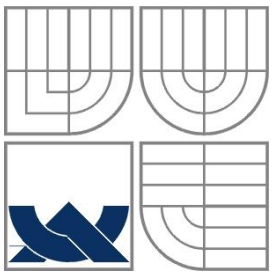
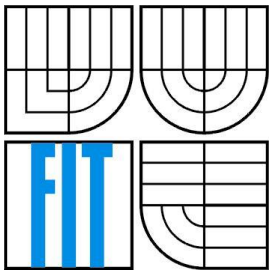


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROHLEDÁVÁNÍ TEXTOVÝCH SOUBORŮ POMOCÍ REGULÁRNÍCH VÝRAZŮ

REGULAR EXPRESSION BASED SEARCHING IN TEXT FILES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Ota Šimek

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jaroslav Rozman, Ph.D.

BRNO 2016

Abstrakt

Tato práce se zabývá návrhem, analýzou a implementací programu, který slouží pro vyhledávání určitých částí textu v textových a xml souborech. Pro vyhledávání je nutné vytvářet šablony pomocí dvou technik porovnávání textu. Jedná se o regulární výrazy a XPath výrazy. Toto je popsáno v první kapitole. Aplikace umožňuje zpracovávat velké množství souborů a také soubory s velkým obsahem.

Abstract

This paper desribes the design, analysis and implementation of the program, whichis used to search for specific parts of text in text and xml files. To search, you must create a template using two text-matching technigues. These are the regular expressions and XPath expressions. This is desribed in the first chapter. Application allows to process large number of files or files with a high content.

Klíčová slova

xml, XPath, regulární výraz, XPath výraz, šablona, prohledávání souborů, mapování, paralelní přístup, datové toky, textové soubory, xml soubory

Keywords

xml, XPath, regular expression, XPath expression, template, file browsing, mapping, parallel access, data stream, text files, xml files

Citace

Šimek Ota: Prohledávání textových souborů pomocí regulárních výrazů, bakalářská práce, Brno, FIT VUT v Brně, 2016

Prohledávání textových souborů pomocí regulárních výrazů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ota Šimek
13.5.2016

Poděkování

Tímto chci poděkovat vedoucímu práce Ing. Jaroslava Rozmana, Ph.D. za jeho pomoc při řešení a ochotu při výběru tématu.

Obsah

1 Úvod

Tématem práce je prohledávání textových nebo XML souborů. V dnešní době už každý editor umožňuje vyhledávat požadovaný řetězec v určitém souboru. Co se ale stane, pokud chce uživatel vyhledávat jeden řetězec, popřípadě více, ve velkém množství souborů. Projít každý soubor a hledat určité části není schůdnou cestou.

Cílem práce je vytvořit aplikaci, která umožní prohledávat velké množství souborů v různých podsložkách, případně velkých souborech. Vyhledávání správné části souboru je dáno na základě šablon, které jsou vytvořeny uživatelem. Šablony jsou vytvořeny pomocí regulárních či XPath výrazů.

Takováto aplikace by měla umožnit monitorovat například logy některých aplikací, které mohou produkovat mnoho souborů. Aby nemusel uživatel logy procházet po jednom, případně hledat ve velkém souboru, využije regulárních výrazů či XPath jazyka, aby získal důležité informace z logů. Použití regulárních a XPath výrazů by mělo umožnit, že se bude dát nalézt téměř cokoli.

I když jsou nutné k používání jisté znalosti, nemění to nic na faktu, že by aplikace měla být jednoduchá a intuitivní, tedy jednoduché uživatelské rozhraní, kde je vše „hned po ruce“ a uživatel se rychle zorientuje již při prvním pohledu.

Druhá kapitola popisuje dvě výše zmiňované technologie, které jsou důležité k vytváření vyhledávacích šablon. V první části kapitoly je popsán jazyk XPath a jeho syntaxe. O regulárních výrazech je psáno v druhé části zmíněné kapitoly.

Ve třetí kapitole je popsán návrh aplikace. Tuto kapitolu můžeme také rozdělit na dvě části. V první části je popsán návrh knihovny, která umožňuje vyhledávání zadaných hodnot. V druhé části třetí kapitoly je vysvětlen návrh uživatelského rozhraní.

Čtvrtá kapitola pojednává o implementaci samotné aplikace. Je zde možné vidět použité technologie.

Pátá a předposlední kapitola ukazuje provedené testy nad hotovou aplikací.

V poslední kapitole je shrnuta tato práce.

2 Popis technologií

V této kapitole jsou popsány dvě techniky, které jsou využívány k vyhledávání v souborech. Díky těmto technikám uživatel vytvoří šablony potřebné k vyhledání správné části textu. Jde o počítačový jazyk XPath, který umožňuje adresovat části xml dokumentu. Druhou technologií jsou regulární výrazy. Regulární výraz je řetězec popisující větší množinu řetězců. Tyto řetězce pomohou k prohledávání textových souborů.

2.1 XPath

XPath je počítačový jazyk. Jednoduše řečeno jde o jazyk, který nám umožní vyjádřit cestu mezi určitými XML uzly. Popřípadě mezi uzlem a atributem. XPath může sloužit k více účelům, přičemž tento výše zmíněný je asi nejdůležitější a nejpoužívanější. Mimo ten také poskytuje základní nástroje pro práci s řetězci, čísly nebo booleovskými prvky. Na první pohled může připomínat cestu k adresářům, jak je známe ze souborových systémů. Výsledkem však jsou XML elementy či atributy.

Je používán prakticky všude, kde je potřeba vyhledávat ve struktuře dokumentu XML.

2.1.1 Cesta uzlu

Asi nejdůležitější konstrukcí XPath je cesta k uzlu XML. Každá tato cesta obsahuje určitý počet kroků. Každý krok se skládá z identifikátoru osy, testu uzlu a podmínky. Identifikátor osy a podmínka jsou nepovinné. Jednotlivé kroky výrazu se spojují dohromady pomocí lomítek. Zpracování XPath výrazu začíná z aktuálního uzlu, pokud není lomítky určeno jinak. Kroky se vyhodnocují zleva doprava. Výsledek může nabývat libovolného typu uzlu. Můžeme tedy narazit nejen na element a atribut, ale také například na textový uzel.

2.1.2 Vyhodnocení výrazu

Celý výraz je vyhodnocován po krocích zleva doprava. Prvním krokem je určení uzlů, které se budou v určitém kroku zpracovávat. K tomu slouží identifikátor osy. Výchozí hodnotou identifikátoru osy jsou všechny podřízené uzly aktuálního uzlu. Dále se pomocí testu uzlu tato množina omezí. Nakonec se ještě upraví na základě dodatečné podmínky. Všechny uzly, které jsou ve výsledné množině, pokračují a jsou zpracovávány v dalším kroku.

2.1.3 Syntaxe

Oddělení kroků – kroky lze oddělovat dvěma způsoby:

- Lomítko **/** – jednotlivé kroky lze oddělit lomítkem, což je stejné jako u adresářových struktur. V případě, že lomítko je první znak celého výrazu, jedná se o určení, že výraz není vztažen k aktuálnímu prvku. Takovýto výraz se vztahuje ke kořenu dokumentu.
- Dvě lomítka **//** – slouží k překonání víceúrovňové struktury. Dokážeme díky tomuto překročit některé uzly. Pokud jsou dvě lomítka na začátku, jde opět o cestu od kořene dokumentu.

Identifikátory osy – předchází testu uzlu. Určuje směr procházení XML dokumentu.

Identifikátor	**Vyhodnocené uzly**
child::	Přímí potomci aktuálního uzlu. Výchozí hodnota identifikátoru osy
self::	Aktuální uzel.
descendand::	Všichni potomci aktuálního uzlu.
descendand-or-self::	Všichni potomci aktuálního uzlu a aktuální uzel.
ancestor::	Všichni předci aktuálního uzlu.
ancestor-or-self	Všichni předci aktuálního uzlu a aktuální uzel.
parent::	Rodič aktuálního uzlu.
following::	Všechny uzly, které se nacházejí za aktuálním uzlem.
preceding::	Všechny uzly, které se nacházejí před aktuálním uzlem.
following-sibling::	Všichni následující sourozenci aktuálního uzlu.
preceding-sibling::	Všichni předchozí sourozenci aktuálního uzlu
attribute::	Atributy aktuálního uzlu.
namespace::	Deklarované jmenné prostory.

Tabulka č. 1 – Identifikátory osy

Testy uzlu – dále určuje množinu uzlů, která byla vybrána na základě identifikátoru osy.

- Uzel určený názvem – nejzákladnější test uzlu. Jsou vybrány všechny XML elementy s odpovídajícím názvem. Zapisuje se jednoduše zapsáním názvu elementu.
- Uzel určený typem – bere ohled na typ uzlu. Vybírá pouze typ, který je určen. Zapisuje se názvem typu uzlu, za kterým následují prázdné kulaté závorky.

Podmínky – použije se v případě, že je nutné upravit předchozí množinu. Zapisuje se do hranatých závorek. Hranaté závorky mohou také určovat polohu elementu, v případě, že je v závorkách pouze číslo.

Operátory – používají se obvyklé operátory jako ve všech jiných programovacích jazycích. Nejčastěji se používají v podmínkách. Pořadí je možné upravovat závorkami.

Funkce – je možné použít celou řadu užitečných funkcí, většinou používaných v podmínkách. Příklad některých funkcí:

Funkce	**Význam**
`string()/boolean()/number()`	Převede hodnotu na řetězec/bool/číslo
`sum()`	Konverze na čísla a vrácen součet
`true()/false()`	Logická 1/0
`position()`	Pořadové číslo aktuálního uzlu

Tabulka č. 2 – Xpath funkce

2.1.4 Zkrácení syntaxe

Z důvodu častého opakování některých konstrukcí je možné použít zkrácené tvary. Tyto slouží zejména ke zpřehlednění výrazu.

Znak	**Význam**
`Hvězdička *`	výběr všech dostupných elementů.
`Tečka .`	jedná se odkaz na sebe sama
`Dvě tečky ..`	umožňují pohyb o jednu úroveň výše
`Zavináč @`	určuje atribut aktuálního uzlu

Tabulka č. 3 – Zkrácení syntaxe

Některé prvky z výše uvedené tabulky mají další speciální vlastnosti. Tečka nahrazuje celý krok, tudíž není možné přidat dodatečné podmínky. Z toho plyne, že dvě tečky opět nahrazují celý krok.

Zavináč nahrazuje identifikátor osy `attribute::`. Obvykle za ním následuje název atributu jako test uzlu.

2.2 Regulární výrazy

Regulární výrazy umožňují efektivně hledat v textových řetězcích pomocí vzoru. Nejčastěji jsou používány k vyhledávání textu. Může jít o zjištění, zda text odpovídá zadanému regulárnímu výrazu, popřípadě zjištění pozice, kde nastává shoda s výrazem. Také se používá k manipulaci s textem. Je možná záměna textu v podvýrazu regulárního výrazu, popřípadě export všech shod do proměnné. V podstatě umožňuje zadat tvar řetězce, který může nabýt více podob. Regulární výraz se skládá z literálů textu a speciálních znaků.

Hledání odpovídající sekvence znaků začíná na začátku řetězce. Následně se hledá první shoda se zadaným výrazem.

2.2.1 Metaznaky

Metaznaky jsou speciální znaky v regulárních výrazech. Jde o znaky, které ve výrazu nereprezentují vlastní hodnotu, ale mají speciální význam. V případě, že chceme použít hodnotu těchto znaků, musí se escapovat pomocí lomítka (**/**). V regulárních výrazech existují tyto metaznaky:

- **Tečka .** – určuje právě jeden libovolný znak.
- **Kvantifikátory** – určují, kolikrát se smí určitý znak opakovat. Opakovaný znak je vždy před kvantifikátorem. Kvantifikátory jsou popsány v kapitole 2.2.2.
- **Hranaté závorky `[]`** – určují skupinu znaků, které se zde mohou vyskytovat. Znaky uvnitř závorek je možné doplnit stříškou (**^**), která určí, že jde o skupinu znaků, které se nesmí vyskytnout. Je možné také definovat skupinu znaků, které jdou po sobě v abecedě a to přidáním mínus (**-**) mezi první a poslední znak posloupnosti. Některé nejpoužívanější skupiny znaků mají speciální značení a tyto jsou popsány v kapitole 2.2.3
- **Kulaté závorky `()`** – slouží k uzavření určité sekvence znaků do bloku, který můžeme následně pomocí kvantifikátoru opakovat.
- **Svislice `|`** – umožňuje určit několik variant textu.
- **Stříška `^`** – určuje začátek řetězce. Žádný znak není před znakem, který následuje. Při kombinaci s hranatými závorkami slouží k určení nežádoucích znaků.
- **Dolar `$`** – určuje konec řetězce. Žádný znak není za znakem, který je před tímto dolarem.

2.2.2 Kvantifikátory

Aby bylo docíleno určité univerzálnosti, je možné zadat, kolikrát se může určitý znak opakovat. Takovýmto prvkům se říká kvantifikátory. Existují tyto typy kvantifikátorů:

Kvantifikátor	Počet opakování
`?`	minimálně 0krát, maximálně 1krát
`*`	minimálně 0krát, maximálně neomezeno
`+`	minimálně 1krát, maximálně neomezeno
`{n}`	právě `n`krát
`{m, n}`	minimálně `m`krát, maximálně `n`krát
`{m, }`	minimálně `m`krát, maximálně `n`eomezeno

Tabulka č. 4 – Kvantifikátory

Pokud výše uvedené kvantifikátory doplníme zprava otazníkem (`?`) získáme `líné kvantifikátory`. Jde o kvantifikátory, které zachytí minimální počet znaků, kterým odpovídají, zatímco kvantifikátory bez otazníku zachytí největší možné množství znaků vstupního textu.

2.2.3 Skupiny znaků

Skupiny znaků jsou definovány pomocí hranatých závorek. Kvůli přehlednosti a snadnějšímu používání jsou zavedeny speciální znaky, které zahrnují nejpoužívanější skupiny:

Znaky	Skupina	Ekvivalent
`\d`	číslice 0-9	`[0-9]`
`\D`	jakýkoliv znak mimo 0-9	`[^0-9]`
`\w`	znaky abecedy a čísla	`[a-zA-Z0-9]`
`\W`	jakýkoliv znak mimo abecedu a čísla	`[^a-zA-Z0-9]`
`\s`	bílé znaky	
`\S`	jakýkoliv znak mimo bílých znaků	

Tabulka č. 5 – Skupiny znaků

3 Návrh aplikace

Tato kapitola se zabývá analýzou problému a návrhem aplikace. Jsou zde popsány požadavky na knihovnu i uživatelské rozhraní. Dále samotné navržení struktury těchto prvků aplikace.

3.1 Analýza

Cílem práce je vytvořit aplikaci, která umožní prohledávat velké množství souborů v různých podsložkách, případně velké soubory. Následně tyto soubory předat uživateli v přehledné formě. Vyhledávání správné části souboru je dáno na základě šablon, které jsou vytvořeny uživatelem. Po nalezení výskytu se hledaná část rozloží pomocí mapování výsledku. Šablony jsou vytvořeny pomocí regulárních či XPath výrazů.

Ačkoli je pro použití aplikace nutno mít určité znalosti, i nadále by měla tato zůstat jednoduchá a intuitivní. Jde tak převážně o přehledné uživatelské rozhraní, kde se uživatel zorientuje již při prvním seznámení. V aplikaci bude příručka pro uživatele, která pomůže s tvořením šablon.

3.2 Požadavky na knihovnu

Požadavky pro výslednou knihovnu je možné shrnout do následujících bodů:

- Zpracování objemných dat
- Zpracování složek a podsložek
- Hledání podle regulárních výrazů
- Hledání podle XPath
- Vrácení rozložených řádků podle seznamu mapování.

3.3 Požadavky na GUI

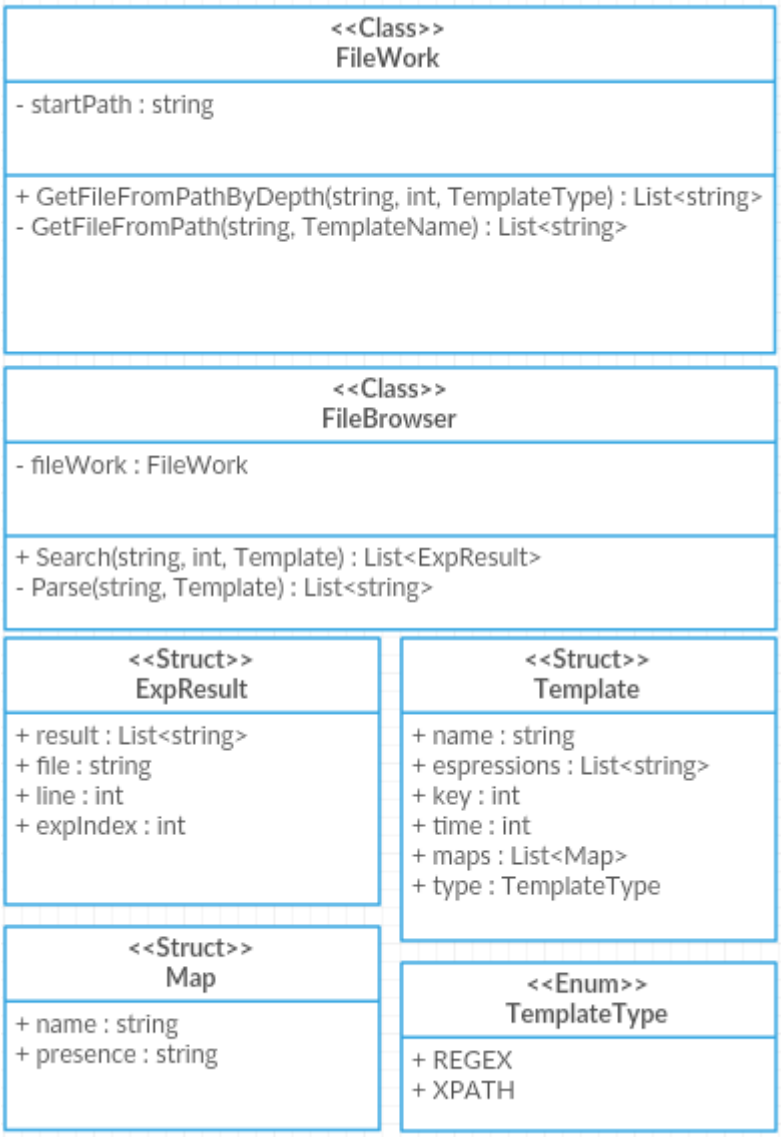
Požadavky pro výsledné uživatelské rozhraní je shrnuto v bodech níže:

- Využívání výše uvedené knihovny
- Umožňuje definovat a ukládat šablony
- Musí být možno vybrat následující parametry pro vyhledávání:
  - Startovní složka a hloubka prohledávání.
  - Šablonu pro prohledávání souborů.
  - Hloubku zanoření podsložek
- Výstup:
  - Seznam, který je možné řadit, stránkovat a prohledávat.

- Celkový počet výskytu dané šablony.
- Celková délka daných událostí.

3.4 Návrh knihovny

Struktura knihovny je popsána na obrázku č. 1.



Obrázek č. 1 – Struktura navržené knihovny

3.4.1 Třídy

Knihovna se bude skládat ze dvou tříd:

- **FileBrowser** – tato třída bude sloužit k samotnému vyhledávání a následnému vrácení hodnot. Funkce `Search(…)` bude využívat třídu `FilesWork` k vyhledání potřebných hodnot.

Následně hodnoty pošle funkci `Parse(…)`, které vrátí hodnoty rozložené na základě mapování a vrátí seznam vyhledaných hodnot.

- **FilesWork –** tato třída bude sloužit k práci se soubory. V privátní proměnné `startPath` si udržuje počáteční složku. Dále zde bude funkce `GetFilesFromPathByDepth(…)`, která vrátí strukturu podsložek a souborů v nich uložených. Slouží k vyhledání všech souborů od výchozí složky, na základě zadané hloubky. K vyhledávání pomůže také funkce `GetFileFromPath(…)`, která vrátí soubory z určité složky.

3.4.2 Struktury

Knihovna bude obsahovat tři struktury potřebné k vyhledávání.

- **Template –** bude obsahovat všechna důležitá data k uchování šablony - jméno šablony pro její identifikaci a dále výrazy, podle kterých se bude nakonec vyhledávat. Proměnné `key` a `time`, ve kterých bude uložen index klíče a času, podle nichž budou výsledky seřazeny. Každá šablona obsahuje mapování pro rozložení výsledků a také typ této šablony.
- **Map –** bude obsahovat všechna data potřebná k mapování hodnot výsledku. Jméno určeného prvku výsledku a jeho výskyt v nalezeném řetězci.
- **ExpResult –** slouží k uložení nalezeného výsledku. Také udržuje důležité informace o tomto výsledku. Je zde seznam rozložených prvků na základě mapování. Pro seřazení a kompletaci bude uložen soubor a řádek, ve kterém byl výsledek nalezen.

3.4.3 Výčet hodnot

Při přidávání některých hodnot uživatelem je nutné, aby vybral z předem definovaných. Proto bude knihovna obsahovat také výčet hodnot:

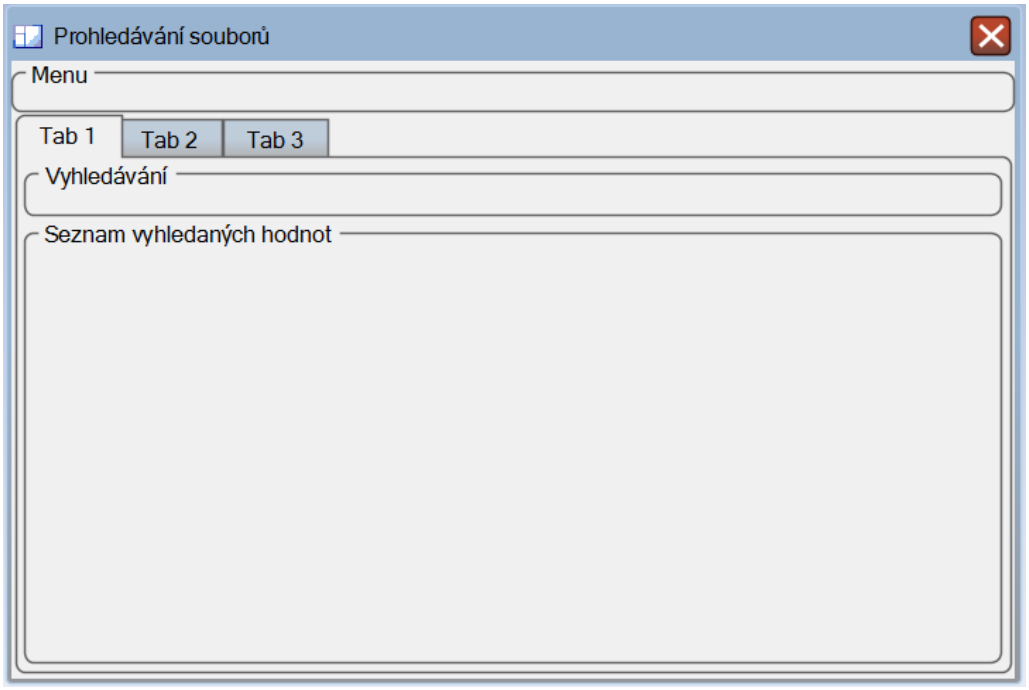
- **TemplateType –** určuje typ šablony. Na výběr je ze dvou možností `XPATH` nebo `REGEX`. Hodnotu typu zadává uživatel a je nutná pro výběr souborů, které se mají prohledávat.

3.5 Uživatelské rozhraní

Uživatelské rozhraní bude jednoduché a intuitivní, aby se práce v aplikaci nijak nekomplikovala a netrvala příliš dlouho.

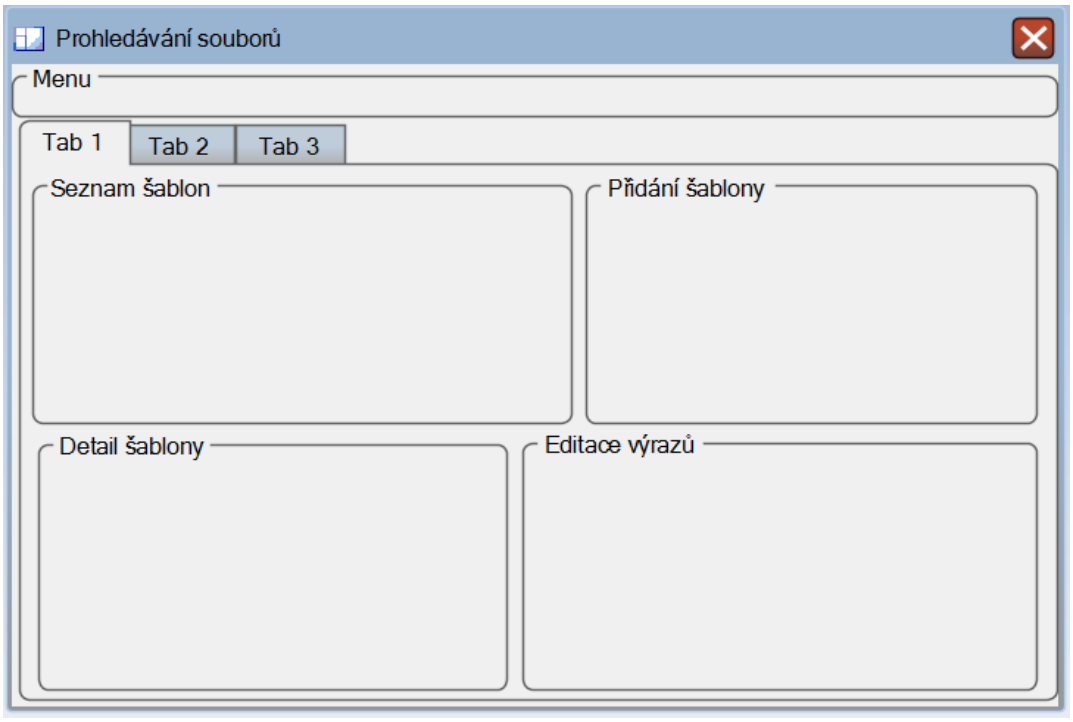
Aplikace se bude skládat pouze z jednoho formuláře. Tento formulář se dá rozdělit do dvou částí. V první části bude menu, ve kterém budou základní funkce aplikace. Druhá bude obsahovat záložky, které budou nahrazovat další formuláře.

- První záložka bude obsahovat samotné vyhledávání. V horní části bude možné vybrat složku, ze které se má vyhledávat a případně hloubka vyhledávání. Dále se zde vybere šablona, podle které se bude vyhledávat. V další části se zobrazí seznam nalezených hodnot.



Obrázek č. 2 – Záložka vyhledávání

- Ve druhé záložce uvidíme seznam všech šablon, které jsou definovány. Po výběru určité šablony se zobrazí její detail. V detailu šablony budou viditelné všechny její prvky. Po výběru jednoho z výrazu či mapování jej bude možné editovat v části editace výrazu. Zde bude také možno tyto vkládat. Přidání šablony najdeme vedle seznamu šablon. Veškerá práce se šablonami je obsažena na druhé záložce. Ukládání šablon bude ošetřeno automaticky po přidání, odebrání či editaci.



Obrázek č. 3 – Záložka Šablony

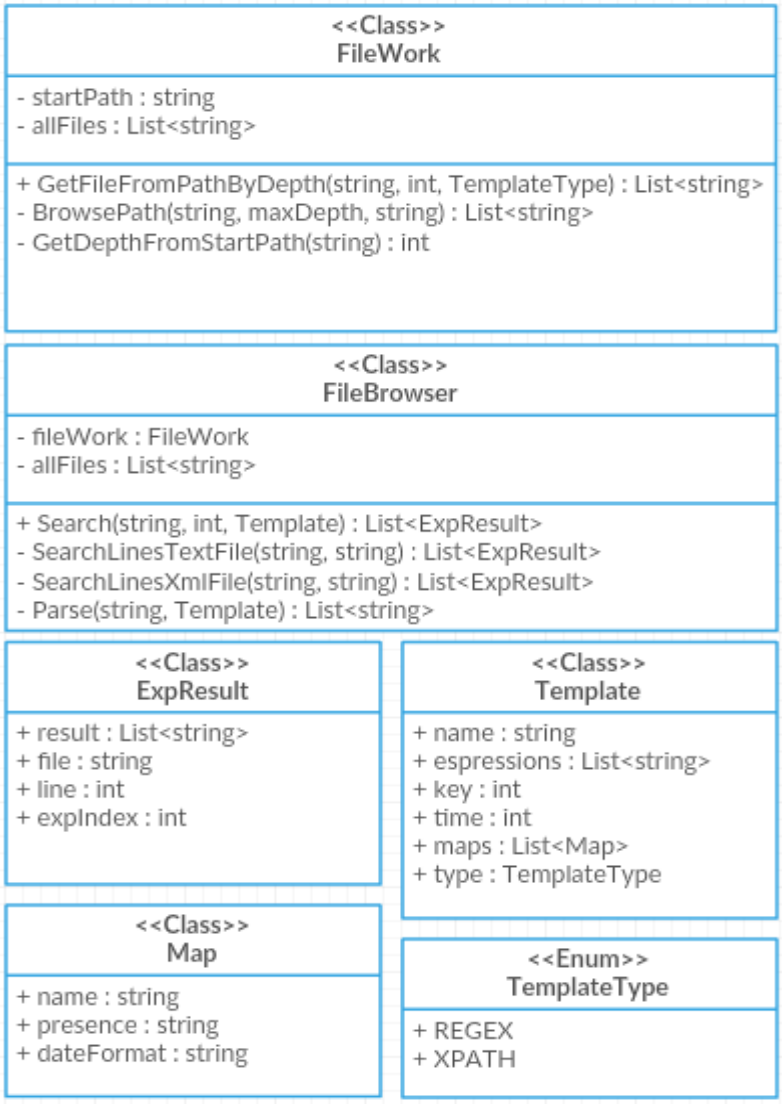
4 Implementace

V této kapitole jsou popsány principy aplikace a jak byla vytvořena. Nejprve je zde popsána implementace knihovny, která se stará o vyhledávání souborů a jejich prohledávání. Zadáním práce bylo také vytvořit uživatelské rozhraní. Implementace uživatelského rozhraní bude popsána v další části této kapitoly.

K vytvoření knihovny i uživatelského rozhraní bylo využito jazyka C#. Pro uživatelské rozhraní konkrétně Windows WinForm knihovna.

4.1 Knihovna

Výsledná knihovna byla vytvořena velice podobně, jak byla navrhována. V průběhu implementace se vyskytly určité problémy a bylo nutné lehce pozměnit strukturu knihovny.



Obrázek č. 4 – Skutečná struktura knihovny

4.1.1 Vyhledávání souborů

První krok, který aplikace provede, je vyhledání množiny všech souborů, které se budou prohledávat. Jedná se o soubory ze všech podsložek startovní složky. Vyhledávání omezuje tedy startovní složka, hloubka prohledávání a samotný typ šablony. Všechny tyto hodnoty zadává uživatel v uživatelském rozhraní. Následně se pošlou do knihovní funkce `Search(…)`. Tato funkce začne již zmíněným vyhledáním množiny prohledávaných souborů. K tomu je využita funkce `GetFilesFromPathByDepth(…)` ze třídy `FileWork`.

Tato funkce obdrží parametry zmíněné výše, důležité k prohledání struktury podsložek. Při volání této funkce se na začátku určí typ hledaného souboru, zda se jedná o `txt` nebo `xml` soubor. Následuje pomocná funkce `BrowsePath(…)`, která vyhledá všechny soubory ve struktuře. Strukturu prohledává od první složky, dokud nedosáhne požadované hloubky.

Funkce `BrowsePath(…)` byla nejdříve implementována jako rekurzivní funkce, která při nalezení nové neznámé cesty, zavolá sebe samou znovu a prohledává tuto cestu. Při následném testování se ukázalo, že při přibývajícím počtu podsložek je tato metoda stále více neefektivní.

Po testech tedy bylo nutné změnit implementaci této funkce. Implementace se změnila na první pohled jen málo, ale při hlubší analýze lze vidět, že jde o rozdíl, který šetří čas. Místo čistě rekurzivního volání funkce je použit paralelní přístup. To znamená, že všechny složky i podsložky jsou prohledávány zároveň. Díky této změně je dosažení výsledku poněkud rychlejší (viz 5.).

Funkce tedy obdrží startovní složku a tu paralelně projde. Všechny zde nalezené podložky projdou paralelním cyklem, pokud vyhovují kritériu hloubky. Poté uloží všechny soubory v této složce, které odpovídají typu hledaného souboru.

4.1.2 Prohledávání souborů

Dalším krokem po nalezení množiny prohledávaných souborů je samotné prohledání této množiny. Využit je znovu paralelní cyklus procházející všechny soubory. V případě, že šablona využívá více výrazů pro hledání, je volán další paralelní cyklus, který každý soubor prohledá paralelně pro každý výraz. Tím pádem se všechny výrazy prohledávají zároveň.

V těchto cyklech se ze všeho nejdříve vyhledá pomocí dvou pomocných funkcí vyhovující řádky. Tyto funkce jsou `SearchLinesTextFile(…)` a `SearchLinesXmlFile(…)`. Výběr funkce je opět dán typem šablony. Obě funkce pracují v podstatě stejným principem, akorát jsou využity odlišné techniky. K nalezení všech vyhovujících řádků je zde využit datový proud, ze kterého se pomocí cyklu načítají data, která se vyhodnocují. Datový proud je využit z důvodu, aby při vyhledávání ve velkém souboru se tento soubor neukládal do paměti. V případě gigového souboru by tato aplikace nemohla fungovat. Následně se správné řádky uloží do seznamu výsledků. Mimo samotný nalezený řádek se díky `ExpResult` uloží také název souboru, kde byl řádek nalezen. Po dosažení konce je datový kanál uzavřen a navrácen seznam nalezených řádků.

První z uvedených funkcí `SearchLinesTextFile(…)` využívá k prohledání klasickou třídu `StreamReader`. Díky této funkci se připojí datový kanál souboru a dále v cyklu čte řádek po řádku. Tato funkce prohledává pouze textové soubory a pro porovnávání řádků využívá regulárních výrazů. Regulární výrazy jsou realizovány pomocí třídy `Regex`.

Funkce `SearchLinesXmlFile(…)` naopak nevyužívá klasickou třídu. Pro účely vyhledávání pomocí XPath výrazů bylo nutné nalézt speciální knihovnu, která umožní připojení datového proudu a zároveň porovnávání s XPath. U klasické třídy pro použití XPath je nutné načíst celý soubor do paměti, což nám možnost velkého souboru neumožňuje. Na druhou stranu klasické využití datového čtení xml souboru je při spojení s XPath velice složité. Tato třída je ovšem již vytvořena, ale není standardem. Použita je tedy funkce `XPathReader`, která je volně ke stažení na webu MSDN [1].

4.1.3 Rozložení výsledků

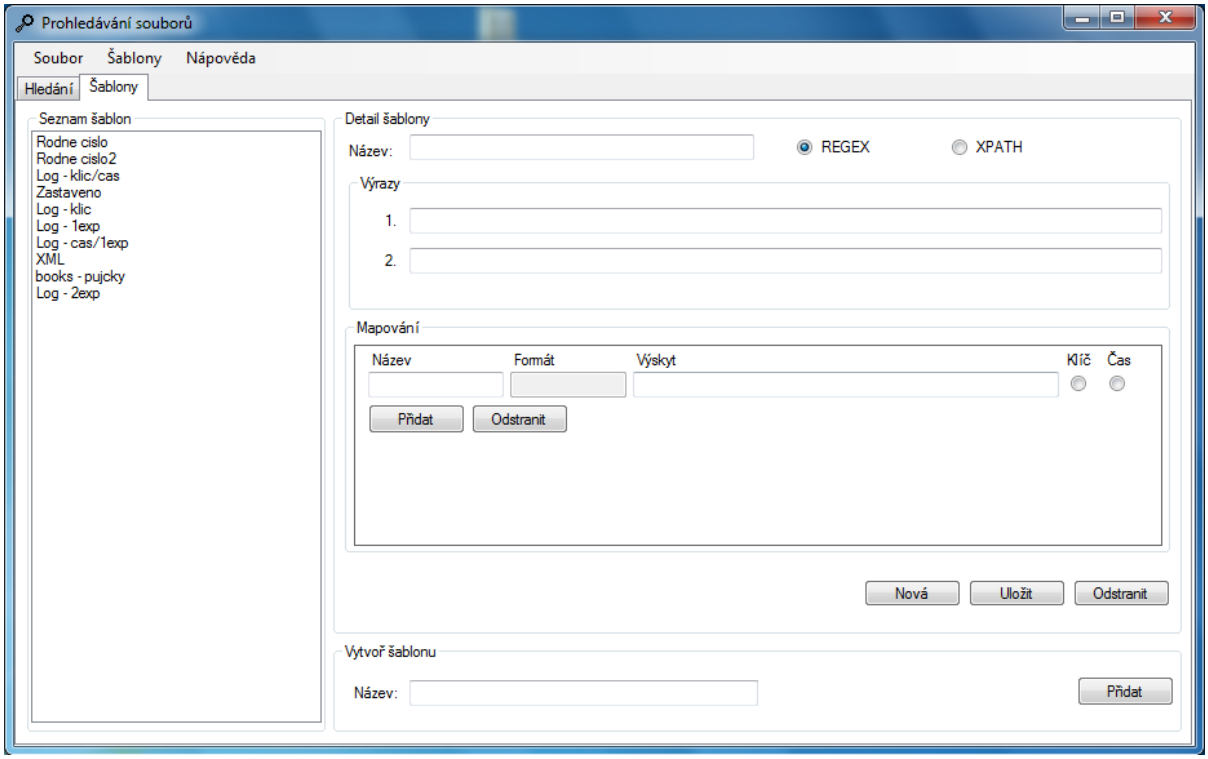
Po nalezení všech odpovídajících řádků je nutné tyto řádky rozložit podle mapování. O toto se stará funkce `Parse(…)`. `Parse(…)` obdrží nalezený řádek a samotnou vyhledávací šablonu. Jsou zde využity dvě hlavní třídy a to `Regex` a `Match`. `Regex` stejně jako při prohledávání textových souborů slouží na porovnání regulárního výrazu s aktuálním řádkem. Díky třídě `Match` je z vyhledaného řádku extrahována požadovaná část. Tato část musí být v regulárním výrazu označena závorkami, viz 4.2.4. V případě, že regulární výraz neodpovídá žádné části řádku, do požadovaného prvku se uloží "`Chyba výrazu`". Po rozložení jsou všechny nalezené výsledky navráceny a práce knihovny tímto krokem končí.

[1] XPathReader knihovna ke stažení - https://msdn.microsoft.com/en-us/library/ms950778.aspx

4.2 Uživatelské rozhraní

Uživatelské rozhraní prošlo při implementaci také několika malými změnami. Největší změny proběhli na druhé záložce, kde se pracuje se šablonami, viz Obrázek č. 5.

V levém rohu lze vidět stále seznam všech uložených šablon. Vpravo od seznamu šablon lze vidět detail samotné šablony, kde může uživatel editovat všechny hodnoty vybrané šablony. Ve spodní části je možné přidat novou šablonu.



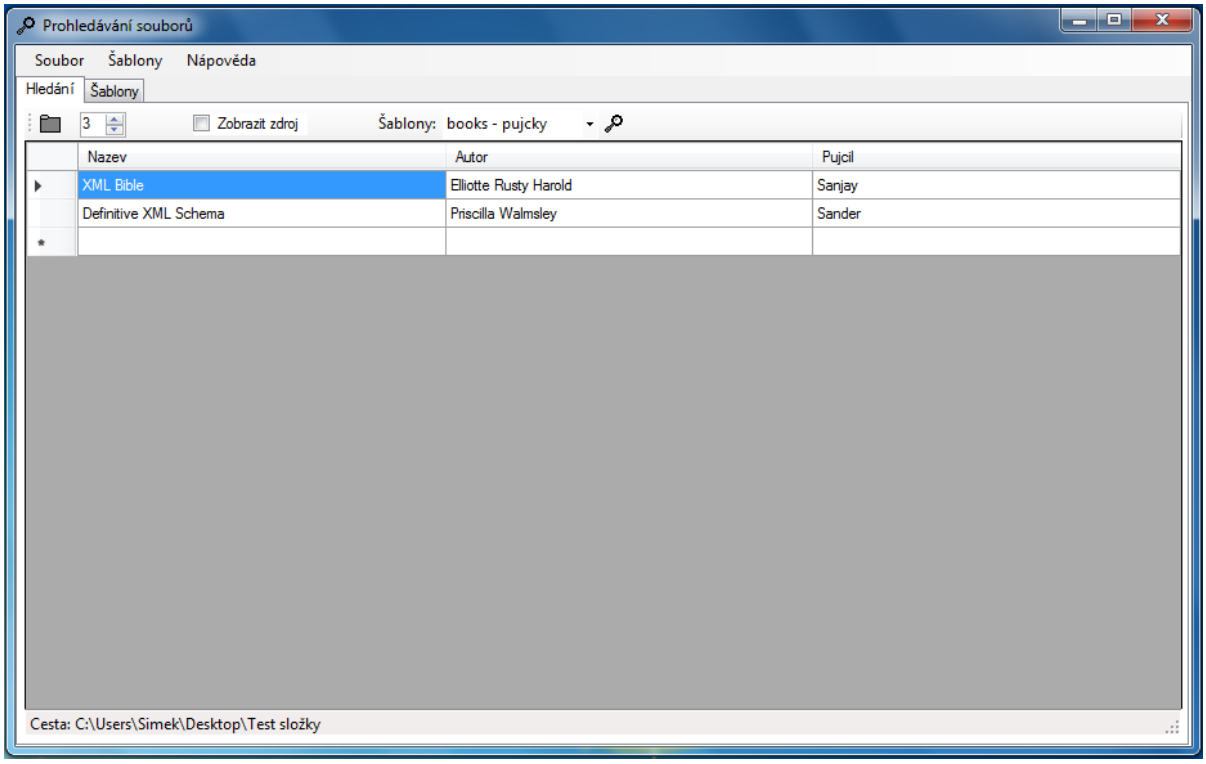
Obrázek č. 5 – Uživatelské rozhraní záložka šablony

Ve druhé záložce se zobrazení prvků oproti návrhu změnilo jen velice nepatrně, viz Obrázek č. 6.

Ve vrchní části okna jsou potřebné údaje k vyhledávání. Zleva výběr počáteční složky, hloubka prohledávaných složek. Dále je zde zaškrtávací políčko `Zobrazit zdroj`. Toto tlačítko bylo přidáno po konzultacích s potencionálními uživateli. Každý z těchto uživatelů odpověděl: „Takže mi to řekne, v jakém souboru se to našlo?“. Tato funkce tedy zobrazí, v jakém souboru byl tento řádek zobrazen. Následuje výběr šablony z uložených a poté tlačítko `Hledej` pro start vyhledávání.

Pod vyhledávacím panelem se nachází tabulka vyhledaných hodnot. Tato tabulka se vyplní a seřadí na základě dat zadaných v šabloně (popsáno v 4.2.1. a 4.2.2). Pod touto tabulkou už vidíme

pouze informativní panel, kde se zobrazuje cesta počáteční složky a v průběhu vyhledávání je zde pomocí `ProgressBaru` indikováno vyhledávání.



Obrázek č. 6 – Uživatelské rozhraní záložka hledání

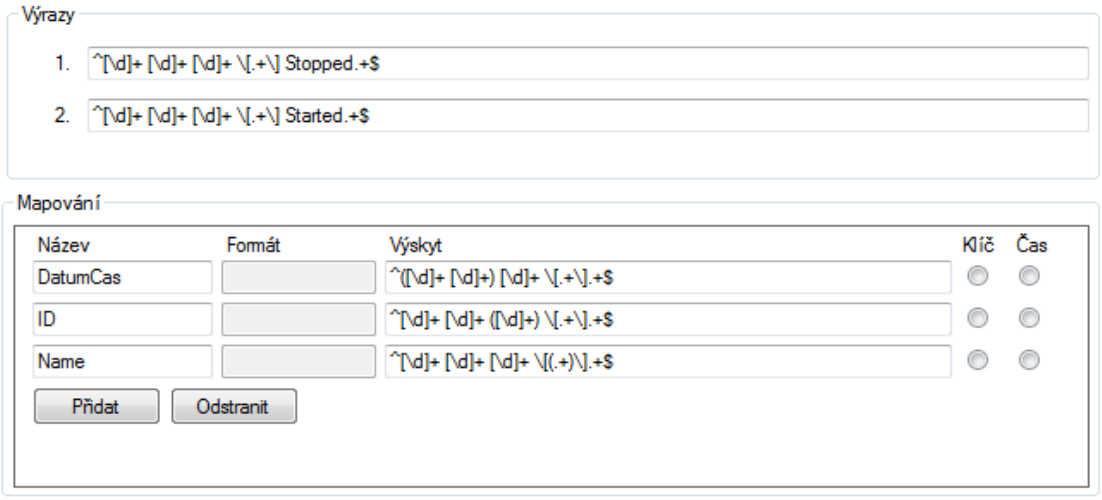
Poté co uživatel vybere všechny důležité parametry k vyhledávání a stiskne tlačítko „Hledej“, zkontrolují se všechny zadané hodnoty. Pokud některé z parametrů chybí, je uživatel upozorněn výstražnou hláškou a vyhledávání se nespustí. Po startu vyhledávání se změní tlačítko „Hledej“ na tlačítko „Vypni hledání“, které nám umožní vypnout spuštěné vyhledávání. Spolu s touto změnou se spustí animace ProgressBaru a je také znemožněn přístup ke změně parametrů. Znemožnění přístupu je nutné, protože aplikace tyto parametry později používá při řazení výsledků. Celé vyhledávání se odehrává ve speciálním vlákně, aby aplikace nezamrzla. Po dokončení vyhledávání se zobrazí informační zpráva s výsledným počtem nalezených výskytů.

4.2.1 Seřazení výsledků

Seřazení výsledků následuje ihned poté, co knihovna vrátí všechny nalezené výskyty (popsáno v 4.1.). Tato část systematicky pokračuje po kroku popsaném v 4.1.3.

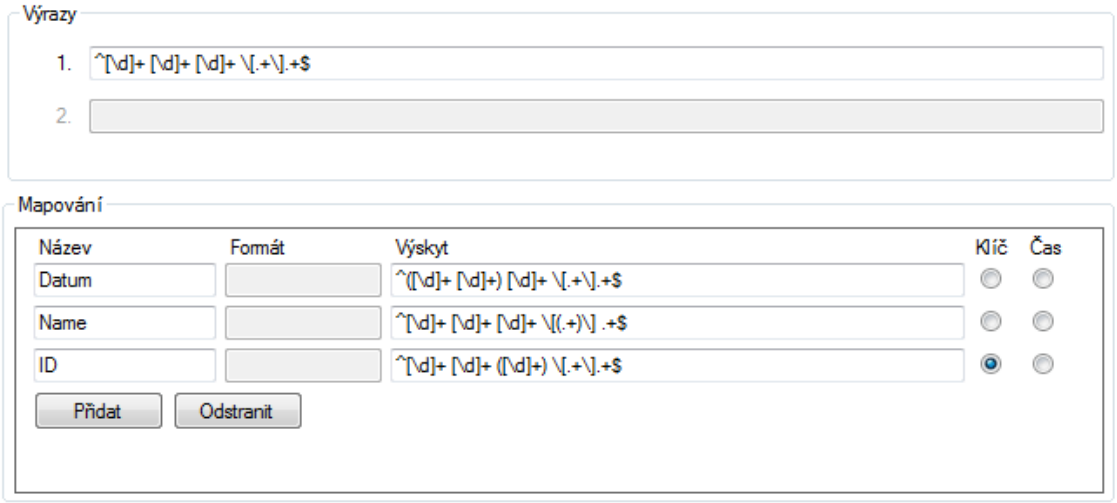
Po obdržení nalezených výsledků následuje seřazení podle vybraného času a klíče. Spolu s možností zadání dvou nebo jednoho výrazu se provádí teoreticky osm způsobů seřazení. V tomto případě je ale použití klíče omezeno pouze na jeden výraz, tudíž se jedná o šest způsobů:

Zadán jeden/dva výraz/y, bez klíče a času – v takovém případě se všechny vyhledané výskyty seřadí nejprve podle zdroje, kde byly nalezeny. Následně se seřadí podle řádku a nakonec podle indexu hledaného výskytu.



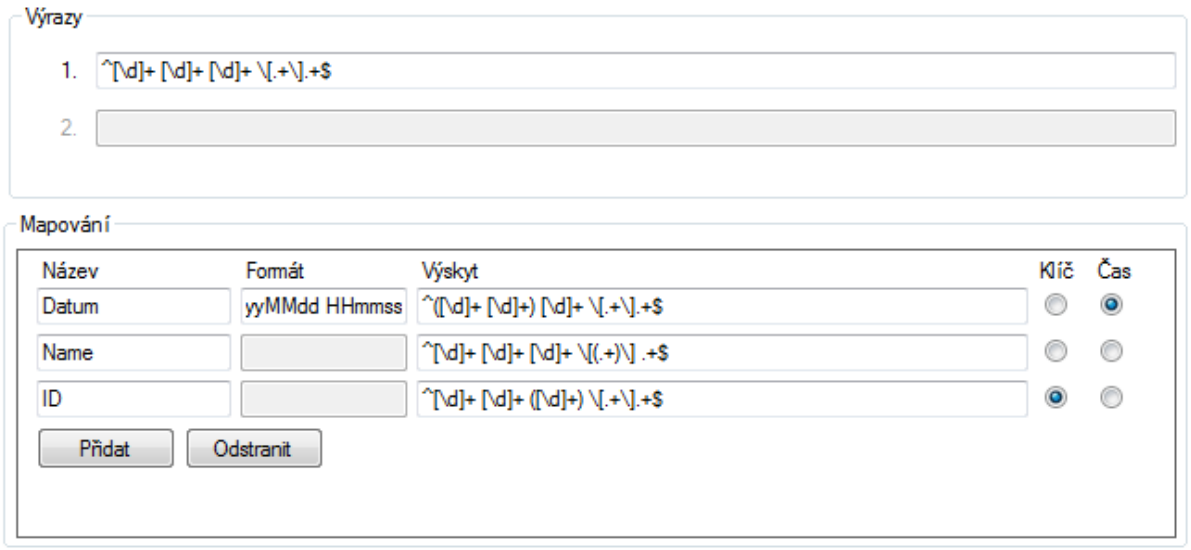
Obrázek č. 7 – Dva výrazy, bez klíče a času

Zadán jeden výraz, klíč a bez času – výsledek je seřazen podle prvku, který je určen jako klíč. V tomto případě odpadá jeden způsob seřazení, protože je možná pouze kombinace jeden výraz a klíč. Naopak dva výrazy a klíč nejsou povoleny. Dále je speciálně zobrazen pouze první řádek a vypočítaný časový rozdíl.



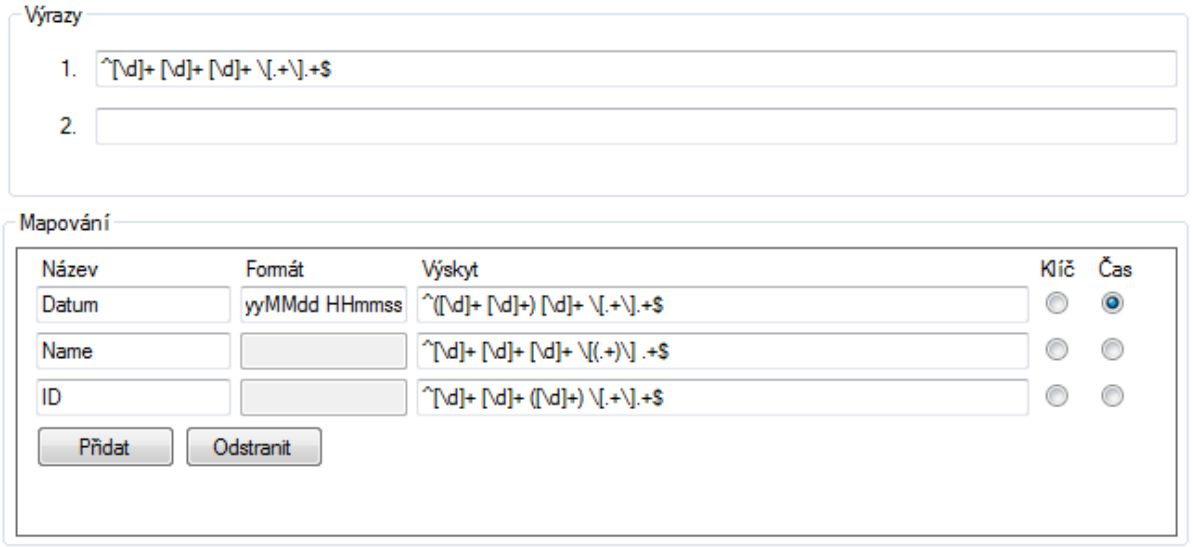
Obrázek č. 8 – Jeden výraz, klíč a bez času

Zadán jeden výraz, klíč i čas – pokud je zadán klíč i čas, jsou nejprve všechny výsledky seřazeny podle klíče. Poté se z množiny těchto výskytů se stejným klíčem vezme první a poslední a zjistí se rozdíl v časech. Dále je speciálně zobrazen pouze první řádek a vypočítaný časový rozdíl.



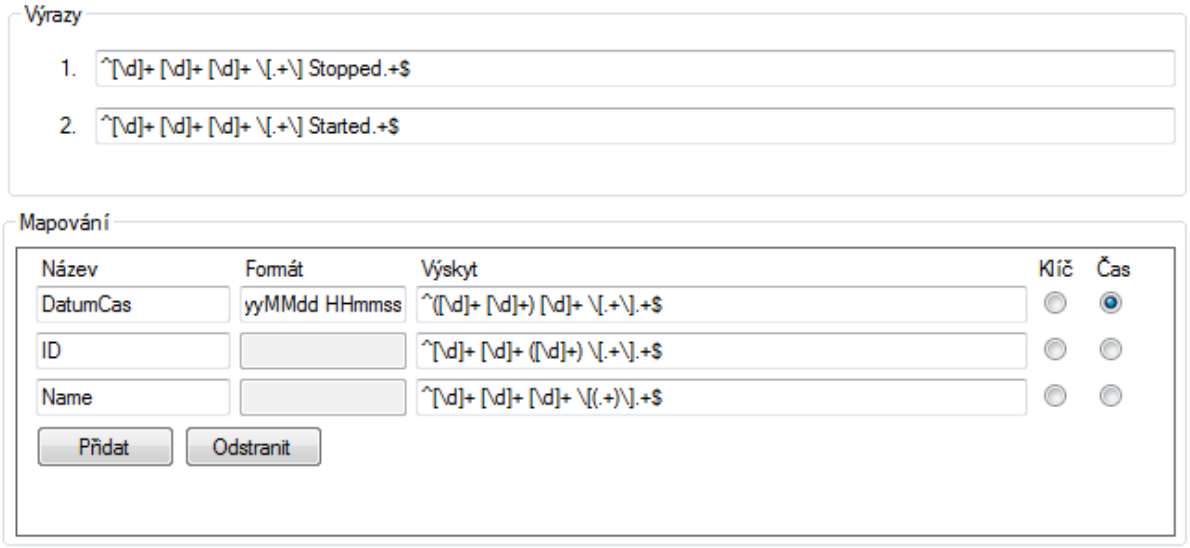
Obrázek č. 9 – Jeden výraz, klíč i čas

Jeden výraz, čas a bez klíče – když je zadán pouze jeden výraz a čas, výsledná množina se seřadí jednoduše podle času.



Obrázek č. 10 – Jeden výraz, čas a bez klíče

Dva výrazy, čas a bez klíče – v tomto případě se seřadí nejprve všechny výskyty stejně jako v prvním případě podle zdroje, řádku a indexu výrazu. Následuje vyhodnocení rozdílu v časech, vždy dvou po sobě jdoucích výsledků, podobně jako výše. Opět speciálně zobrazeno (popsáno v 4.2.2.).

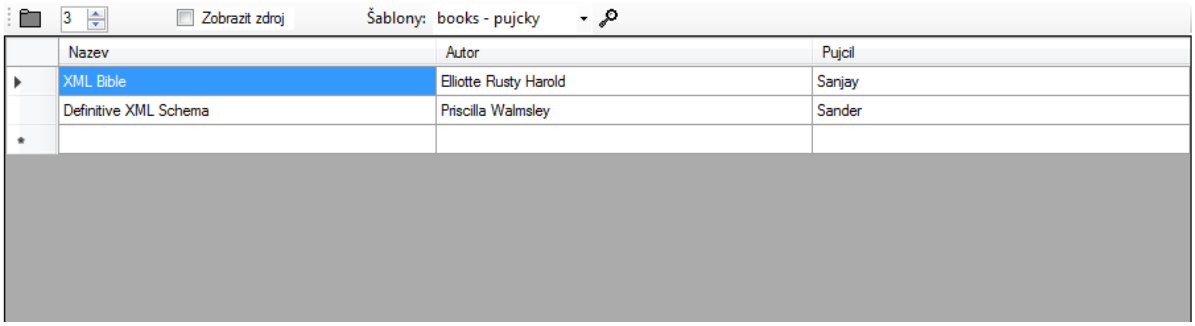


Obrázek č. 11 – Dva výrazy, čas a bez klíče

4.2.2 Zobrazení výsledků

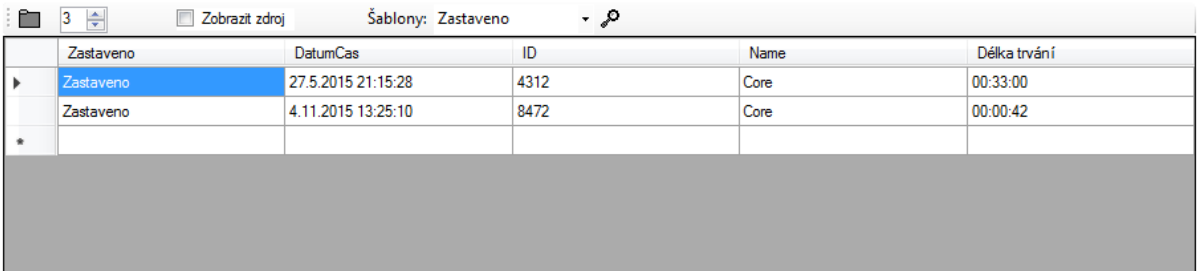
Při seřazování výsledků se zároveň určuje, jak se budou výsledky zobrazovat. Výsledky se zobrazují na základě mapování zadaného uživatelem. Zobrazení by se dalo rozdělit do dvou podob.

První podobou je klasické zobrazení, kdy se zobrazí vše, co bylo rozděleno pomocí mapování. V takovém případě odpovídá počet sloupečků zobrazené tabulky počtu prvků v mapování.



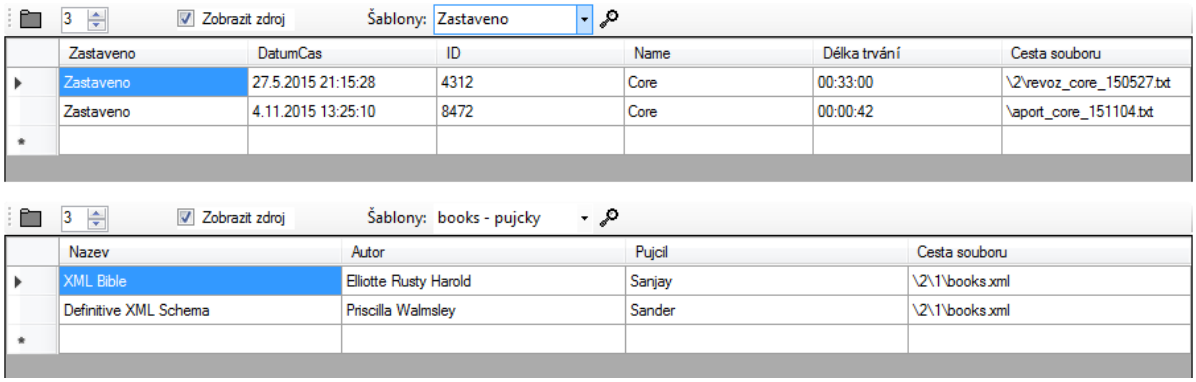
Obrázek č. 12 – Klasické zobrazení

V případě, že chce uživatel vidět rozdíl časů mezi dvěma určitými záznamy případně mezi prvním a posledním určitého klíče, je využito speciálního zobrazení. Speciální zobrazení přidává dva nové sloupečky. Prvním přidaným sloupečkem tabulky je pouze název použité šablony. Dalším sloupečkem, který je umístěn na konec, je již zmiňovaný rozdíl časů.



Obrázek č. 13 – Speciální zobrazení rozdílu časů

Oba z těchto způsobů zobrazení lze doplnit ještě o jeden sloupeček, který nám umožní vidět zdroj nalezeného řádku. Tento sloupeček se přidá na konec tabulky. Cesta zdroje je uvedena od počáteční složky.

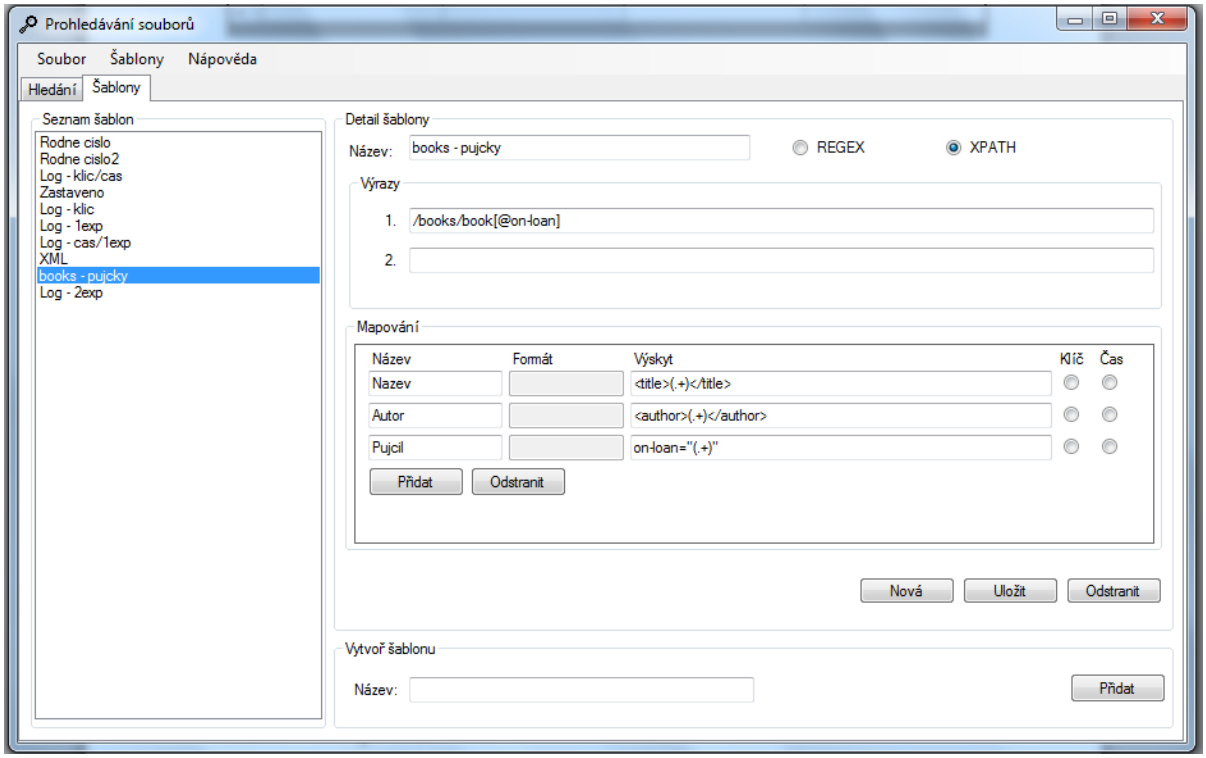


Obrázek č. 14 – Obě zobrazení a zdroj

4.2.3 Šablony

Nedílnou součástí aplikace jsou šablony. Bez těchto šablon by nemohla aplikace vyhledávat. Zde je popsáno, jak se šablony definují a jak je k tomu uspořádáno uživatelské rozhraní.

Již výše je popsáno, že pro definici šablon slouží druhá záložka. Na této záložce je vlevo seznam všech uložených šablon. Tyto šablony jsou uloženy v souboru `template.xml` a po spuštění aplikace se z tohoto souboru nahrají. Po stisknutí některé ze šablon se automaticky vyplní vpravo detail šablony, viz Obrázek č. 15.



Obrázek č. 15 – Detail šablony

Jako první v detailu šablony je možné editovat název. Název šablony musí být unikátní. Jedná se o identifikátor, který určuje jedinečnost šablony. Každá šablona může nabývat jednoho ze dvou typů, podle toho na jaké soubory se bude zaměřovat. Typ `REGEX` pro textové soubory a `XPATH` pro xml soubory.

Na základě těchto typů musí být následně vyplněny výrazy šablony. Pro vyhledávání v textových souborech je nutné používat regulární výrazy a pro xml soubory XPath výrazy. Syntaxe výrazů není nijak zkoumána aplikací, je totiž možné, že v určitém případě mohou vypadat výrazy stejně. Následuje mapování, které rozloží jednotlivé nalezené hodnoty (popsáno v 4.2.4).

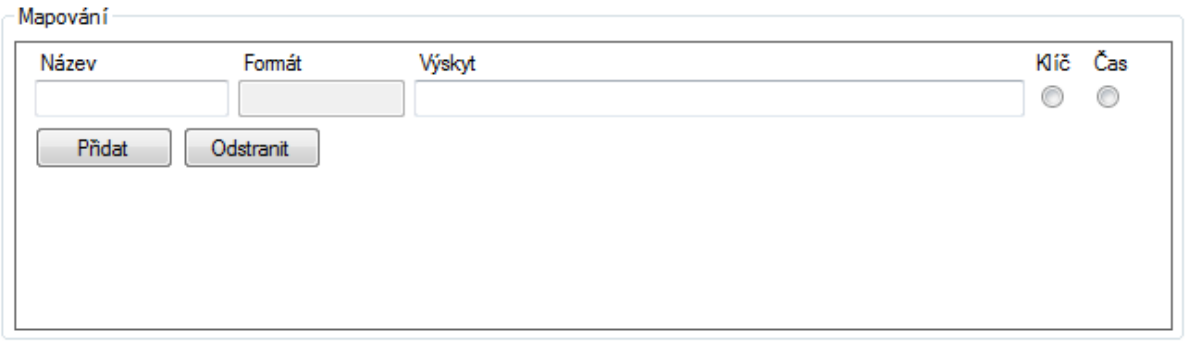
Pod mapováním je soustava tlačítek. První z tlačítek s názvem „Nová“ slouží ke zkopírování aktuální vybrané šablony a vytvoření nové. Hned vedle něj je tlačítko „Uložit“, které uloží všechny změny provedené v detailu šablony. Při použití obou těchto tlačítek se kontroluje, zda je jméno unikátní. V případě opaku je uživatel upozorněn a změna se neprovede. Poslední v soustavě je tlačítko „Odstranit“. Toto tlačítko odstraní aktuální vybranou šablonu.

Nakonec je zde kolonka „Vytvoř šablonu“, která umožňuje rychlé vytvoření šablony. Po stisknutí tlačítka „Přidat“ se zkontroluje kolize jmen šablon, a poté se vytvoří jednoduchá šablona. Automaticky se poté vyplní detail a je hned možné přidanou šablonu editovat.

4.2.4 Mapování

Díky mapování jsou vyhledaná data přehledná a seřazená. Žádná šablona se neobejde bez mapování. V případě, že by nebylo zadáno mapování, aplikace vrátí prázdnou tabulku výsledků, i kdyby některé byly nalezeny. Mapování je seznam prvků, které se dají rozložit z vyhledaného řetězce.

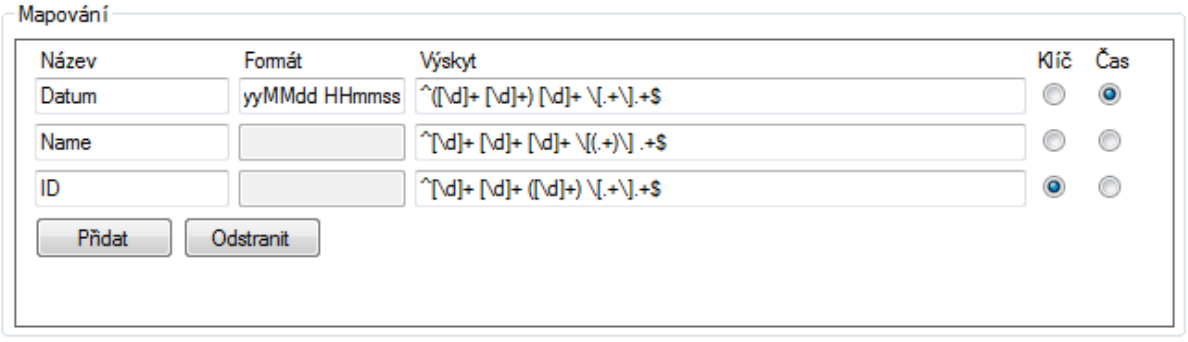
Každý tento prvek má povinné jméno a výskyt v řetězci. Jméno je pro orientaci ve výsledkové tabulce. Pomocí výskytu se určí, jaká část nalezeného řetězce odpovídá tomuto prvku. Každý prvek je možné označit jako klíč či čas. Prvek může být označen jako klíč i čas zároveň, ale není možné zadat více jak jeden klíč nebo prvek. Mapování může být také bez klíče i prvku. V případě, že je vybrán prvek jako čas, je nutné zadat formát času, aby bylo možné jej správně převést (popsáno v 4.2.4.1).



Obrázek č. 16 – Mapování šablony

Při různé kombinaci výrazů, klíče a času, jsou výsledná zobrazená data jinak seřazena (toto je popsáno v 4.2.1).

V kolonce `Výskyt` je očekáván vždy regulární výraz, který určuje výskyt v již nalezeném souboru. Regulární výraz musí vždy obsahovat závorky `( )`, které znamenají, že odpovídající prvek se nachází mezi těmito závorkami.



Obrázek č. 17 – Mapování s časem i klíčem

Defaultně je v každém mapování jeden řádek pro detail jednoho mapování. Pokud chce uživatel přidat další prvek mapování, slouží k tomu tlačítko „Přidat“, které přidá další řádek. Jak již napovídá název dalšího tlačítka „Odstranit“, to slouží k vymazání některého řádku mapování. Pokud není označen žádný řádek, je odstraněn poslední v pořadí. Pro vybrání řádku k odstranění je nutné označit název mapování. V jakém řádku je název označen, ten bude smazán.

4.2.4.1 Formát času

Aplikace umožňuje jakýkoliv formát času. Toto umožňuje zadání řetězce, který naformátuje nalezený čas. Formát času je očekáván ve standardu jazyka C# dostupné z MSDN[2]. Zde je uvedeno několik základních příkladů:

Specifikátor	Popis	Příklad
`d`	Den v měsíci, od 1 do 31	2009-06-01T13:45:30 -> 1
`dd`	Den v měsíci, od 01 do 31	2009-06-01T13:45:30 -> 01
`h`	Hodiny ve 12hodinovém režimu, od 1 do 12	2009-06-01T13:45:30 -> 1
`hh`	Hodiny ve 12hodinovém režimu, od 01 do 12	2009-06-01T13:45:30 -> 01
`H`	Hodiny ve 24 hodinovém režimu, od 1 do 24	2009-06-01T13:45:30 -> 13
`HH`	Hodiny ve 24 hodinovém režimu, od 1 do 24	2009-06-01T13:45:30 -> 13
`m`	Minuty od 0 do 59	2009-06-01T13:05:30 -> 5
`mm`	Minuty od 00 do 59	2009-06-01T13:45:30 -> 45
`MM`	Měsíc od 01 do 12	2009-06-01T13:45:30 -> 06
`yy`	Rok od 01 do 99	2009-06-01T13:45:30 -> 09
`yyyy`	Rok jako čtyřmístné číslo	2009-06-01T13:45:30 -> 2009

Tabulka č. 6

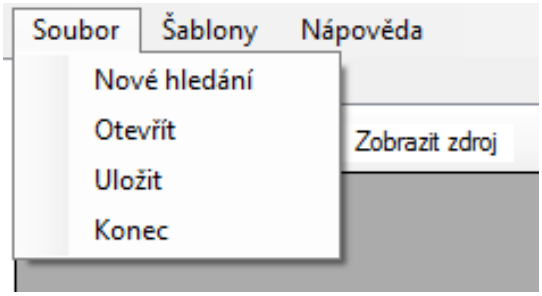
4.2.5 Další funkce

Uživatelské rozhraní dále obsahuje některé další funkce, které mohou uživateli pomoci. Všechny tyto funkce jsou dostupné z menu aplikace.

V první položce pojmenované `Soubor`, jako u většiny aplikací, jsou čtyři základní pomocné funkce. Jako první je zde tlačítko `Nové hledání`. Toto umožní uživateli vymazat zobrazovací tabulku. Pokud je zobrazovací tabulka prázdná, žádná akce se neprovede. V případě, že jsou vyhledaná

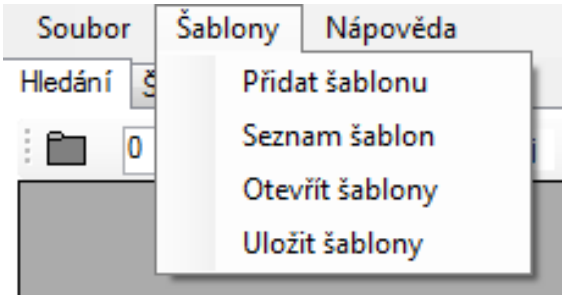
[2] Vlastní řetězec formátu data a času - https://msdn.microsoft.com/cs-cz/library/8kb3ddd4(v=vs.110).aspx

data neuložena, uživatel je dotázán, zda chce aktuální výsledky uložit. Další v pořadí je položka `Otevřít`, která dovolí, aby uživatel otevřel staré vyhledané výsledky. Případně výsledky vyhledané jinde. Na toto plynule navazuje funkce `Uložit`. Díky ní uživatel může uložit vyhledaná data. Poslední v pořadí je funkce `Konec`, která pouze vypne aplikaci.



Obrázek č. 18 – Menu položka Soubor

Další v menu je odrážka Šablony, která zpřístupňuje pomocné funkce při práci se šablonami. První dvě odrážky jsou velice jednoduché funkce, které pouze přesměrují uživatele na záložku Šablony. Zde může poté, buďto přidat šablonu popřípadě zde vidět seznam šablon. Dále v pořadí je Otevřít šablony. Zde můžou být načteny šablony z jiného zdroje. Nově načtené šablony se neuloží do první editace některé z šablon. Pokud před vypnutím aplikace není provedena žádná změna, šablony se při dalším spuštění načtou z defaultního souboru templates.xml obsaženém v adresáři aplikace. Poslední položkou je Uložit šablony. Jednoduše uloží aktuální šablony do uživatelem vybraného souboru.



Obrázek č. 19 – Menu položka Šablony

Posledním prvkem v menu je Nápověda. Tato položka zobrazí pdf soubor s pomocnými materiály k vytváření šablon a práci s aplikací.

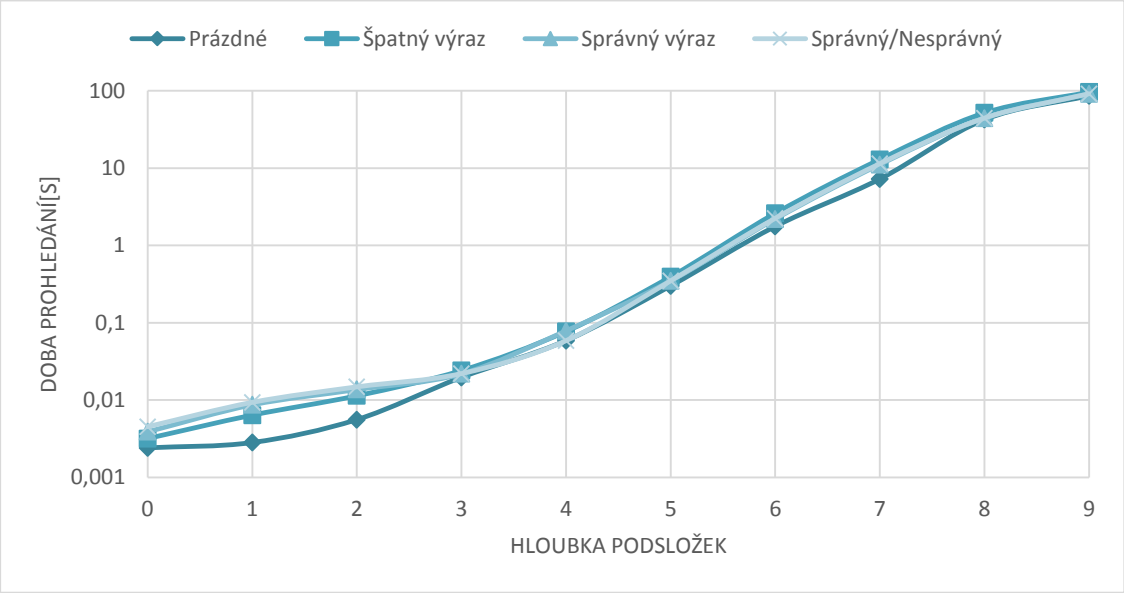
5 Testy

V této kapitole jsou popsány výsledky a poznatky z testování. Aplikace byla vystavena třem skupinám testů. V prvních testech bylo testováno velké množství podsložek s malou koncentrací prohledávaných souborů. Další test naopak ukazuje velké množství souborů s malou strukturou podsložek. Ve třetí sadě testů byly aplikaci zadány velké soubory.

5.1 Velké množství podsložek

V prvním kroku testování byla aplikace zkoumána z hlediska potřebného času k prohledání málého množství souborů ve velkém množství složek. Tato část testování by se dala rozdělit do čtyř menších testů. V první řadě byly testovány pouze prázdné složky bez souborů. Následně bylo přidáno pět souborů do každé jedné úrovně podsložek. To znamená pět souborů pro úroveň nula a padesát souborů pro hloubku devět.

Po přidání těchto souborů bylo dalším testem prohledávání s neodpovídající šablonou. Tudíž nebyl vypsán žádný výsledek. Poté byla využita šablona, které odpovídaly všechny soubory, a v každém tato šablona nalezla dva výskyty. Nakonec bylo přidáno do každé úrovně dalších pět souborů, které neodpovídaly vyhledávací šabloně.



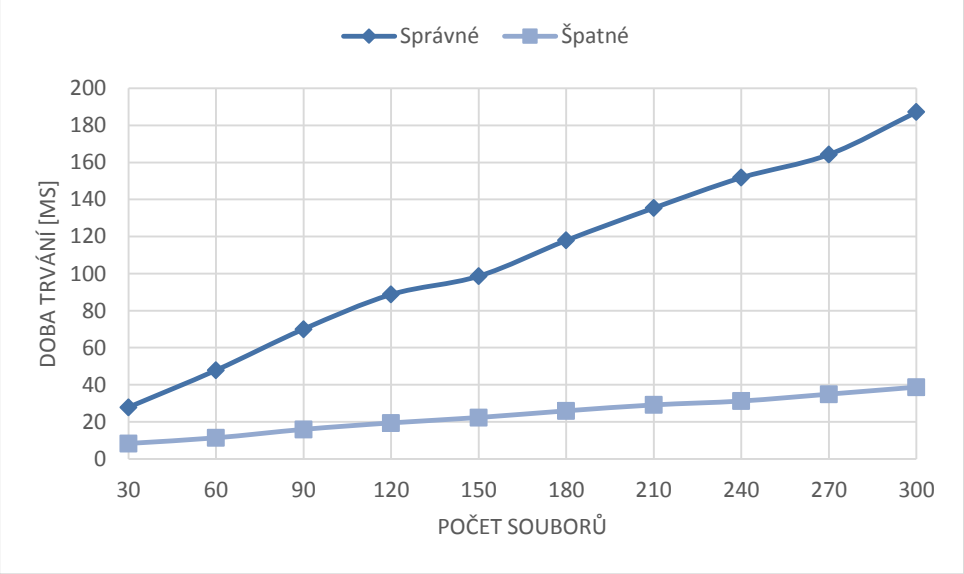
Graf č. 1 – Časový průběh vzhledem k hloubce podsložek

Z grafu lze vyčíst, že prohledávání je rychlejší, pokud nejsou žádné soubory ve složkách. To samozřejmě vyplývá z faktu, že není co prohledávat, tím pádem se aplikace úplně vyhne procházení souborů. Je ovšem viditelné, že do hloubky čtyř podsložek skončí aplikace prohledávání pod 0,1 vteřiny, což v podstatě uživatel ani nepostřehne.

Na první pohled je vidět, že se stoupajícím počtem podsložek, roste i doba zpracování. Tato skutečnost udává, že je více efektivní prohledávání v menší hloubce podsložek. Z grafu je také zřejmé, že při menším počtu podsložek se více projeví počet souborů. Ovšem vzhledem k rychlosti aplikace v těchto podmínkách se nejedná o žádný problém.

5.2 Velké množství souborů

Při dalším testování byla hloubka podsložek vždy rovna nule. Test začínal s jednou složkou s 30 soubory a po každé soustavě testů bylo přidáno dalších 30. Provedeny byly dvě soustavy testů. V jedné soustavě byla použita šablona, která vyhovovala všem souborům. Při druhém kroku byla šablona špatná a nevyhovovala žádnému z prohledávaných souborů.



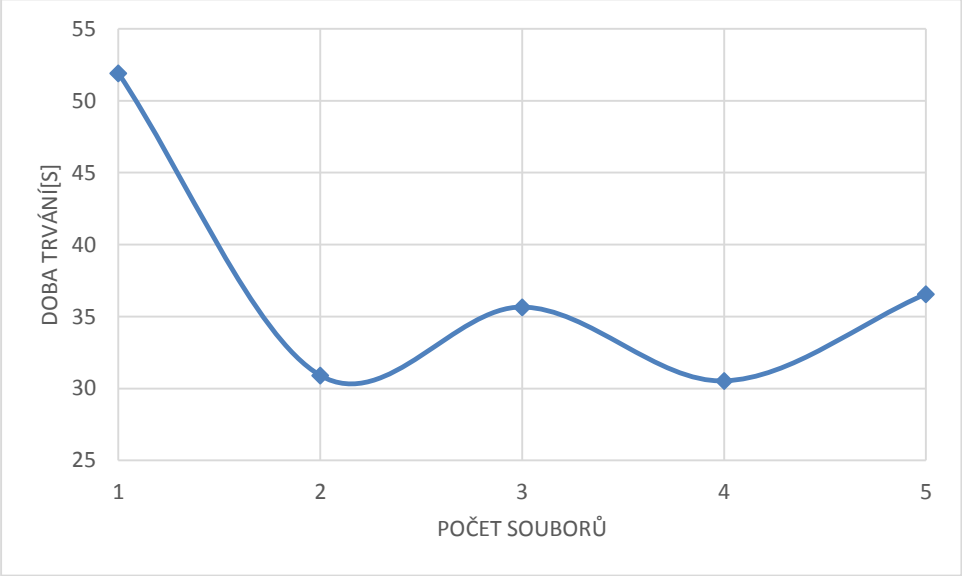
Graf č. 2 – Zpracování souborů za čas

Jak bylo před testem předpokládáno, při každé iteraci a přidání stejného počtu souborů se doba trvání zvýšila o přibližně stejnou hodnotu. Tato hodnota je průměrně 17,7ms pro správnou šablonu. Pro nesprávnou šablonu vychází odchylka 3,4ms v průměru.

V grafu je vidět časový rozdíl, kdy vyhledávání najde výskyty oproti nenalezení výskytů. Je ovšem vidět stejně jako v předchozím testu, že při počtu 300 souborů je aplikace zase tak rychlá, že uživatel ani nepozná rozdíl. Při očekávání lineárního průběhu bychom museli mít zhruba 1400 souborů, aby aplikace pracovala alespoň jednu vteřinu. V tomto případě je možné říct, že aplikace je velice efektivní při práci s malými soubory.

5.3 Obsáhlé soubory

Další prováděné testy byli trochu odlišnější od předchozích. Cílem bylo změřit, jak dlouho trvá aplikaci prohledat velké soubory o velikosti gigabajtů. V tomto testu bylo nejdříve vytvořeno pět souborů o velikosti asi 270 MB. Těchto pět souborů dohromady dává zhruba 1,3 GB. V každé iteraci testu byl ubrán jeden soubor a navýšena velikost každého tak, aby výsledný součet všech souborů dával vždy 1,3 GB.



Graf č. 3 – Doba trvání velkých souborů

Hned při prvním pohledu na graf je vidět velký skok mezi jedním a dvěma soubory. Mezi ostatními kroky je zhruba pět vteřin rozdíl, který ovšem není tak zásadní jako rozdíl prvních dvou měření. Tento rozdíl je dán tím, že vše je prohledáváno paralelně. Tím pádem dva zaráz zkoumané soubory jsou rychlejší než jeden o stejné velikosti. Z toho plyne, jak již bylo řečeno výše, že je efektivnější práce s více menšími soubory. Ovšem při velikosti souboru 1,3 GB je analýza souboru za 52 vteřin dostatečná. Do budoucna by bylo vhodné toto optimalizovat.

6 Závěr

Tato práce se zabývá prohledáváním souborů. Cílem práce bylo prostudovat regulární výrazy a XPath výrazy. Následně navrhnout a vytvořit knihovnu i uživatelské rozhraní, které prohledávají soubory na základě zadaných šablon. Výsledkem programu jsou dva soubory. Jedním je výsledná spustitelná aplikace, díky které uživatel vyhledává. Dalším souborem je knihovna, která obsahuje třídy pro vyhledávání.

Pro implementaci byl zvolen jazyk C#. Prohledávání podsložek bylo provedeno klasickou třídou Directory, která umožňuje práci se soubory. Navíc jsou podsložky prohledávány paralelně, aby hledaná struktura byla nalezena rychleji. Paralelního přístupu je využito i při hledání výskytu v samotném souboru. Na procházení souborů jsou využity datové proudy, kdy se soubor připojí a čte po částech, což umožňuje prohledání velkých souborů, bez toho aby byl nahrán do paměti. Pokud by tomu tak nebylo, je velice možné, že by PC při velice obsáhlých souborech zamrznul.

Výsledná aplikace byla nakonec otestována na několika sadách souborů. Program byl testován jak s velkými a malými soubory, tak i velkou strukturou podsložek. Testovací soubory byly vytvořeny z poskytnutých logů aplikace externího zadavatele. Cílem testování bylo zjistit efektivitu a rychlost zdrojových dat. V první řadě bylo zjištěno, že pokud není příliš velké množství podsložek, aplikace pracuje dostatečně rychle. Naopak při větším počtu podsložek je prohledávání stále delší. Dále bylo po přidání souborů zjištěno, že i prohledávání malých souborů je velice efektivní a rychlé. Při počtu 300 souborů bylo naměřeno prohledání za 187ms. Nakonec bylo testováno zpracování velkých souborů. Zde se ukázalo, že aplikace není zcela optimalizovaná. Při práci s jedním souborem trvalo aplikaci prohledání déle než při zpracování dvou souborů. Tyto dva soubory dohromady měly stejnou velikost jako první testovaný soubor. Navzdory tomu jsou výsledky rychlosti stále uspokojivé.

Literatura

[1] IEEE 1003.1 Standard for Information Technology: Regular Expressions, [online], 2013, [cit. 18.12.2015]. Dostupné z http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html

[2] XML Path Language: XPath, [online], [cit. 18.12.2015]. Dostupné z http://www.w3.org/TR/xpath/

[3] W3schools.com: XPath Tutorial [online], 2014 [cit. 5.2.2016]. Dostupné z http://www.w3schools.com/xsl/xpath_intro.asp

[4] Itnetwork.cz: Regulární výrazy v C# .NET [online], 2013 [cit. 5.2.2016]. Dostupné z http://www.itnetwork.cz/csharp/pokrocile/tutorial-csharp-dot-net-regularni-vyrazy/

[5] ESPOSITO, Dino. *XML: efektivní programování pro .NET*. Praha: Grada, 2004. Moderní programování. ISBN 80-247-0775-6.

[6] SKONNARD, Aaron a Martin GUDGIN. *XML: pohotová referenční příručka : referenční příručka programátora ke XML, XPath, XSLT, XML Schema, SOAP a dalším*. Praha: Grada, 2006. Průvodce (Grada). ISBN 80-247-0972-4.

[7] GOYVAERTS, Jan a Steven LEVITHAN. *Regulární výrazy: kuchařka programátora*. Brno: Computer Press, 2010. ISBN 978-80-251-1935-8.

[8] MSDN. *MSDN* [online]. [cit. 2016-05-12]. Dostupné z msdn.microsoft.com

Seznam příloh

Příloha 1 – Obsah přiloženého CD

Adresář	Obsah adresáře
Aplikace	Zdrojové soubory knihovny a uživatelského rozhraní
Spustitelná verze	Spustitelná verze aplikace s testovacími šablonami
Testovací soubory	Sada testovacích souborů
Text	Bakalářská práce v elektronické podobě