VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PROHLÍŽENÍ A POROVNÁVÁNÍ STRUKTUROVANÝCH SOUBORŮ PRO ALTAP SALAMANDER

VIEWING AND COMPARING OF STRUCTURED FILES FOR ALTAP SALAMANDER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN MORES**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20757/2017/xmores00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Mores Martin**
Obor: Informační technologie
Téma: **Prohlížení a porovnávání strukturovaných souborů pro Altap Salamander**
Viewing and Comparing of Structured Files for Altap Salamander
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s možnostmi vývoje pro správce souborů Altap Salamander. Dále nastudujte techniky porovnávání strukturovaných souborů (např. XML, JSON) a zobrazení výsledků porovnání.
2. Dle konzultací s vedoucím navrhněte modul pro Altap Salamander pro prohlížení vybraného formátu (např. XML, JSON, apod.) a pro porovnávání dvojic těchto souborů a zobrazení výsledků porovnání.
3. Návrh implementujte a testujte.
4. Realizaci zhodnoťte a navrhněte budoucí rozvoj a vylepšení.

Literatura:

- Developing for Altap Salamander File Manager. *Altap Salamander* [online]. ALTAP, 2017 [cit. 2017-10-22]. Dostupné z: https://www.altap.cz/developers/
- Luuk Peters: *Change Detection in XML Trees: a Survey*. In: 3rd Twente Student Conference on IT, University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science, 2005.
- dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Křivka Zbyněk, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Témou tejto práce je zobrazovanie a porovnávanie štruktúrovaných súborov v programe Altap Salamander. Cieľom bolo vytvoriť zásuvný modul, ktorý by umožňoval zobrazovať a porovnávať formáty XML a JSON. Práca popisuje motiváciu stojacu za vytvorením zásuvného modulu, požiadavky na jeho funkcionalitu, proces návrhu modulu, jeho implementácie a následného testovania. Podstatnou súčasťou práce je tiež teoretický základ týkajúci sa algoritmu na porovnávanie XML súborov.

Abstract

The topic of this thesis is to display and compare structured files in Altap Salamander application. The aim is to create a plug-in capable of viewing and comparing XML and JSON file formats. This thesis explains the motivation behind creating the plug-in, its functional requirements, the process of designing the plug-in, its implementation and subsequent testing. An important part of this thesis is also the theoretical foundation of the algorithm used to compare XML files.

Kľúčové slová

JSON, XML, porovnávanie štruktúrovaných súborov, zásuvný modul, C++, Altap Salamander, Windows API, X-Diff

Keywords

JSON, XML, structured files comparison, plug-in, C++, Altap Salamander, Windows API, X-Diff

Citácia

MORES, Martin. *Prohlížení a porovnávání strukturovaných souborů pro Altap Salamander*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Prohlížení a porovnávání strukturovaných souborů pro Altap Salamander

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Zbyňka Křivku, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Mores
14. mája 2018

Poďakovanie

Ďakujem pánovi Ing. Zbyňkovi Křivkovi, Ph.D. za poskytnutú pomoc pri tvorení a testovaní práce.

Obsah

Kapitola 1

Úvod

Za posledných niekoľko dekád sa dôležitosť informačných technológií v životoch ľudí neustále zvyšovala. Zároveň s dôležitosťou sa zvyšuje aj zložitosť informačných systémov a vznikajú programy, ktoré nie sú určené na pomoc s určitou činnosťou, ale priamo na zjednodušenie obsluhy týchto systémov. Medzi tieto typy programov patria aj správcovia súborov. Prví správcovia systémov umožňovali iba jednoduchú prácu so súbormi – prehliadanie adresárov, kopírovanie, presúvanie súborov. S postupom času vznikajú pokročilejší správcovia súborov, ktorí okrem jednoduchej práce so súbormi pridávajú aj ďalšiu funkcionalitu, napríklad možnosť zobraziť základné formáty súborov priamo zo správcu, pracovať so vzdialenými súborovými systémami, atď. Okrem toho mnoho z nich adoptuje filozofiu modularity a umožňuje aj tretím stranám vytvárať rozšírenia funkcionality, ktoré môžu byť pridané do základného programu. Práve táto vlastnosť dnešných správcov súborov je využitá v tejto práci.

Cieľom práce je rozšírenie funkcionality správcu súborov Altap Salamander o možnosť zobrazovať a porovnávať štruktúrované súbory typu XML a JSON. Zásuvný modul (plugin) vytvorený v rámci tejto práce doplňuje už existujúcu funkcionalitu porovnávania textových súborov poskytovanú iným pluginom, berie však ohľad na špecifické požiadavky vznikajúce pri porovnávaní štruktúrovaných súborov, ako je napríklad nezávislosť porovnávania na poradí elementov. Jedným z hlavných motivačných faktorov, stojacich za vznikom tejto práce, bola snaha umožniť prístup k porovnávaniu štruktúrovaných súborov priamo z programu Altap Salamander, bez nutnosti inštalácie a spúšťania externých nástrojov.

Kapitola 2 je úvodnou kapitolou, ktorá rozoberá históriu a dôvody vzniku správcov súborov, popisuje formát XML a JSON a tiež algoritmus vybratý na porovnávanie XML súborov. V kapitole 3 je vysvetlená motivácia, ktorá stojí za týmto projektom a požiadavky, ktoré sú na projekt kladené. Kapitola 4 popisuje návrh implementácie z niekoľkých hľadísk. Kapitola 5 obsahuje popis nástrojov, ktoré sú využité pri vytváraní tohto projektu. Popísané nástroje sú primárne Altap Salamander SDK, knižnice na spracovanie súborov formátu JSON a XML a Visual Studio 2015. Kapitola 6 sa už venuje samotnej implementácii pluginu, sú v nej detailnejšie popísané podstatné časti implementácie. Kapitola 7 popisuje spôsob testovania pluginu vytvoreného v tejto práci. Kapitola 8 je záverečná kapitola popisujúca výsledky práce a jej možné rozšírenia.

Kapitola 2

Úvod do tématiky

Táto kapitola popisuje v krátkosti históriu a dôvod vzniku správcov súborov. Venuje sa správcovi súborov Altap Salamander, ktorého funkcionalitu rozširuje táto práca. Tiež popisuje formáty dát podstatné pre túto prácu (XML, JSON). Na záver rozoberá algoritmus pre porovnávanie XML súborov.

2.1 Správcovia súborov

Význam použitia správcov súborov súvisí s jedným zo základných konceptov súčasných operačných systémov – s konceptom súborového systému.

Na začiatku počítače, takzvané strediskové počítače, fungovali bez akéhokoľvek operačného systému. Vždy sa vykonávala iba jedna úloha, ktorú vždy bolo treba ručne načítať zo série diernych štítkov.

S postupom času sa táto operácia automatizuje tak, že najprv sa v dávke načíta viacero úloh na magnetickú pásku. Na počítači sa spúšťa rutina, ktorá po vložení pásky načíta a spustí prvú úlohu, ktorá potom svoj výstup zapisuje na inú magnetickú pásku a po jej dokončení sa automaticky načítava z pásky ďalšia úloha [22]. Tieto rutiny sú predchodcami dnešných operačných systémov.

Dáta ukladané na magnetických páskach a neskôr na magnetických diskoch je však potrebné nejakým spôsobom organizovať. Vznikajú pomenované úseky dát, takzvané súbory. Pri väčšom počte je však nutné organizovať aj samotné súbory, a na tento účel vznikajú súborové systémy.

Prvé súborové systémy boli jedno-úrovňové, teda všetky súbory sa nachádzali v jednom adresári. Akonáhle sa začali zdieľať počítače, bolo nutné oddeliť dáta rozdielnych užívateľov. Jedným z prvých operačných systémov, ktorého súborový systém podporoval hierarchickú štruktúru bol MULTICS – tento udržiaval dáta rôznych užívateľov v rôznych adresároch a viedol si záznamy o tom, kto môže kam pristupovať [11]. S postupom času vzniká mnoho ďalších pokročilejších hierarchických súborových systémov, napríklad: FAT, NTFS, ext3, ext4. Súborové systémy využívajú čoraz pokročilejšie koncepty, ktoré zvyšujú prístupnosť, bezpečnosť dát a ich odolnosť proti poruchám médií.

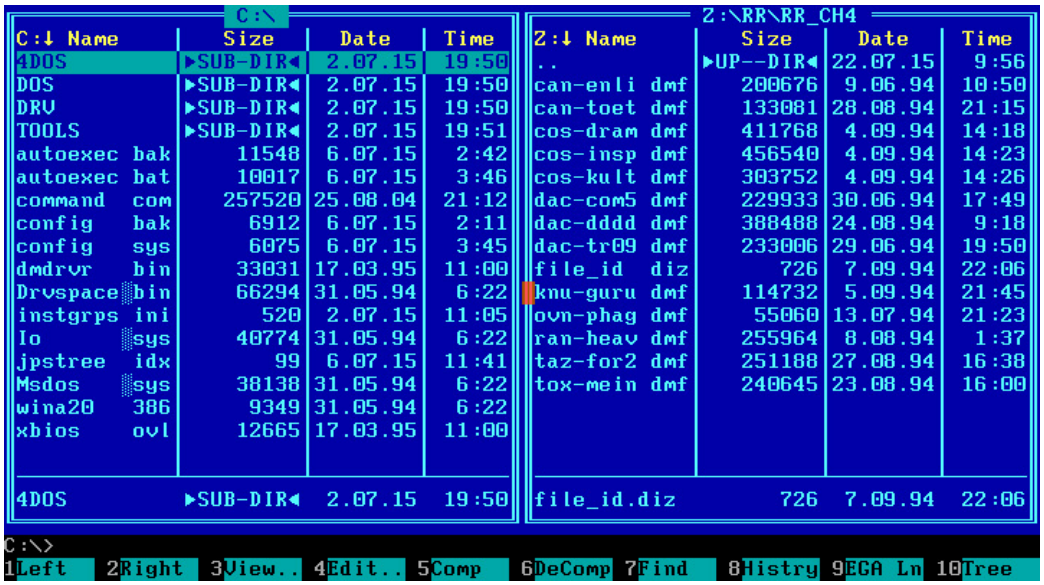
Okrem toho, akým spôsobom sa skladujú a zabezpečujú dáta, je však nutné uvažovať nad tým, ako s týmito dátami môže manipulovať užívateľ z operačného systému. Operačné systémy môžu byť textovo orientované alebo graficky orientované a na tomto potom bude závislý aj spôsob práce so súbormi. Pre grafické systémy to bude manipuláciou grafických prvkov, zatiaľ čo pre textové systémy to bude vykonávaním príkazov v príkazovom riadku.

Jeden z prvých systémov s grafickým užívateľským rozhraním vzniká už v roku 1973 [23]. Avšak tieto systémy sa príliš nerozšírili a viac sa rozširujú počítače založené na platforme IBM PC s operačným systémom MS-DOS, ktorý bol textovo orientovaný.

S rozšírením používania medzi širokú verejnosť sa ukazuje manipulácia so súbormi pomocou príkazov v príkazovom riadku ako nedostačujúca a preto začínajú vznikať dedikované programy určené na prezeranie a manipuláciu súborov.

Najstaršie z týchto programov umožňovali iba prezerať súbory na disku a radiť ich podľa ich atribútov. Jeden z prvých správcov súborov bol Dired, ktorý funguje ako špeciálny mód editora Emacs a umožňuje prezeranie súborov v priečinku a prácu s nimi pomocou špeciálnych príkazov [14].

Ďalšia generácia sú tzv. ortodoxní správcovia súborov. Jedným z prvých správcov tohto typu bol PathMinder vydaný v roku 1984. V roku 1986 bol vydaný program Norton Commander (pozri obrázok 2.1), jeden z prvých a najznámejších správcov súborov vyznačujúci sa dvomi panelmi. Tieto programy mali kvázi grafické rozhranie, keďže bežali v textovom režime a toto rozhranie tvorili vypisovaním textu.



Obr. 2.1: Užívateľské rozhranie programu Norton Commander [15]

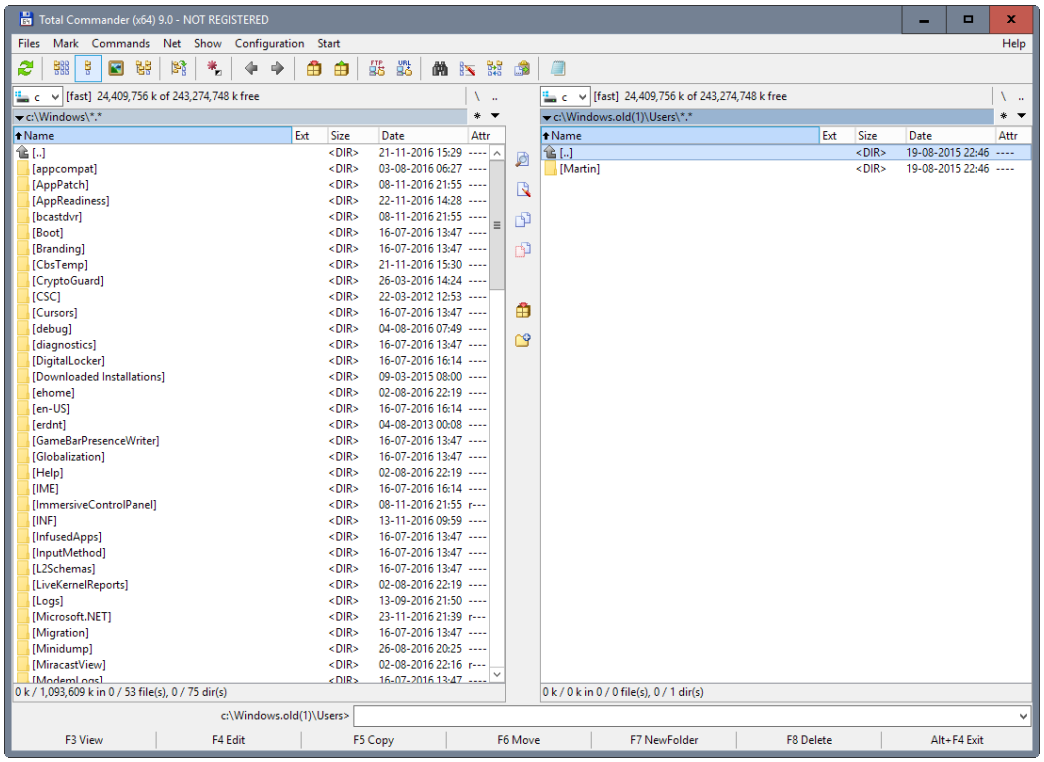
Ortodoxní správcovia súborov zvyčajne pozostávajú z troch častí: príkazový riadok a dva panely na zobrazovanie súborov. Príkazový riadok poskytuje možnosť vykonávať príkazy v príkazovom riadku v danom adresári.

Existencia dvoch panelov poskytuje užívateľovi niekoľko výhod, ktoré predošlí správcovia súborov nemali. Umožňuje simultánne zobrazovať dve lokácie v súborovom systéme naraz a teda užívateľ je schopný pracovať so súbormi na dvoch miestach bez zmeny adresára. Tiež to umožňuje zjednodušenie vykonávania operácií, ktoré typicky vyžadujú ako vstup dve miesta v súborovom systéme, ako napríklad kopírovanie súborov, presúvanie súborov. Vždy je označený iba jeden z týchto panelov a pri operáciách typu kopírovanie súborov sa označený panel chová ako zdroj a neoznačený ako cieľ operácie.

V prípade správcu Norton Commander bolo v druhom paneli možné zobrazovať informácie o prvom paneli alebo druhý adresár na disku, zatiaľ čo v dnešných správcoch súborov je typicky možné zobrazovať iba miesto v súborovom systéme v oboch paneloch.

Títo správcovia súborov majú niekoľko čŕt, ktoré sú pre nich typické, napríklad poskytujú veľké množstvo klávesových skratiek na zjednodušenie práce užívateľa, poskytujú možnosť navigácie v súborovom systéme a vykonávania operácií so súbormi bez použitia myši a väčšinou majú vstavané prehliadače súborov aspoň pre základné formáty. Jednou z častých čŕt je to, že správca súborov poskytuje možnosť užívateľovi rozšíriť funkcionalitu pomocou pluginov. Ďalšou z častých čŕt, typicky pri správcoch s grafickým rozhraním, je podpora záložiek. Tieto sa chovajú podobne ako v internetovom prehliadači a typicky sú pre každý panel osobitne. Týmto získavame možnosť mať naraz otvorených ešte viac než dve lokácie v súborovom systéme.

Niekoľko súčasných príkladov ortodoxných správcov súborov sú Total Commander, Double Commander, Altap Salamander. Na obrázku 2.2 možno vidieť užívateľské rozhranie správcu súborov Total Commander.



Obr. 2.2: Užívateľské rozhranie programu Total Commander [17]

Keďže súčasní počítačoví užívatelia častokrát nepotrebujú všetky možnosti poskytované ortodoxnými správcami súborov, a často sú pre nich títo správcovia zbytočne zložití, rozšírenosť ortodoxných správcov súborov klesá a bežní užívatelia začínajú používať iné možnosti správy súborov. Príkladom je napríklad File Explorer vo Windows, ktorý poskytuje iba jedno okno so súbormi. Tieto sú typicky súčasťou operačného systému a teda nie je potreba ich inštalovať a okrem toho poskytujú navigačné možnosti, ktoré bežnému užívateľovi uľahčia prácu (rýchly prístup k niektorým zložkám, prehľad často používaných zložiek...).

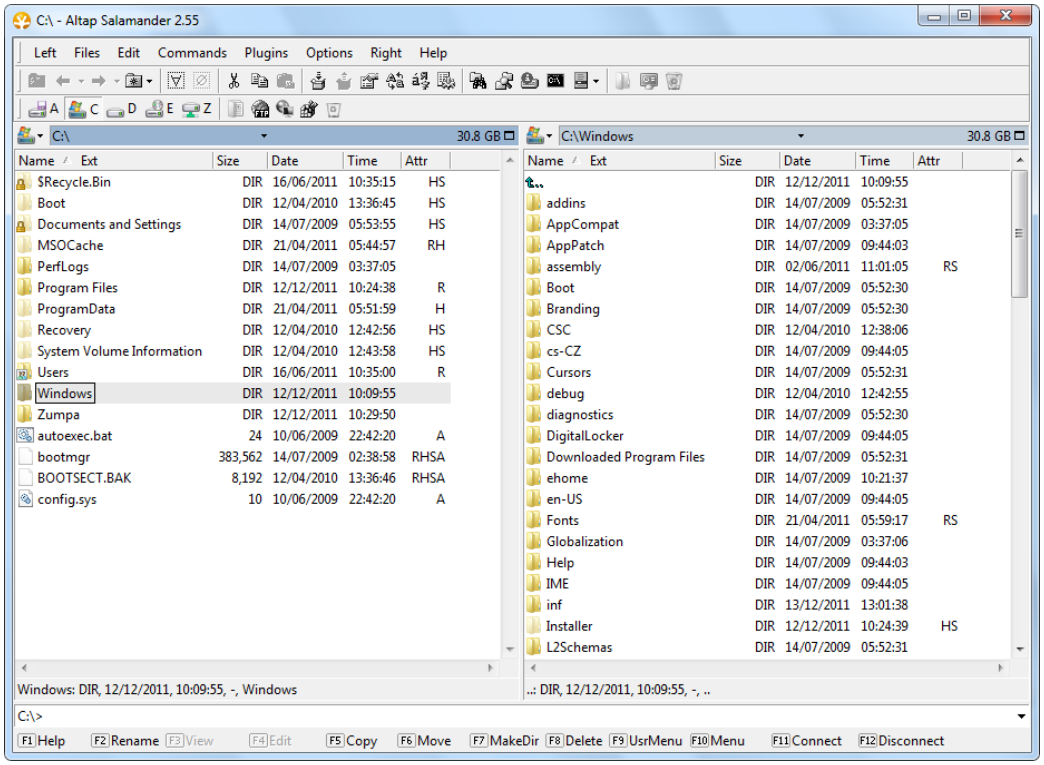
V dnešnej dobe používajú ortodoxných správcov súborov skôr pokročilejší užívatelia (správcovia systémov, programátori), ktorí vedia do väčšej miery využiť funkcionalitu, ktorú títo poskytujú.

2.2 Altap Salamander

Altap Salamander patrí medzi ortodoxných správcov systémov s dvomi panelmi a príkazovým riadkom, viď obrázok 2.3. Vyvíja ho česká firma ALTAP a pôvodne bol distribuovaný ako freeware zadarmo až do verzie 2.0, pričom posledná verzia zadarmo (1.52) je stále k dispozícii na stiahnutie na oficiálnej stránke. Verzia 2.0 bola v roku 2001 vydaná už ako shareware [4].

Základné vlastnosti a schopnosti programu Altap Salamander [2]:

- má vstavané prehliadače dát pre množstvo formátov,
- rozšíriteľný pomocou vlastných pluginov,
- veľké množstvo klávesových skratiek,
- možnosť pracovať s archívmi,
- schopnosť pripojiť sa na vzdialené súborové systémy pomocou protokolov FTP, SFTP.



Obr. 2.3: Hlavné okno správcu súborov Altap Salamander [3]

Jednou z nevýhod programu Altap Salamander je to, že nepodporuje unicode [20]. To znamená, že Altap Salamander nevie korektne pracovať so súbormi, ktorých mená obsahujú

znaky, ktoré sa nenachádzajú v znakovej sade nastavenej v systéme. Napríklad teda Altap Salamander na českom Windows systéme nebude vedieť pracovať so súbormi, ktorých mená obsahujú čínske znaky. Okrem toho, na rozdiel od napríklad Total Commandera, nepodporuje záložky.

V tejto práci je využitá možnosť rozšíriteľnosti funkcionality programu Altap Salamander pomocou vlastných pluginov používajúcich Altap Salamander SDK. Bol navrhnutý a implementovaný plugin pre porovnávanie a zobrazovanie štruktúrovaných formátov (XML, JSON).

2.3 XML formát

XML (Extensible Markup Language) je dátový formát určený na ukladanie (reprezentáciu) štruktúrovaných dokumentov, ktorý vznikol v neskorých 90-tych rokoch. V roku 1986 bol SGML stanovený ako štandard pre vytváranie značkovacích jazykov [16]. Neskôr v roku 1990 fyzik Tim Berners-Lee špecifikuje jazyk HTML, ktorý považuje za aplikáciu SGML [6].

S postupom času sa čoraz viac prejavuje potreba univerzálneho formátu, ktorý by umožňoval prenos dát cez internet. HTML je však jazyk určený na prezentačné účely a nie je vhodný na prenos dát, zatiaľ čo SGML je zbytočne zložité a v niekoľkých oblastiach problematické pri prenose cez internet, napríklad nemá dostatočnú podporu v prehliadačoch [16]. Preto v rokoch 1996-1998 bol vytvorený jazyk XML, ktorý je kompatibilný so štandardom SGML [19] [8].

Jedným z cieľov a výhod jazyka XML je to, že je čitateľný aj človekom aj strojom, teda je pre užívateľa pomerne jednoduché vytvoriť XML dokument napríklad aj v obyčajnom textovom editore [19].

Nedostatkom na druhú stranu je jeho prehnaná výrečnosť, keďže XML dokument obsahuje množstvo formátovacích informácií. Toto však ani nie je cieľom jazyka XML, keďže štandard hovorí, že stručnosť má pri vývoji minimálnu prioritu [19].

V súčasnej dobe je formát XML jedným z najpoužívanejších dátových formátov. Slúži ako základ veľkého množstva ďalších formátov, napríklad SVG, XHTML, Atom, atď.

Má niekoľko využití, medzi ktorými je napríklad skladovanie dát. Používať sa môže napríklad na skladovanie dát, ktoré často menia svoju štruktúru, alebo pre skladovanie prirodzene hierarchických dát. Jeden z príkladov, kde sa XML v praxi používa na skladovanie dát sú konfiguračné súbory aplikačných serverov.

Okrem toho v oblasti systémovej integrácie, kde sa prenášajú dáta v určitom formáte medzi dvomi systémami, formát XML takmer úplne nahradil v minulosti používané technológie používajúce binárne formáty, napríklad pred tým populárnu technológiu CORBA.

Prácu s XML uľahčujú a obohacujú aj nasledujúce technológie a špecifikácie:

- XSD – umožňuje programátorovi definovať aký formát a špecifikáciu by mali mať XML elementy v danom dokumente. Na základe tohto je potom možné validovať XML dokument oproti tejto schéme,
- XSLT – deklaratívny turingovsky úplný jazyk používaný na transformáciu XML dokumentov. Vstupom je typicky jeden alebo viac XML súborov a XSLT stylesheet, na základe ktorého sa vyprodukujú nové súbory,

- XPath – umožňuje vyhľadávanie v stromovej reprezentácii daného XML dokumentu. Je to dotazovací jazyk, ktorý okrem výberu informácií z dokumentu na základe dotazu umožňuje aj určité základné operácie nad jednoduchými dátovými typmi.

2.4 JSON formát

Podľa definície špecifikácie [10] je JSON (JavaScript Object Notation) dátový formát určený pre výmenu dát, implementovaný minimalistickým spôsobom - neobsahuje takmer žiadne formátovacie informácie. Pozostáva zo skupín dvojíc meno a hodnota a zoradených zoznamov prvkov. Na rozdiel od formátu XML, ktorý nerozlišuje typy uložených dát, špecifikácia definuje pre formát JSON niekoľko dátových typov: reťazec, číslo, objekt, pole a boolean.

JSON formát bol vytvorený, alebo ako hovorí jeho autor Douglas Crockford objavený [9], v roku 2001. Tiež hovorí, že nebol prvým, ktorý našiel tento koncept. Je však tým, kto sa zaslúžil o jeho pomenovanie, špecifikáciu a rozšírenie popularity. Ako napovedá jeho názov (pozri vyššie), JSON je odvodený od jazyka JavaScript, avšak sám o sebe je to jazykovo nezávislý formát dát [10].

JSON má niekoľko využití, napríklad serializácia objektov, či už za účelom ich zachovania po ukončení behu systému alebo ich prenosu po sieti. V dnešnej dobe sa čoraz viac používa na prenos dát medzi serverom a prehliadačom. Napríklad odpoveď na dotaz v rámci protokolu REST častokrát vracia dáta vo formáte JSON.

JSON má pri prenose dát vo webových službách oproti XML niekoľko výhod. Jednou z nich je, že nemá takú veľkú réžiu ako XML. Ďalšou výhodou je jeho jednoduchšie spracovanie, keďže pre väčšinu jazykov existuje mapovanie na natívne dátové štruktúry. Napríklad v mobilných internetových prehliadačoch môže spracovávanie XML súborov spôsobiť problémy s nedostatočným výkonom, zatiaľ čo spracovanie JSON je menej náročné na zdroje.

V dnešnej dobe sa poskytovatelia web služieb presúvajú od technológií používajúcich XML a čoraz viac používajú technológie, ktoré pracujú s JSON formátmi. Napríklad pri prenosoch určených na prezentačné účely na webe alebo jednoduchšej komunikácii so serverom je vhodnejšie použiť JSON, keďže má ako jazyk jednoduchšiu štruktúru a produkuje dáta menšieho rozsahu ako XML.

V určitých prípadoch je však stále výhodnejšie používať XML formát. Zoberme si napríklad systémovú integráciu, kedy dáta prenášame z jedného systému do druhého. Častokrát platí, že prijímateľ dát si potrebuje overiť ich validitu, čo XML umožňuje ale JSON nie. Tiež systémový integrátor využíva možnosť transformácie pomocou XSLT, keďže dáta v integrovaných systémoch majú typicky rozdielnu štruktúru a je nutné pri prenose dát ich transformovať zo zdrojovej štruktúry do cieľovej.

2.5 XML porovnávanie

XML súbory reprezentujú stromové dátové štruktúry a teda ich porovnávania je problémom z oblasti porovnávania grafov. Porovnávanie (detekciu zmien) stromových štruktúr je možné rozdeliť do dvoch kategórií: porovnávanie zoradených stromov a nezoradených stromov. V nezoradených stromoch na rozdiel od zoradených stromov nezáleží na horizontálnych vzťahoch medzi súrodencami (ich poradí z ľava do prava).

Pre porovnávanie XML súborov existujú algoritmy obidvoch kategórií, pre zoradené stromy je to napríklad XyDiff [7] a X-Tree Diff+ [18], zatiaľ čo pre nezoradené stromy je to X-Diff.

Pri výbere algoritmu pre túto prácu boli zvažované dva faktory: rýchlosť a kvalita výsledku. Pre rozhodnutie o tom, na ktorý z týchto faktorov bude kladený väčší dôraz bola použitá nasledujúca úvaha:

- práca je vyvíjaná ako plugin, ktorý je súčasťou programu Altap Salamander a má grafické užívateľské rozhranie – teda výsledky porovnania budú prehliadané graficky človekom,
- keďže plugin bude manuálne používaný užívateľom, je predpoklad, že sa budú primárne porovnávať súbory menšieho rozsahu,
- zároveň však užívateľ bude chcieť mať čo najspoľahlivejšie a najpresnejšie porovnanie.

Teda záleží menej na rýchlosti algoritmu a viac na presnosti jeho výsledkov. Z porovnania algoritmov X-Diff a XyDiff [24] s ohľadom na tieto faktory vyplýva, že bude vhodnejšie použiť algoritmus X-Diff.

2.5.1 X-Diff

Nasledujúci text je voľným popisom a zhrnutím informácií z článku [24], ktorý definuje špecifikáciu tohto algoritmu.

V algoritme X-Diff sa v rámci štruktúry XML stromu uvažuje o troch typoch uzlov:

- element (nelistový, má iba meno),
- atribút (listový uzol, má meno a hodnotu),
- textový uzol (taktiež listový uzol, má iba hodnotu).

Pri práci s uzlami sa používa koncept, ktorý je nazvaný signatúra na určenie toho, ktoré uzly je vhodné porovnávať. Táto signatúra určuje kontext uzlu v strome. Na jeho určenie sa použije zoznam predkov uzlu, od neho samotného až ku koreňu stromu, spolu s určením jeho typu.

Algoritmus je určený na porovnávanie stromov a teda je nutné vedieť, kedy sú dva stromy zhodné (ekvivalentné) – dva stromy sú ekvivalentné, pokiaľ je ich štruktúra zhodná s výnimkou poradia súrodeneckých prvkov.

Rozdiel medzi dvomi stromami sa hľadá tak, že sa hľadá taká transformácia prvého stromu, aby bol ekvivalentný s druhým stromom. Transformácia pozostáva z operácií, pričom tieto operácie vlastne reprezentujú rozdiely daných stromov, pretože po ich vykonaní sú dané stromy ekvivalentné.

Pri porovnávaní stromových štruktúr však niekedy nie je jasné, ktorá časť stromu 1 pri porovnávaní zodpovedá ktorej časti stromu 2, pozri príklad 1.

Element parkovisko má v oboch XML stromoch dve deti s menom auto. Nie je ale jasné, ktoré auto zodpovedá ktorému, keďže v XML nezáleží na poradí elementov.

Ak by sme priradili prvé auto v strome 1 k prvému autu v strome 2 a druhé auto k druhému, dostaneme nejakú transformáciu popisujúcu rozdiel stromov. Zatiaľ čo ak by sme priradili prvé auto k druhému a naopak, dostaneme inú transformáciu. Teda je možné zostaviť niekoľko rôznych možných transformácií, pričom niektoré budú obsahovať viac operácií a niektoré menej, ale všetky tieto transformácie dosiahnu ten istý výsledok: po ich vykonaní nad stromom 1 bude tento strom ekvivalentný so stromom 2.

Príklad 1 Možná dvojica porovnávaných súborov.

```
<parkovisko>
    <auto>
        <značka>Ford</značka>
        <model>Mustang</model>
        <farba>biela</farba>
        <rok>2018</rok>
    </auto>
    <auto>
        <značka>Volkswagen</značka>
        <model>Touareg</model>
        <farba>modrá</farba>
        <rok>2015</rok>
    </auto>
</parkovisko>
```

```
<parkovisko>
    <auto>
        <značka>Volkswagen</značka>
        <model>Touareg</model>
        <farba>čierna</farba>
        <rok>2005</rok>
    </auto>
    <auto>
        <značka>Ford</značka>
        <model>Mustang</model>
        <farba>červená</farba>
        <rok>2010</rok>
    </auto>
</parkovisko>
```

O čo sa teda snažíme pri porovnávaní XML súborov je nájsť taký prípad transformácie, kedy priradíme k sebe uzly, ktoré sa čo najviac podobajú a tým pádom výsledná transformácia bude obsahovať čo najmenej zmien (operácií). V našom príklade teda budeme brať autá s rovnakou značkou a modelom ako navzájom zodpovedajúce na porovnanie, pretože v tom prípade sa mení iba farba a rok výroby.

Okrem toho je ešte nutné zvážiť, akú cenu majú vykonávané operácie. V tomto algoritme sú definované nasledujúce operácie, pričom všetky majú rovnakú váhu:

- vkladanie uzlu ako dieťa iného uzlu,
- vymazanie uzlu,
- aktualizácia hodnoty uzlu.

Okrem toho sú definované dve zložené operácie, ktoré pracujú so stromami a ich váha zodpovedá súčtu váh operácií, z ktorých sa skladajú:

- vloženie stromu ako podstrom nejakého uzlu,
- vymazanie podstromu s koreňom v určitom uzle.

Samotná implementácia algoritmu pozostáva z niekoľkých častí. Najprv sa v prvej časti spracujú vstupné dáta a vytvoria stromy reprezentujúce XML súbory. Pre každý uzol v strome je vygenerovaná hash hodnota, ktorá reprezentuje daný uzol a celý podstrom v ňom zakorenený. Umožňuje rýchlo spárovať dvojice úplne zhodných uzlov a tak ich vylúčiť z procesu porovnávania v ďalších krokoch.

Druhá časť algoritmu je rozdelená do niekoľkých krokov:

Krok 1: Najprv zostavíme množinu všetkých listových uzlov v oboch porovnávaných stromoch. Táto množina je redukovaná o všetky uzly, ktorých rodičov je možné spárovať na základe zhody ich hash hodnôt.

Krok 2: Následne iterujeme cez všetky možné dvojice uzlov zo zostavených množín, teda (x, y), kde x patrí prvému stromu a y druhému stromu. Pre každú dvojicu, kde uzly x a y majú zhodnú signatúru vypočítame vzdialenosť medzi týmito dvomi uzlami a uložíme

ju do tabuľky. Následne pomocou minimum cost maximum flow algoritmu zistíme, akým najlepším spôsobom môžeme spárovať tieto listové uzly – nájdeme párovanie listových uzlov tak, že výsledná cena transformácie na tejto úrovni je čo najnižšia. Týmto získavame cenu transformácie pre rodičovské uzly týchto detských uzlov.

Krok 3: Vytvoríme novú množinu, do ktorej zaradíme všetky rodičovské uzly uzlov, ktoré sme spracovávali v predošlom kroku a opakujeme nad ňou krok 2. Takto pokračujeme, až kým nedostaneme prázdnu množinu (teda sme dosiahli najvyššiu úroveň porovnávaných stromov).

Po skončení kroku 3 máme kompletnú informáciu o optimálnom vzájomnom priradení medzi uzlami porovnávaných stromov.

V poslednej časti sa generuje editovací skript – je to zoznam operácií potrebných na vykonanie transformácie.

2.5.2 Zložitosť algoritmu

Pri výpočte zložitosti algoritmu budeme postupne uvažovať o troch častiach algoritmu. Prvou je spracovávanie súborov, druhá je nachádzanie zodpovedajúcich si uzlov a tretia je vytváranie editovacieho skriptu.

Prvá časť zahŕňa aj vytváranie hash hodnôt pre jednotlivé uzly, pričom pred ich vytvorením je nutné uzly zoradiť, preto zložitosť tejto časti je ohraničená výrazom 2.1 [24]:

$$O(|T_1| * \log(|T_1|) + |T_2| * \log(|T_2|)) \tag{2.1}$$

Notácia $|T|$ vyjadruje počet uzlov v strome T.

Druhá časť sa týka hľadania zodpovedajúcich si uzlov. Zložitosť výpočtu vzdialenosti dvoch listových uzlov je určená ako O(1) a teda zložitosť výpočtu vzdialeností všetkých možných dvojíc (a, b), kde $a \in T_1, b \in T_2$ a platí Signature(a) == Signature(b) je ohraničená výrazom 2.2 [24]:

$$O(|T_1| * |T_2|) \tag{2.2}$$

Pre dvojice nelistových uzlov (x, y), kde $x \in T_1, y \in T_2$ a platí Signature(x) == Signature(y) zložitosť ohraničuje výraz 2.3 [24]:

$$O(\deg(x) * \deg(y) * \max(\deg(x), \deg(y)) * \log_2(\max(\deg(x), \deg(y)))) \tag{2.3}$$

kde $\deg(x)$ určuje počet detských uzlov elementu x. Zložitosť výpočtu vzdialenosti všetkých dvojíc nelistových uzlov so zhodnou signatúrou bude ohraničená výrazom 2.4 [24]:

$$\sum_{k=1}^{M} \sum_{i=1}^{N_{1k}} \sum_{j=1}^{N_{2k}} O(\deg(x_{ki}) * \deg(y)_{kj} * \max(\deg(x)_{ki}, \deg(y)_{kj}) * \log_2(\max(\deg(x)_{ki}, \deg(y)_{kj}))) \tag{2.4}$$

kde M vyjadruje celkový počet takých signatúr, že v strome ich má viac než jeden uzol. N_{1k} vyjadruje počet uzlov v strome 1, ktoré majú konkrétnu signatúru S a N_{2k} vyjadruje počet uzlov v strome 2, ktoré majú signatúru S.

Na základe [24] je možné výraz 2.4 zjednodušiť na výraz 2.5:

$$O(|T_1| * |T_2| * \max(\deg(T_1), \deg(T_2)) * \log_2(\max(\deg(T_1), \deg(T_2)))) \tag{2.5}$$

Notácia $\deg(T)$ vyjadruje počet detí prvku stromu, ktorý má najviac detí.

Výsledná zložitosť kroku 2 teda bude súčet zložitosti výpočtu vzdialeností dvojíc listových uzlov (výraz 2.2) a dvojíc nelistových uzlov (výraz 2.5). Keďže sa však jedná o výpočet

zložitosti, je tento súčet možné zjednodušiť iba na jeho prvok s najväčšou mocnosťou a teda výsledná zložitosť kroku 2 je vyjadrená rovnicou 2.5.

V tretej časti sa už iba generuje editovací skript, a na to je potrebné iterovať cez všetky uzly oboch stromov. Toto nám dáva zložitosť ohraničenú výrazom 2.6 [24]:

$$O(|T_1| + |T_2|) \tag{2.6}$$

Výsledná zložitosť algoritmu bude súčtom zložitostí všetkých troch krokov, ale na základe podobného princípu ako pri sčítavaní zložitostí v kroku 2, bude zložitosť algoritmu reprezentovaná zložitosťou kroku 2 (výraz 2.5).

2.5.3 Optimalizácia algoritmu

V prípade, že by užívateľovi nepostačovala štandardná rýchlosť algoritmu bola autormi algoritmu navrhnutá optimalizácia, ktorá algoritmus zrýchli za cenu malého zníženia presnosti algoritmu pri hľadaní najlepšieho rozdielu. Optimalizovaný algoritmus je označený X-Diff+.

Optimalizácia spočíva v redukcii množiny prehľadávaných uzlov, teda namiesto toho, aby sa vždy hľadala vzdialenosť medzi všetkými uzlami dvoch množín, ako je spomínané vyššie, sa určí hodnota, ktorá slúži ako prah. Pokiaľ je vzdialenosť medzi aktuálne porovnávanými uzlami menšia ako zvolený prah, potom tieto dva uzly sú automaticky priradené ako zodpovedajúce a nie je nutné počítať ich vzdialenosť k ostatným uzlom.

Hodnota prahu na každej úrovni bude dynamicky vypočítaná tak, že na začiatku kroku porovnávania dvoch množín sa zoberie niekoľko náhodných uzlov z množiny 1 a vypočíta sa ich vzdialenosť so všetkými uzlami z druhej množiny. Následne sa pre každý z nich určí na základe vypočítaných hodnôt uzol s najmenšou vzdialenosťou z množiny 2. Potom zoberieme všetky tieto vzdialenosti a ich priemer nám bude slúžiť ako prah.

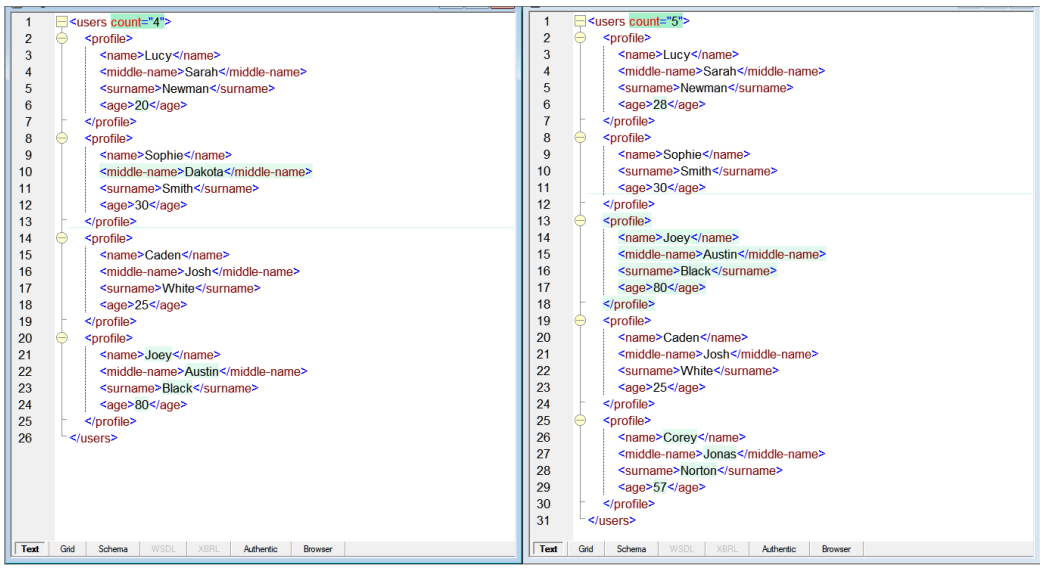
Kapitola 3

Motivácia a požiadavky

Táto kapitola sa venuje motivácii stojacej za vývojom tohto projektu. Definuje špecifikáciu požiadaviek zadávateľa a tiež niektoré požiadavky plynúce zo štúdia podobných nástrojov.

3.1 Motivácia práce

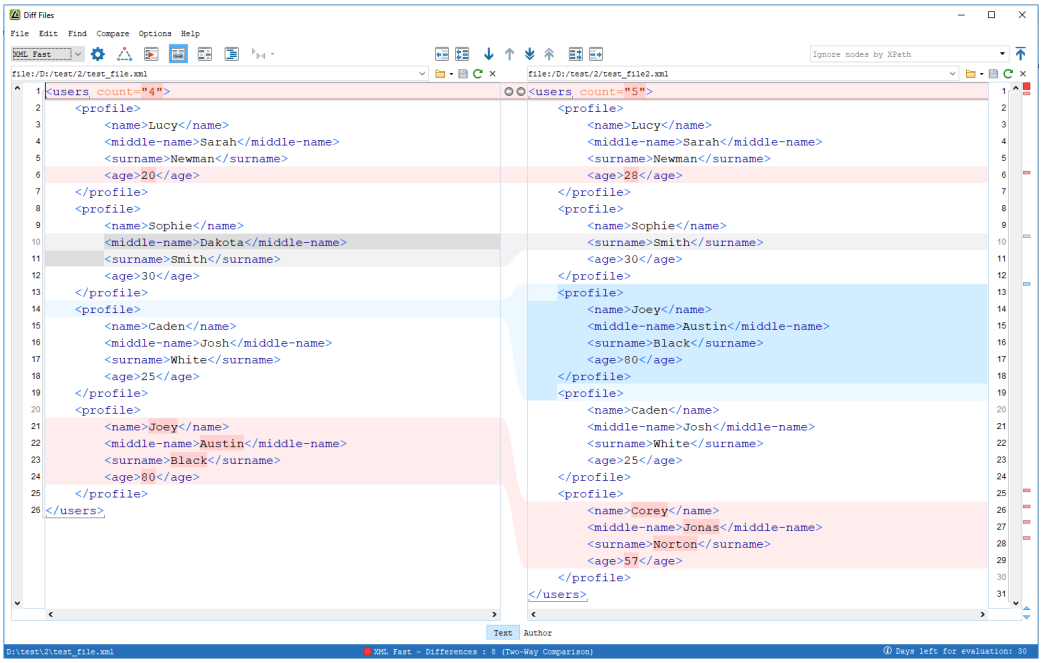
V súčasnosti existuje niekoľko nástrojov, ktoré umožňujú porovnávať štruktúrované súbory formátu XML alebo JSON a častokrát sú sofistikovanejšie a rozsiahlejšie ako je rozsah tejto práce. Dva z nástrojov na porovnávanie XML súborov sú Altova XMLSpy XML Editor [5] a Oxygen XML Editor [21].



Obr. 3.1: Porovnávanie XML súborov pomocou nástroja Altova XMLSpy XML Editor

Obidva tieto nástroje primárne slúžia ako editory XML súborov, ale v rámci pokročilej funkcionality poskytujú aj možnosť porovnávať tieto súbory. Na obrázku 3.1 možeme vidieť príklad porovnania pomocou nástroja XMLSpy, zatiaľ čo na obrázku 3.2 vidíme porovnanie nástrojom Oxygen XML Editor.

Čo odlišuje túto prácu od iných riešení a tiež bolo hlavnou motiváciou vytvorenia tejto práce bola potreba implementovať túto funkcionalitu ako súčasť správcu súborov. Integrácia so správcom súborov poskytuje zjednodušený prístup k aplikácii a tiež šetrenie času, keďže



Obr. 3.2: Porovnávanie XML súborov pomocou nástroja Oxygen XML Editor

nie je nutné súbory buď nahrávať na internet ako v prípade online riešení alebo otvárať samostatnú aplikáciu ako v prípade aplikačných riešení.

Konkrétny správca súborov bol zvolený zadávateľom na základe frekvencie používania a jedná sa o program Altap Salamander.

Súčasťou zadania bolo aj implementovať zobrazovanie týchto súborových formátov, keďže základný integrovaný prehliadač súborov v programe Altap Salamander nepodporuje štruktúrované súbory.

3.2 Požiadavky práce

Požiadavky na prácu môžeme rozdeliť do dvoch kategórií: funkčné požiadavky a požiadavky na grafické rozhranie. Funkčné požiadavky boli primárne definované zadávateľom práce, zatiaľ čo časť grafických požiadaviek vyplynula zo štúdia pluginu File Comparator (viď nižšie).

Porovnávanie štruktúrovaných súborov

V tomto projekte je cieľom rozšíriť funkcionalitu nástroja Altap Salamander o porovnávanie štruktúrovaných súborov, konkrétne súborov typu JSON a XML. Altap Salamander obsahuje vstavanú funkcionalitu porovnávania textových súborov avšak neposkytuje možnosť porovnávať štruktúrované súbory.

Rozdiely medzi textovým formátom a štruktúrovanými formátmi:

- štruktúrované súbory majú stromovú logickú štruktúru,
- majú zameniteľné poradie elementov – teda napríklad dva súbory typu XML, ktoré majú prehodené elementy, sú považované za zhodné,

- v štruktúrovaných súboroch nie je text štruktúrovaný pomocou nových riadkov, ale pomocou značiek, syntaxe.

Rozbaľovanie uzlov

Pri štruktúrovaných súboroch väčšieho rozsahu je žiadúce, aby bolo možné zbaliť a rozbaliť uzly (elementy). Keď je uzol zbalený, tak sa zobrazuje iba jeho prvý riadok. Užívateľ má potom možnosť tento uzol rozbaliť a pozrieť si celý jeho obsah. Toto dodáva súboru na prehľadnosti a zároveň to uľahčuje orientáciu.

Vyznačovanie syntaxe

XML a JSON ako štruktúrované formáty majú určitú syntax, ktorá sa používa v dokumentoch. Jednou z požiadaviek na plugin bolo pri zobrazovaní obsahu týchto súborov vyznačiť túto syntax pre zlepšenie prehľadnosti v súbore.

Grafické informácie

Štúdiom pluginu File Comparator, ktorý je súčasťou programu Altap Salamander a slúži na textové porovnávanie súborov bolo identifikovaných niekoľko grafických prvkov, ktoré by bolo vhodné implementovať do pluginu (navigácia medzi rozdielmi, zobrazovanie mien porovnávaných súborov, číslovanie riadkov).

Kapitola 4

Návrh

Kapitola popisuje návrh a štruktúru pluginu. Je rozdelená na dve časti: užívateľské rozhranie a dátový model.

Tento plugin bol vytvorený upravovaním vzorového pluginu FileViewer ponúkaného v rámci Altap Salamander SDK (došlo k zásadným úpravám, ale kostra pochádza z tohto pluginu). V oblasti grafického aj implementačného návrhu bol častý námet inšpirácie nejaký modul programu Altap Salamander, jednak za účelom určitej konzistencie s tým, na čo je užívateľ zvyknutý pri používaní programu Altap Salamander, a tiež za účelom konzistencie kódu.

4.1 Užívateľské rozhranie

Užívateľské rozhranie je grafickým užívateľským rozhraním, ktoré je založené na oknách. Altap Salamander SDK používa Windows API a za účelom kompatibility a konzistentného kódu sa Windows API používa aj v plugine. Toto môže ovplyvňovať alebo obmedzovať spôsob, akým sú riešené určité potreby a problémy.

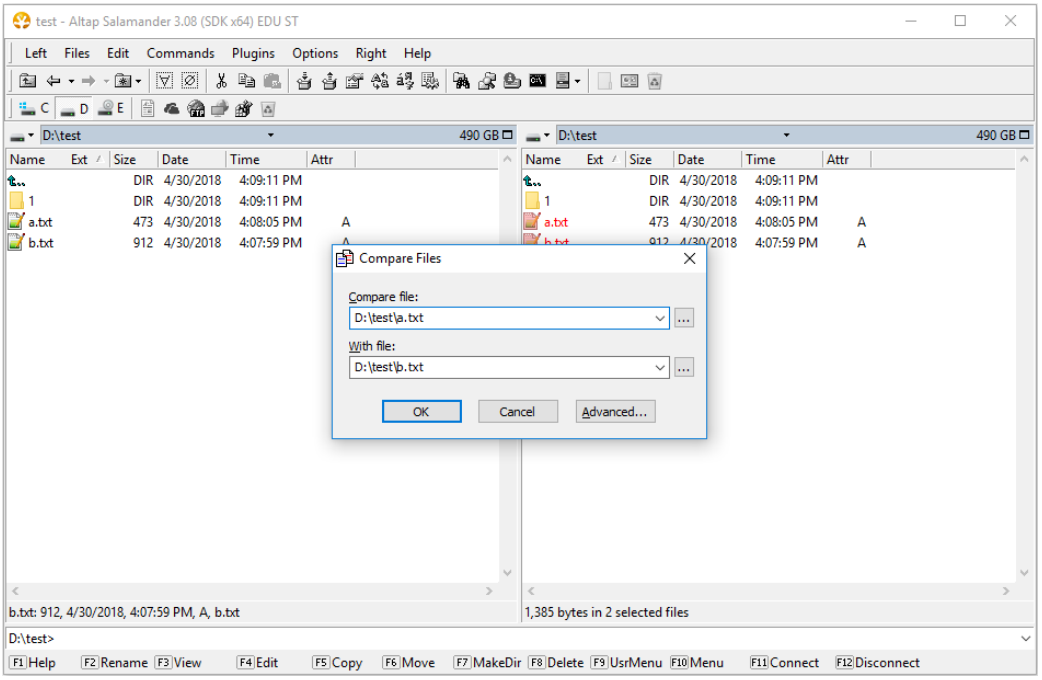
4.1.1 Spúšťanie pluginu a výber súborov

Plugin bude možné spustiť pomocou klávesovej skratky Ctrl+Shift+X. Túto skratku som zvolil, pretože je blízko skratke Ctrl+Shift+C, ktorá je použitá pre plugin File Comparator, ktorý Altap Salamander poskytuje na textové porovnávanie súborov.

Po spustení pluginu sa užívateľovi zobrazí dialóg box, ktorý obsahuje cesty ku porovnávaným súborom. Tento box bude takmer úplne zhodný s boxom, ktorý sa používa v plugine File Comparator (pozri obrázok 4.1) pre zachovanie konzistentnosti užívateľského rozhrania. Jediná funkcionalita, ktorá nebude implementovaná, je história porovnávaných súborov.

Pri otvorení boxu sa na základe toho, aké súbory sú označené alebo vybrané v paneloch programu Altap Salamander inicializujú cesty k porovnávaným súborom. To akým spôsobom sa tieto cesty inicializujú na základe panelov je popísané v kapitole 6.

Po dokončení výberu ciest ku súborom užívateľ stlačí tlačidlo OK a spustí sa porovnávanie a zobrazenie výsledku. V tejto fáze sa kontroluje validita zadaných vstupných súborov, a pokiaľ nastala nejaká chyba, tak program zobrazí dialógové okno s popisom danej chyby a po tom, čo užívateľ klikne OK, znovu sa zobrazí to isté okno s výberom ciest.



Obr. 4.1: Dialógové okno pre výber porovnávaných súborov pluginu File Comparator

4.1.2 Zobrazovanie výsledkov porovnávania

Pokiaľ všetko prebehlo v poriadku, spúšťa sa zobrazovanie výsledkov porovnávania v novom okne, ktoré sa skladá z hlavného okna a troch detských okien. Hlavné okno má ohraničenie a titulnú lištu. Jedno z detských okien slúži na zobrazovanie menu. Ostatné dve slúžia na zobrazovanie porovnávaných súborov, v každom okne jeden.

Okrem týchto prvkov sú ešte ďalšie informácie, ktoré je nutné zobraziť. Jednou z týchto informácií sú mená súborov, ktoré sa zobrazujú v detských oknách. Graficky boli tieto mená umiestnené nad obsah daných súborov, podobne ako v oficiálnom plugine File Comparator od firmy ALTAP.

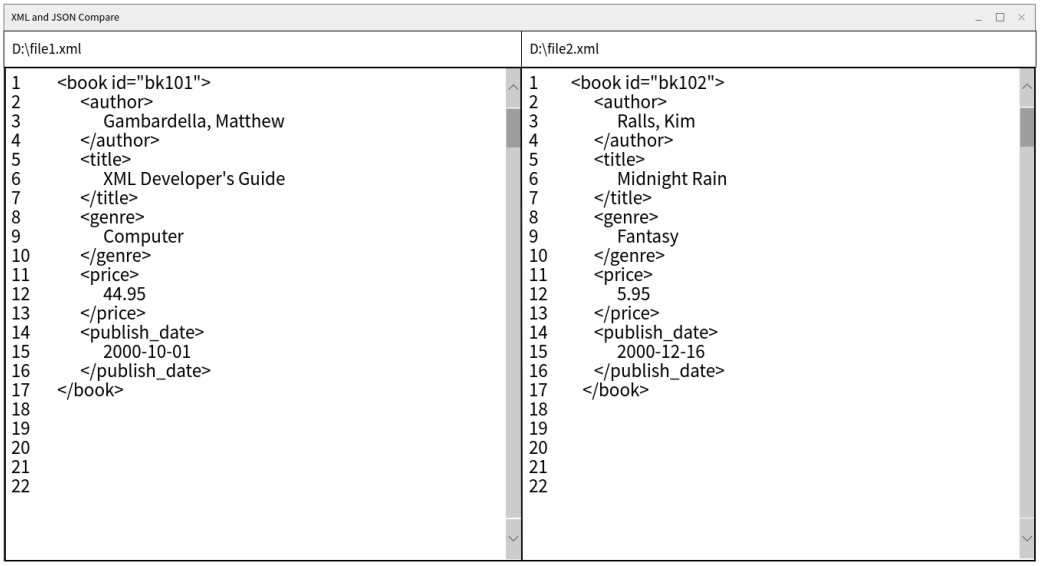
Ďalšia funkcionalita, ktorá je súčasťou návrhu za účelom zlepšenia užívateľského zážitku je možnosť voliť medzi horizontálnym a vertikálnym zobrazením detských okien a možnosť voľby pomeru plochy, ktorú zaberajú.

Grafická možnosť zmeny tohto pomeru bude implementovaná pomocou vynechania miesta medzi oknami porovnávaných súborov. Toto vizuálne vytvára lištu, ktorú je možné ťahať po obrazovke a tým meniť pomer.

Súčasťou implementácie je aj menu a jeho položky. Tieto budú prevzaté zo vzorového pluginu a upravené do potrebného tvaru.

Jednoduchý návrh užívateľského rozhrania pre zobrazenie výsledkov porovnávania súborov možno vidieť na obrázku 4.2.

Obsah zobrazovaných súborov je počas behu programu uložený vo vlastnom modeli. Tento je dizajnovaný tak, že uchováva všetky informácie potrebné pre korektné zobrazenie a jeho návrh je popísaný v nasledujúcej podkapitole.



Obr. 4.2: Návrh zobrazovania výsledkov porovnávania XML súborov.

4.2 Model

Do modelu sa ukladá objektová reprezentácia načítaných štruktúrovaných súborov (XML, JSON), a teda to bude stromový model.

Pre načítavanie súborov sú použité open-source knižnice, ktoré majú vlastné modely pre načítanie dát zo súborov. Zobrazovanie dát na obrazovku teda môže prebiehať priamo pomocou týchto modelov, alebo je možné vytvoriť ďalší model, ktorý je špecificky určený na zobrazovanie dát. V tomto plugine je využitý nový model z niekoľkých dôvodov: neobsahuje žiadne zbytočné informácie, ktoré nie je potrebné uchovávať po dokončení spracovania a porovnávania. Operácia presunu dát z pôvodného do nového modelu sa vykonáva iba jednorazovo a umožňuje potenciálnu zmenu použitej knižnice s minimálnymi úpravami kódu. Napríklad ak by sa v budúcnosti rozširovala funkcionalita pluginu a aktuálna knižnica na spracovanie XML súboru by nepodporovala nutné operácie, potom by bolo možné použiť inú knižnicu a upraviť iba tú časť kódu pluginu, ktorá je zodpovedná za presun dát medzi modelmi.

Pri návrhu som zvažoval, či použiť jeden spoločný model pre XML aj JSON súbory, alebo navrhnúť dva samostatné modely. Výhodou jedného modelu je možnosť porovnávať nielen v rámci zhodných formátov, ale aj medzi rôznymi formátmi. Nevýhodou je vyššia náročnosť návrhu a implementácie. Okrem toho nie je presne definované vzájomné mapovanie určitých elementov týchto formátov, napríklad atribúty uzlu v XML nemajú žiaden presne definovaný tvar po transformácii do JSON. Tieto špecifiká si typicky určí ten, kto implementuje daný transformačný algoritmus. Obvykle však vstupom porovnávania sú XML a JSON súbory, ktoré nemajú presne definovaný vzájomný transformačný vzťah a teda nie je možné určiť mapovanie niektorých elementov. Preto sú v tomto plugine využité dva rôzne modely.

4.2.1 Porovnávanie modelov

Porovnávanie XML modelov je vykonávané pomocou algoritmu popísaného vyššie v kapitole 2.5. Porovnávanie JSON modelov je implementované pomocou vlastného kódu (pozri kapitolu 6.3.1), hlavne kvôli podstatne nižšej implementačnej zložitosti.

V JSON súboroch podobne ako v XML súboroch nezáleží na poradí detských elementov v objektoch, takže na prvý pohľad by zložitosť implementácie porovnávania mala byť podobná. Avšak zásadný rozdiel je v tom, že každý z týchto detských elementov má svoj jednoznačný identifikátor (meno), ktorý je možné použiť na určenie toho, ktoré elementy sa budú porovnávať. V rámci polí zase jednoznačne identifikuje detský element jeho index. Teda pri porovnávaní JSON súborov je vždy jasné, ktorý element v prvom súbore je príslušný na porovnanie ktorému elementu v druhom súbore, keďže každý element má v rámci svojich súrodeneckých elementov jednoznačný identifikátor.

Kapitola 5

Nástroje

Táto kapitola popisuje nástroje, ktoré boli použité pri vývoji tohto softvéru. Podrobne sa venuje hlavne platforme Altap Salamander SDK, ktorá nie je spomínaná v kapitole 6, keďže sa nejedná o kód implementovaný autorom práce. Okrem toho sa zbežne venuje vývojovému prostrediu, jazyku, externým knižniciam a softvéru použitému na správu verzií a zálohovanie kódu.

5.1 Altap Salamander SDK

Altap Salamander SDK pozostáva z nástrojov, dokumentácie a úsekov zdrojového kódu, ktoré sú potrebné pre vytváranie rozšírení, ktoré sú integrovateľné do programu Altap Salamander. Tieto rozšírenia (pluginy) sú implementovateľné v jazyku C++ a na ich kompiláciu sa používajú prostredia Microsoft Visual Studio 2008 alebo Microsoft Visual Studio 2015. V rámci tohto balíku je poskytnutých niekoľko vzorových pluginov, od ktorých sa vývojár môže odraziť pri implementácii vlastného pluginu, a to konkrétne [1]:

- DemoMenu – najjednoduchší zo vzorových pluginov, pridáva jeden príkaz do menu v položke pluginy,
- DemoView – vzorový plugin použiteľný ako základ pre prehliadače súborov,
- DemoPlug – najkomplexnejší zo vzorových pluginov, snaha predviesť väčšinu funkcionality, ktorá by mohla byť potrebná v rámci navrhovania vlastných pluginov,
- UnFAT – voľne distribuovaný kód skutočného pluginu, ktorý je súčasťou klasickej distribúcie progrmu Altap Salamander; slúži ako základ pre archivačné pluginy.

Súčasťou vývojového balíku je tiež skompilovaná SDK verzia aplikácie Altap Salamander, ktorá umožňuje používať makrá, ktoré vypisujú ladiace a chybové výpisy. Na zachytenie týchto správ sa používa Altap Trace Server, ktorý sa taktiež nachádza v balíku. V určitých prípadoch je toto pomerne vhodný spôsob ako odladiť nejakú funkcionalitu (najmä spracovanie správ Windows API súvisiacich s vykresľovaním), avšak vo väčšine prípadov je skôr lepšie použiť ladenie poskytované Visual Studiom ako je spomenuté nižšie.

5.1.1 Základy Altap Salamander SDK

Táto podkapitola sa venuje funkcionalite Altap Salamander SDK, ktorá je využitá pri implementácii pluginu. Okrem toho popisuje podstatné časti kódu, ktoré boli prebraté od

vzorového pluginu bez takmer akýchkoľvek úprav. Popis funkcií typicky zahŕňa iba ich podstatnejšie časti.

Základné informácie o pluginoch

Plugin má formát dynamicky linkovanej knižnice s príponou spl. Autori programu Altap Salamander v dokumentácii popisujú rozdelenie pluginov do týchto kategórií podľa funkcionality:

- zobrazovacie pluginy – spustené pomocou klávesy F3
- archivačné pluginy – napríklad 7-Zip – spustené pomocou klávesových skratiek Alt+F5, Alt+F9 alebo vstupom do pluginu v paneli programu Altap Salamander
- súborové systémy – napríklad FTP Client – zmena súborových systémov je možná pomocou skratiek Alt+F1, Alt+F2
- rozšírenia menu – napríklad File Comparator – tieto pluginy majú svoje menu v menu Plugins, v menu je možné nájsť voľbu pre spustenie, prípadne aj klávesovú skratku
- pluginy načítavajúce náhľady (obrázky) – registrované a potom automaticky volané programom Altap Salamander

Plugin vyvíjaný v rámci tejto práce je možné považovať za kombináciu kategórií zobrazovacích a menu rozširovacích pluginov, keďže umožňuje zobrazovať XML a JSON formáty a ich porovnávanie je implementované ako rozšírenie menu.

Každý plugin exportuje dve vstupné funkcie a to `SalamanderPluginEntry()` a `SalamanderPluginGetReqVer()`. Prvá z nich umožňuje jadru programu Altap Salamander nadviazať komunikáciu s pluginom, zatiaľ čo druhá umožňuje kontrolu kompatibility verzií pluginu a programu Altap Salamander.

V rámci funkcie `SalamanderPluginEntry()` sa deje niekoľko inicializácií, napríklad kontroluje, či je plugin určený pre verziu programu Altap Salamander, ktorá je používaná. Okrem toho sa načítava jazykový modul a rozhrania programu Altap Salamander (jedno obecné rozhranie a jedno poskytujúce upravené grafické prvky). Volá sa aj funkcia `InitPlugin()`, ktorá zabezpečuje nastavenia ohľadom WinLiblt (pozri nižšie). Funkcia upravuje mená univerzálnych tried objektovej nadstavby WinLiblt, aby nedošlo k ich kolízii medzi viacerými pluginmi (prispôsobuje ich pomocou mena pluginu). Nastavuje tiež callback pre pripojenie na HTML nápovedu a načítava akcelerátory. Potom sa nastavujú základné informácie pluginu pomocou funkcie `SetBasicPluginData()`, pričom túto funkciu je nutné volať, inak by nebolo možné volať metódu `CPluginInterface::Connect()` (pozri nižšie). Táto metóda vracia ukazovateľ na objekt, ktorého metódy bude potom Altap Salamander volať.

Po skompilovaní projektu získavame dva podstatné súbory, jedným z nich je vyššie spomínaný `.spl` súbor, druhým je jazykový zdroj s príponou `.slg`. Pri nasadzovaní pluginu je možné plugin nasadiť do programu Altap Salamander z akéhokoľvek adresára, ale pre `.spl` a `.slg` súbory je nutné dodržať nasledujúcu adresárovú štruktúru:

- `./plugin.spl`
- `./lang/lang.slg`

CPluginInterface

Táto trieda implementuje rozhranie (abstraktnú triedu) *CPluginInterfaceAbstract*. Trieda definuje metódy, ktoré potom volá hlavný proces programu Altap Salamander pri určitých úkonoch s pluginom, napríklad pripojenie, odpojenie, práca s konfiguráciou pluginu.

Metóda `Connect()` sa volá pri pripájaní pluginu do programu Altap Salamander. V tejto metóde sa nastavuje ikona pluginu, okrem toho sa definuje položka v menu tohto pluginu, ktorá je naviazaná na klávesovú skratku (CTRL+SHIFT+X), pomocou ktorej je potom možné plugin spustiť.

Metóda `Release()` sa volá pred odpájaním pluginu z programu Altap Salamander, čo sa napríklad deje, keď užívateľ v menu Plugins > Plugin Manager zvolí buď Remove alebo Unload pre daný plugin. V rámci tejto metódy sa uvoľňuje kód pluginu z pamäte a musia sa ukončiť všetky jeho spustené vlákna.

Metóda `LoadConfiguration()` zabezpečuje načítanie konfigurácie z registrov podľa privátneho kľúča, ktorý je poskytnutý programom Altap Salamander. Pokiaľ je predaný prázdny kľúč, potom táto metóda nastavuje predvolenú konfiguráciu. Táto metóda je jednou z prvých metód, ktorú volá hlavný proces programu Altap Salamander, hneď po `SalamanderPluginEntry()`.

Metóda `SaveConfiguration()` zabezpečuje ukladanie konfigurácie do registrov podľa privátneho kľúča.

Metóda `Configure()` sa volá v reakcii na spustenie konfigurácie pluginu, či už z Plugin Manager alebo priamo z pluginu. Pokiaľ už nie je otvorené konfiguračné okno, metóda ho otvorí a keď užívateľ úspešne dokončí prácu s konfiguráciou, metóda pošle správu, ktorá upozorní okná, že sa zmenila konfigurácia.

Trieda okrem toho implementuje metódy pre získanie základných rozhraní pre prácu s pluginom: `GetInterfaceForArchiver()`, `GetInterfaceForViewer()`, `GetInterfaceForMenuExt()`, `GetInterfaceForFS()`, `GetInterfaceForThumbLoader()`. Z týchto metód sú v tomto plugine použité iba dve (`GetInterfaceForViewer()`, `GetInterfaceForMenuExt()`), ostatné vracajú hodnotu `NULL`.

CPluginInterfaceForViewer

Implementuje rozhranie (abstraktnú triedu) *CPluginInterfaceForViewerAbstract*. Definuje metódu `ViewFile()`, ktorú Altap Salamander volá, pokiaľ užívateľ stlačí príkaz View (F3) nad XML alebo JSON súborom.

V implementácii pluginu sú preddefinované prípony súborov, ktoré bude tento plugin zobrazovať. Po načítaní do programu Altap Salamander je však možné toto nastavenie upraviť v menu Altap Salamander `Options > Configuration... > Viewers and Editors > Viewers`.

CPluginInterfaceForMenuExt

Implementuje rozhranie (abstraktnú triedu) *CPluginInterfaceForMenuExtAbstract*. Definuje metódu `ExecuteMenuItem()`, ktorú Altap Salamander volá, ak užívateľ zvolí nejakú položku tohto pluginu v menu. Každá z týchto položiek môže mať priradenú klávesovú skratku, ktorá ju spustí (podobne ako kliknutie). V rámci metódy sa musí dekódovať, o ktorú z položiek sa jedná.

V prípade, že nejaká klávesová skratka položky menu použitá v tomto plugine je použitá aj v inom plugine, potom prioritu má ten plugin, ktorý bol do programu Altap Salamander

pridaný ako prvý. Funkcionalitu neskôr pridaného pluginu je stále možné spustiť kliknutím na položku v menu, avšak nemá asociovanú klávesovú skratku.

Altap Salamander však umožňuje zmeniť preddefinované skratky v menu pomocou `Plugins > Plugins Manager... > Keyboard...`, teda v prípade konfliktu preddefinovanej skratky je možné pre neskôr pridaný plugin definovať novú skratku.

WinLiblt

Altap Salamander SDK obsahuje aj časť, ktorá je jednoduchou objektovou nadstavbou nad Windows API. Táto nadstavba obsahuje niekoľko základných tried:

- *CWindowsObject* – rodič všetkých WinLiblt okienkových tried,
- *CWindow* – základná trieda reprezentujúca grafické okno, viac menej okresaná verzia MFC *CWnd* triedy (pridáva určitú funkcionalitu potrebnú na korektné fungovanie v kontexte pluginu),
- *CTransferInfo* – trieda umožňujúca automatické prenášanie dát medzi premennou v programe a grafickým obsahom editovacieho okna na obrazovke (používa sa v dialógových oknách),
- *CDialog* – základná trieda reprezentujúca dialógové okno,
- *CPropertyDialog* – dedí od *CDialog*, trieda určená na viac-záložkové dialógové okná,
- *CPropSheetPage* – dedí od *CDialog*, trieda reprezentuje jednu záložku viac-záložkového dialógového okna,
- *CWindowQueue* – trieda, ktorá drží záznam všetkých okien, ktoré má plugin otvorené, umožňuje napríklad ich vynútené uzatvorenie v prípade odstránenia pluginu, atď.,
- *CWindowQueueItem* – reprezentuje jednu položku v kolekcii v inštancii triedy *CWindowQueue*.

5.2 Visual Studio 2015

Projekt bol vyvíjaný v prostredí Microsoft Visual Studio 2015. Tento nástroj bol zvolený, pretože vzorové pluginy z Altap Salamander SDK poskytovali už vytvorený Visual Studio projekt, takže bolo možné veľmi rýchlo začať pracovať na projekte.

Jednou z funkcií Visual Studia podstatne využívaných v tomto projekte bolo ladenie. Visual Studio poskytuje možnosť ladiť procesy tak, že sa pripojí k danému procesu a je možné krokovať vykonávanie daného procesu. Po pripojení k procesu Visual Studio prechádza do ladiacej perspektívy, ktorá umožňuje vkladať zarážky do kódu, na ktorých potom proces zastaví vykonávanie, ďalej sledovať hodnoty premenných, a pohybovať sa v rámci zásobníku volaní. V prípade fatálnej chyby v programe je poskytnutá aspoň základná chybová hláška.

V prípade tohto projektu bolo ladenie pomocou pripojenia sa k procesu veľmi užitočné, keďže sa neladí hlavný program, ktorý by sa dal spustiť v ladiacom režime, ale DLL knižnica, ktorú tento program používa.

Pre zjednodušenie práce s pripájaním k procesu je možné stiahnuť rozšírenie do Visual Studia, ktoré sa volá ReAttach (v prípade Visual Studia 2017 je toto už súčasťou Visual

Studia a nie je nutné ručne nič pridávať). Pripájanie k procesu totiž funguje tak, že Visual Studio má v menu možnosť Pripojiť k procesu a toto spustí okno, kde je možné vybrať proces, ku ktorému sa chce užívateľ pripojiť. ReAttach umožní užívateľovi pripojiť sa pomocou klávesovej skratky (Ctrl+R, Ctrl+1) k poslednému vybranému procesu, čo podstatne redukuje čas spúšťania ladenia.

Visual Studio umožňuje pohodlný spôsob výberu, ako bude daná aplikácia skompilovaná. Na hornej lište si je možné vybrať pomocou rozbaľovacej ponuky medzi ladiacou alebo produkčnou kompiláciou a tiež pre akú platformu kompilujeme danú aplikáciu (x86, x64). Potom už len kedykoľvek chceme prekompilovať program stačí stlačiť klávesovú skratku.

5.3 Bitbucket

Na správu verzií a zálohovanie zdrojových kódov som používal Bitbucket od firmy Atlassian. Túto možnosť som zvolil, pretože s ňou mám dobré skúsenosti a tiež ponúka možnosť vytvorenia bezplatného privátneho repozitára, čo napríklad v GitHube nie je možné. Pracoval som s privátnym git repozitárom a ako klienta som používal aplikáciu SourceTree, ktorá je tiež od firmy Atlassian. Poskytuje grafické rozhranie pre všetky možné operácie, ktoré je možné robiť s git repozitárom a umožňuje automatickú autentifikáciu, takže nie je nutné sa zakaždým prihlasovať.

5.4 Knižnice pre spracovávanie súborov

Knižnice v tejto práci boli vyberané s pomocou webovej stránky poskytujúcej udržiavanú kolekciu knižníc určených pre jazyk C++ (C) [13].

5.4.1 XML súbory

Na spracovávanie XML súborov je v tejto práci použitá knižnica Apache Xerces-C++[1]. Hlavným dôvodom použitia tejto knižnice je jej integrácia v implementácii porovnávacieho algoritmu pre XML súbory. Knižnica umožňuje spracovávanie súborov pomocou DOM, SAX a SAX2 API. XML verzia, ktorá je podporovaná touto knižnicou je 1.0. V plugine sa používa spracovávanie súborov pomocou SAX API.

Pri používaní tejto knižnice sa vyskytol jeden väčší problém. Táto knižnica je dynamicky linkovaná, teda je nutné mať príslušný DLL súbor k programu, ktorý používa jej funkcie. Problém vznikol tým, že samotný plugin je dynamicky linkovaná knižnica, ktorá sa integruje do programu Altap Salamander. Altap Salamander vyžaduje, aby všetky závislosti daného pluginu boli prístupné ešte pred tým, než je tento plugin pridaný. A keďže plugin je závislý na DLL súbore knižnice Xerces-C++, tak to pri pokuse o načítanie pluginu vyhodí chybu.

Jedno možné riešenie je, že pred pridaním pluginu do programu Altap Salamander sa DLL súbor pridá do Windows cesty, vtedy k nemu má Altap Salamander prístup.

Ďalším riešením je zmeniť knižnicu Xerces-C++ a skompilovať ju ako staticky linkovanú. Nakoniec bola zvolená táto možnosť, keďže nevyžaduje, aby užívateľ vykonával žiadne dodatočné úkony. Týmto sa síce zvyšuje veľkosť výsledného skompilovaného pluginu, ale toto navýšenie je tolerovateľné v kontexte výhod, ktoré tento prístup poskytuje.

[1] https://xerces.apache.org/xerces-c/index.html

Pri kompilácii knižnice sa používal nástroj CMake, ktorý slúži na platformovo nezávislé vytváranie Makefile súborov. Umožňuje na základe jednej sady nastavení generovať Makefile súbor pre prekladač používaný systémom, na ktorom je spúšťaný CMake.

5.4.2 JSON súbory

Pre spracovávanie JSON súborov je v tomto projekte použitá knižnica JSON for Modern C++[2]. Knižnica pozostáva z jediného hlavičkového súboru, takže je možná jednoduchá integrácia do projektu.

Táto knižnica pôvodne nepodporuje ukladanie poradia, v akom boli z pôvodného súboru načítané detské uzly pre JSON objekty. Avšak v rámci tejto práce bolo uvážené, že je vhodné aby zobrazovaný súbor mal čo najväčšiu podobnosť pôvodnému súboru. Bolo treba nutné vykonať mierne úpravy v kóde, čo však má za následok to, že v prípade zmeny API knižnice prestane riešenie fungovať [12] a bude nutná aktualizácia. Pri týchto úpravách bola použitá FIFO štruktúra[3] osobitne implementovaná autorom knižnice pre spracovanie JSON súborov.

[2] https://github.com/nlohmann/json
[3] https://github.com/nlohmann/fifo_map

Kapitola 6

Implementácia

Táto kapitola podrobne popisuje implementáciu pluginu. Kapitola obsahuje konkrétne pomenovanie funkcií a premenných. Pre lepšiu čitateľnosť bude notácia funkcií vynechávať argumenty, ktoré je možné dohľadať v kóde.

6.1 Vstupný bod programu

Altap Salamander SDK používa niekoľko rôznych rozhraní pre prácu s rôznymi typmi akcií pluginov. V tejto práci sú použité nasledujúce dve rozhrania:

- trieda *CPluginInterfaceForMenuExt* odvodená od abstraktnej triedy *CPluginInterfaceForMenuExt* – metóda `ExecuteMenuItem()` tejto triedy je volaná v prípade, že užívateľ zvolí v menu tohto pluginu možnosť porovnania dvoch súborov (alebo využije zodpovedajúcu klávesovú skratku Ctrl+Shift+X),
- *CPluginInterfaceForViewer* odvodená od abstraktnej triedy *CPluginInterfaceForViewerAbstract* – metóda `ViewFile()` tejto triedy je volaná v prípade, že užívateľ sa bude snažiť prezerať súbory typu XML alebo JSON pomocou funkcionality v Altap Salamander menu Files > View (odpovedajúca klávesová skratka F3). Pri pripájaní pluginu k programu Altap Salamander plugin indikuje, že práve on má byť volaný pre prehliadanie súborov s príponou .xml a .json.

6.1.1 Zobrazovanie súborov

Pri volaní funkcie `ViewFile()` dostáva funkcia ako jeden z argumentov meno súboru, ktorý má byť zobrazený. Táto funkcia je volaná ako súčasť hlavného vlákna programu Altap Salamander, ale keďže plugin nemôže blokovať vykonávanie akcií v hlavnom vlákne, je nutné vytvoriť nové vlákno slúžiace špecificky pre vykonávanie akcií pluginu. V tomto plugine na to slúži trieda *CViewerThread* (dedí od triedy *CThread*, ktorá je súčasťou Altap Salamander SDK).

V rámci tohto vlákna sa najprv načíta súbor, ktorý sa má zobrazovať. Pokiaľ tento súbor nie je v platnom XML alebo JSON formáte, potom sa užívateľovi zobrazí chybové okno upozorňujúce na túto skutočnosť.

Po úspešnom načítaní dát do modelu sa vytvára zobrazovacie okno triedy *CViewerWindow*. Nakoniec sa spustí slučka správ, ktorá dekóduje správy prijímané vláknom a distribuuje ich oknu.

6.1.2 Porovnávanie súborov

Pri volaní metódy `ExecuteMenuItem()` dostáva táto metóda ako argument ID položky v menu, ktorá bola vybraná. Menu pluginu v tejto práci obsahuje iba jednu položku, ktorá umožňuje porovnávať dva súbory.

Pri porovnávaní súborov umožňuje tento plugin vyberať, ktoré súbory budú porovnávané pomocou panelov v programe Altap Salamander, podobne ako plugin File Comparator poskytovaný firmou Altap na porovnávanie textových súborov.

V metóde `ExecuteMenuItem()` najprv zistíme, aký je stav panelov programu Altap Salamander, teda aké súbory sú označené a koľko je vybratých súborov v jednotlivých paneloch.

Vybratý súbor je taký, nad ktorým bola vykonaná akcia vybratia a Altap Salamander zobrazuje jeho meno červenou farbou, označený súbor je ten, na ktorom sa aktuálne nachádza "kurzor" programu Altap Salamander.

Na základe stavu panelov následne určíme, ktoré súbory budú predbežne zvolené ako porovnávané (predbežne, keďže užívateľ ich môže následne upraviť v dialógovom okne). V závislosti na tom, aký tento stav je, sa môže stať, že budú predbežne vybrané dva súbory, jeden súbor, alebo dokonca žiadny súbor (napríklad, keď v aktívnom paneli je vybratých viac ako dva súbory, potom nie je jasné, ktoré dva by sa mali porovnávať a teda nie je možné určiť ani jeden súbor na porovnávanie).

S panelmi programu Altap Salamander sa pracuje v metóde `ExecuteMenuItem()`, keďže táto je súčasťou hlavného vlákna vykonávania programu Altap Salamander. Ďalšia funkcionalita pluginu beží v samostatnom vlákne reprezentovanom triedou *CCompareThread* (taktiež dedí od triedy *CThread*).

Vláknu sú predávané (dve) cesty k predbežne určeným súborom na porovnávanie. Následne je užívateľovi zobrazený dialóg výberu porovnávaných súborov, pričom ak predané predbežné cesty neboli prázdne, tak sa zobrazia v tomto dialógu (pozri kapitolu 4 pre obrázok tohto dialógu). Po stlačení OK sa skontroluje validita daných ciest (existujú súbory? nie sú to priečinky?) a pokračuje sa načítavaním súborov. Ak sa pri načítavaní súborov zistí, že nejaký zo súborov nie je platný XML alebo JSON súbor, potom sa užívateľovi zobrazí chybová hláška, po ktorej odkliknutí sa mu znovu zobrazí výberový dialóg. To isté sa deje v prípade, že jeden zo súborov je typu XML a druhý typu JSON, keďže tento plugin nepodporuje XML-JSON porovnávanie. Ak sú obidva súbory platné a toho istého typu, potom sa pokračuje ich porovnaním. Nakoniec sa vytvára okno triedy *CComparatorWindow* (pomocou metódy `CreateEx()`) a spúšťa sa slučka správ, v ktorej sa správy prichádzajúce tomuto vláknu dekódujú a posielajú oknu.

6.2 Model

Plugin implementovaný v rámci tejto práce využíva dva typy modelov, jeden pre XML súbory a druhý pre JSON súbory. V prípade oboch formátov je najprv súbor načítaný do modelov používaných knižnicami, ktoré táto práca používa na spracovávanie XML a JSON súborov a až potom je načítaný do modelov popisovaných v tejto kapitole. Modely popísané v tejto kapitole slúžia na skladovanie a zobrazovanie načítaných dát - na rozdiel od modelov používaných knižnicami, ktoré sú z pamäte odstránené po tom, čo sú dáta načítané do zobrazovacích modelov.

Trieda *CModel* slúži ako spoločné rozhranie pre prácu s XML a JSON modelmi, od tejto triedy potom dedia triedy *CXmlModel* a *CJsonModel* reprezentujúce XML a JSON dátové

stromy. CModel definuje tri abstraktné metódy, ktoré sú spoločné pre obidva typy modelov, konkrétne:

- `getType()` – slúži na získanie typu modelu (XML/JSON),
- `determineNodeLines()` – zavolaním tejto metódy sa pre každý uzol (element) určí, na ktorom riadku pri zobrazovaní začína a na ktorom končí; toto sa potom využíva pri implementácii skrolovania pri zobrazovaní súborov,
- `getLineCount()` – metóda vracia celkový počet riadkov, ktorý bude mať zobrazený model.

6.2.1 XML model

XML model je implementovaný nasledujúcou sériou tried.

CXmlModel

Potomok triedy *CModel*, reprezentuje dáta načítané z XML súboru v stromovom modeli. Obsahuje ukazovateľ na koreňový uzol XML modelu.

CXmlNode

Táto abstraktná trieda je predkom všetkých typov uzlov v modeli a implementuje funkcionalitu, ktorú majú spoločnú. Toto zahŕňa primárne status porovnávania uzlu (trieda *NodeStatus*). Okrem toho poskytuje rozhranie pre určovanie typu uzlu a funkcionalitu, ktorá je zdieľaná viac než jedným z jej potomkom, ale nie všetkými (napríklad získanie indexu prvého a posledného riadku – toto je nepotrebné pri uzloch typu atribút).

CXmlAttribute

Potomok triedy *CXmlNode*, ktorý reprezentuje atribút určitého elementu.

CXmlElement

Trieda reprezentujúca konkrétny XML element. Obsahuje zoznam detí typu *CXmlNode* a zoznam atribútov typu *CXmlAttribute*. Taktiež obsahuje informácie o tom, na ktorom riadku v rámci zobrazovaného súboru sa bude nachádzať prvý riadok tohto elementu a na ktorom sa bude nachádzať posledný riadok.

CXmlText

Táto trieda reprezentuje textový uzol. Obsahuje hodnotu textového uzlu rozdelenú do viacerých riadkov podľa formátovania v pôvodnom súbore.

CXmlNodeEmpty

Trieda reprezentuje prázdne miesto pri zobrazovaní. Je použitá v prípadoch, kedy pri porovnávaní neexistuje k uzlu jeho zodpovedajúci uzol v druhom strome. Vypísanie prázdneho miesta zaistí, že nasledujúce zodpovedajúce riadky sa budú zobrazovať vedľa seba.

Určovanie hraničných riadkov elementov

Na určovanie indexov prvého a posledného riadku pri zobrazovaní modelu pre všetky elementy modelu slúžia metódy `determineElementLines()` a `determineElementLines_proc()`, pričom prvá z nich je verejná a slúži na počiatočné volanie (s argumentami zodpovedajúcimi koreňovému elementu) druhej, ktorá potom rekurzívne volá samú seba. Metóda `determineElementLines_proc()` si odovzdáva dva argumenty: element, ktorý je aktuálne spracovávaný a index riadku, na ktorom tento element začína.

V rámci tejto metódy sa určí, akú dĺžku (počet riadkov) má spracovávaný element, nastaví sa index jeho prvého a posledného riadku (index prvého riadku dostáva metóda ako argument, index posledného riadku určí na základe dĺžky elementu) a metóda vracia dĺžku spracovávaného elementu.

Pri určovaní dĺžky sa najprv kontroluje, či element je zbalený. Ak áno, potom dĺžka elementu je 1 a index jeho posledného riadku je rovnaký ako index prvého riadku. Ak nie, potom sa určuje jeho dĺžka na základe toho, akého typu je. Prázdny element (`NodeType::empty`) obsahuje informáciu o svojej dĺžke a teda ju netreba počítať. Dĺžka textového uzlu je rovná počtu nových riadkov, ktoré obsahuje – pri formátovaní textového elementu sa prihliada na to, akým spôsobom bol rozdelený na riadky v pôvodnom súbore. A pokiaľ sa jedná o element, tak jeho dĺžka zodpovedá súčtu dĺžky všetkých jeho detí, pre ktoré je rekurzívne volaná metóda `determineElementLines_proc()` plus dva riadky na otváraciu a zatváraciu značku. Výnimku predstavuje element, ktorý je prázdny alebo obsahuje iba jeden jednoriadkový textový uzol. V tom prípade sa celý element zobrazí na jednom riadku.

6.2.2 JSON Model

Na implementáciu JSON modelu slúžia nasledujúce triedy.

CJsonModel

Potomok triedy *CModel*, ktorý reprezentuje dáta načítané z JSON súboru v stromovom modeli. Podobne ako trieda *CXmlModel* obsahuje ukazovateľ na koreňový uzol JSON modelu.

CJsonNode

Táto abstraktná trieda je predkom všetkých tried ktoré sa nachádzajú v implementácii JSON stromovej štruktúry.

Obsahuje meno uzlu, príznak určujúci, či je daný uzol pomenovaný (určené tým, či sa daný uzol nachádza v poli alebo je súčasťou objektu). Ďalej obsahuje status porovnania uzlu reprezentovaný triedou NodeStatus a index prvého a posledného riadku uzlu v rámci zobrazovaného súboru.

Táto trieda deklaruje niekoľko abstraktných metód, ktoré sú potom zodpovedajúcim spôsobom implementované v detských triedach. Metóda `compareTypes()` umožňuje porovnať typ uzlu s iným uzlom predaným ako argument, metóda `isComposite()` určuje, či uzol je jednoduchého alebo zloženého typu (pole, objekt). Metóda `isEmpty()` určuje, či sa jedná o normálny uzol alebo o uzol, ktorý je použitý pre zobrazovanie prázdneho miesta.

CJsonValueBase

Trieda reprezentujúca jednoduché uzly. Pamätá si hodnotu uzlu vo formáte reťazca a k tomu si pamätá informáciu o jej type.

CJsonCompositeEntity

Je to abstraktná trieda, ktorá slúži ako spoločný predok uzlov typu objekt a pole. Obsahuje vektor detských uzlov typu *CJsonNode*. Umožňuje nastaviť status uzlu, ktorý je propagovaný ku všetkým detským uzlom pomocou metódy `setStatus()`.

CJsonArray

Potomok triedy *CJsonCompositeEntity*, reprezentuje JSON uzol typu pole. Obsahuje mapu, ktorá mapuje kľúče na ku nim zodpovedajúce detské uzly. V prípade poľa tieto kľúče pozostávajú z čísel reprezentujúcich poradie v poli.

CJsonObject

Taktiež potomok triedy *CJsonCompositeEntity*, reprezentuje JSON uzol typu objekt. Podobne ako pole obsahuje mapu, ktorá mapuje kľúče na ku nim zodpovedajúce detské uzly. V prípade objektu tieto kľúče pozostávajú z reťazca typu string a reprezentujú meno položky objektu.

Určovanie hraničných riadkov uzlov

Podobne ako pri XML modeli, na určovanie indexov prvého a posledného riadku uzlov slúžia dve metódy: `determineNodeLines()` a `determineNodeLines_proc()`. Logika použitia týchto metód ako aj argumenty metód sú zhodné s metódami v modeli XML. Taktiež celková štruktúra metódy `determineNodeLines_proc()` je taká istá: zistí sa dĺžka uzlu, nastaví sa index prvého a posledného riadku a metóda vráti zistenú dĺžku.

V metóde `determineNodeLines_proc()` sa teda najprv testuje to, či je uzol zbalený – ak áno, potom dĺžka uzlu je jedna. Ak nie, potom sa testuje, akého typu je uzol. JSON uzol môže byť buď zložený alebo jednoduchý. Pre jednoduché uzly to môže byť buď uzol reprezentujúci prázdny uzol, ktorý v sebe obsahuje informáciu o svojej dĺžke, alebo nejaký z jednoduchých JSON dátových typov – tieto budú mať vždy dĺžku (počet riadkov) jedna, pretože JSON špecifikácia ani v prípade reťazcov nepodporuje znak nový riadok (musí byť reprezentovaný ako `"\n"`).

Ak je uzol zložený, teda pole alebo objekt, potom jeho dĺžka zodpovedá súčtu dĺžok jeho potomkov, pre ktorých je rekurzívne volaná metóda `determineNodeLines_proc()`, plus dva (jeden otvárací riadok kde je meno a otváracia zátvorka a jeden zatvárací riadok so zatváracou zátvorkou).

6.3 Porovnávanie súborov - CFileComparator

CFileComparator je základná trieda použitá na načítanie, skladovanie a porovnávanie súborov. Obsahuje dva ukazovatele na typ *CModel*, ktoré reprezentujú porovnávané súbory. Taktiež sú jej súčasťou mená týchto súborov, ktoré sa pri zobrazovaní rozdielov zobrazujú nad obsahom súboru. Pri vytváraní objektu triedy je nutné konštruktoru odovzdať názvy oboch súborov, ktoré chceme porovnávať.

Načítavanie a porovnávanie súborov je inicializované z konštruktora triedy *CFileComparator*. Pri určovaní typu súboru sa nespoliehame na to, akú má súbor príponu, ale na to, či je ho možné úspešne načítať ako XML alebo JSON súbor pomocou použitých knižníc.

Najprv sa pokúšame načítať obidva súbory ako JSON. Pokiaľ sú obidva súbory validné JSON súbory a úspešne sa načítali, potom ďalej pokračujeme s JSON porovnávaním súborov. Pokiaľ aspoň jeden z týchto súborov nebol úspešne načítaný ako JSON, potom sa pokúsime obidva súbory načítať ako XML súbory. Pokiaľ sú obidva súbory úspešne načítané, potom sa pokračuje s XML porovnávaním. Pokiaľ nie, tak konštruktor vyhadzuje výnimku, na základe ktorej sa zobrazí užívateľovi chybová hláška.

Po dokončení porovnávania sa ešte zavolá metóda `determineNodeLines()`, ktorá zabezpečí, že sa v modeloch na zobrazovanie určia indexy začiatočných a koncových riadkov elementov.

6.3.1 JSON porovnávanie

Porovnávanie JSON súborov je vykonávané metódou `CFileComparator::compareFilesJson()`, ktorá ako argumenty dostáva dva JSON uzly, ktoré má porovnávať a vracia príznak zhody uzlov.

Zároveň nastavuje NodeStatus spracovávaných uzlov, ktorý určuje ich stav porovnania. Prípustné hodnoty sú:

- `NodeStatus::same` – pre uzol bol nájdený zodpovedajúci uzol v druhom strome so zhodnou hodnotou
- `NodeStatus::changed` – pre uzol bol nájdený zodpovedajúci uzol v druhom strome, tento uzol však má inú hodnotu
- `NodeStatus::added` – uzol sa nenachádza v strome 1, ale nachádza sa v strome 2
- `NodeStatus::missing` – uzol sa nachádza v strome 1, ale nenachádza sa v strome 2

Pri porovnávaní sa najprv kontroluje, či vstupné uzly sú zhodného typu. Ak nie, potom vieme, že tieto uzly nie sú zhodné.

Pokiaľ porovnávané uzly sú zhodného typu, potom nás zaujíma, či sa jedná o zložený alebo jednoduchý typ. Pri porovnávaní jednoduchých typov nám stačí využiť funkcionalitu implementovanú v rámci modelu JSON pomocou metódy `CJsonValue::compare()`.

Pre zložené uzly typu objekt najprv zoberieme prvý uzol a iterujeme cez všetky jeho deti. Pre každé dieťa z prvého uzlu hľadáme zodpovedajúce dieťa z druhého uzlu. Pokiaľ sa nám ho podarí nájsť, tak rekurzívne zavoláme metódu `compareFilesJson()` pre tieto detské uzly. Okrem toho sa ešte prehodí poradie detských uzlov druhého stromu, keďže zodpovedajúce detské uzly sa majú zobrazovať na rovnakej úrovni. Ak sa zodpovedajúci detský uzol nenašiel, vytvorí sa prázdny uzol, ktorý sa vloží na zodpovedajúce miesto medzi detskými elementami uzlu z druhého stromu.

Mená všetkých takto spracovaných detských uzlov prvého uzlu sa zapamätajú. Po skončení spracovania detí prvého uzlu sa spracujú všetky deti druhého uzlu. Iteruje sa cez všetky jeho detské uzly a hľadajú sa tie, ktoré nereprezentujú prázdne miesto a tiež nemajú meno, ktoré bolo už spracované pri prechode prvého uzlu. Takto sa nájdu všetky unikátne deti druhého uzlu. Pre každý z nich je vytvorený prázdny uzol, ktorý je vložený ako dieťa prvého uzlu na koniec jeho detských uzlov.

Pre zložené uzly typu pole sa algoritmus správa podobne, až na to, že v tomto prípade sú definujúcim kľúčom objektov indexy. Teda keď sa iteruje cez detské uzly prvého uzlu,

tak sa vždy pozerá na ten istý index v druhom uzle. Ak má prvý uzol viac detí ako druhý uzol, potom všetky zvyšné budú mať status (`NodeStatus::missing`), zatiaľ čo ak druhý uzol má viac detí, všetky zvyšné deti druhého uzlu budú mať status (`NodeStatus::added`).

6.3.2 XML porovnávanie

Pre XML porovnávanie je použitý algoritmus X-Diff, ktorý je popísaný v kapitole 2.5. Bola nájdená jeho vzorová implementácia, ktorá je začlenená do projektu. Táto sa skladá z niekoľkých tried, pričom jej hlavná trieda zodpovedná za porovnávanie a vypisovanie rozdielov dvoch súborov je *XDiff*.

Pôvodne som nechcel upravovať originálny kód, a preto som odvodil od triedy *XDiff* detskú triedu *XDiffFc*, v ktorej boli reimplementované metódy použité na vypisovanie zmien a iniciáciu porovnávania – `writeDiff()`, `writeDiffNode()`, `writeMatchNode()` a konštruktor triedy.

Nakoniec však bolo nutné zasahovať aj do originálneho kódu, kvôli implementácii podpory unicode zobrazovania a kvôli oprave niekoľkých chýb nájdených v originálnom kóde.

Konštruktor triedy

V rámci konštruktora triedy sa najprv načítajú obidva XML súbory pomocou triedy *XParser* a načítané súbory sú uložené do modelu typu *XTree*, pričom obidve tieto triedy sú súčasťou prevzatej implementácie algoritmu X-Diff.

Po úspešnom načítaní sa začnú súbory porovnávať, kde najprv sa porovnajú mená koreňových uzlov. Ak sa líšia, potom sa dokumenty líšia na úrovni koreňových uzlov a stačí koreňovým uzlom priradiť príslušný status porovnania a vypísať ich pomocou metódy `writeDiff()`. Ak sú zhodné, potom sa volá pre tieto uzly metóda `xdiff()`, ktorá implementuje algoritmus popísaný v kapitole 2.5.

V konštruktore triedy došlo k dvom zmenám oproti pôvodnej implementácii. Jednou z nich je test, či sa líšia koreňové uzly. Ak áno, potom sa užívateľovi zobrazí informačné okno, pretože v tomto prípade nemá zmysel zobrazovať výsledok porovnania.

Druhá je opravou chyby v uvoľnovaní pamäte, ku ktorej dochádzalo, pokiaľ sa líšili mená koreňových uzlov.

Metódy pre transformáciu uzlov

V pôvodnej implementácii slúžili metódy `writeDiff()`, `writeDiffNode()` a `writeMatchNode()` na vypisovanie textového súboru popisujúceho rozdiely medzi porovnávanými súbormi.

V plugine je však nutné tieto rozdiely graficky zobrazovať, a preto je žiadúce namiesto výpisu rozdielového súboru transformovať stromové štruktúry typu *XTree* do stromových štruktúr typu *CXmlModel*, ktoré sú v tejto práci použité na skladovanie a zobrazovanie dát (aj s informáciami o ich porovnaní). Na toto sa využíva metóda `writeDiff()`, ktorá vykoná operáciu tranformácie obidvoch *XTree* stromov na *CXmlModel* stromy.

Metódy `writeDiffNode()` a `writeMatchNode()` boli upravené tak, že zoberú uzol z pôvodnej stromovej štruktúry a vrátia uzol transformovaný do zodpovedajúceho dátového typu stromovej štruktúry *CXmlModel*.

Obom metódam je ako jeden z argumentov predaná indikácia toho, ktorý strom je aktuálne spracovávaný. Strom 1 je označený ako primárny a strom 2 ako sekundárny.

writeDiff

Metóda `writeDiff()` rozlišuje dva prípady výsledku porovnania koreňových uzlov. Buď sú rozdielne mená koreňových uzlov, vtedy volá pre obidva uzly metódu `writeMatchNode()` so zodpovedajúcim príznakom výsledku porovnania. Alebo sú mená koreňových uzlov zhodné, ale líši sa ich obsah, vtedy dvakrát volá metódu `writeDiffNode()`, raz pre primárny a raz pre sekundárny strom.

writeDiffNode

Táto metóda je určená na transformáciu uzlov, ktorých hodnota bola zmenená. Ako argumenty dostáva dvojicu navzájom zodpovedajúcich si uzlov (uzol 1 patrí primárnemu stromu, uzol 2 patrí sekundárnemu), pričom vráti transformovaný jeden z nich, podľa toho či je aktuálne spracovávaný primárny alebo sekundárny strom.

Pokiaľ sa jedná o textové uzly, je uzol spracovávaného stromu transformovaný na objekt triedy *CXmlElement* typu `NodeType::text` a hodnota tohto uzlu (teda text) sa získava pomocou metódy `constructText()` poskytovanej triedou *XDiff*.

Najprv sa vygenerujú zodpovedajúce objekty triedy *CXmlAttribute* v závislosti od výsledku porovnania atribútov uzlov.

Následne sa iteruje cez všetky detské elementy uzlov. Najprv sa iteruje cez všetky detské elementy uzlu 1, pričom pokiaľ má aktuálne spracovávaný detský element uzlu 1 zodpovedajúci detský element v uzle 2, ktorého výsledok porovnania je zhoda, volá sa metóda `writeMatchNode()` s príznakom `WRITENODE_SAME`. Pokiaľ nemá zodpovedajúci detský element, volá sa metóda `writeMatchNode()` s príznakom `WRITENODE_DELETE`. Pokiaľ zodpovedajúci detský element sa líši, volá sa rekurzívne metóda `writeDiffNode()`, pričom argumenty identifikátorov uzlov sú detský element a jemu zodpovedajúci detský element uzlu 2.

Následne sa iteruje cez detské elementy uzlu 2, a pokiaľ nemajú zodpovedajúci detský element v uzle 1, volá sa metóda `writeMatchNode()` s príznakom `WRITENODE_INSERT`.

Návratové hodnoty metód volaných nad deťmi vždy vrátia transformovaný element, ktorý je pridaný do zoznamu detí uzlu.

writeMatchNode

Metóda `writeMatchNode()` sa používa na transformáciu uzlov, ktoré sú buď zhodné, boli odstránené, alebo boli pridané. Jedným z argumentov metódy je identifikátor uzlu na konverziu. Argumentom metódy je tiež príznak výsledku porovnania tohto uzlu.

Pokiaľ predaný identifikátor uzlu neexistuje v strome, cez ktorý sa aktuálne iteruje, ale existuje v druhom strome, tak sa vytvorí prázdny uzol zodpovedajúci tomuto uzlu v druhom strome a tento prázdny uzol je potom vrátený. Dĺžka prázdneho uzlu sa počíta pomocou metódy `computeLineLength()`, ktorá ju počíta ako dĺžku zodpovedajúceho elementu v druhom strome (ktorý existuje).

Ak uzol existuje v strome, cez ktorý sa iteruje, môže to byť element, alebo textový uzol. Ak sa jedná o uzol typu element, tento je transformovaný na objekt triedy *CXmlElement* s typom `NodeType::element` tak, že sa najprv prenesie jeho meno. Potom sa iteruje cez všetky atribúty pôvodného uzlu, kde pre každý atribút je vytvorený zodpovedajúci objekt typu *CXmlAttribute* a tento je pridaný do zoznamu atribútov transformovaného elementu. Nakoniec sa rekurzívne volá metóda `writeMatchNode()` pre všetky deti pôvodného uzlu a tieto sú pridané do zoznamu detí transformovaného elementu.

Ak uzol nie je elementom, potom to znamená, že sa jedná o textový uzol. Tento je transformovaný na objekt triedy *CXmlElement* typu `NodeType::text` a hodnota tohto uzlu (text) sa získa pomocou metódy `constructText()`.

6.4 Užívateľské rozhranie

Implementovaný plugin má dve základné použitia, vie prehliadať súbory a zobrazovať výsledky porovnávania súborov. Podľa tohto je implementované užívateľské rozhranie, ktoré pre každé z týchto použití používa iný typ hlavného okna. Každé hlavné okno potom má svoje detské okná, pričom v každom z nich sa zobrazuje obsah jedného súboru. Hlavné okno určené pre zobrazovanie súborov má iba jedno detské okno, zatiaľ čo hlavné okno určené pre zobrazenie výsledkov porovnania má dve.

CBaseWindow

Abstraktná trieda, ktorá implementuje spoločnú funkcionalitu hlavných okien. Dedí od triedy *CWindow*.

Pre implementovanie rozdielneho chovania na špecifických miestach sú podľa potreby použité abstraktné metódy, ktoré každý potomok implementuje podľa svojich požiadaviek.

CComparatorWindow

Trieda reprezentujúca hlavné okno použité pri zobrazovaní výsledkov porovnávania dvoch súborov. Je potomkom triedy *CBaseWindow*.

Pri zobrazovaní výsledkov porovnávania súborov sú zobrazené aj mená týchto porovnávaných súborov. Meno súboru je nezávislé od operácií skrolovania a malo by sa vždy zobrazovať v takom istom formáte a na tom istom mieste, je teda vhodné ho neumiestniť do oblasti obrazovky, ktorá je ovplyvnená operáciou skrolovania – príslušného detského okna – ale umiestniť ho do hlavného okna.

CViewerWindow

Trieda reprezentujúca základné okno použité pri zobrazovaní jedného súboru typu XML alebo JSON. Dedí od triedy *CBaseWindow*.

V tejto triede ostáva veľká časť zo zdedených abstraktných funkcií prázdna, keďže tieto nie sú určené pre implementáciu funkcionality tohto typu okna (sú určené pre *CComparatorWindow*).

6.4.1 Zobrazovanie obsahu súborov - CRendererWindow

Je to trieda reprezentujúca okno, ktoré je použité na zobrazovanie obsahu jedného súboru XML alebo JSON formátu z modelu určeného na zobrazovanie. Taktiež je potomkom triedy *CWindow*.

Určovanie súradníc výpisu

Pri vypisovaní informácií na obrazovku sa používa Windows API, ktoré pozíciu vypisovaného textu adresuje v pixeloch. Naopak model použitý pre skladovanie dát pracuje na úrovni čísel riadkov. Je teda nutné prepočítať číslo vypisovaného riadku na súradnice na osi x a y v pixeloch. Súradnice, na ktorých sa začína vypisovanie riadku vypočítame ako:

```
int x = _xChar * (1 - _xPos);
int y = _yChar * (currentLine - _yPos);
```

Kód 6.1: Výpočet súradníc riadku

Premenná *_xChar* v kóde 6.1 reprezentuje jednotku skrolovania na x-ovej osi (zodpovedá priemernej šírke jedného znaku v pixeloch), zatiaľ čo premenná *_xPos* reprezentuje aktuálny stav skrolovania (koľko krokov skrolovania bolo urobených) v rozmedzí $1 \rightarrow$ max.

V prípade, že sa skrolovalo, bude pri vypisovaní riadku počiatočná súradnica na x-ovej osi záporná a tým sa posúva tá časť textu, ktorá je viditeľná pre užívateľa.

Premenná *_yChar* v kóde 6.1 reprezentuje jednotku skrolovania na y-ovej osi (výška jedného riadku v pixeloch), zatiaľ čo premenná *_yPos* reprezentuje aktuálny stav skrolovania v rozmedzí $1 \rightarrow$ max. Premenná *currentLine* reprezentuje číslo vypisovaného riadku.

Podobne ako pri x-ovej osi, výpočet je tiež závislý na stave skrolovania. V prípade, že sa skrolovalo sa teda tiež prvý riadok súboru bude vypisovať na záporné súradnice a tým sú posunuté čísla riadkov, ktoré užívateľ vidí.

Okrem toho, keďže je nutné rôznym spôsobom formátovať vypisovaný text, tak nie je vždy vypísaný celý riadok naraz, ale vypisuje sa postupne. Pri týchto postupných výpisoch sa nemení súradnica y, keďže sa stále jedná o ten istý riadok, ale je nutné zmeniť súradnicu x tak, aby ukazovala na miesto za posledným vypísaným textom na danom riadku. Preto sa vždy po vypísaní textu pripočíta k súradnici x dĺžka vypísaného textu a tým získavame aktuálnu súradnicu pre ďalší výpis. Dĺžka vypísaného textu je počítaná pomocou funkcie `GetTextExtentPoint32`, ktorá je poskytovaná v rámci Windows API a vracia dĺžku (v pixeloch) textu predaného ako argument.

Zobrazovanie textového obsahu

Vykresľovanie JSON modelu sa deje pomocou rekurzívnej metódy `paintNodeJson()`, pomocou ktorej sa postupne iteruje cez uzly JSON stromu.

Táto metóda poskytuje dva módy volania. Jeden sa používa na emulovaný prechod stromu, pričom sa hľadá dĺžka najdlhšieho riadku, ktorý bude vykreslený. Toto je potrebné pri určovaní dimenzií horizontálneho posuvníku, keďže je použitý proporcionálny font. Druhý mód sa používa priamo na vykreslenie.

Úplne na začiatku metódy sa skontroluje, či daný uzol stromu aspoň čiastočne spadá do rozmedzia medzi indexom prvého a posledného zobrazovaného riadku. Pokiaľ nie, metóda sa vracia. Táto akcia je podstatnou časťou algoritmu zobrazovania, pretože zamedzí iterácii cez uzly, ktoré sa nemajú vykresľovať a týmto znižuje čas vykreslenia.

Ak sa uzol má vykresliť, potom sa vykonávajú akcie v závislosti na jeho type. Pokiaľ sa jedná o prázdny uzol, tak sa vypisujú prázdne riadky, až dokým nenastane jedna z dvoch podmienok: index aktuálneho vypisovaného riadku neprevýši index posledného vypisovaného riadku, alebo bola vypísaná celá dĺžka prázdneho uzlu. Prázdne riadky sú pri vypisovaní očíslované a vždy majú pozadie sivej farby.

Ak sa nejedná o prázdny uzol, potom sa rozlišujú dva rôzne typy uzlov: zložené a jednoduché. Pre obidva typy uzlov je nutné pred výpisom nového riadku skontrolovať, či sa aktuálne vypisovaný riadok nachádza v rozmedzí vypisovaných riadkov. Pred výpisom textu na riadok sa volá metóda `prepareLine()`, ktorá zobrazí číslo riadku, nastaví pozadie riadku a zabezpečí jeho správnu indentáciu.

Ak ide o jednoduchý uzol, potom stačí vypísať jeho meno a hodnotu, oddelené dvojbodkou.

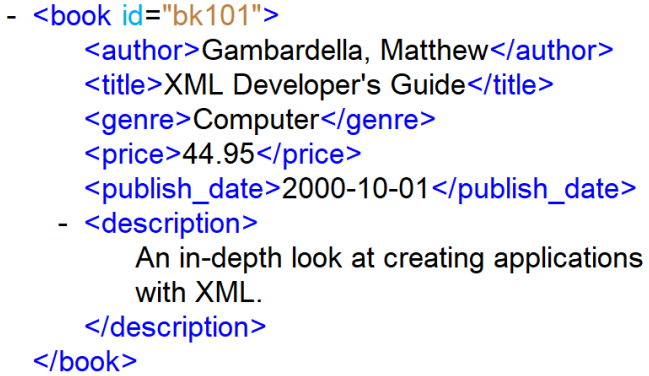
Ak je spracovávaný zložený uzol, tak sa zisťuje, či je zbalený. Pokiaľ áno, tak sa vypíše iba jeho prvý riadok upravený tak, že je na koniec vložená uzatváracia zátvorka. Pokiaľ nie, tak sa vypíše prvý riadok, potom sa iteruje cez všetky detské uzly tohto objektu a rekurzívne sa pre ne volá metóda `paintNodeJson()`. Potom sa vypíše uzatváracia zátvorka na nový riadok. Výsledné formátovanie môžeme vidieť na obrázku 6.1.

Obr. 6.1: Formát zobrazenia JSON súboru.

Pri vypisovaní neprázdnych uzlov je nutné sledovať, či je vypisovaný uzol posledným potomkom svojho rodiča. Štandard JSON totiž vyžaduje, aby všetky detské uzly boli oddelené čiarkou, zatiaľ čo za posledným čiarka byť nesmie. Toto sa overuje pomocou príznaku predaného ako argument metódy, pričom tento je vždy nastavený pri rekurzívnom volaní metódy počas iterácie cez potomkov.

Vykresľovanie XML modelu sa deje pomocou rekurzívnej metódy `paintNodeXml()`, pomocou ktorej sa postupne iteruje cez uzly XML stromu.

Tak ako metóda `paintNodeJson()` poskytuje dva módy volania. Takisto tiež na začiatku metódy prebieha kontrola, či sa daný uzol nachádza v rozsahu výpisu a vypisovanie prázdnych uzlov funguje rovnako ako v metóde `paintNodeJson()`. Líši sa až ostatnými typmi uzlov.



Obr. 6.2: Formát zobrazenia XML súboru.

Pre textové uzly sa zachováva formátovanie novými riadkami z načítavaného súboru, teda ak v načítavanom súbore bol text rozdelený na tri riadky, potom bude aj pri vykresľovaní rozdelený na tri riadky.

Pre elementy sa najprv vypíše riadok s otváracou značkou, v rámci ktorej sa iteratívne vypíšu všetky atribúty uzlu. Následne sa iteruje cez všetky detské uzly a nakoniec sa vypisuje riadok s uzatváracou značkou. V prípade, že je element zbalený, potom sa vypisuje iba otváracia značka (so všetkými atribútmi) a zatváracia značka na tom istom riadku. Pre ukážku výsledného formátovania pozri obrázok 6.2.

Pre vykresľovanie čísel riadkov, zafarbenie pozadia riadku a odsadenie riadku je použitá metóda `prepareLine()`.

Pozadie riadku je vykresľované ako obdĺžnik farby, ktorú špecifikuje stav uzlu, ktorého je vypisovaný riadok súčasťou. Odsadenie riadku je vypočítané na základe toho, v akej hĺbke stromu sa nachádza uzol, ktorého je riadok súčasťou.

Metóda potom vráti dĺžku vypísaného čísla riadku plus dĺžku odsadenia riadku, pričom o túto návratovú hodnotu je upravená súradnica x-ovej osi pred vypisovaním na riadok.

Skrolovanie

Pri implementácii skrolovania bolo nutné zabezpečiť, aby pokiaľ je zobrazovaný viac než jeden súbor, bolo skrolovanie synchrónne pre obidva súbory.

Toto je zabezpečené pomocou triedy *CMsgDistributor*. Táto trieda si udržuje referencie na spárované okná typu *CRendererWindow*. Pokiaľ do *CRendererWindow* príde udalosť skrolovania, potom je volaná metóda `CMsgDistributor::distribute()`, ktorá túto udalosť distribuuje druhému oknu.

Program podporuje skrolovanie pomocou kláves, koliečka myši aj posuvníku.

Zbaľovanie uzlov

Užívateľské rozhranie umožňuje zbaľovať a rozbaľovať uzly pomocou jednoduchého kliknutia myšou na ich prvý riadok. Pri prijatí udalosti kliknutia myši však máme dostupné iba súradnice kliknutia x a y v pixeloch. Musí sa teda určiť, či bolo kliknuté na prvý riadok nejakého elementu.

Najprv sa vypočíta, na ktorý riadok zobrazovaného stromu bolo kliknuté:

```
GetCursorPos(&p);
GetWindowRect(HWindow, &r);

int currentLine = (p.y - r.top) / _yChar + _yPos;
```

Kód 6.2: Výpočet čísla kliknutého riadku

Premenná p v kóde 6.2 reprezentuje bod kliknutia (jeho x-ovú a y-ovú súradnicu). Premenná r je obdĺžnik určujúci pozíciu okna, na ktoré bolo kliknuté. Premenná *_yChar* reprezentuje jednotku skrolovania na y-ovej osi, a premenná *_yPos* reprezentuje aktuálny stav skrolovania.

Následne sa pomocou rekurzívnej metódy `nodeCollapseTest()` iteruje cez všetky uzly stromu a kontroluje sa, či číslo prvého riadku daného uzlu je zhodné s číslom kliknutého riadku. Ak áno, je daný uzol zbalený.

6.4.2 Dialógové okná

V tomto projekte je využitých niekoľko dialógových okien. Ich formátovanie, teda font, rozloženie jednotlivých komponentov ako napríklad popisné texty, editovacie riadky atď. je popísané v súbore typu `.rc`, konkrétne sa jedná o súbor `lang.rc`.

V rámci týchto súborov je aj udané, na akých súradniciach sa zobrazí toto dialógové okno, ale Altap Salamander SDK umožňuje vycentrovať dialógové okno vzhľadom k jeho rodičovskému oknu, čo je tu využité.

Altap Salamander SDK poskytuje dve funkcionality, ktoré je dôležité spomenúť v súvislosti s dialógovými oknami, a to je naviazanie obsahu editovateľných komponentov na premennú v programe a validácia hodnôt. Naviazanie funguje tak, že pri zobrazení dialógového okna sa ako predvolený obsah zobrazí hodnota danej premennej a po potvrdení okna sa do premennej automaticky zapíše aktuálna hodnota obsahu komponentu v okamihu potvrdenia. Validácia funguje tak, že pri potvrdzovaní dialógového okna sa volá validačná metóda, ktorá má prístup k hodnotám editovateľných komponent a vie skontrolovať ich validitu podľa zvolených kritérií.

Dialógové okná sú použité na dva primárne účely, jeden je pri výbere toho, ktoré súbory sa majú porovnávať, zatiaľ čo druhý je pri zmene nastavení pluginu.

Kapitola 7

Testovanie

Testovanie implementovaného pluginu prebiehalo na viacerých úrovniach. Testovalo sa užívateľské rozhranie, funkčnosť porovnávacieho algoritmu a výkon programu. Testovanie bolo vykonávané viacerými užívateľmi, čo pomohlo odhaliť niektoré chyby, ktoré neboli odhalené pri základnom testovaní autorom práce.

Užívateľské rozhranie

Pri testovaní užívateľského rozhrania bol kladený dôraz na správnu funkčnosť komponentov:

- funkčnosť skrolovania (posuvník, koliečko myši, klávesnica),
- správne vykresľovanie pri skrolovaní (ostávajú po skrolovaní artefakty? atď.),
- chovanie pri zmene veľkosti okna,
- synchronizácia skrolovania medzi súbormi pri zobrazovaní výsledkov porovnania,
- funkčnosť dialógových okien (zobrazujú sa správne popisy, informácie, atď.?),
- funkčnosť klávesových skratiek.

Porovnávací algoritmus

Najprv boli testované základné rozdiely tak, že boli vytvorené dvojice testovacích súborov, z ktorých každá obsahovala iba jednu jednoduchú zmenu.

Zmeny testované pre XML súbory:

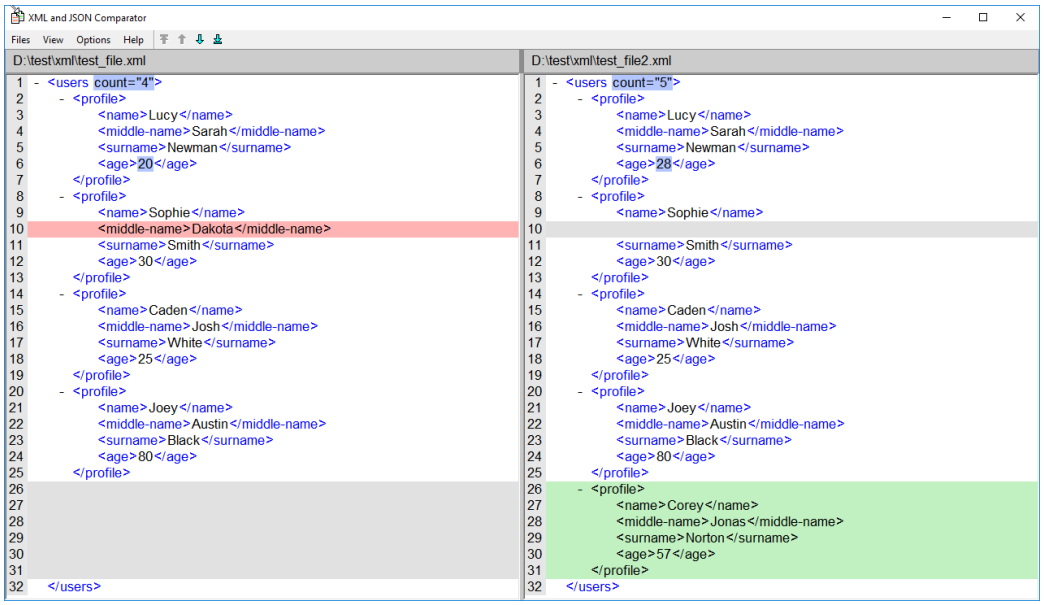
- pridanie uzlu,
- odstránenie uzlu,
- zmena poradia uzlov,
- zmena hodnoty atribútu,
- zmena hodnoty textového uzlu.

Zmeny testované pre JSON súbory:

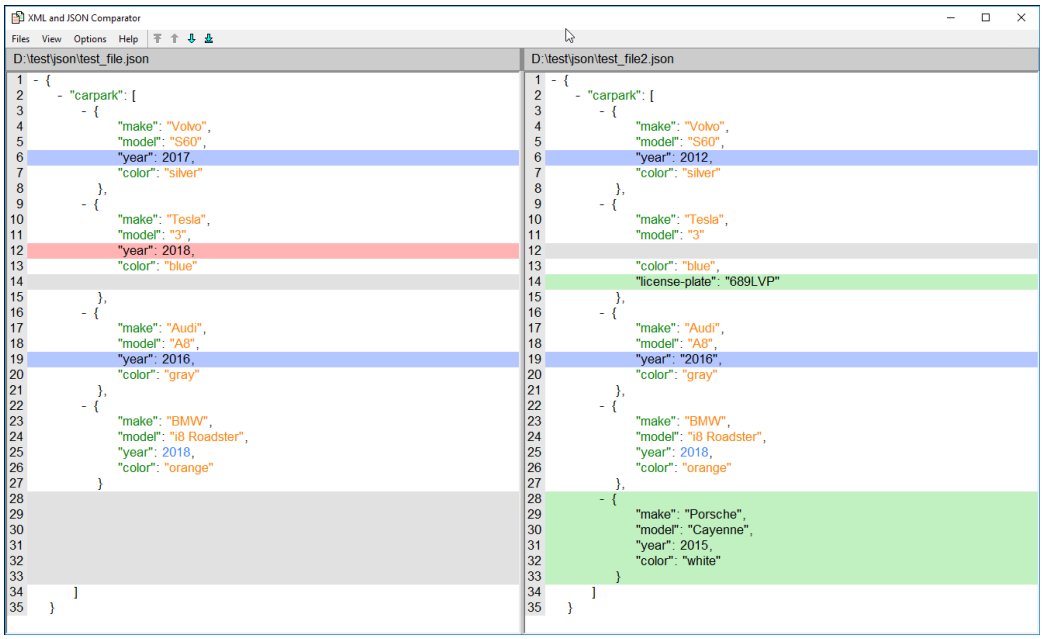
- pridanie položky,
- odobratie položky,
- zmena poradia položiek,
- zmena hodnoty položky,
- zmena typu položky.

Potom bolo vytvorených niekoľko dvojíc súborov, ktoré spájali niekoľko takýchto zmien. Na obrázku 7.1 môžme vidieť výsledok porovnávania dvoch XML súborov, ktoré spájajú

viacero zmien a na obrázku 7.2 vidíme výsledok pre JSON súbory. Súbory použité na testovanie je možné nájsť na sprievodnom CD.



Obr. 7.1: Výsledok porovnávania dvoch XML súborov demonštrujúci viacero typov možných zmien.

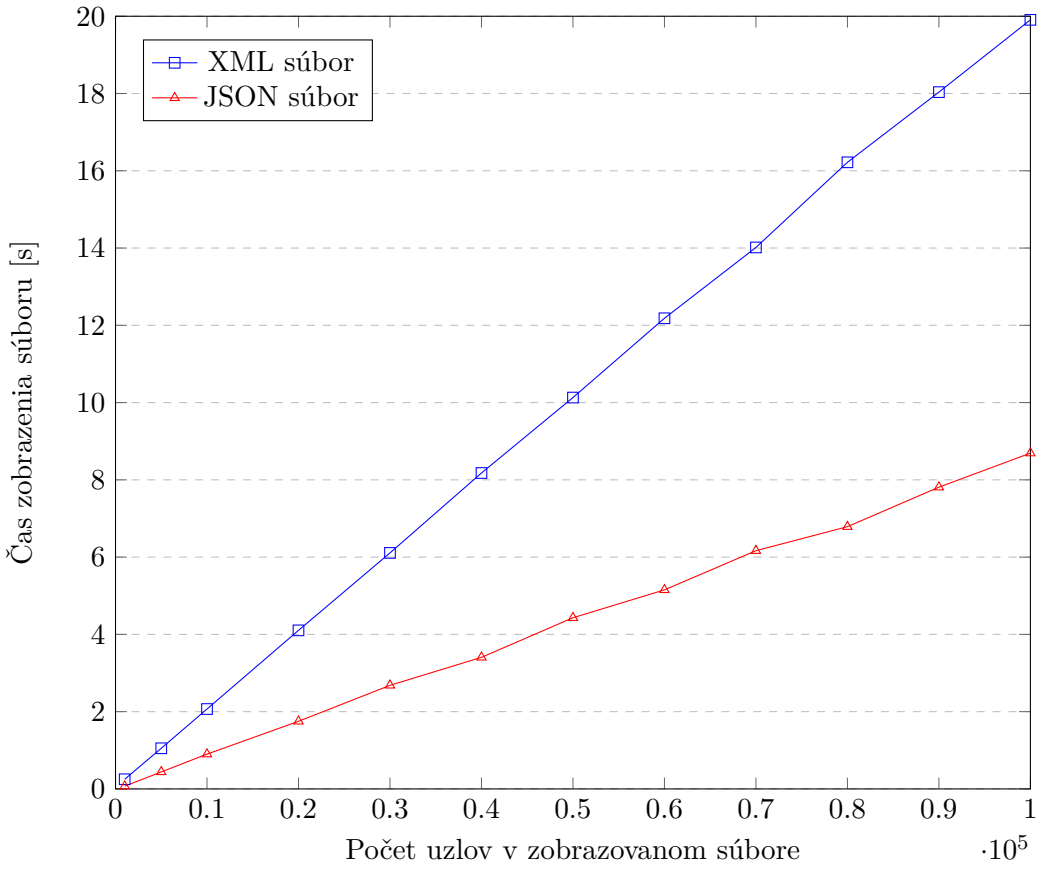


Obr. 7.2: Výsledok porovnávania dvoch JSON súborov demonštrujúci viacero typov možných zmien.

Výkon

Pre testovanie výkonu boli vytvorené generátory XML a JSON súborov v jazyku Java, ktoré vedia na základe predaného cieľového počtu uzlov a percenta odlišnosti vygenerovať dva súbory pre testovanie porovnávania. Oba z týchto súborov majú počet uzlov blízky cieľovému počtu uzlov a druhý súbor sa líši od prvého zhruba o cieľové percento odlišnosti, vztiahnuté k počtu uzlov. Zo stromu sa vyberú náhodne zvolené elementy, a aplikuje sa na ne náhodne zvolená zmena (napríklad odstránenie detského elementu, pridanie atribútu, zmena hodnoty atribútu). V prípade JSON formátu sa aplikujú analogické zmeny, ako napríklad odstránenie položky objektu, pridanie položky objektu, zmena hodnoty položky.

Pomocou generátora bola vytvorená séria súborov s postupne rastúcim počtom uzlov, pre ktoré potom bol meraný čas, za ktorý je ich možné prehliadať. Výsledky meraní sú zobrazené v grafe 7.3. Podobne čas, za ktorý je možné tieto súbory porovnať je prezentovaný v grafe 7.4.



Obr. 7.3: Čas čakania na zobrazenie súboru v závislosti na počte uzlov, ktoré obsahuje.

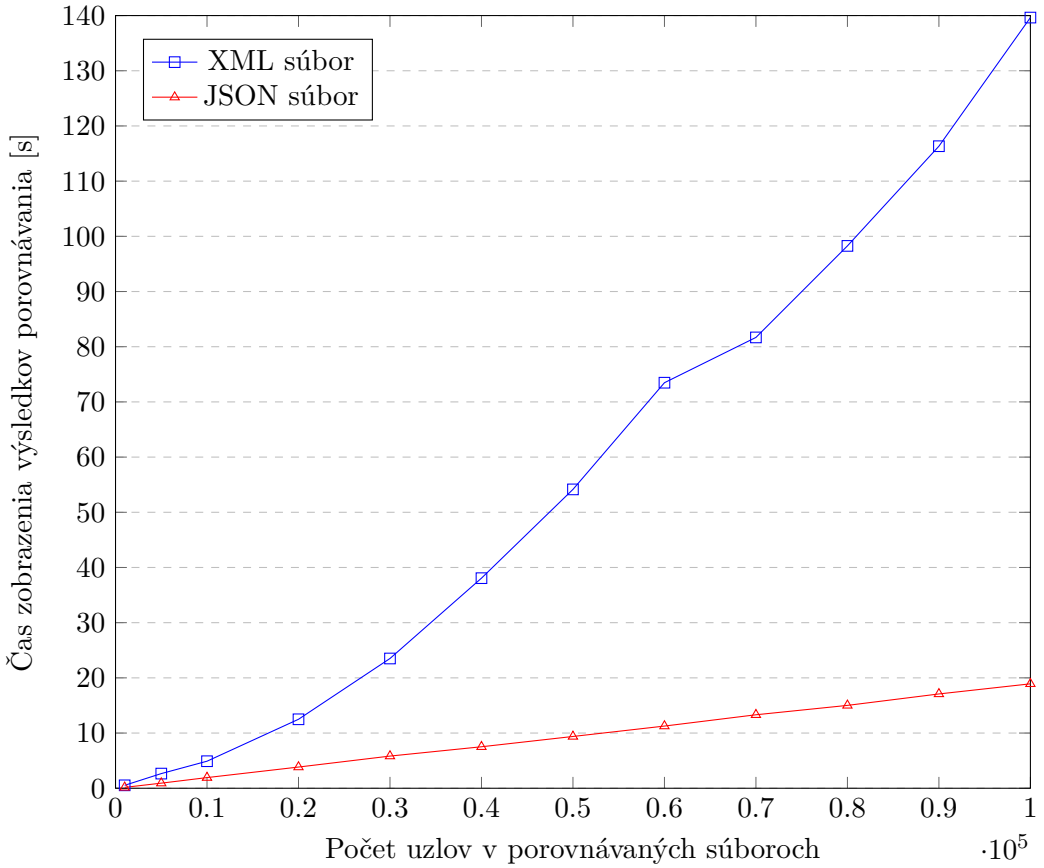
Plugin bol testovaný aj pre väčšie rozsahy než sú zobrazené v grafe a je ich teoreticky schopný spracovať, napríklad zobrazovanie XML súboru s počtom uzlov 5 miliónov a veľkosťou 227 MB mu trvá približne 20 minút.

Keďže však tento plugin typicky nebude spúšťaný v pozadí, teda užívateľ po spustení čaká na výsledky, je plugin rozumne použiteľný do času zobrazenia výsledku asi 10 sekúnd. Teda je možné rozumne zobrazovať XML súbory s 50 tisíc uzlami, čo zodpovedá veľkosti

súboru približne 2 MB, a porovnávať XML súbory do rozsahu okolo 12-13 tisíc uzlov, čo je súbor s veľkosťou približne 550 KB.

V prípade JSON formátu je 10 sekundový limit dosiahnutý pri zobrazovaní súborov s asi 110 tisíc uzlami (veľkosť 3.2 MB) a porovnávaní do rozsahu uzlov okolo 50 tisíc, čo zodpovedá súboru veľkosti približne 1.5 MB.

Generátor aj súbory, ktoré boli použité na meranie rýchlosti zobrazenia v grafoch 7.3 a 7.4 je taktiež možné nájsť na sprievodnom CD.



Obr. 7.4: Čas čakania na zobrazenie výsledku porovnania dvoch súborov v závislosti na počte uzlov, ktoré obsahujú.

Kapitola 8

Záver

Cieľom tejto práce bolo rozšíriť funkcionalitu správcu súborov Altap Salamander o zobrazovanie a porovnávanie XML a JSON súborov. Na dosiahnutie tohto cieľa bolo nutné sa zoznámiť s technológiou určenou na vývoj zásuvných pluginov Altap Salamander SDK.

Práca bola implementovaná v jazyku C++, na ktorom je založená platforma Altap Salamander SDK a bolo použitých niekoľko knižníc, ktoré umožňujú spracovávanie XML a JSON súborov. V priebehu vývoja tohto pluginu bolo využitých niekoľko ďalších nástrojov. Služba Bitbucket bola využitá na správu verzií kódu, program Visual Studio 2015 slúžil ako vývojové prostredie a nástroj CMake bol použitý pre kompiláciu jednej z knižníc použitých v tomto projekte.

Okrem nástrojov bolo treba vybrať aj vhodný algoritmus na porovnávanie XML a JSON súborov. Kvôli zložitosti problému porovnávania XML súborov bol pre implementáciu vybratý už existujúci algoritmus (a využitá jeho existujúca implementácia). Naopak formát JSON má jednoduchšiu logiku porovnávania, a tak bol na porovnávanie JSON súborov implementovaný vlastný algoritmus.

Výsledkom tejto práce je funkčný zásuvný modul splňujúci vytýčené ciele. Zásuvný modul však má niekoľko oblastí, ktoré aktuálne nie sú podporované a pri jeho rozširovaní by boli primárnymi prioritami: nepodporuje textové porovnávanie textových uzlov a unicode zobrazenie pre JSON súbory.

V rámci možných vylepšení tejto práce by bolo vhodné uvažovať o rozšírení možnosti XML porovnávania o alternatívny algoritmus, ktorý by umožňoval detekovať zmeny v názvoch uzlov. V aktuálnej implementácii algoritmu sú totiž dva elementy, ktoré majú úplne zhodný obsah, ale rozdielne meno, považované za nespojiteľné. Toto je síce správne z hľadiska špecifikácie XML, keďže meno identifikuje element, avšak môžu byť aj situácie, keď by užívateľa viac zaujímalo, že tieto elementy sa obsahovo zhodujú, len majú rozdielne meno.

V oblasti užívateľského rozhrania by bolo vhodné umožniť užívateľovi označiť a kopírovať zobrazovaný text. Ďalším možným rozšírením by bolo implementovať porovnávanie medzi panelmi takým spôsobom, že v jednom paneli sa označí viac súborov a potom sa vždy ku každému z nich hľadá súbor s rovnakým menom v druhom paneli, s ktorým je porovnaný (v prípade, že neexistuje, sa môže vypísať varovanie a pokračuje sa ďalej). Táto funkcionalita by uľahčila hromadné porovnávanie súborov.

Veľkou pomocou užívateľovi by taktiež bola implementácia vyhľadávania v súboroch, ktorá by mohla napríklad poskytovať možnosť hľadať iba v rámci mien elementov, hodnôt atribútov, atď.

Literatúra

[1] ALTAP: *Altap Salamander SDK*. [Online; navštíveno 02.04.2018].
URL https://www.altap.cz/cz/salamander/downloads/sdk/

[2] ALTAP: *Altap Salamander – Výkonný dvoupanelový správce souborů*. [Online; navštíveno 01.04.2018].
URL https://www.altap.cz/

[3] ALTAP: *Hlavní okno (snímek obrazovky)*. [Online; navštíveno 15.04.2018].
URL https://www.altap.cz/cz/salamander/screenshots/altap-salamander-main-window/

[4] ALTAP: *Seznam změn pro Altap Salamander*. [Online; navštíveno 22.04.2018].
URL https://www.altap.cz/cz/salamander/changelogs/

[5] Altova: *XML Compare Tool*. [Online; navštíveno 23.04.2018].
URL https://www.altova.com/xmlspy-xml-editor/compare-xml

[6] Berners-Lee, T.: *Email – Re: SGML/HTML docs, X Browser*. [Online; navštíveno 24.04.2018].
URL http://lists.w3.org/Archives/Public/www-talk/1991NovDec/0020.html

[7] Cobéna, G.; Abiteboul, S.; Marian, A.: Detecting Changes in XML Documents. In *Proceedings of the 18th International Conference on Data Engineering*, IEEE, Únor 2002, ISBN 978-0769515311, s. 41–52.

[8] Cover, R.: *SGML and XML as (Meta-) Markup Languages*. [Online; navštíveno 28.04.2018].
URL http://xml.coverpages.org/sgml.html

[9] Crockford, D.: *Douglas Crockford: The JSON Saga*. [Online; navštíveno 28.04.2018].
URL https://www.youtube.com/watch?v=-C-JoyNuQJs

[10] Crockford, D.: *Introducing JSON*. [Online; navštíveno 10.04.2018].
URL https://www.json.org/

[11] Daley, R. C.; Neumann, P. G.: A General-Purpose File System For Secondary Storage. In *AFIPS '64 (Fall, part I) Proceedings of the October 27-29, 1964, fall joint computer conference, part I*, 1965, ISBN 978-1-4419-2881-8, s. 213–229.

[12] Daniel599: *Discussion post – Using fifo-map*. [Online; navštíveno 26.04.2018].
URL https://github.com/nlohmann/json/issues/485#issuecomment-333652309

[13] fffaraz: *Awesome C++*. [Online; navštíveno 25.04.2018].
URL https://github.com/fffaraz/awesome-cpp#xml

[14] Free Software Foundation, Inc.: *Dired, the Directory Editor*. [Online; navštíveno 29.04.2018].
URL https://www.gnu.org/software/emacs/manual/html_node/emacs/Dired.html

[15] Giebel, P.: *MS-DOS ON A 486 DX2-66*. [Online; navštíveno 15.04.2018].
URL https://www.stimpyrama.org/blog/17-computer/127-msdos9

[16] IBM Knowledge Center: *History and Relationships of SGML, HTML and XML*. [Online; navštíveno 19.04.2018].
URL https://www.ibm.com/support/knowledgecenter/en/SS6RBX_11.4.3/com.ibm.sa.xml.design.doc/topics/c_history.html

[17] Khanal, S.: *Best alternative file managers for Windows 10*. [Online; navštíveno 15.04.2018].
URL https://windowsreport.com/top-file-managers-windows-10/

[18] Lee, S. K.; Kim, D. A.: X-Tree Diff+: Efficient Change Detection Algorithm in XML Documents. In *Embedded and Ubiquitous Computing*, Springer, Srpen 2006, ISBN 978-3-540-36679-9, s. 1037–1046.

[19] Maler, E.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: Extensible Markup Language (XML) 1.0 (Fifth Edition). Technická zpráva, W3C, Listopad 2008, https://www.w3.org/TR/xml.

[20] Ryšavý, J.: *Diskusný príspevok – Diakritika v názvoch súborov*. [Online; navštíveno 26.04.2018].
URL https://forum.altap.cz/viewtopic.php?t=8486

[21] SyncRO Soft SRL: *Oxygen Compare and Merge Tools*. [Online; navštíveno 23.04.2018].
URL https://www.oxygenxml.com/xml_editor/xml_diff_and_merge.html

[22] Tanenbaum, A. S.; Bos, H.: *Modern Operating Systems*. Pearson, Čtvrté vydání, 2014, ISBN 978-0133591620, 6-14 s.

[23] Thacker, C. P.; McCreight, E. M.; Lampson, B. W.; aj.: Alto: A personal computer. In *Computer Strutures: Principles and Examples*, McGraw-Hill, 1982, s. 549–572.

[24] Wang, Y.; DeWitt, D. J.; Cai, J.-Y.: X-Diff: An Effective Change Detection Algorithm for XML Documents. In *Proceedings. 19th International Conference on Data Engineering*, IEEE, Březen 2003, ISBN 978-0780376656, s. 519–530.