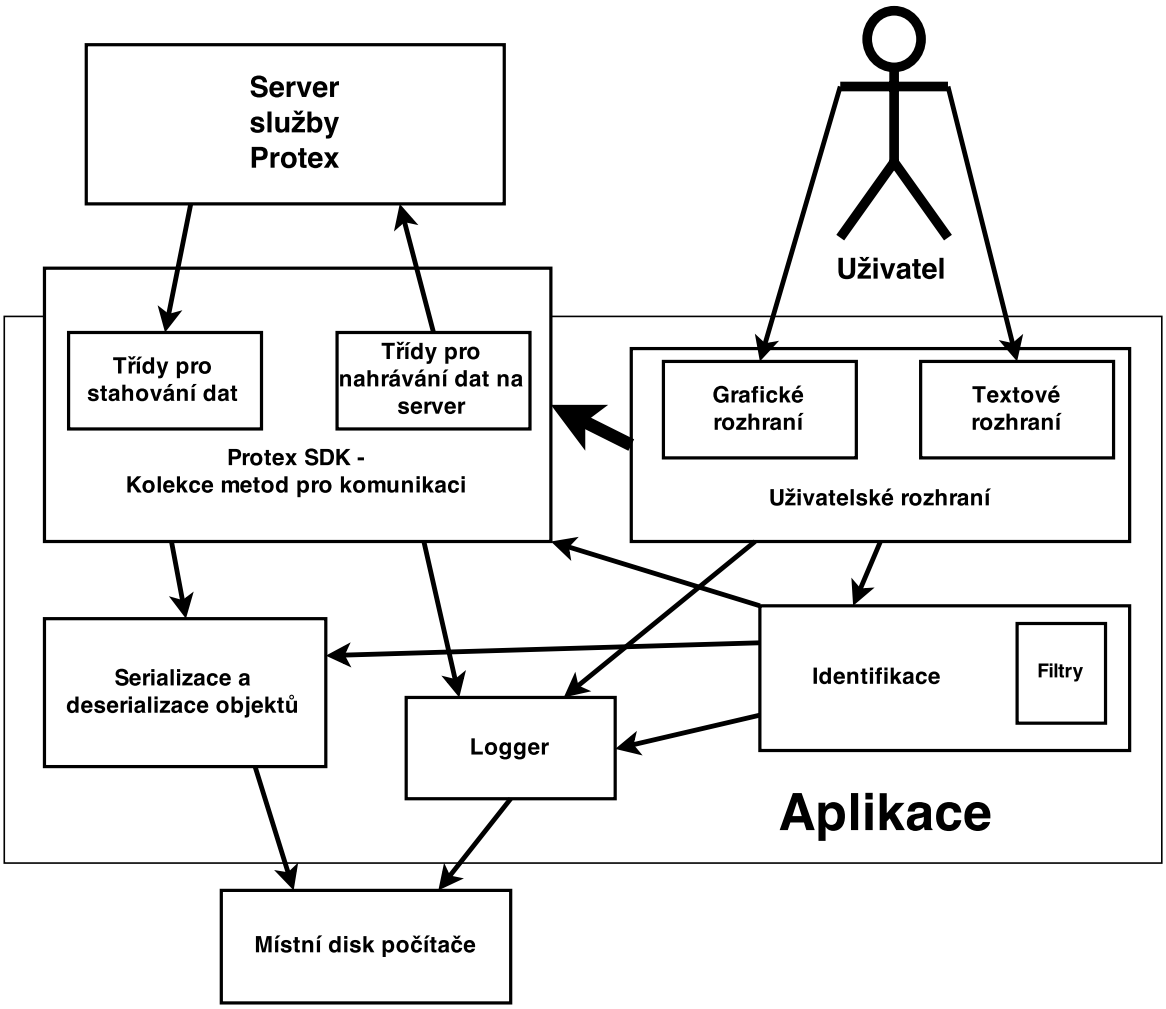
Návrh nového systému

Tato kapitola se bude detailněji zabývat jednotlivými částmi návrhu nového systému. Jedná se o sadu knihoven, které poskytuje služba samotná, využité prvky programovacího jazyka Java, který bylo nutné použít, jeho technologií pro tvorbu uživatelského rozhraní, využité vývojové prostředí, testovací nástroje a systém pro správu verzí atd.

6.1 Struktura aplikace

Obrázek 6.1 znázorňuje strukturu celé aplikace, její propojení s okolím a jejích jednotlivých částí mezi sebou. Všechny prvku z obrázku (a nejen ony) budou detailněji popsány ve zbytku kapitoly, nyní následuje popis jejich propojení.

- Uživatel komunikuje s aplikací přes uživatelské rozhraní. To se může reprezentovat jako grafické nebo textové, avšak se zbytkem systému komunikují obě varianty jednotným způsobem. Uživatelské rozhraní ovládá všechna volání serveru služby a celý proces identifikace.
- Uživatel uživatelským rozhraním ovládá server služby Protex přes kolekci tříd a metod pro to určených. Ty jsou postaveny na části knihoven pojmenovaných Protex SDK. Nejsou použity všechny části knihoven. Třídy v SDK jsou jediným možným způsobem komunikace se zbytkem služby.
- Uživatel spouští svými akcemi identifikaci shod. Ta se projevuje dvěma způsoby:
  1. V případě manuální identifikace se načítají lokálně uložené soubory, na jejich shodách se provede identifikace a soubory se opětovně uloží na původní místo.
  2. V případě automatizačních úloh proces identifikace sám vyžaduje a řídí stahování dat, na těch provede změny a obratem zažádá o jejich poslání zpět na server služby Protex.
- Aplikace ukládá data na místní disk serializací. Při stahování dat vzniká potřeba serializovat a uložit je na disk. Nahrávání dat zpět na server používá proces opačný. Identifikace využívá postupně obou procesů, a to krátce po sobě.
- Aplikace vytváří za běhu výpisy na standardní výstup a do logovacího souboru uloženého na místním disku. Tyto výpisy jsou vytvářeny prakticky všemi třídami systému.



Obrázek 6.1: Diagram důležitých částí aplikace, jejich propojení s okolím a mezi sebou.

6.2 Protex SDK

Přímo ve službě je k dispozici ke stažení soubor knihoven, kterých je třeba pro veškerou komunikaci. Není zcela zřejmé, zda původní webové rozhraní využívá stejné metody přístupu k datům, ovšem pro tvorbu externího nástroje není v tomto ohledu možnost volby.

Z knihoven jsou relevantní pouze dvě. Zbylé jsou jen jejich závislosti. Převážná většina z nich slouží k síťové komunikaci, zpracovávání různých zpráv a XML elementů a k tvorbě vzdálených databázových dotazů. Těmi dvěmi důležitými jsou:

1. `protex-sdk-client` – obsahuje všechny třídy, které dohromady tvoří SDK a jsou potřebné k jakékoliv práci se službou Protex. Jsou rozděleny do několika základních balíčků, z nichž se každý zaměřuje na jinou část služby.

2. `protex-sdk-utilities` – tato knihovna se stará o propojení první knihovny se zbytkem, který slouží pouze jako nástroje.

6.3 Jazyk implementace

Všechny knihovny potřebné pro vytvoření nového systému jsou implementovány v programovacím jazyce Java (viz. [7]). Tato skutečnost nebyla shledána jako problém a jejím důsledkem je pouze záruka přenositelnosti výsledné aplikace mezi různými platformami (zaručenou virtuálním strojem Javy).

Potenciální hrozba menší rychlosti a větší paměťové náročnosti výsledného systému nemá žádný vliv, jelikož většinu času aplikace čeká na zprávy od serveru či pevného disku, což jsou všechno blokující operace – aplikace (nebo její vlákno) jen čeká, předá řízení jinému procesu – viz. [10].

6.4 Soubor API tříd

Knihovna `protex-sdk-client` zmíněná v kapitole 6.2 obsahuje soubor tříd, které dohromady tvoří tzv. Protex API). Jedná se v podstatě o třídy s množinou komplexnějších metod pro komunikaci mezi klientskými aplikacemi a serverem. Zbylé třídy a jejich metody obsažené v knihovně lze používat jakkoliv je třeba, avšak neinvokují žádnou formu síťové komunikace – API je tedy jediným způsobem přístupu k datům na serveru. Tyto třídy jsou inicializovány hned po vytvoření připojení se serverem a přihlášení uživatele.

Podle zaměření se API dělí na několik částí, z nichž budou ty klíčové pro naši práci podrobněji popsány spolu s metodami, které z nich používáme, v následujících podkapitolách. Ovšem je nejprve třeba upřesnit vztah mezi třídami knihoven a prvky služby, které reprezentují. Důvodem zabývání se těmito třídami je snaha o přiblížení se a porozumění procesům uvnitř SDK, ke kterým vede nutnost vytvořit vlastní kolekci metod pro jednotnou komunikaci se serverem.

Je nutno zmínit i fakt, že právě metody obsažené v API třídách jsou částí blokujících operací, o kterých byla řeč v předešlé kapitole.

6.4.1 Důležité třídy vracené API metodami

Následující třídy jsou vraceny serverem při volání API metod. Jejich význam je popsán v kapitole 3.6, zde je uvedeno, k jakým částem služby patří, aby mohlo být dále objasněno, jakým způsobem je zpřístupnit.

- `Project` – projekt vytvořený uživatelem (viz. kapitola 3.1).
- `Component` – komponenta z databáze (viz. kapitola 3.1).
- `CodeTreeNode` – soubor, složka nebo archív ve stromu projektu či komponenty.
- `CodeMatchDiscovery` – shoda mezi soubory (viz. definice 1).
- `RelatedSnippet` – informace o úsecích, ve kterém se dva soubory shodují.

6.4.2 Třída `ProjectApi`

Jedna z nejdůležitějších částí SDK sloužící ke správě uživatelských projektů – vytváření, mazání a změny projektů, spouštění analýzy. Pro tuto aplikaci je relevantní pouze metoda `getProjects()`, která stáhne základní informace o projektech přístupných aktuálně přihlášenému uživateli ve formě seznamu objektů třídy `Project`. Bez těchto informací nelze nadále pracovat, a proto je metoda použita hned po přihlášení, tedy na začátku běhu aplikace.

6.4.3 Třída `CodeTreeApi`

API sloužící ke správě souborů v rámci uživatelských projektů, tedy jejich nahrávání a mazání. Potřebnými metodami jsou:

- `getFileTreeNode()` – slouží k získání informací o souborovém stromu zvoleného projektu a jednotlivých složkách či souborech pomocí stažení seznamu uzlů – objekty třídy `CodeTreeNode`. Tyto informace jsou nutné pro jakoukoliv práci na projektu. Bez těchto uzlů není možné dostat se ke shodám. Stažení probíhá jen pro specifikovanou cestu relativní ke kořenu projektu.
- `getFileContent()` – stažení zdrojového kódu souboru reprezentovaného uzlem (`CodeTreeNode`). Tuto metodu lze aplikovat pouze na soubory z uživatelských projektů, nikoliv na soubory jakékoliv jiné komponenty.

6.4.4 Třída `DiscoveryApi`

API sloužící ke stáhnutí samotných shod ve zdrojových kódech, které jsou dostupné v databázi služby po spuštění analýzy. Metoda `getCodeMatchDiscoveries()` stáhne shody reprezentované seznamem objektů typu `CodeMatchDiscovery` pouze pro zadané uzly souborového stromu projektu, ne pro projekt celý.

6.4.5 Třída `FileComparisonApi`

API sloužící k získání seznamu úseků (seznam objektů typu `RelatedSnippet`), v nichž se dva soubory shodují. Využívána je metoda `getFileSimilarities()`, která potřebuje odkazy dvou souborů označených shodou – jeden z uživatelského projektu a jeden z komponenty v databázi.

6.4.6 Třída `ComponentApi`

Tato třída aplikaci slouží ke stažení informací jako jméno či identifikační číslo komponent, na které se odkazují shody (`CodeMatchDiscovery`). Tato funkce je využita, pokud má být provedena identifikace úseku zdrojového kódu jako původního ke komponentě z databáze. Metoda `getComponentsByKey()` stáhne informace o komponentě reprezentované třídou `Component` a to pokaždé pro jednu shodu `CodeMatchDiscovery`.

6.4.7 Třída `IdentificationApi`

Tato část knihovny API slouží k práci s identifikacemi shod v projektech.

- `getAppliedIdentifications()` – metoda slouží ke zjištění, zda už v projektu nějaké identifikace proběhly. Pokud ano, stáhne přesné informace o všech shodách, kterých se týkají. Tohle je důležité u projektů, kde již nějaká práce proběhla, protože není žádoucí zabývat se shodami znovu. Identifikace jsou stáhnuty pro zadané shody typu `CodeMatchDiscovery`.
- `addCodeMatchIdentification()` – nahrávání lokálně vytvořených žádostí o identifikaci na server, kde budou vyhodnoceny a v případě schválení službou i aplikovány.

6.4.8 Třída `BomApi`

Toto API slouží k obnovení seznamu komponent projektu (nazývaného „Bill of Materials“, popsáno v kapitole 3.5) a informací o identifikovaných a nedentifikovaných shodách pomocí metody `refreshBom()`, která bere jako argument objekt třídy `Project` – specifikace projektu, jehož data obnovujeme.

6.5 Stahování a ukládání dat

Jednou z nejdůležitějších schopností aplikace je možnost stahování dat ze serveru bez účasti nebo dohledu uživatele a případně jejich následné uložení pro práci v libovolný čas kvůli zmírnění časové náročnosti celého procesu. Data jsou stahována pomocí API metod popsaných v kapitole 6.4. Pokud se uživatel rozhodne na projektu pracovat, nebylo by efektivní stahovat jej znovu při každém zapnutí aplikace – proto je třeba obdržené objekty rozumným způsobem ukládat.

6.5.1 Serializace

K ukládání dat na disku byla zvolena metoda serializace (viz. [8]) objektů na místní, pevný disk. Pokud třída v Javě implementuje rozhraní `Serializable`, lze její instance svévolně ukládat na disk a následně je znovu snadno načítat. Po přečtení dat z disku vznikne totožný objekt s identickými hodnotami atributů.

Problémem však je, že většina tříd v Protex SDK serializaci nepodporuje. Byly proto vytvořeny třídy nové s úkolem je reprezentovat. Po stáhnutí jsou objekty služby rozebrány a relevantní informace jsou uloženy do nových instancí tříd pro uložení. Většinou jsou převzety pouze atributy typu `String` (textový řetězec) a `int` (celé číslo), jako například identifikační čísla či jména, takže je splněna rekurzivní podmínka, že objekt lze serializovat jen můžeme-li serializovat všechny jeho atributy.

6.5.2 Serverová volání – přihlašování a stahování dat

SDK je postavené tak, že k datům na nižší úrovni abstrakce (pro projekt) je třeba se dostat z vyšší úrovně. Typický průběh volání API metod pro uživatele, který se chce přihlásit a stáhnout data k jednomu projektu by byl asi takovýto:

1. Ustanovení spojení se serverem pomocí URI.
2. Přihlášení uživatele a ověření správnosti přihlašovacích údajů.
3. Inicializace všech používaných API tříd.
4. Stažení seznamů projektů dostupných přihlášenému uživateli.
5. Stažení souborového stromu pro zvolený projekt (`CodeTreeNode`).
6. Stažení zdrojových kódů pro vybraný soubor.
7. Stažení shod (`CodeMatchDiscovery`) pro vybrané soubory.
8. Stažení již aplikovaných identifikací na straně serveru pro označení stáhnutých shod jako již vyřešených.
9. Stažení informací o úsecích, ve kterých se soubory shodují – „snippetu“.

Po tomto procesu případně proběhne získání relevantních informací ze získaných objektů a serializace jejich reprezentací.

Některá z těchto volání se týkají více souborů či objektů najednou, avšak jiná je potřeba volat na každý soubor zvlášť. Z toho je patrné, že pro stáhnutí celého projektu je potřeba velké množství volání, což zabere mnoho času.

Maximální množství volání

Maximální počet vzdálených volání služby pro stahování i nahrávání dat lze vyjádřit pomocí vzorce. Následující vzorec popisuje vpočet serverových volání pro postup popsaný v kapitole 6.5.2:

$$\text{Maximum volání} = 5 + n + m + \frac{m}{o} + 2p \tag{6.1}$$

kde platí, že

- 5 volání je pro:
  1. Připojení k serveru
  2. Přihlášení uživatele
  3. Ověření přihlašovacích údajů uživatele
  4. Stažení seznamu projektů přístupných uživateli
  5. Stažení všech uzlů projektu
- n je počet inicializovaných API tříd
- m je počet všech uzlů projektu – pro každý uzel zvlášť stáhnout zdrojový soubor

- o je počet souborů řešených v jednom vlákně – shody se stahují jednou pro každé vlákno
- p je počet všech shod všech uzlů m – pro každou shodu zvlášť se stahují informace o již provedených identifikacích a informace o úsecích shody

6.5.3 Serverová volání – nahrávání identifikací na server

Tento proces je na serverové požadavky ještě náročnější. Pokud předpokládáme již stáhnutý projekt s provedenými identifikacemi, ovšem právě nanovo zaplou aplikaci, je proces takovýto:

1. Ustanovení spojení se serverem pomocí URI.
2. Přihlášení uživatele (a ověření správnosti přihlašovacích údajů).
3. Inicializace všech používaných API tříd.
4. Stažení seznamů projektů dostupných přihlášenému uživateli.
5. Stažení souborového stromu pro zvolený projekt (`CodeTreeNode`).
6. Stažení shod (`CodeMatchDiscovery`) pouze pro soubory, jež byly označeny jako pozměněné.

Tyto volání jsou však pouze první vlnou. Po nich následuje zpětná identifikace serializovaných souborů a shod s těmi znovu staženými. Tento proces není příliš efektivní, avšak nezbytný pokud chceme pracovat offline a ukládat data lokálně.

Po identifikaci původních shod, jež chceme identifikovat, musíme vytvořit žádosti o identifikaci a ty nahrát na server. Tohle ovšem nelze provést najednou, protože proces by pak nekoreloval s původním modelem identifikace popsaném v jedné z úvodních kapitol – 3.3. Mezi nežádoucí efekty patří například nutnost resetu všech identifikací v každém úseku shody místo pouze jedné, aktivní identifikace. Kromě toho není zcela patrné, jakým způsobem řeší server služby stále nové identifikace na stále tentýž úsek souboru.

Bezpečný, avšak časové náročný proces lze popsat takto:

1. Vyber všechny soubory, na kterých byly provedeny identifikace, a identifikované shody k nim patřící.
2. Stáhni ze serveru služby všechny shody čekající na identifikaci a přiřaď je k jejich serializovaným, identifikovaným reprezentacím vybraných souborů.
3. Od každého souboru nahraj na server jednu, nejdříve vytvořenou identifikaci. Uchovej informaci o souborech, jejichž data byly nahrány.
4. Stáhni informace o všech souborech dosud vybraných.
5. V seznamu vybraných ponechej pouze ty, které stále nejsou zcela identifikovány.
6. Pokud seznam vybraných souborů není prázdný, pokračuj krokem 2.

Druh volání	Jeden objekt	Více objektů
`getCodeMatchDiscoveries()`	2.15824 s	2.16826 s
`getFileSimilarities()`	2.19107 s	2.17352 s
`getAppliedIdentifications()`	2.11419 s	2.14518 s

Tabulka 6.1: Tabulka srovnání trvání některých API volání pro jeden objekt zvlášť a více objektů najednou. Každý výsledek je aritmetickým průměrem pěti hodnot. Jednotkou jsou sekundy.

6.5.4 Doba trvání volání

Z tabulky 6.1 je patrné, že u většiny běžných volání nezpůsobí zásadní rozdíl množství přenášených dat. Naopak v případě metody `getFileSimilarities()` je to právě naopak, průměrná doba trvání pro více objektů byla nižší, než pro objekt jeden. Problémem je spíše nutnost žádat o nepotřebné informace pro přístup k informacím potřebným, či nutnost ptát se na stejnou věc pro každý soubor zvlášť. Zde je vhodné povšimnout si, že serveru trvalo vždy více než dvě sekundy odpovědět. Je velice nepříjemné čekat takto dlouho na data po každém kliknutí v rozhraní.

6.5.5 Nároky a omezení

Dosud zmíněné fakta o procesu stahování a ukládání dat mají negativní důsledky pro nového klienta.

Prvním omezením je mnoho souborů a ještě více shod v projektu. Vzdálená volání mohou potencionálně vracet nekonečně objektů, avšak v praxi může při zbytečně velkém požadavku (a to i přestože mohou metody vracet teoreticky nekonečně velký seznam objektů) dojít k těmto problémům:

1. Na serveru dojde při příliš velkém databázovém dotazu k chybě.
2. Objektů může být tolik, že bude mít virtuální stroj Javy problém je uchovat v jednom seznamu.

Pokud se podaří předat všechny objekty v jednom seznamu, dalšími problémy, jak díky velikosti, tak potenciální neuspořádanosti výsledku, jsou:

1. Výsledný seznam objektů zabere nepřípustné množství operační paměti.
2. Nad výsledným seznamem je třeba vykonávat operace v cyklech, a to často – hlavně při vyhledávání a řazení. Příliš velký seznam zpravidla není nejefektivnější pro takovéto algoritmy.

Z těchto všech faktorů vyplývá, že je nutné žádat o menší množiny výsledků – omezení například rozdělením na podstromy v souborovém stromu uzlů projektu. To ovšem způsobuje další limitaci – doba stahování a ukládání dat. Volání velkého množství blokujících operací znamená nepřípustné zdržení a to i při běhu aplikace na pozadí. V praxi lze hovořit až o několika dnech na stažení těch největších projektů (nižší desítky tisíc souborů, ověřeno manuálním testem aplikace).

6.6 Vícevláknový přístup

Pro zmírnění problémů popsaných v minulé kapitole byl zvolen vícevláknový přístup pro download a upload dat. Při spuštění jednoho z těchto dvou procesů se aplikace přepne do režimu, kdy nereaguje na podněty uživatele, avšak zobrazuje výsledky svých operací v textové podobě.

Díky tomuto přístupu může aplikace posílat další požadavek službě či ukládat soubor na disk, zatímco čeká jiné vlákno na výsledek stejné operace spuštěné dříve.

6.6.1 Způsob dělby práce, velikost vláken

Definice 5. Velikostí vlákna rozumíme počet souborů jemu přiřazen k řešení.

Na výsledný způsob definice působnosti jednotlivých vláken mají vliv především tyto tři spolu související faktory:

- Pokud ve vláknu dojde k chybě související s voláním metody z SDK, obvykle nemá smysl pokračovat s další prací, jelikož chybí nějaký důležitý druh dat.
- Některé metody potřebných API podporují jako parametr více souborů, avšak jiné nikoli. Pokud vláknu přiřadíme větší počet souborů k řešení, potenciálně lze ušetřit čas snížením počtu celkových volání. Avšak zvýší se tímto možnost chyby ve vláknu, protože posléze musí volat vícekrát metody určené pro soubory zvlášť.
- Dopředu není známo, kolik informací a požadavků o informace ze serveru vlákno musí ve výsledku provést. Může se tedy při nevhodné velikosti vláken stát, že některá obsahující nerelevantní data již dávno ukončila svou činnost

Z těchto důvodů bylo zvoleno rozdělení všech relevantních souborů mezi jednotlivá vlákna tak, aby každé vlákno obstaralo veškerou činnost potřebnou pouze pro jemu dané soubory, ukončilo se a uvolnilo místo dalšímu vláknu, které dělá shodnou činnost. Počet souborů pro přidělení jedinému vláknu je určen systémovou konstantou – z bezpečnostních důvodů nepřístupno uživateli ke změně.

6.6.2 Počet souběžných vláken

V průběhu vývoje aplikace docházelo k průběžnému manuálnímu testování ideálního počtu vláken tak, aby byla komunikace se serverem co nejrychlejší z pohledu uživatele, ale příliš server nezatěžovala. Problémem v tomto ohledu je absence přístupu na stroj serveru, kde služba běží, lze tedy zkoumat jen změny, které se projeví lokálně.

Při velkém počtu vláken byla zaznamenána narůstající chybovost při posílání odpovědí na požadavky – tedy na straně serveru. Tato chybovost při jistém (z bezpečnostních důvodů nezveřejněném) počtu vláken může způsobit stav, kdy služba není schopna nadále odpovídat a je nutný manuální restart.

Maximální počet souběžných vláken je určen systémovou konstantou z bezpečnostních důvodů nepřístupnou uživateli, který má k dispozici aplikaci pouze ve zkompilované podobě.

6.7 Filtry

Pro jednodušší řešení shod byl navržen systém filtrů. Ten funguje rozdílně pro grafické a textové rozhraní, ale v principu v obou případech aplikaci říká, jakých shod si všímat a jakých ne.

Bylo implementováno několik základních druhů filtrů a architektura tohoto systému umožňuje ty stávající relativně snadno rozšířit o další. Filtry lze kombinovat. Avšak ne všechny jsou k dispozici pro obě rozhraní, spíše naopak, každý je určen pro jiný druh práce.

K vyhodnocování, zda jsou filtry aplikovatelné, dochází v několika různých okamžicích identifikace podle toho, k čemu se filtry vztahují – například filtrace podle komponent, ke kterým se shody vztahují, podle zdrojových kódů apod.

Nejzákladnější (a nejčastěji využívané) filtry jsou například:

- Zobrazení pouze souborů, ke kterým se vztahují nějaké shody. Tento filtr je aplikovatelný pouze u grafického rozhraní.
- Identifikace souborů, jejichž všechny shody jsou s komponentami vypsanými v seznamu, a které jsou původem z databáze (tedy standardní komponenty, viz. kapitola 3.1).
- Identifikace souborů, jejichž všechny shody jsou s komponentami vypsanými v seznamu, a které jsou původem od stejné společnosti (tedy Custom komponenta, viz. kapitola 3.1)
- Identifikace souborů, jejichž všechny shody jsou s komponentami vypsanými v seznamu filtrovaných. V tomto případě se nekontroluje druh komponent.
- Identifikace shod se soubory třetí strany obsahující někeré z textových řetězců vypsaných v seznamu filtrovaných. Příkladem je přítomnost informace o licenci.

6.8 Tvorba grafického rozhraní a NetBeans

Grafické prostředí bylo vytvořeno pomocí vývojového prostředí NetBeans[1], které podporuje snadnou tvorbu rozhraní typu Swing [5]. Z tohoto důvodu je celá aplikace při překladu závislá na jeho použití.

Poměrně jednoduchým úkolem bylo sestavit jednotlivá okna aplikace, přidat do nich funkční GUI komponenty (seznamy, tlačítka, checkboxy apod.) a nastavit jejich vzájemné pozicování. Veškerá tato práce je realizována pomocí NetBeans.

Pro funkčnost grafických komponent však bylo potřeba vytvořit větší množství metod. Velká část z těchto metod pouze načítá a zobrazuje data, tedy stačilo použití výchozích tříd a jen jim implementovat žádoucí chování. Výjimkou je v tomto ohledu ovládání komponenty zobrazující souborový strom projektu.

6.8.1 Souborový strom

`JTree` [6] je třída grafické komponenty pro zobrazení souborů a složek ve stromu jak jsme zvyklí z běžných aplikací. Mezi její hlavní schopnosti patří:

- Načtení dat do speciální třídy `DefaultMutableTreeNode`.

[1] Více informací o vývojovém prostředí lze nalézt na https://netbeans.org/

- Sestavení stromu z uzlů (instance třídy z předchozího bodu).
- Rozlišení složek a souborů atributem.
- Sbalování a rozbalování (expanze) složek.
- Detekce akcí nad jednotlivými elementy – každý uzel je zobrazen na jednom řádku.

Při implementaci aplikace bylo zjištěno, že výchozí třída `DefaultMutableTreeNode` nesplňuje požadavky pro rychlost práce a aplikaci filtrů (popsány v kapitole 6.7), tedy je nutno ji rozšířit využitím dědičnosti. Tato nová třída `ProjectNode` obsahuje atributy určené pro indikaci vlastností využívaných filtry a často potřebovaná data z disku, která se nemusí často opakovaně načítat.

Virtuální souborový strom z objektů této třídy je sestaven při zvolení projektu a to pouze jednou. Důsledkem je tedy čekání jen jednou při začátku práce, i když delší dobu, avšak posléze již máme k dispozici všechna potřebná data o souborech uložených na disku a nemusíme pořád dokola procházet pevný disk k získání těchto dat či zjištění přítomnosti souborů samotných.

Díky uchovávání několika informací o obsahu se i aplikace filtrů stává téměř okamžitou. Nová třída totiž umožňuje jednotlivé uzly jednoduše nezobrazovat a dokonce uzel komponentě oznámí, že už nemusí prohledávat celý jeho podstrom.

6.9 Chybějící funkcionalita, chyby

V implementaci chybí jedna část služby a jedna nefunguje korektně. Tento fakt způsobuje omezení při práci, ale ne natolik závažná, aby se s novým systémem nedalo pracovat.

6.9.1 Zdrojové kódy třetí strany

Zcela úmyslně není uvedeno, kde přesně se tato chyba nalézá, jelikož to znamená bezpečnostní riziko pro službu. Jedna z API tříd zmíněných v kapitole 6.4 obsahuje metodu pro obdržení zdrojových kódů nikoli vlastních, ale třetí strany. Tato metoda vrací URL na server s databází, kde se mají data nalézat. Tohle URL však vrací jen zcela prázdnou HTML stránku. Není doposud jasné, zda je to problém oprávnění či technický, ale byl vznesen dotaz na technickou podporu služby. V konečném efektu tato funkcionalita doposud není v aplikaci zahrnuta.

6.9.2 Absence informací o úsecích shody pro některé soubory

V případě některých souborů (někdy i všech v určitých komponentách) nejsou k dispozici informace o tom, v jakých úsecích se shodují se souborem uživatelem prověřovaného projektu. Tato chyba se projevuje obdržením prázdného seznamu objektů třídy `RelatedSnippet`. Přitom když je počet úseků nulový, je návratová hodnota vždy `NULL`. Z webového rozhraní patrné, že úseky existují. V případě této chyby byl také vznesen dotaz na technickou podporu služby.

Během identifikace se chyba projevuje nekorektním označováním shod. Shody ležící ve stejném úseku, jako nově identifikovaná shoda, mají být označeny za nadále neplatné, to ovšem nelze, když chybí informace o tom, zda vůbec ve stejném úseku jsou. Uživatel si však tohoto faktu nemusí být ani vědom, stačí identifikovat více shod. Při korektně provedené identifikaci se na serveru žádná chyba neprojeví.

6.10 Výpisy a hlášení

Aplikace obsahuje soubor tříd určených pro výpis hlášení, který byl označen za „Logger“. Výpisy jsou realizovány na standardní výstup, standardní chybový výstup v případě chyb a do speciálního `.log` souboru.

Vypisování informací na standardní výstupy má smysl především při spuštění aplikace v textovém režimu. Některé z těchto zpráv jsou upravené i pro tisk do speciálních grafických komponent k tomu určených. Tohoto se využívá hlavně při stahování dat a jejich následném nahrávání zpět na server.

Důvod je v obou případech stejný – uživatel má za každých okolností informace o tom, co se děje. To znamená hlášení postupu ve vykonávané akci a oznámení o chybách.

6.10.1 Logovací soubor

Nový soubor je vytvořen pro každé spušení aplikace do `log/` adresáře v kořenovém adresáři projektu. Jedinečné jméno souboru je datum a čas jeho vytvoření.

Soubor obsahuje stejná hlášení, která se vypisují na standardní výstupy, plus některá další. Jsou dva hlavní důvody pro jeho existenci:

- Uživatel má možnost při pádu aplikace zaslat vývojáři zprávu o tom, co se stalo. Ten pak má k dispozici přesně ty informace, které potřebuje k nápravě. Tento faktor se stal aktuálním již během začátků vývoje aplikace, jelikož ji velice rychle začalo používat několik uživatelů. Ovšem k implementaci došlo až při rozrostení systému do podoby, kdy už nelze ohlídat vše.

- V případě automatizačních skriptů uživatel nemá tušení o tom, s jakými soubory skript pracuje. Logovací soubor proto obsahuje informace:

  1. Které soubory se skript rozhodl identifikovat.
  2. U kterých souborů začal s identifikací.
  3. U kterých souborů identifikaci dokončil.

6.11 Testování

Projekt aplikace obshauje několik testovacích tříd. Původně byly testy čistě jednotkové [11], avšak v průběhu vývoje se zjistilo, že je nutné některé z nich rozšířit na integrační. Knihovna `JUnit` původně využívaná k jednotkovému testování nebyla nahrazena ani doplněna jinou.

Testovanou funkcionalitou aplikace je především vzdálené volání serveru, stahování dat ze serveru, lokální práce s nimi a jejich konečné nahrání zpět.

Třídy pro uživatelská rozhraní, kterých je v projektu více, není třeba testovat takovouto automatickou formou, ovšem je třeba investovat čas do manuálních zkoušek, které mohou prozradit více a za kratší čas. Kromě toho rozhraní pro příkazovou řádku neobsahuje žádné zvláštní prvky, které je potřeba testovat.

Čas hrál u testování zásadní roli, jelikož automatickým způsobem by se rozhodně nestihlo otestovat všechny části aplikace, která je relativně mohutná.

6.12 Správa verzí

Kvůli rychlému rozrůstání aplikace, rozšiřování o nové funkce a potřebě rychlé a snadné distribuce bylo rozhodnuto aplikaci umístit na vlastní repozitář systému pro správu verzí Git [12].

Ve zkratce je vhodné zmínit, že tento systém podporuje vývoj na několika větvích současně, přičemž z nich je jedna hlavní a ostatní se od ní odrážejí nebo se k ní zpátky připojují. V rámci každé větve pak existuje množina bodů časové osy vývoje, kdy byla práce uložena – „commit“.

Tento systém umožnil uživatelům stáhnout si celý projekt aplikace kdykoliv a odkudkoliv (podmínkou je být připojen k internetu a do firemní sítě). S aplikací pak mohli pracovat již během vývoje a kdykoliv se objevila nová funkcionalita, mohli si okamžitě aktualizovat lokální data.

Kapitola 7

Popis nového rozhraní, jeho ovládání a přínosu

Tato kapitola se zabývá předvedením nového rozhraní a jeho přínosu uživatelům. Zároveň bude popsáno základní ovládání celé aplikace. Součástí dat odevzdaných v příloze na CD jsou i jednoduché uživatelské materiály (viz. příloha A) v podobě vyžadované pracovníky, kteří již mají zkušenosti s původním rozhraním a vědí, jak se službou pracovat.

7.1 Režimy spuštění

Aplikace obsahuje jak grafické, tak textové rozhraní. Obě jsou součástí jednoho celku, ovládány jednou hlavní třídou (s metodou `main()`). Při distribuci aplikace jako spustitelného soubor typu `.jar` je nutné specifikovat, které rozhraní chce uživatel použít.

Realizace je snadná – při zadání libovolného, nenulového počtu parametrů příkazové řádky (nepočítaje nultý parametr se jménem aplikace) se spustí textový režim, jinak režim grafický. Oba si nyní přiblížíme.

7.2 Grafické rozhraní

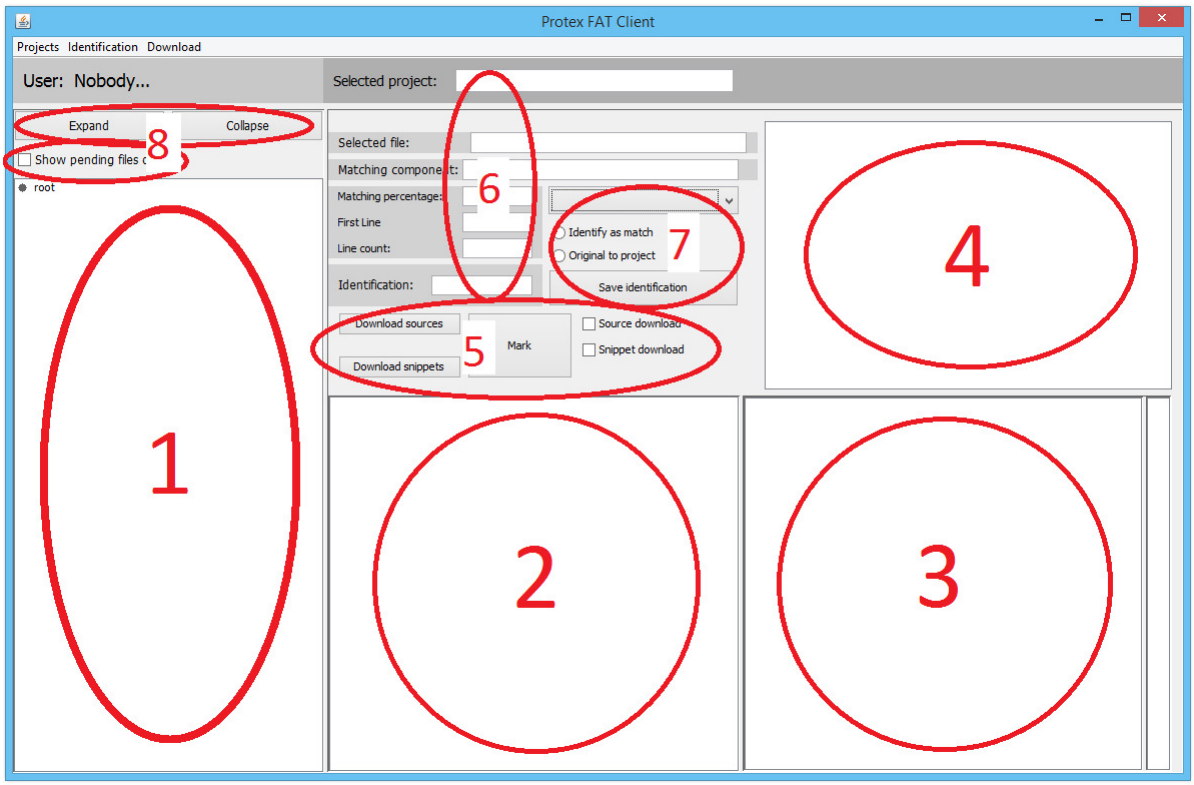
Grafické rozhraní je rozsáhlou částí projektu. Skládá se především z jednoho hlavního okna a několika oken dialogových:

- Přihlašovací okno – spouštěno jako první při startu grafického rozhraní, požaduje od uživatele přihlašovací jméno a heslo pro připojení a přihlášení se ke službě. Při správných přihlašovacích údajích se spustí hlavní okno.
- Okno pro stahování projektů – spuštěno pokud chce uživatel stáhnout a uložit projekt pro lokální práce. Obsahuje výběr projektu podle jména a možnosti pro redukci stahovaného objemu dat (lze zaškrtnout co stahovat a co ne). Přítomna je i komponenta pro vypisování průběhu stahování, aby uživatel věděl, co se děje.
- Okno výběru projektu – využívá se k výběru projektu pro načtení a práci na něm nebo ke smazání.
- Logovací okno – spuštěno při nahrávání dat na server či při stahování chybějících dat u již staženého projektu uživatelem. Slouží jen pro vypisování informací o průběhu těchto procesů.

- Okno pro výpis chyb – při neplatné operaci či chybném stavu aplikace oznámí uživateli, co je špatně.

Hlavním přínosem některých těchto oken je informování uživatele o prováděných operacích, takže není v nevědomosti, jak tomu bylo u původního rozhraní.

7.2.1 Hlavní okno



Obrázek 7.1: Hlavní okno grafického rozhraní aplikace rozdělené na významné části.

Na obrázku 7.1 jsou vyznačeny základní funkčí části okna. Je značně podobné původnímu webovému rozhraní (viz. 3.2). Hlavní okno je vytvořeno tak, aby šly měnit rozměry většiny jeho komponentů, ale pouze pokud si to uživatel přeje. Nyní budou rozebrány jednotlivé oblasti včetně nových možností, které přináší.

Oblast 1 – souborový strom

Zde jsou zobrazeny soubory a složky projektu. Hlavní výhodou oproti původnímu rozhraní je možnost výběru více položek najednou – lze provádět více akcí nad více soubory.

Oblast 2 – zdrojové kódy vlastního souboru

Zobrazí zdrojový kód uživatelem zvoleného souboru. Úsek souboru, který se shoduje s cizím souborem, je vyznačen žlutým pozadím.

Oblast 3 – zdrojové kódy souboru 3. strany

Tato komponenta není schopna zobrazit žádná data, jelikož je aplikace momentálně neumí stáhnout – viz. kapitola 6.9.

Oblast 4 – seznam shod souboru

Tato komponenta zobrazuje všechny shody aktuálně vybraného souboru. Oproti původnímu rozhraní jsou shody automaticky zobrazeny od největší po nejmenší a lze vybrat více shod najednou.

Oblast 5 – dodatečné stahování

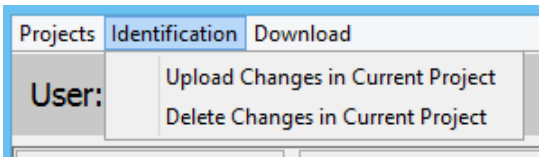
Možnosti pro stažení zdrojových souborů a informací o shodovaných úsecích k projektu, který je už stažen. Uživatel se mohl rozhodnout nestahovat tyto informace, ale během práce zjistil, že je potřebuje. Takto nemusí stahovat celý projekt znovu. Data lze stáhnout okamžitě, či si poznačit více předmětů/shod a stáhnout jich posléze více najednou. To umožní uživateli zabývat se jen podstatnými fakty, ale ponechat si možnost kdykoliv si doplnit informace důležité k rozhodování o identifikacích.

Oblast 6 – informační panely

Tyto komponenty zobrazují informace o aktuálně zvoleném souboru a shodě. Jsou zobrazeny jen ty nejdůležitější informace a to v co nejpřehlednějším formátu.

Oblast 7 – identifikace

Komponenty pro vytvoření identifikace – zvolení druhu a úrovně identifikace (viz. 3.3). Identifikace se zde jen vytvoří, není ještě nahrána na server. To se provádí najednou položkou menu „Upload Changes in Current Project“. Je také možné všechny identifikace dosud nenahrané na server smazat položkou „Delete Changes in Current Project“. Tyto možnosti vidíme na obrázku 7.2.



Obrázek 7.2: Menu položky pro nahrávání a mazání lokálně vytvořených identifikací.

Oblast 8 – manipulace se souborovým stromem

Tato oblast obsahuje dvě tlačítka určená pro rozbalení a sbalení souborového stromu. Ve větších projektech je rozdíl rychlosti oproti původnímu rozhraní v několika řádech.

Je zde i přítomno aktivační pole pro filtr, který způsobí zobrazení jen souborů s dosud neidentifikovanými shodami. Rozdíly oproti webovému rozhraní služby jsou:

1. Po aplikaci filtru zůstanou rozbalené složky nadále rozbalené.
2. Aplikace je prakticky okamžitá.

7.3 Příkazová řádka

Toto rozhraní je určeno pouze k automatizačním úlohám. Veškeré používané argumenty jsou popsány v uživatelské příručce (příloha A). Ta je psána v angličtině.

Jen ve zkratce – je třeba zadat parametry:

- Pro příhlášení – `--username=""` a `--password""` nebo `--fileLogin` (načítá přihlašovací údaje ze speciálního souboru `credentials` v kořenu projektu, kde si je může uživatel uložit pro automatické přihlašování, ovšem tohle řešení znamená bezpečnostní hrozbu v případě přístupu jiné osoby k přihlašovacím údajům). Je nutné zadat první dva nebo třetí parametr.

- Pro zvolení projektu a podstromu v souborech projektu – povinný parametr `--project=""` a volitelný `--subtree=""` (v nemž musí být podstrom specifikován cestou relativní ke kořenu projektu a začínající znakem "/").

- Pro zvolení automatizačních pravidel, neboli filtrů – `--rules=(,)` musí obsahovat jména filtrů, oddělené čárkou a bez mezer. Lze vybrat neomezeně mnoho filtrů a ty budou aplikovány v pořadí, v jakém byly uvedeny. Argument je povinný

- Pro způsob identifikace shod, na které se vztahují filtry – povinný argument `--usage=""` specifikuje jak identifikovat shody, či jako originální k projektu, či jako převzané z jiné komponenty.

- Volitelný argument `--localComponent` třídí vykonané identifikace k volitelným komponentám (viz. 3.1) podle toho, jaký filtr identifikaci způsobil.

- Volitelný argument `--reportIdentified` způsobí vytvoření hlášení o identifikacích – bude podrobněji popsáno dále.

Po spušení skriptu se může uživatel věnovat jiné práce či odejít od počítače, podmínkou pro úspěšné dokončení je jen připojení k internetu a napájení počítače.

Po dokončení se uživatel může dozvedět detailnější informace o průběhu z výpisů konzole či logovacího souboru. Při použití volitleného argumentu příkazové řádky `--reportIdentified` ho v adresáři s logovacími soubory po dokončení identifikací čeká i soubor s hlášením cest souborů, které byly identifikováný a do jaké míry. Toto hlášení může být použito i pro další hlášení jejich nadřízeným.

Kapitola 8

Závěr

Za použití knihoven Protex SDK se mi podařilo vytvořit nového klienta pro práci s touto službou. Tento klient není schopen zcela nahradit původní webové rozhraní služby Protex, avšak jeho používání jakožto nástroje pro ulehčení práce přináší jeho uživatelům nové možnosti a úsporu času.

Podařilo se mi v klientovi propojit grafické a textové rozhraní, která mají obě své klady a zápory a obě znamenají přínos. Tento přínos nelze exaktně změřit či spočítat, vyplývá spíše z uživatelských reakcí. Ti tento nástroj chtějí používat i nadále a již během vývoje se aktivně zapojovali k jeho rozvíjení svými připomínkami a požadavky.

Bohužel se nepodařilo splnit všechny požadavky, jelikož se narazilo na chyby v SDK služby. Konkrétně se jedná o nekorektní či chybějící data vracená službou, konkrétně nefunkční URL ke zdrojovým souborům třetí strany a chybějící informace o shodovaných úsecích souborů. Ohledně těchto chyb byly vzneseny dotazy na technickou podporu služby a čeká se na řešení.

Aplikace je v době odevzdání již běžně využívána, a to především k automatizaci a řešení jednoznačných, triviálnějších úloh, které uživateli zaberou nejvíce času. Někteří uživatelé uvedli, že nový klient jim v případě větších projektů ušetří 30-70% úsilí. Díky oblíbenosti je plánován její další vývoj a rozrůstání. Hlavními cíli pro další vývoj je zrychlení komunikace se serverem služby, zjednodušení vícevláknového přístupu aplikace v přístupu ke zdrojům, doplnění funkcionality chybějící kvůli chybám v SDK zmíněným v předešlém odstavci a rozšíření automatizačních úloh o další scénáře.

V zimě 2014/2015 probíhala i komunikace se zástupci firmy Black Duck, kteří přiznali, že část jejich zákazníků pracuje na automatizačních skriptech pro identifikaci shod, avšak nikdo zatím nepřišel s takto rozsáhlým projektem zahrnujícím i grafické rozhraní. Jedním zástupcem byla i přislíbena pomoc a odpovědi na otázky k SDK služby, kdyby takové nastaly. Nový klient se však nestane oficiálním nástrojem pro práci se službou Protex.

Literatura

[1] Freescale MQX™ Real-Time Operating System (RTOS). [Online].
URL http://www.freescale.com/webapp/sps/site/overview.jsp?code=MQXRTOS

[2] Black Duck Software, I.: Protex SDK documentation. [Online].
URL http://protex-bd.freescale.net/

[3] Black Duck Software, I.: Protex, Automate Open Source Compliance. [Online], 2015.
URL https://www.blackducksoftware.com/products/black-duck-suite/protex

[4] CollabNet, I.: Black Duck Protex Screenshots. [Online].
URL http://www.collab.net/

[5] Oracle®: Creating a GUI With JFC/Swing. [Online].
URL http://docs.oracle.com/javase/tutorial/uiswing/

[6] Oracle®: How to Use Trees. [Online].
URL
https://docs.oracle.com/javase/tutorial/uiswing/components/tree.html

[7] Oracle®: Java. online.
URL http://www.oracle.com/technetwork/java/index.html

[8] Oracle®: Java Object Serialization Specification. [Online].
URL http://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html

[9] Peták, T.: Obecná veřejná licence GNU v.3. [Online], 29. června 2007, neoficiální překlad.
URL http://www.gnugpl.cz/v3/

[10] tutorialspoint.com: Java - Multithreading. [Online].
URL http://www.tutorialspoint.com/java/java_multithreading.htm

[11] Vogel, L.: Unit Testing with JUnit - Tutorial. [Online], 2007, poslední aktualizace 15. května 2015.
URL http://www.vogella.com/tutorials/JUnit/article.html

[12] Vogel, L.: Git - Tutorial. [Online], 2009, poslední aktualizace 14. prosince 2014.
URL http://www.vogella.com/tutorials/Git/article.html

[13] Wikipedia: POSIX. [Online].
URL http://en.wikipedia.org/wiki/POSIX

Příloha A

Obsah CD

Kořenový adresář CD obsahuje:

- `client/` – složka s aplikací
  - `docs/` – složka obsahující uživatelské příručky
  - `filters/` – složka obsahující soubory pro filtry
  - `libs/` – složka obsahující knihovny
  - `src/` – složka obsahující zdrojové soubory
  - Soubory k projektu pro vývojové prostředí NetBeans
- `thesis/` – složka se zdrojovými soubory technické zprávy
- `thesis.pdf` – technická zpráva ve formátu PDF
- `LICENSE.txt` – soubor s licencí společnosti Freescale Semiconductor, Inc

Příloha B

Manuál

B.1 Prerekvizity

Tato aplikace je určena pro služby Protex společnosti Black Duck Software, Inc. Bez dostupného serveru této služby a přihlašovacích údajů k němu není možné s aplikací nijak pracovat.

Pro kompilaci a spuštění aplikace je potřeba mít nainstalován tento software:

- Java JDK – Java Development Kit, nutný pro vyvíjení aplikací v programovacím jazyce Java. Jakákoliv verze z roku 2015 či pozdější.
- NetBans IDE – vývojové prostředí NetBeans. Potřebné kvůli grafickému uživatelskému rozhraní, které v něm bylo vytvořeno. Obvykle bývá součástí balíčku JDK. Je potřeba verze 8 nebo vyšší.

V době vzniku práce se oba produkty dají stáhnout na adrese
http://www.oracle.com/technetwork/articles/javase/jdk-netbeans-jsp-142931.html

B.2 Instalace a spuštění

Na CD je odevzdán celý projekt pro vývojové prostředí NetBeans. Stačí vývojové prostředí spustit a v něm otevřít již existující projekt, který je třeba načíst z CD. Doporučuje se ale zkopírovat jej z CD na místní disk.

Projekt je nakonfigurován tak, aby byl přeložitelný ihned po načtění. Ihned po kompilaci se i spustí.