PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Systém pro správu diplomových prací za použití fulltextového vyhledávání

2010 Marek Vávra

Anotace

Obsahem této diplomové práce je vytvoření malého systému pro správu diplomových prací s možností fulltextového vyhledávání. Při tvorbě práce byly použity nejmodernější technologie a programovací techniky, které jsou v současné době k dispozici. Cílem práce však není pokrýt kompletní problematiku fulltextového vyhledávání, jako spíše ukázka praktického využití fulltextu v praxi. Práce byla vyvíjena v programovacím jazyku C# ve verzi 4.0, grafická vrstva pak ve WPF ve verzi rovněž 4.0, pro jejichž bezproblémový chod je nutné použití .Net frameworku taktéž ve verzi 4.0. Jako datové uložiště slouží libovolná instance SQL Serveru 2008 se spuštěnou a nakonfigurovanou službou umožňující fulltextové vyhledávání.

Mé poděkování patří Ing. Jiřímu Hronkovi za jeho trpělivost a ochotu při psaní mé bakalářské práce a také mé přítelkyni, která se nezvyklou mírou tolerance a podpory výrazně podepsala na úspěšném dokončení této práce.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

1.1. Zadání projektu

Účelem této bakalářské práce bylo napsat jednoduchý systém pro správu dokumentů v univerzitním prostředí. Tento systém musí obsahovat možnost fulltextového vyhledávání nad vloženými dokumenty. Fulltextový vyhledávač pak musí umožňovat širokou škálu vyhledávacích kritérií tak, aby bylo možné dokumenty vyhledávat podle jména autora, podle typu dokumentu, podle klíčových slov obsažených v textu.

Systém pro správu dokumentů bude využívat jako datové uložiště SQL server poslední generace, který disponuje fulltextovým vyhledáváním, a jeho užití na platformě Windows v kombinaci s .Net frameworkem je nejvíce přirozené.

Smyslem této bakalářské práce je pak demonstrovat sílu a možnosti tohoto fulltextového vyhledávání napojedného na reálnou aplikaci. Systém pro správu dokumentů musí být schopen zpracovat texty známých textových formátů, jako jsou například PDF dokumenty, Word dokumenty, Excel dokumenty, HTML stránky, dokumenty v XML i RTF formátu, stejně tak jako plain text, a to vše až do velikosti jednoho záznamu o maximální velikosti 2GB.

Uživatelské rozhraní aplikace musí být uživatelsky příjemné a intuitivní, což bude zajištěno nejnovější technologií Windows Presentation Framework užívanou pro tvorbu grafického rozhraní pro systémy Windows v posledních letech.

Aplikace musí být schopna zobrazovat obsah nalezených dokumentů, případně umožnit uložení obsahu dokumentu lokálně.

Aplikace bude vyvíjena výhradně pro platformu Microsoft Windows s instalovaným .Net frameworkem verze 4.0, který je standardní součástí posledních verzí operačních systémů.

1.2. Zpracování projektu

Desktopová aplikace bude napsána v prostředí Microsoft Visual Studio 2010 v jazyce C# verze 4.0.
Grafická vrstva aplikace bude psána rovněž v prostředí Microsoft Visual Studio 2010 za použití Windows Presentation Foundation verze 2.0.
Serverovou část aplikace bude tvořit SQL Server 2008 ve verzi Enterprise s instalovanou fulltextovou službou.
Pro přístup k datům uloženým na SQL Serveru bude využito Entity Frameworku verze 2.0.

1.3. Zaměření

Aplikace, jak již její samotný název napovídá, je jednoduchým systémem pro správu dokumentů, jehož primární funcí je spravovat dokumenty vznikající při studiu a práci na univerzitě. Jejími největšími výhodami oproti podobně zaměřeným systémům je jednoznačně velmi příjemné uživatelské rozhraní vytvořené za pomoci nejnovější verze Windows Presentation Foundation a také široká škála vyhledávacích kritérií, která jsou dostupná v posledních verzích SQL serveru 2008.

Aplikace je primárně určena pro platformu Microsoft Windows a její používání na jiných platformách není podporováno a jakýkoliv pokus spustit aplikaci na jiné platformě, než pro kterou je určena, může vést k nefunkčnosti programu.

Ačkoliv aplikace poběží na lokálním počítači, datové uložiště v podobě SQL Serveru 2008 by mělo být nainstalováno na samostatném počítači, jehož výpočetní výkon, kapacita diskového a paměťového místa by měla odpovídat zátěži, kterou na něj budou jednotlivé klientské aplikace vyvíjet.

Počet klientských aplikací, které mohou přistupovat k databázi, je omezen jen a pouze výpočetním výkonem a typem licence používaného SQL Serveru.

I když existuje nepřeberné množství aplikací podobného zaměření, jen málo z nich poskytuje opravdu plnohodnotnou podporu fulltextového prohledávání dokumentů.

Aplikaci Document Management System je pochopitelně možné rozvíjet a přizpůsobovat potřebám a požadavkům zadavatele, stejně tak jako je možné programovou logiku oddělit od prezentované aplikace a integrovat ji jako součást jiného programu. Předváděná aplikace se nesnaží pokrýt kompletní problematiku fulltextového vyhledávání, ale spíše předvést jeho užití v praktické aplikaci.

2. SQL SERVER 2008

Aplikace využívá databázový server společnosti Microsoft ve verzi SQL Server 2008, která oproti předchozím verzím nabízí bohatší škálu fulltextového vyhledávání. Tato verze také jako první umožňuje naistalovat originální český thezaurus, který tak již není nutné do programu doinstalovávat.

2.1. Fulltextové vyhledávání

Pro správnou funkci fulltextového vyhledávání je nezbytné vytvořit fulltextový index na sloupci, nad kterým má být vyhledávání prováděno. Je možné indexovat i několik sloupců, což má za následek přesnější vyhledávání, nicméně za cenu snížení výkonu, což může být problém u malých a hodně vytížených serverů. Fulltextový index bohužel není možné vytvořit nad každým datovým typem.

Podporovány jsou pouze následující datové typy:

- char
- varchar
- nchar
- nvarchar
- text
- ntext
- image
- xml
- varbinary
- varbinary(max)

Změnou oproti dřívější verzi SQL Server 2005 je integrace fulltextového vyhledávání pro více než 50 jazyků, mezi nimiž nově nechybí ani podpora českého a slovenského jazyka, což je velkou změnou oproti verzi předchozí, ve které bylo integrováno jen 23 jazyků, a podpora češtiny pro indexování, respektive pro fulltextové prohledávání dokumentů byla možná až po doinstalování pluginu dodávaného třetími stranami.

2.2. Vytvoření fulltextového indexu

Pro umožnění fulltextového vyhledávání je nutné vytvořit nad dotazovanou databází fulltextový katalog, který může obsahovat jeden nebo více fulltextových indexů. Je nutné mít na paměti některá omezení, která se týkají zejména rozsahu katalogu a indexů, které do něj náležejí. Zde bych rád uvedl některá omezení, která mohou zabránit konfliktům, či nefunkčnosti systému:

- Jedna databáze může obsahovat n katalogů
- Jeden katalog může náležet pouze jedné databázi
- Jeden katalog může obsahovat indexy pro n tabulek
- Jedna tabulka může mít pouze jeden fulltextový index
- Jedna tabulka může náležet pouze jednomu fulltextovému katalogu

Dále je nutné mít na paměti, že není možné vytvářet index ani na pohled (view), ani na dočasnou tabulku, která neexistuje fyzicky v databázi. Stejně tak není možné vytvářet index na systémové tabulky, které nemohou být indexovány za použití fulltextového indexu. Index je nutné definovat na tabulce s unikátním klíčem, což bývá zpravidla primární klíč, jak je tomu i v popisované aplikaci. Je také nutné definovat index na jednom nebo n sloupcích typu umožňujícího fulltextové indexování. Na sloupcích typu image, xml, varbinary nebo varbinary(max), které neobsahují přímo textovou informaci, je nutné aplikovat Ifilter, který dokáže získat plain-text i ze souborů typu html, xml, xdoc a mnoha dalších. Je také možné doinstalovat IFilter pack od společnosti Microsoft, který rozšiřuje filtrovací schopnosti SQL Serveru na několik nových formátů. Stejně tak je možné doinstalovat i další filtry poskytované třetími stranami.

2.3. Streamer

Pro každý z podporovaných jazyků poskytuje SQL Server nástroj pro rozpoznávání rozdělených slov, tvarů jednotlivých slov, čísel a také formátů datumu a času, tzv. STREAMER. Tento nástroj hraje zcela klíčovou roli při vlastním překladu dotazu, kdy k zadanému dotazu hledá různé tvary slov generované interními algoritmy, které popisují gramatiku daného jazyka.

Pro český jazyk dokáží tyto algoritmy rozpoznat slovní druh, pád, slovesný tvar, čas a všemožné tvary zadaného slova. Tvary jsou generované jen v rámci jednoho slovního druhu. Podstatná jména jsou generována ve všech tvarech jednotného i množného čísla, slovesa jsou generována ve všech časech, v rozkazovacím způsobu, či infinitivu. Pro slovesa se dokonce generují přechodníky. U přídavných jmen a zájmen se generují všechny tvary v jednotném a množném čísle pro všechny rody. Slovesná podstatná jména, přivlastňovací přídavná jména a příčestí jsou zpracovávána jako samostatná slova, tzn. negenerují se slovní druhy, ze

kterých jsou odvozené (přesvědčení a přesvědčit jsou považována za různá slova). Záporné tvary slov vytvářené pomocí předpony "ne" jsou zpracovávány odděleně od kladných tvarů, proto například uspět a neuspět nejsou generována v jednom dotazu pro jeden základ slova.

Práci STREAMERU si tedy můžeme představit tak, jakoby fulltextový dotaz na "červené jablko" přeložil na dotaz typu "červené" OR "červený" OR "červená" OR ... AND "jablko" OR "jablka" OR "jablku" OR ...

Důležitou vlastností (nejen) pro český jazyk je schopnost generovat ohyby slov také pro slova, která se nenacházejí ve slovnících daného jazyka (jako jsou například jména a příjmení, novotvary nebo slova počeštěná).

2.4. Thezaurus

Editovatelný thezaurus, neboli slovník synonym je využíván pro hledání slov podobného významu. Tento slovník synonym je defaultně prázdný a nenaplní se ani při vytvoření fulltextového indexu. K jeho naplnění dojde při prvním pokusu o jeho použití nebo může být načten spuštěním dávkového příkazu:

```
sys.sp_fulltext_load_thesaurus_file lcid[,@loadOnlyIfNotLoaded=action]
```

LCID je jednoznačný identifikátor jazyka, pro který požadujeme thezaurus, a kde action může nabývat hodnot:

- 1 – což znamená, že bude slovník synonym načten a přepsán bez ohledu na to, zda již načten byl
- 0 – což znamená, že bude slovník synonym inicializován pouze pokud tak zatím nebylo učiněno (například spuštěním příkazu, který jej vyžaduje)

Pro každý z podporovaných jazyků poskytuje SQL Server také slovník méně významných slov (stop words nebo noise words), která nemají v daném jazyku příliš velkou váhu pro vyhledávání, a mohou tak být z indexu jednoduše vynechána. Tato vlastnost má pozitivní vliv převážně na velikost indexu a tedy i na rychlost vyhledávání. Pro český jazyk představují tato méně významná slova například předložky a spojky.

Je důležité mít na paměti, že slova, která se v indexu nevyskytují, mohou dotaz významným způsobem ovlivnit. Z výše uvedeného můžeme snadno vyvodit, že fulltextový dotaz na frázi "a tak to tedy máte" bude v indexu nakonec vyhledávat pouze slovo "máte", což může výsledek dotazu nečekaně zkreslit. Pro slova, která mají v daném jazyku menší váhu, je proto lepší použít klasický dotaz LIKE namísto fulltextového vyhledávaní, který neprohledává fulltextový index.

2.5. Predikát CONTAINS

Tento predikát je podobně jako predikát FREETEXT použit ve WHERE nebo HAVING klausuli s použitím SELECT příkazu. Může být libovolně kombinován s jinými transakčními predikáty, jako je LIKE nebo BETWEEN. Predikát vrací jako návratovou hodnotu TRUE, nebo FALSE a je používám jako rozhodovací podmínka pro to, zda zadaný řádek vyhovuje zadanému fulltextovému dotazu. Výsledný SELECT příkaz vrací tabulku (neboli RESULT SET).

Predikát CONTAINS je používán pro nalezení záznamů obsahujících naprostou shodu nebo tzv. "fuzzy" shodu. Jedná se tedy o shodu, kdy vyhledávání není zcela přesné, ale je založené na nalezení přesně specifikované části slova. Pomocí CONTAINS lze vyhledávat:

- Jednotlivé slovo
- Skupinu slov neboli frázi
- Slovo blízké jinému slovu nebo fráze
- Frázi blízko jiného slova nebo fráze
- Slovo s určitou platností v porovnání s jiným slovem

Predikát CONTAINS hledá slova s velkou přesností, proto jsou jeho výsledky zpravidla přesnější, než použití predikátu FREETEXT, avšak nastává problém s hledáním slov nebo frází, která obsahují nevýznamná slova (tzv. noise words jako bylo zmíněno výše). Pro takovéto fráze vrátí dotaz výsledek FALSE, tedy neúspěch hledání, byť by fráze v textu existovala.

2.5.1. Syntax

```
CONTAINS
    ( { [NÁZEV SLOUPCE] | [SEZNAM SLOUPCŮ] | * }
          , '< vyhledávací podmínka >'
 [ , LANGUAGE [JAZYK]]
      )
< vyhledávací podmínka > ::=
    { < jednoduchý výraz >
    | < předpona >
    | < obecný výraz >
    | < přibližný výraz >
    | < posuzovaný výraz >
    }
```

```
    | { ( < vyhledávací podmínka> )
    [ { < AND > | < AND NOT > | < OR > } ]
    < vyhledávací podmínka > [ ...n ]
    }
< jenoduchý výraz > ::=
         slovo| " fráze "
< předpona > ::=
    { "slovo * " | "fráze*" }
< obecný výraz > ::=
    FORMSOF ( { STREAMER | THESAURUS } ,
     <jednoduchý výraz > [ ,...n ] )
< přibližný výraz > ::=
    { < jednoduchý výraz > | < předpona > }
    { { NEAR | ~ }
    { < jednoduchý výraz > | < předpona > }
    } [ ...n ]
< posuzovaný výraz > ::=
    ISABOUT
       ( { {
  < jednoduchý výraz >
  | < předpona >
  | < obecný výraz >
  | < břibližný výraz >
  }
   [ VÁHA ( weight_value ) ]
   } [ ,...n ]
       )
< AND > ::=
    { AND | & }
< AND NOT > ::=
    { AND NOT | & !}
< OR > ::=
    { OR | | }
```

2.5.2. Argumenty

`[NÁZEV SLOUPCE]`

Název sloupce určuje název fulltextového indexovaného sloupce tabulky, která je specifikovaná ve FROM klausuli. Sloupce mohou být typu char, varchar, nchar, nvarchar, text, ntext, image, xml, varbinary, or varbinary(max).

`[SEZNAM SLOUPCŮ]`

Argument specifikuje dva, či více názvů sloupců, které jsou od sebe odděleny čárkami a uvozovkami. Jestliže [JAZYK] není specifikován musí být jazyk všech sloupců ze zadaného seznamu stejný.

`LANGUAGE [JAZYK]`

LANGUAGE [JAZYK] je jazyk, který se používá pro dělení slov, inicializaci slovníku synonym, nahrazení slov a pro odstranění tzv. noise word jako části dotazu. Tento parametr je nepovinný. Pokud jsou dokumenty s různými jazyky uloženy pohromadě jako binární objekty, tzv. BLOBs v jednom sloupci, pak lokální identifikátor, tzv. LCID daných dokumentů předurčuje, jaký jazyk bude určen k indexování jejich obsahu. V případě dotazování takového sloupce právě upřesnění jazyka LANGUAGE [JAZYK] může zvýšit možnost nalezení shody s hledaným záznamem.

Za předpokladu, že definujeme tento nepovinný parametr, pak daný jazyk bude použit pro vyhledávání v celém dotazu. Pokud se tak nestane, tzn. nedefinujeme [JAZYK], bude automaticky pro vyhledávání použit fulltextový jazyk sloupce.

[JAZYK] můžeme definovat jako:

- řetězec
- celé číslo
- hexadecimální hodnotu k LCID jazyka

V prvním případě, tedy pokud je [JAZYK] definován jako řetězec, pak musí být [JAZYK] ve shodě s alias hodnotou sloupce. Řetězec musí být vždy uzavřen jednoduchými citačními znaménky, jak je patrno na následujícím příkladu: ´[JAZYK]´ V případě, že máme [JAZYK] definován jako celé číslo, pak je jako argument použit LCID aktuálního jazyka.

Pokud [JAZYK] definujeme jako hexadecimální hodnotu, pak [JAZYK] je retězec 0x následován hexadecimální hodnotou LCID. Hexadecimální hodnota má vždy délku osmi znaků, kdy jsou případné volné znaky nahrazeny nulou.

SQL Server vyhodnotí jako chybu stav, kdy dojde k tomu, že specifikovaný jazyk je neplatný či pokud neexistují zdroje korespondující s daným jazykem.

`Slovo`

Slovo definujeme jako řetězec znaků nepřerušený mezerami či interpunkčními znaménky.

`Fráze`

Frází může být jedno či více slov oddělené mezerami mezi každým slovem.

`[Jednoduchý výraz]`

[Jednoduchý výraz] specifikuje výskyt zcela identických slov a frází. Fráze by měly být ohraničeny dvojitými uvozovkami. Slova se musí v dané frázi objevit ve stejném pořadí, jako se objeví v databázovém sloupci. Při vyhledávání se nerozlišují velká a malá písmena. V potaz se neberou ani interpunkční znaménka.

`Předpona`

Vyhledává výskyt slov nebo frází, které začínají specifickým uskupením slov, tzv. prefixem. Předpona je vždy označena dvojitými uvozovkami. Před vložením koncových uvozovek je třeba připojit symbol hvězdičky (*). Klauzule pak vypadá následovně: CONTAINS (column, "text*"]. Symbol * je v tomto případě zástupným znakem pro žádný, jeden či více libovolných symbolů.

V případě, že text a symbol * není uvozen dvojitými uvozovkami, pak je dotaz přeložen jako CONTAINS (column, 'text*'), fulltextový vyhledáváč považuje * za znak a hledá přesný výskyt k text*. Takový text ovšem nebude nalezen, neboť vyhledávač automaticky ignoruje takové znaky.

Pokud je předpona fráze, každé slovo obsažené v dané frázi je zvažováno jako samostatná předpona. Proto dotaz specifikovaný předponou "lokální vín" vyhledá všechny řádky s textem "lokální vinice", "lokální vinohrad", atd.

`Obecný výraz`

Definuje výskyt slov, pokud zahrnutý jednoduchý termín obsahuje varianty původního hledaného slova.

`STREAMER`

Rozlišuje, zda bude docházet k časování, či skloňování, což záleží na typu slovního druhu. Je třeba myslet na to, že každý jazyk má svá specifická pravidla pro ohýbání slov. Neutrální jazyk nemá svá pravidla pro skloňování/časování.

Jazyk sloupce v rámci zkoumaných sloupců je užit jako výchozí zdroj pro identifikování, o jaká pravidla ohýbání slov se jedná. Pokud je jazyk definován, pak jsou s ním pravidla pro ohýbání slov totožná.

`Přibližný tvar`

Určuje výskyt slov nebo frází, které musí být obsažené v hledaném dokumentu. Podobně jako AND operátor, přibližný termín vyžaduje, aby se hledaný termín vyskytoval v dokumentu, ve kterém se vyhledává.

`NEAR | ~`

Signalizuje, že slovo nebo fráze na každé straně NEAR nebo ,se vyskytuje v dokumentu v blízkosti jiného slova. Některé přibližné termíny mohou vytvářet spojený řetěz, jako v případě NEAR a NEAR b NEAR c nebo a b̃ c̃. Aby bylo dosaženo návratu, musí být v takto spojeném řetězci obsaženy všechny termíny.

Pokud použijeme funkci CONTAINSTABLE, tak blízkost hledaných termínů ovlivní hodnocení každého dokumentu. Čím bližší jsou si hledané výrazy v dokumentu, tím výše je dokument oceněn.

NEAR ukazuje logickou vzdálenost mezi termíny, spíše než absolutní vzdálenost mezi nimi.

Posuzovaný výraz

Výsledkem je tabulka obsahující hledaná slova nebo fráze s tím, že záznamy obsahující slova s větší vahou jsou vráceny dříve.

`ISABOUT`

Definuje váhu klíčového slova při vyhledávání.

`VÁHA`

Udává váhu hodnoty, která se pohybuje na stupnici od 0,0 do 1,0. Každá část posuzovaného výrazu může zahrnovat váhu. Definice váhy je způsob, kterým je možné změnit a mnohem přesněji specifikovat dotaz, aby nalezený výsledek přesněji vyhovoval zadaným kritériím. Slovům s větší vahou je tak při vyhledávání dávána vyšší priorita před slovy s vahou nižší.

Váha nemá vliv na výsledek CONTAINS dotazů, ale ovlivňuje hodnocení u CONTAINSTABLE dotazů.

`AND a OR`

Definuje logické operace mezi dvěmi vyhledávacími podmínkami a určuje, zda dvě obsahové vyhledávací podmínky musí být naplněny, aby došlo k výskytu. Znak & se používá namísto klíčového slova AND.

`AND NOT | \&!`

Vypovídá o tom, že druhá vyhledávací podmínka nesmí být přítomna ve výskytu. V případě, že se jedná o AND NOT operátor, použijeme namísto klíčového slova AND NOT znak &!

`OR`

Podobně jako ve všech programovacích jazycích definuje, že jedna nebo druhá CONTAINS vyhledávací podmínka musí být platná.

Pokud podmínka obsahuje skupiny v závorce, pak tyto závorkové skupiny jsou hodnoceny jako první.

Po vyhodnocení výrazů v závorce jsou jednotlivé operátory užity následovně:

- NOT je užito před AND
- NOT se vyskytne za AND jen v případě AND NOT
- NOT nemůže být upřesněno před prvním termínem
- AND se užívá před OR

Fulltextové vlastnosti a funkce jsou schopné pracovat s jednoduchou tabulkou, která v sobě zahrnuje FROM vlastnost. K vyhledávání ve více tabulkách je třeba použít spojené tabulky ve FROM klauzuli tak, aby bylo možno dojít k výsledku založeném na informacích ze dvou či více tabulek.

2.6. Predikát CONTAINSTABLE

Operátor CONTAINSTABLE vrací tabulku s žádným, jedním, čí více řádky pro takové sloupce, které obsahují data s fuzzy shodou jak pro jednotlivá slova, tak i celé fráze, nebo slova s určenou vzdáleností jednoho od druhého. Případně také můžeme zadat váhu jednotlivých slov, čímž upřesníme vyhledávací podmínku. CONTAINSTABLE může být použit pouze ve FROM klauzuli příkazu SELECT, jako by se jednalo o název tabulky.

Dotazy obsahující CONTAINSTABLE upřesňují fulltextový dotaz, který vrací hodnotu RANK a fulltextový klíč KEY pro každý řádek. Funkce CONTAINSTABLE používá pro vyhledávání stejné vyhledávácí podmínky jako predikát CONTAINS.

2.6.1. Syntax

```
CONTAINSTABLE ( tabulka ,
{ název sloupce | (seznam sloupců ) | * } ,
' < vyhledávací podmínka> '
     [ , LANGUAGE [JAZYK]]
  [ , prvních n výskytů ]
         )
< vyhledávací podmínka > ::=
    { < jednoduchý výraz >
    | < předpona >
```

```
    | < obecný výraz >
    | < přibližný výraz >
    |  < posuzovaný výraz >
    }
    | { ( < vyhledávací podmínka > )
    { { AND | & } | { AND NOT | &! } | { OR | | } }
     < vyhledávací podmínka > [ ...n ]
    }
< jednoduchý výraz > ::=
         slova | " fráze "
< předpona > ::=
    { "slovo * " | "fráze *" }
< obecný výraz > ::=
    FORMSOF ( { STREAMER | THESAURUS } ,
             < jednoduchý výraz > [ ,...n ] )
< přibližný výraz > ::=
    { < jednoduchý výraz > | < předpona > }
    { { NEAR | ~ }
    { < jednoduchý výraz > | < předpona > } } [ ...n ]
< posuzovaný termín > ::=
    ISABOUT
       ( { {
  < jednoduchý výraz >
  | < předpona >
  | < obecný výraz >
  | < přibližný výraz >
  }
   [ VÁHA ( weight_value ) ]
   } [ ,...n ]
        )
```

2.6.2. Argumenty

`[TABULKA]`

Tabulka je jméno tabulky, která obsahuje jeden nebo více sloupců, nad kterými je vytvořen fulltextový index. Tabulka dále nemůže specifikovat jméno serveru a nemůže být použita pro dotazování na tabulky uložené na jiných serverech.

`[prvních n výskytů]`

Tento argument omezuje počet řádků vrácených dotazem pouze na prvních n nalezených výskytů vyhovujících zadaným podmínkám. A to za předpokladu, že

hodnota celého nezáporného čísla n je stanovena. Pokud argument specifikující počet vrácených řádků je kombinován s ostatními parametry, pak dotaz dokáže vrátit méně řádků, než je ve skutečnosti počet řádků, které vyhovují všem tvrzením. Tento argument nám tedy umožňuje vyhledávat jen nejvíce relevantní shody.

Zbylé argumenty jsou definovány shodně jako pro CONTAINS.

2.7. Predikát FREETEXT

Predikát FREETEXT je podobně jako predikát CONTAINS použit ve WHERE klauzuli k hledání sloupců obsahující data pro hodnoty, které zachycují význam slov, a ne jen přesný text zadaný ve vyhledávacích podmínkách. Pokud je FREETEXT použit, fulltextové vyhledávání vnitřně spustí následující akce na vyhledávaný řetězec a přiřadí ke každému termínu váhu a počet výskytů.

Predikát FREETEXT dále odděluje řetězce do jednotlivých slov založených na tzv. dělení slov. Vyhledává ohyby slov stejně jako dokáže najít synonyma pro hledané výrazy na základě slovníku synonym (tzv. thezauru).

2.7.1. Syntax

```
FREETEXT ( { jméno sloupce | (seznam sloupců) | * }
          , 'hledaný řetězec' [ , LANGUAGE [JAZYK]] )
```

2.7.2. Argumenty

`[Hledaný řetězec]`

Tento argument je stěžejním vyhledávacím kritériem pro fulltextový dotaz, kdy se hledají jeho výskyty v daném fulltextovém indexu. Textem mohou být jakákoli slova, fráze či věty. Dotaz vrátí TRUE pokud je slovo nebo jeho tvar nalezen ve fulltextovém indexu.

Zbylé argumenty jsou definovány shodně jako pro CONTAINS a CONTAINSTABLE.

Narozdíl od vyhledávacích podmínek CONTAINS a CONTAINSTABLE, ve kterých je AND klíčovým slovem, hledaný řetězec považuje AND za vyloučené slovo a z hledaného řetězce jsou tak jeho výskyty odstraněny.

Použití WEIGHT, FORMSOF, WILDCARDS, NEAR a jiné jazykové skladby není povoleno. Pokud je hledaný řetězec vložen ve složených citačních znaménkách, tzn. “hledaný řetězec“, je dána přednost shodě celé frázi a funkce thezauru se nepoužívá.

Hledaný řetězec je vždy typu nvarchar, pokud je jako počáteční vstup použit i jiný znak než písmeno, proběhne implicitní konverze.

Fulltextové vyhledávání za použití FREETEXT predikátu je ve výsledcích méně přesné než použití predikátu CONTAINS, protože se hledají nejen výskyty zadaného slova nebo fráze, ale také všechny jeho tvary.

Fulltextové vyhledávání za použití FREETEXT predikátu dokáže rozpoznat důležitá slova stejně jako fráze, ovšem klíčovým slovům není přidáván žádný druhotný význam, jako v případě upřesňujících kritérií u predikátu CONTAINS.

2.8. Predikát FREETEXTTABLE

FREETEXTABLE vrací tabulku s žádným, jedním nebo více řádky takových sloupců, které zahrnují data s podobným významem, tzn. nemusí se jednat o naprosto shodná slova textu tak, jak je určen hledaným řetězcem. FREETEXTTABLE je možno použít jen ve FROM klauzuli se SELECT funkcí, jako je běžný název tabulky, podobně jako je tomu u CONTAINSTABLE.

Dotazy používající FREETEXTTABLE určují freetext-type fulltextový dotaz, který vrací RANK a fulltextový klíč KEY pro každý řádek.

2.8.1. Syntax

```
FREETEXTTABLE (tabulka , { název sloupce | (seznam sloupců) | * }
          , 'vyhledávaný řetězec'
     [ , LANGUAGE [JAZYK]]
     [ , prvních n výskytů] )
```

Fulltextové vlastnosti a funkce jsou schopné pracovat s jednoduchou tabulkou, definovanou pomocí FROM. K vyhledávání ve více tabulkách je třeba použít spojení na více tabulkách a definovat FROM klauzuli tak, aby bylo možno dojít k výsledku založenému na informacích ze dvou či více tabulek.

FREETEXTTABLE je třeba používat za stejných podmínek jako predikát FREETEXT (viz podkapitola). Stejně jako CONTAINSTABLE, vyhledané tabulky osahují KEY a RANK.

2.8.2. Argumenty

Argumenty jsou definovány shodně jako pro CONTAINS a CONTAINSTABLE a FREETEXT.

3. Instalace Aplikace

Instalace aplikace pro fulltextové vyhledávání dokumentů se skládá z několika na sobě nezávislých kroků.

- Instalace "tlustého klienta" na klientském počítači
- Změna konfigurace připojení na databázi na klientském počítači
- Instalace a konfigurace SQL serveru
- Založení databáze na SQL serveru, vytvoření fulltextových katalogů a indexů

3.1. Instalace klientské části

Před instalací programu Dokument Management System je nutné mít nainstalován Microsoft .Net Framework ve verzi 4.0 a vyšší. Přesvědčte se proto prosím, že tato součást systému Windows je na vašem počítači nainstalována. Pokud tomu tak není, nainstalujte program .Net Framework podle návodu popsaného níže. V případě, že máte .Net Framework na vašem počítači nainstalován stačí spustit instalační soubor z přiloženého CD. Soubor se nachází v adresáři bin\DocumentManagementSystem. Po spuštění instalačního souboru bude spuštěn instalátor, který vyžaduje zadání adresáře, kam bude program nainstalován. Před prvním spuštěním aplikace je nutné zkontrolovat a případně upravit connection string k databázi, jak je popsáno níže.

3.1.1. Minimální systémové požadavky

- Operační systém Microsoft Windows 2003/2008/XP/Vista/Win7
- Microsoft .NET Framework 4.0

Program Dokument mangement System je určen výlučně pro platformu Windows společnosti Microsoft a to od verze Windows XP s instalovaným servisním balíkem verze 3.0 až po nejnovější verze Windows 7 a Windows Server 2008. Program vyžaduje ke svému bezproblémovému chodu Microsoft .Net framework verze 4.0 a vyšší, což limituje jeho použití v mobilních zařízeních s operačním systémem Windows Mobile. Dále je nezbytné připojit počítač k databázovému serveru, na kterém je umístěna databáze k programu. Je-li tento server na intranetové síti či internetové síti, je nezbytné připojit počítač k této síti tak, aby bylo možné nakonfigurovat spojení se serverem, jak je popsáno níže.

3.2. Instalace serverové části

Než začněte s vlastní instalací instance SQL Serveru, na kterém poběží databáze k programu, ujistěte se, že máte v počítači nainstalovány následující komponenty: Microsoft .NET Framework 3.5, Windows Powershell a Windows Installer 4.5 Redistributable. Ve všech současných verzích operačního systému jsou tyto nástroje dostupné a lze je tak doinstalovat z instalačního CD Windows. V průběhu instalace je nutné vybrat možnost fulltextového vyhledávání (Fulltext indexing services) a také doinstalovat nástroj Microsoft SQL Server Management Studio, který budeme používat pro konfiguraci instalované instance SQL Serveru namísto konzolové aplikace Microsoft Management Console.

3.2.1. Minimální systémové požadavky

- Operační systém Microsoft Windows 2003/2008/Vista/Win7
- Microsoft .NET Framework 3.5
- Windows Powershell
- Windows Installer 4.5 Redistributable

3.2.2. Vytvoření databáze

Pro bezproblémový chod programu je nezbytné mít správně nastavenou databázi, kterou program využívá pro ukládání informací o spravovaných dokumentech. Pro vytvoření prázdné databáze je možné využít připravený skript, který se nachází na přiloženém CD v adresáři: `src\DB-SCRIPTS\DB-CREATE.sql`. Pro spuštění databázového skriptu je nezbytné, aby uživatel, který skript spouští, disponoval na SQL serveru alespoň rolí dbcreator. Pokud spuštění systému neproběhlo úspěšně, je nutné kontaktovat administrátora příslušného SQL Serveru. V případě, že spuštění skriptu proběhlo v pořádku, objeví se v Object Exploreru námi vytvořená databáze s názvem DocumentManagementSystem. Dalším nezbytným krokem je vytvoření fulltextového katalogu a indexu.

3.2.3. Vytvoření fulltextového indexu

Pro správnou funkci fulltextového vyhledávání je nutné vytvořit fulltextový katalog a index. K tomu nám poslouží předpřipravený skript, který se nachází na přiloženém CD v adresáři: `src\DB-SCRIPTS\FULLTEXT-INIT.sql`. Po vykonání databázového skriptu, který nám zpřístupní funkce fulltextového vyhledávání pro naši aplikaci, zbývá už jen přidat předpřipravená data do databáze.

3.2.4. Vytvoření uložených procedur

Vzhledem k faktu, že ani poslední verze Entity Frameworku neumožňuje fulltextové dotazování přímo z objektového modelu aplikace, je nutné vytvořit na straně SQL serveru uložené procedury, které nám zajistí vykonávání samotných fulltextových dotazů. Pro vytvoření procedur musíme spustit připravený databázový skript, který se taktéž nachází na přiloženém CD v adresáři: `src\DB-SCRIPTS\PROCEDURES-INIT.sql`. Po vykonání databázového skriptu můžeme přejít k nepovinnému kroku, kterým je inicializace tabulek výchozími daty.

3.2.5. Inicializace databázových tabulek

I když tento krok není nezbytně nutný pro chod aplikace, je možné do programu importovat některé typy dokumentů, pro které je tato aplikace primárně určena. Pokud se tak rozhodnete učinit, můžete použít připravený databázový skript, který se taktéž nachází na přiloženém CD v adresáři `src\DB-SCRIPTS\DOCUMENT-TYPES-INIT.sql`. Po vykonání databázového skriptu, který nám naplní tabulku obsahující typy dokumentů, můžeme přejít k poslednímu kroku nastavení, kterým je konfigurace connection stringu.

3.3. Napojení klientské aplikace na databázi

Po instalaci aplikace Document Management System, instalaci a konfiguraci SQL Serveru 2008, inicializaci databázových tabulek, načtení uložených procedur, a vytvoření fulltextového katalogu a indexu je nutné nastavit správný connection string. Jedná se o textovou informaci o databázovém serveru, uživateli, který je použit pro přístup k databázi, jeho loginu a heslu. V libovolném textovém editoru otevřete soubor DocumentManagementSystem.exe.config, který naleznete v adresáři který jste vybrali při instalaci aplikace. V tomto souboru vyhledejte sekci, která bude mít následující konfiguraci:

```
<connectionStrings>
    <add name="DocumentManagementSystem.ModelFramework.
                Properties.Settings.ConnectionString"
    connectionString="Data
    Source=[DATABASE SERVER];
    Integrated Security=True"
    providerName="System.Data.SqlClient" />
    <add name="DocumentManagementSystemEntities2"
    connectionString="metadata=
    res://*/DocumentManagementSystemContainer.csdl|
    res://*/DocumentManagementSystemContainer.ssdl|
    res://*/DocumentManagementSystemContainer.msl;
    provider=System.Data.SqlClient;provider
```

```
    connection string=&quot;Data
    Source=[DATABASE SERVER];Initial
    Catalog=DocumentManagementSystem;Integrated
    Security=True;MultipleActiveResultSets=True&quot;"
    providerName="System.Data.EntityClient" />
</connectionStrings>
```

Zde nahraďte všechny výskyty řetězu [DATABASE SERVER] názvem vámi používaného databázového serveru.

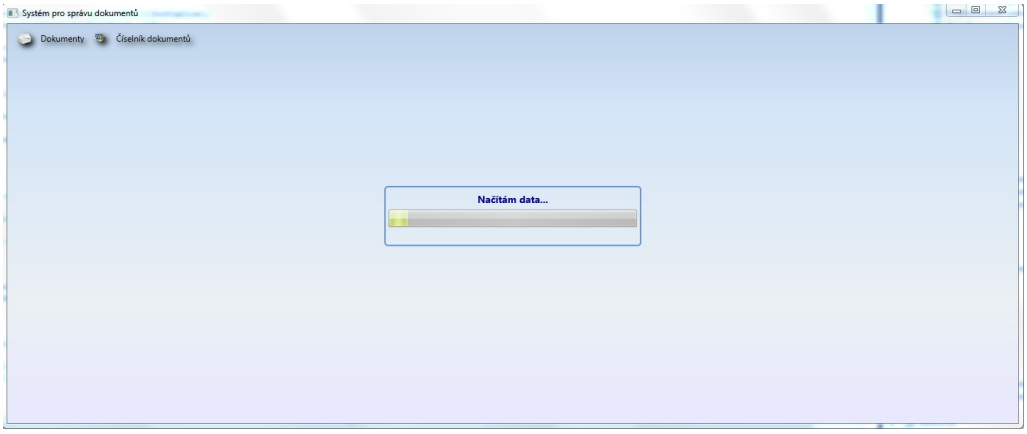
4. Uživatelská dokumentace

4.1. Uživatelský manuál k programu

Při vývoji uživatelského rozhraní byl kladen důraz především na co nejjdnodušší a nejpříjemnější uživatelskou obsluhu celé aplikace. Jak je na první pohled patrné, ovládání aplikace je velice intuitivní a přímočaré a dodržuje veškerá zažitá pravidla a standardy pro vývoj podobných aplikací. Celá aplikace je ovládána myší v kombinaci s klávesnicí, přičemž myš může být plnohodnotně zastoupena jakýmkoliv dotykovým monitorem.

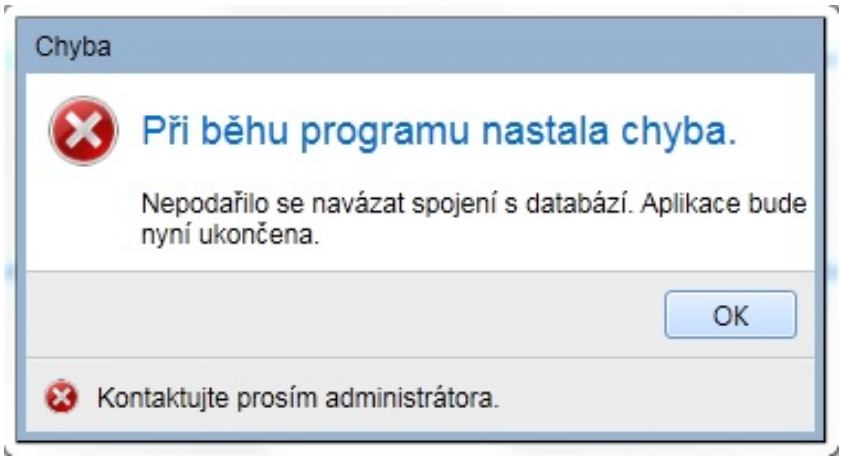
O grafickou prezentaci dat uživateli se stará grafická vrstva vyvýjená na nejnovější technologii Windows Presentation Foundation, která je mnohem líbivější, než jiné dostupné technologie, jakými jsou například často používané Windows Forms.

4.1.1. Start aplikace



Obrázek 1.: Výchozí okno aplikace

Po startu aplikace se uživateli zobrazí úvodní okno s probíhajícím progress barem. Ihned po startu aplikace se ověří spojení s databází a zkontroluje se konzistence dat v tabulkách. Pokud vše proběhne v pořádku, pak se zobrazený progress bar zavře a zůstane zobrazené pouze okno aplikace. V případě že nastane chyba při přístupu k databázi, pak se uživateli zobrazí chybové hlášení:



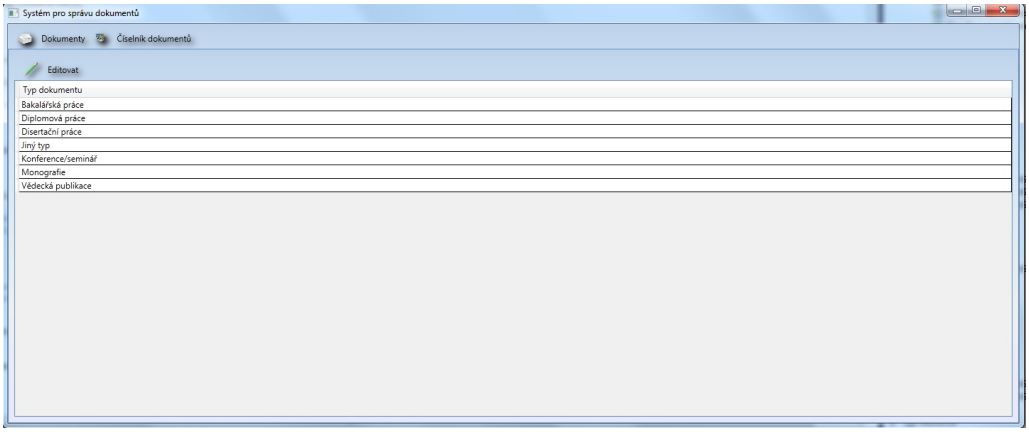
Obrázek 2.: Chyba při inicializaci databáze

V případě, že spojení s databází bylo navázáno úspěšně, pak se uživateli zpřístupní hlavní okno aplikace ve kterém jsou k diuspozici dvě tlačítka.

- Dokumenty – pro vkládání a vyhledávání dokumentů.
- Číselník dokumentů - pro správu typů jednotlivých dokumentů

4.1.2. Číselník dokumentů

Po klidnutí na tlačítko [Čísleník dokumentů] se zobrazí následjící okno:



Obrázek 3.: Čísleník dokumentů

Tato volba umožňuje v aplikaci zadávat nové typy dokumentů. Typ dokumentu je jedním z atributů, které jsou uchovávány v databázi a je tak jedním z kritérií, které lze použít pro pozdější vyhledávání dokumentu.

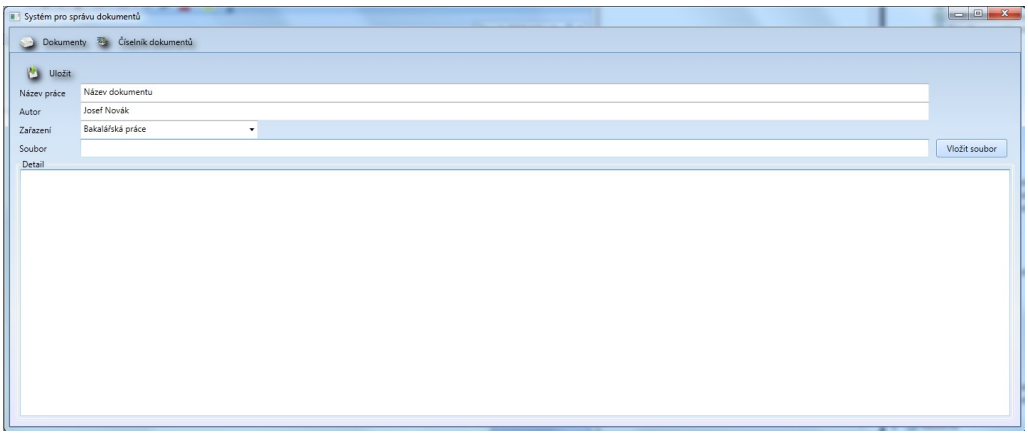
Tato volba tak umožňuje rozšiřovat stávající typy dokumentů, případně definovat zcela novou sadu typů. Vzhledem k tomu, že je databázová tabulka s typy dokumentů plně editovatelná, je možné v programu definovat zcela nové typy, což znamená, že apliakce se dá jednoduše využít pro ukládání zcela jiných

dokumentů například v malém podnikovém systému.

Pro editaci tabulky je nutné kliknout na tlačítko [Editovat], čímž se tlačítko změní na tlačítko [Uložit]. Když uživatel skončí s editací tohoto formuláře, je nutné provedené změny potvrdit tlačítkem [Uložit].

4.1.3. Vložení nového dokumentu

V případě, že uživatel klikne na tlačítko [Vložit dokument] zobrazí se mu následující okno:



Obrázek 4.: Přidání nového dokumentu

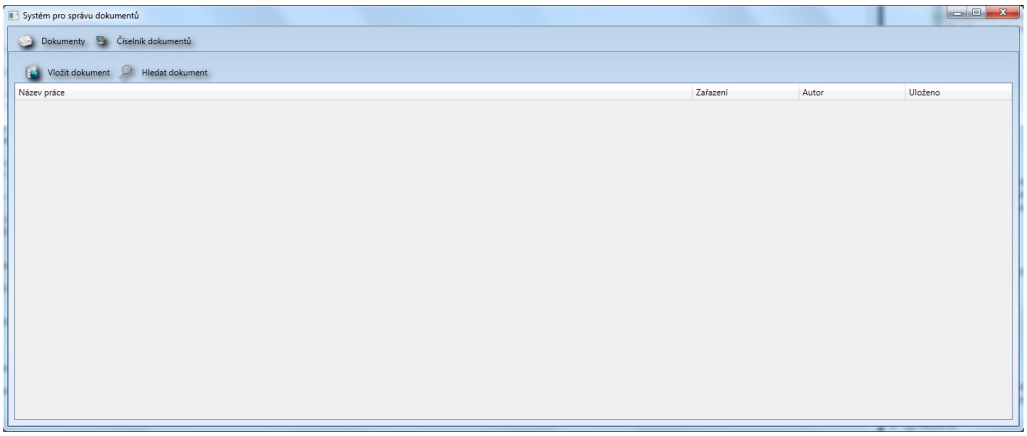
Z tohoto formuláře je možné přidávat nové záznamy do databáze. Uživatel může zadat následující údaje:

- Název práce
- Autora
- Zařazení - typ dokuemntu
- Soubor

Po přidání souboru, se pokusí aplikace získat všechny dostupné informace, a v příápadě, že tak ještě uživatel neučinil sám, vyplní název práce, autora dokumentu, zařazení práce a pokusí se získat text z vybraného souboru. Pokud aplikace nedokáže získat plain text z dokumentu, nebude možné nad daným dokumentem použít fulltextového vyhledávání.

4.1.4. Seznam dokumentů

V případě, že uživatel klikne na tlačítko [Dokumenty] zobrazí se mu následující okno:

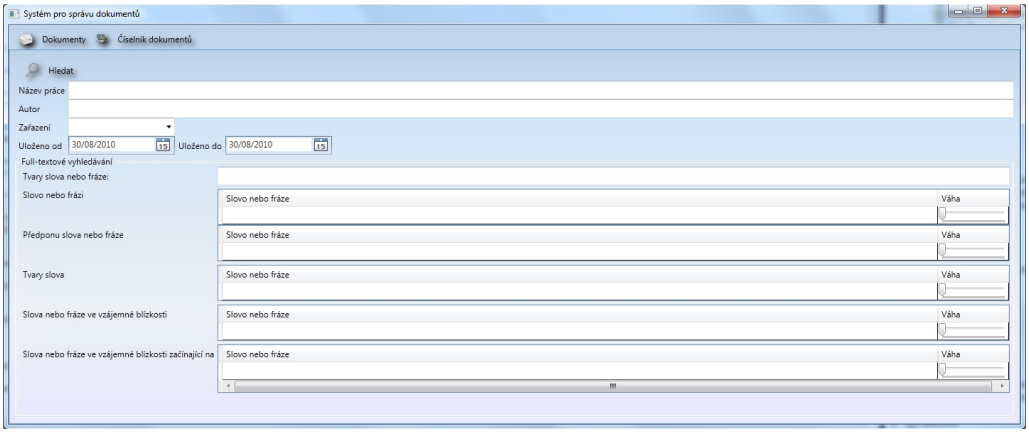


Obrázek 5.: Seznam nalezených dokumentů

Toto okno slouží k zobrazení výsledků nalezených pomocí fulltextového vyhledávání. Při prvním kliknutí na toto tlačítko, je okno vždy prázdné, jinak je zde zobrazen výsledek posledního vyhledávání. Nalezené dokumenty je možné zobrazit, poku uživatel 2x klikne na vybraný dokument. Je nutné aby byl na lokálním počítači nainstalovám prohlížeč daného typu dokumentu. Pokud tomu tak není, bude uživatel vyzván, aby vybral jiný program ve kterém se má daný soubor otevřít.

4.1.5. Vyhledávání dokumentů

Pro vyhledávání dokumentů slouží následující okno:



Obrázek 6.: Vyhledávání dokumentů

Tento formulář je klíčovým místem celé aplikace. Uživatel má možnost zadat zde celu řadu kritérií, která složí k nalezení hledaného dokumentu.

- Název práce
- Autor
- Zařazení - typ dokuemntu
- Uloženo od a Uloženo do - datum uložení dokumentu
- Tvary slova nebo fráze - hledá ohyby slova
- Předponu slova nebo fráze
- Tvary slova
- Slova nebo fráze ve vzájemné blízkosti
- Slova nebo fráze ve vzájemné blízkosti začínající na

5. Programová dokumentace

5.1. Použité návrhové vzory

5.1.1. MODEL VIEW CONTROLLER(MVC)

Existuje několik zavedených modelů pro interakci mezi uživatelským rozhraním a daty. Ve své aplikaci jsem se rozhodl pro osvědčený návrhový vzor MVC, který se jeví pro použití s grafickým rozhraním WPF jako nejvíce praktický. Standardní návrhový vzor užívaný WPF není dostatečně vhodný pro testování dat pomocí unit testů a navíc nastavají nemalé komplikace při použití data bindingu. Tehdy je grafická část aplikace načítána do paměti souběžně s daty a zejména kód využívající více procesů může způsobit nemalé problémy s interkací.

MVC představuje vzorec pro tento vztah, neboť dokáže oddělit vrstvy aplikace do následujících tří částí:

- Model – data aplikace ve formě samostatné třídy.
- View (pohled) - je zodpovědný za zobrazování dat uživateli a odchycení událostí a předání jejich zpacování controlleru
- Controller – je řídící třída, která obsahuje jak modely jako data, tak view.

Je důležité poznamenat tři základní vztahy mezi Model, View a Controllers:

- Model - existuje samostatně bez ohledu na View nebo Controller
- Controller - vlastní všechny modely a své View
- View - vlastní svůj Controller a přes controller dokáže přistupovat k datům v modelech.

V následujících odstavcích stručně popíši implementaci MVC modelu v mé aplikaci.

Model Každý model musí být typu parcial EntityObject, což znamená že je možné použít entity framework pro práci s tímto objektem. Objekty musí být partial, protože entity framework s každou změnou v modelu vygeneruje objekt znovu a všechny uživatelem provedené změny jsou tak zahozeny, avšak změny v příslušné partial třídě zůstanou nedotčeny.

Modely dělíme na dva druhy:

- Jednoduchý Model, který je nutno zaregistrovat ke Controller použitím SetModel funkce.

- Kolekci modelů, kterou je nutno zaregistrovat ke Controlleru použitím SetModelCollection funkce.

Ukázka SetModel funkce:

```
protected void SetModel<T>(Expression<Func<T>>
                           fieldExpression, EntityObject model)
where T : EntityObject
{
  if (model == null)
    throw new ArgumentNullException("model");
  ObservableCollection<EntityObject> modelCollection =
            new ObservableCollection<EntityObject>();

  modelCollection.Add(model);

  SetModelCollection(fieldExpression, modelCollection);
}
```

Potřeba registrovat modely ke controllerům existuje jen v případě, pokud je možné model editovat pomocí view. To je nutné z toho důvodu, že BaseController volá funkci Save a Validate pro všechny modely které mu byly přiřazeny.

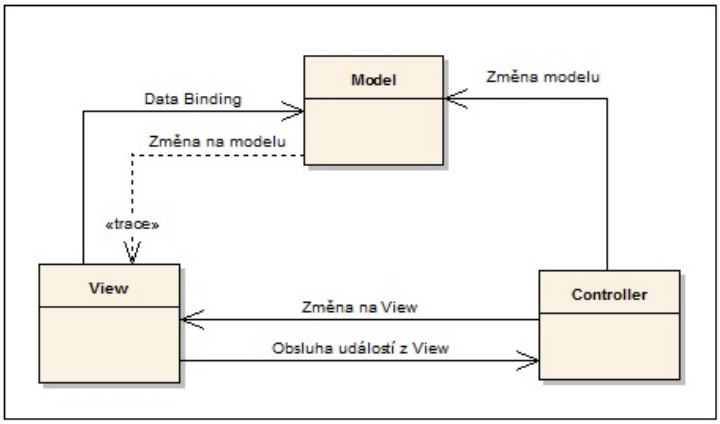
Pro načtení modelu slouží funkce GetModel:

```
protected T GetModel<T>(Expression<Func<T>> fieldExpression)
        where T : EntityObject
{
  string key = ReflectionHelper.GetPropertyName<T>(fieldExpression);
  if (_models.ContainsKey(key))
  {
    IList modelCollection = _models[key];
    if (modelCollection.Count > 0)
      return (T)modelCollection[0];
    else
      return default(T);
  }
  else
    return default(T);
}
```

View Ve své aplikaci používám dva druhy. První typ je zděděn z ControllerBase a reprezentuje modifikovaný UserControl. Jako příklad můžeme uvést View zobrazující seznam nalezených dokumentů. Druhý typ je zděden z WindowBase a reprezentuje modifikaci Window. Jako příklad slouží všechna dialogová okna

v aplikaci. Pro oba typy je typické, že samy o sobě nedrží žádná data a ani s nimi nijak nepracují. Všechny události jsou předávány ke zpracování příslušnému controlleru, který provádí jejich zpracování. View tak slouží opravdu jenom jako UI aplikace a načítá všechna data pomocí data bindingu přes controller. Každé View obsahuje informaci o controlleru, který jej vytvořil. Proto je nutné vytvořit pro každé View takovou implementaci, která použije správnou instanci daného controlleru. Všechna View v aplikaci jsou tvořena pomocí statické třídy ViewFactory, která vytváří a následně vrací vždy jednu a tu samou instanci View, čímž se šetří čas i paměť počítače při jeho opětovném použití. Tato třída také při vytváření nového View nastaví jako DataContext svůj controller, který volá vytvoření View.

Controller Controller je řídící element celé trojice. Obsahuje jeden či více modelů a má informaci o svém view. Každý controller je zděděn z ControllerBase a vždy existuje pouze jedna instance daného controleru. Ke správě controllerů slouží statická třída ControllerFactory, která hlídá, že v daný moment existuje vždy pouze jeden controller. Nedochází tak k situaci že by pro jedno view existovalo více controllerů, což brání případným kolizím (kdy by jedny data existovaly v aplikaci dvakrát), a také to šetří čas a paměť počítače.



Obrázek 7.: Model-View-Controller

5.1.2. Návrhový vzor Factory

Jádro frameworku celé aplikace je založeno na návrhovém vzoru Factory neboli česky továrna. Ten to je jedním ze základních návrhových vzorů z množiny takzvaných creation patterns. Implementací tohoto návrhového můžeme najít

ve statické DynamicControllerFactoryBase. Vzhledem k faktu, že zmíněná třída se nachází ve jmenném prostoru Core.Base a vzhledem k tomu, že se jedná o třídu statickou, lze k ní přistupovat z hlavního jmenného prostoru celé aplikace v každém jejím bodě. V kombinaci s návrhovými vzory singleton a Model-View-Controller nám pak tato třída vždy vrací právě jednu instanci controlleru. Výhoda tohoto zapouzdření spočívá především v tom, že tato třída je zodpovědná za vytváření a uchovávání instance controllerů, což v našem případě znamená, že je tato třída zodpovědná za správu celé trojice Model-View-Controller a za udržení správných referencí mezi nimi. Další nespornou výhodou této konstrukce je fakt, že kdykoliv za běhu aplikace potřebujeme získat instanci daného controlleru, případně jeho View, získáme dané objekty vždy ve stejném stavu v jakém jsme je zanechali po posledním volání.

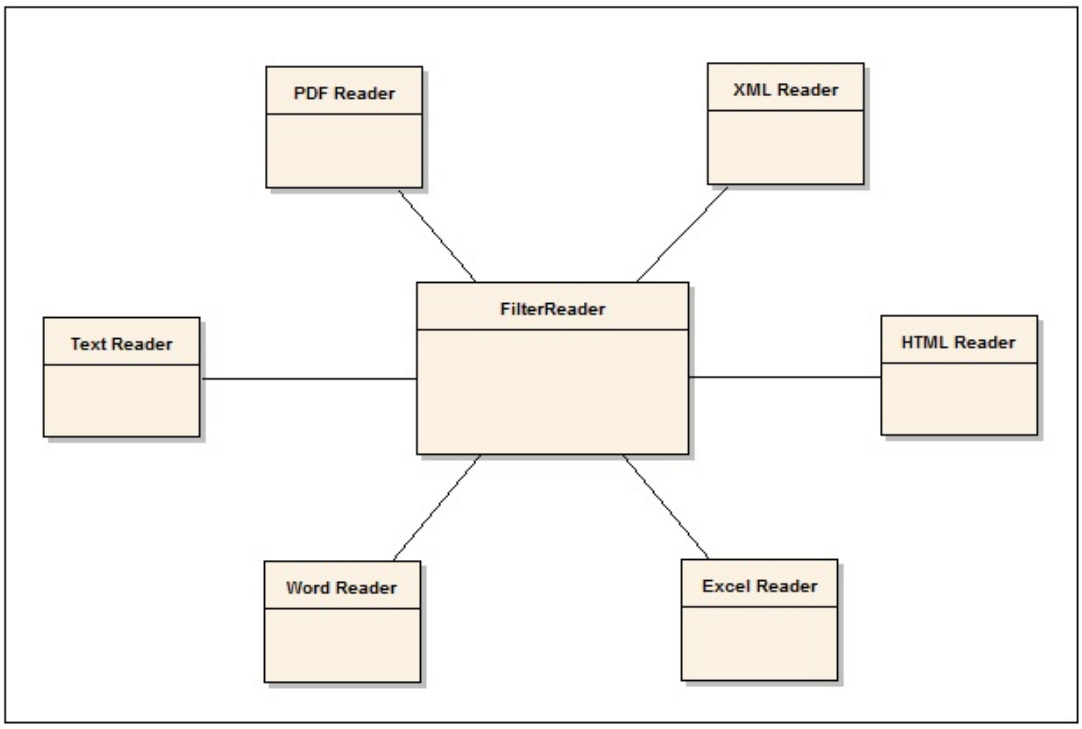
5.1.3. Návrhový vzor Singleton

Nutností pro návrh frameworku aplikace bylo využití dalšího návrhového vzoru z rodiny takzvaných creation patterns, tedy z rodiny vytvářecích návrhových vzorů. Implementace tohoto návrhového vzoru nám zajistí, že aplikace bude při svém chodu vždy využívat pouze jednu instanci konkrétního datového typu. Výhodou tohoto řešení je, že tímto způsobem navržená aplikace zbytečně neplýtvá dostupnými zdroji, protože v každém momentu životního cyklu aplikace existuje vždy právě jedna nebo žádná instance daného objektu.

Implementace tohoto návrhového vzoru společně s implementací návrhového vzoru factory nám zajistí, že kdykoliv během běhu programu potřebuje získat controller daného View, případně controller okna, získáme vždy stejný controller, který vždy drží instanci svého View, které řídí a zároveň drží instanci svých modelů, jež obsahují zobrazovaná data.

5.1.4. Návrhový vzor Facade

Tento návrhový vzor se využívá k zapouzdření sytému tříd pokrývajících podobnou funkcionalitu, nicméně s rozdílnou implementací, což zvyšuje udržovatelnost a čistotu architektury.
Důvodem užití tohoto návrhového vzoru v mé bakalářské práci bylo napsat jeden jednoduchý interface, jehož jediným vstupem bude cesta k dokumentu, kdy na výstupu vždy získáme plain text zadaného dokumentu. Tento přístup nám tak umožňuje zapouzdření vlastních filtrovacích algoritmů do jedné třídy, která na základě svého argumentu sama vybere vhodný typ algoritmu pro získání požadovaných dat.



Obrázek 8.: Facade

5.1.5. Implementace návrhový vzorů

Vzhledem k faktu, že výše popsané návrhové vzory tvoří jádro frameworku tohoto systému pro správu dokumentů, popíšeme si blíže implementaci třídy DynamicControllerFactoryBase, která je využívána při inicializaci nových controllerů. Implementaci celé třídy tvoří dvě veřejné statické metody a jedna privátní statická kolekce:

Privátní statická kolekce slouží pro uchování již jednou vytvořených controllerů a její využití nám reprezentuje návrhový vzor Singleton.

```
private static readonly Dictionary<string, ControllerBase>
  _controllers = new Dictionary<string, ControllerBase>();
```

První generická metoda:

```
/// <summary>
/// Creates base controller for a DynamicViewBase
/// </summary>
/// <typeparam name="ControllerType">Type of the controller</typeparam>
/// <typeparam name="DynamicViewType">Type of the View</typeparam>
/// <returns>Controller</returns>
public static ControllerType CreateViewController
  <ControllerType, DynamicViewType>()
where DynamicViewType : DynamicViewBase
```

```
where ControllerType : DynamicViewController
{
ControllerType controller;
string typeName = typeof(ControllerType).FullName;
string viewtypeName = typeof(DynamicViewType).FullName;
  string key = typeName;

  if (_controllers.ContainsKey(key))
{
//existing controller
controller = _controllers[key] as ControllerType;
}
else
{
Assembly executingAssembly = typeof(ControllerType).Assembly;
object[] args = new object[] { };
controller =
    (ControllerType)executingAssembly.CreateInstance(typeName,
    false, BindingFlags.Instance | BindingFlags.Public, null,
    args, System.Globalization.CultureInfo.CurrentCulture, null);

   DynamicViewBase view =
    (DynamicViewBase)executingAssembly.CreateInstance(viewtypeName,
    false, BindingFlags.Instance | BindingFlags.Public,
    null, null, System.Globalization.CultureInfo.CurrentCulture, null);

    controller.SetView<DynamicViewType>(view);

view.SetController<ControllerType>(controller);
_controllers.Add(key, controller);
}

  return controller;
}
```

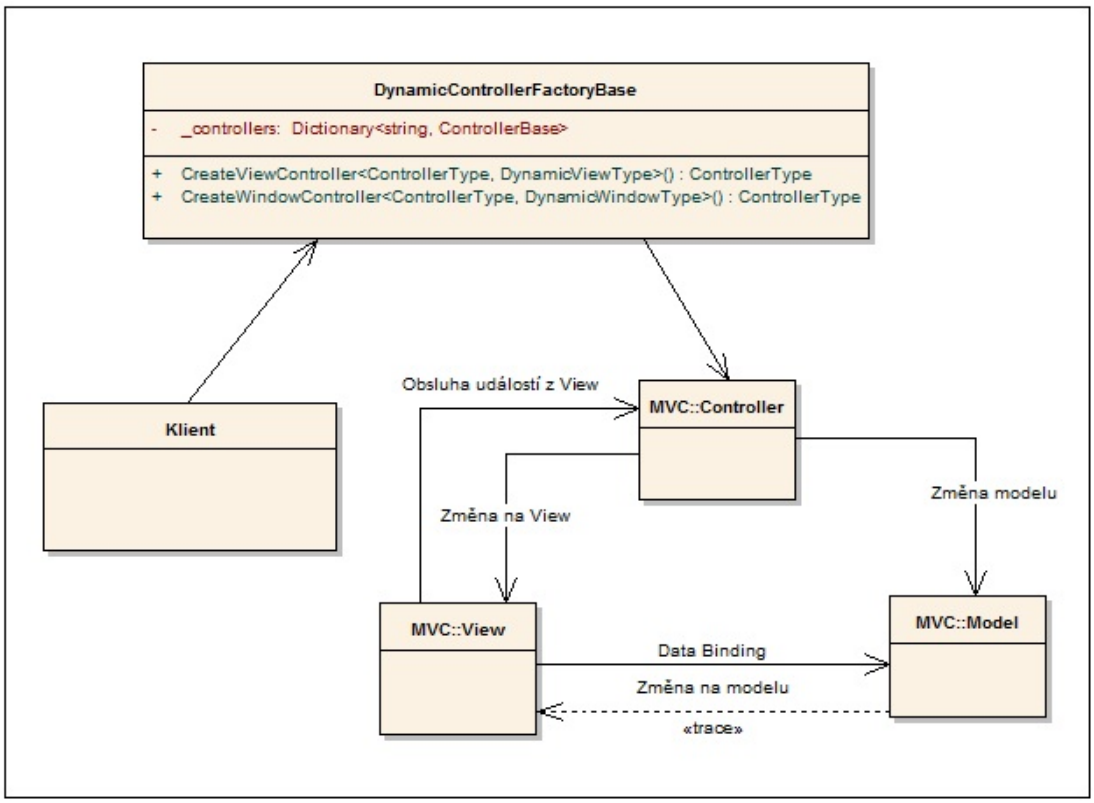
Tato metoda je volána se dvěma generickými typy, které určují typ vytvářeného controlleru a typ jeho View. Metoda zkontroluje privátní kolekci controllerů, zda již byl controller tohoto typu vytvořen. Nebyl-li controller ještě vytvořen, tak se za použití reflection vytvoří nová instance controlleru stejně tak, jako nová instance jeho View, a nastaví se vazby mezi nimi. Nakonec se instance nově vytvořeného controlleru přidá do privátní kolekce, ve které bude při příštím volání metody nalezen a vrácen.

Obdobně funguje druhá metoda této třídy:

```
/// <summary>
/// Creates base controller for a DynamicWindowBase
/// </summary>
/// <typeparam name="ControllerType">Type of the Controller</typeparam>
/// <typeparam name="DynamicViewType">Type of the Window</typeparam>
/// <returns>Controller</returns>
public static ControllerType CreateWindowController
  <ControllerType, DynamicWindowType>(DynamicWindowType window = null)
where DynamicWindowType : DynamicWindowBase
where ControllerType : DynamicWindowController
{
ControllerType controller;
string typeName = typeof(ControllerType).FullName;
string viewtypeName = typeof(DynamicWindowType).FullName;
  string key = typeName;
Assembly executingAssembly = typeof(ControllerType).Assembly;

if (_controllers.ContainsKey(key))
{
//return existing controller
return controller = _controllers[key] as ControllerType;
}
else
{
object[] args = new object[] { };
controller =
    ControllerType)executingAssembly.CreateInstance(typeName,
    false, BindingFlags.Instance | BindingFlags.Public, null,
    args, System.Globalization.CultureInfo.CurrentCulture, null);

controller.SetView<DynamicWindowType>(window);
}

  if (window == null)
{
DynamicWindowBase view =
    (DynamicWindowBase)executingAssembly.CreateInstance(viewtypeName,
    false, BindingFlags.Instance | BindingFlags.Public, null, null,
    System.Globalization.CultureInfo.CurrentCulture, null);

   controller.SetView<DynamicWindowType>(view);
```

```
view.SetController<ControllerType>(controller);
}
else
{
window.SetController<ControllerType>(controller);
}

  _controllers.Add(key, controller);

//return new created controller
return controller;
}
```

Jediným rozdílem oproti implementaci metody CreateViewController<ControllerType, DynamicViewType>(), je možnost předat metodě jako argument existující objekt typu DynamicWindowBase. Tato konstrukce je zde z toho důvodu, že hlavní okno apliakce je vždy vytvořeno dříve, než jeho controller a pro tento specifický případ je zde možnost předat již existující okno.



Obrázek 9.: Factory model

6. Entity Framework

Při návrhu svého programu jsem se rozhodl použít .Net Entity Framework verze 2.0. Tento framework funguje jako ORMapper mezi relační databází SQL Serveru 2008 a objektovým prostředím .Net frameworku. Entity framework tak nabízí širokou škálu funkcí jak přistupovat k datům z SQL Serveru a zachází s těmito daty jako s objekty. Programátor se tak nemusí starat o přesnou implementaci každé třídy, která reprezentuje databázová data, protože Entity Framework pro něj dokáže implementaci vygenerovat. Tento přístup značně šetří čas při vývoji aplikace a minimalizuje takzvaný refactoring při změnách v databázi.

Entity Framework tak představuje jakousi alternativu ke staršímu, a velmi často používanému LINQ to SQL, se kterým se nikterak nevylučuje, a dokonce s ním lze velmi vhodně kombinovat. Pomocí LINQ lze totiž nejen přistupovat k datovým zdrojům jako jsou in-memory datové struktury, xml dokumenty či SQL, SQL Compact, nebo Oracle databáze, ale také entity models a DataSets, které Entity Framework využívá k mapování.

6.1. LINQ a Entity Framework

Aktuální verze .Net frameworku zahrnuje vrstvu, která může obsahovat data v databázi jako regulérní .Net objekty. Navíc, je možno vygenerovat pro tyto objekty takzvané partial třídy, které reprezentují Entity Data Model schéma v prostředí .Net Frameworku. Toto vytváří z předmětové vrsty ideální cíl pro LINQ, který umožnuje formulovat dotazy pro databázi přímo ve zdrojového kódu progaramovacího jazyka. Tato vlastnost je nazývána jako LINQ to Entities a její nespornou výhodou je to, že sestavený dotaz je kontrolován při kompilaci programu, a je tedy kontrolován hned při překladu a ne až při pokusu spustit dotaz na SQL serveru. Pro jednoduché dotazy tak odpadá složité ladění pomocí SQL Profileru. Bohužel, ani Linq to Entities, ani Entity Framework neumožňují přímo sestavovat fulltextové dotazy, proto jedinou cestou jak lze zakomponovat fulltextové dotazování do objektového modelu generovaného Entity Frameworkem je volání takzvaných uložených procedur přímo z SQL Serveru. Entity Framework se ke všem uloženým procedurám chová jako k metodám a proto nám tento postup umožní zachovat k databázi objektový přístup. Alternativou uloženým procedurám na straně SQL Serveru by mohlo být použití TableAdaptérů a DataSetů, čímž bychom ale narušili objektový přístup k relační databázi.

Závěr

Cílem mé bakalářské práce bylo vytvoření jednoduchého a uživatelsky příjemného programu, který na základě uživatelem zadaných parametrů, které popisují hledaný dokument, vyhledal v databází SQL Serveru relevantní dokumenty, které nejpřesněji splňují kritéria vyhledávací podmínky.

Smyslem celého projektu bylo vytvoření apliakce, která by demonstrovala možnosti využití fulltextového vyhledávání nad různými typy dokumentů v praxi, nikoliv však pokrytí kompletní problematiky fulltextového vyhledávání jako celku.

Bakalářská práce se skládá ze dvou částí. Klientské části, tedy programu Document Management System, který musí být nainstalován na každém počítači, ze kterého bude aplikace používána a ze serverové části tvořené databází běžící na SQL Servereru společnosti Microsoft.

Celá aplikace byla navržena a naprogramována tak, aby splňovala přísné standardy společnosti Microsoft, pro vývoj aplikací v prostředí .Net frameworku a jazyku C# . Aplikace také využívá návrhových vzorů určených pro tento typ programů a snaží se předvést nějnovější trendy ve vývoji software.

Conclusions

The aim of my Bachelor thesis was to create simple and user friendly program, which would be able to find relevant documents on SQL Server, which are described by inserted parameters.

The main reason of building this software was to show possibilities and capabilities of fulltext search over different document types and application of this knowledge in real application, instead try to cover whole fulltext search functionality offered by SQL Server.

My Bachelor thesis is based on two separate parts. Client software called Document Management System, which needs to be installed on each client computer where we wants to use this application and from server part, which is created by SQL Server database from Microsoft company.

This application keeps all patterns and standards developed by Microsoft company, which are used for software development on .Net framework and C# language. This application also follows design patterns used from this kind of programs and it shows the latest trends in software development.

Reference

[1] Ben-Gan, Itzik – Kollar, Lubor – Sarka, Dejan – Kass, Steve. *Inside Microsoft® SQL Server® 2008: T-SQL Querying.* Microsoft Press, 2009.

[2] Fritchey, Grant – Dam, Sajal. *SQL Server 2008 Query Performance Tuning Distilled.* Apress, 2009.

[3] Lopez Azpeita, Fernando. *SQL Server 2008 Full-Text Search: Internals and Enhancements.* Microsoft Corp, 2008.

[4] *MSDN Microsoft Developer Network.* Elektronická publikace, 2008.

A. Popis obsahu přiloženého CD

Součástí této práce je i přiložené médium CD. Jeho obsahem jsou čtyři adresáře.

- Adresář bin obsahuje podadresář Install s instalačním souborem aplikace
- Adresář doc obsahuje dokumentaci této práce, včetně zdrojových kódů této dokumentace.
- Adresář Net Framework obsahuje instalační soubor s poslední verzí Microsoft .NET Frameworku.
- Adresář src obsahuje zdrojové kódy aplikací rozdělené v podadresářích.
  - Podadresář DB-SCRIPTS, který obsahuje sql skripty nutné pro vytvoření databáze, inicializaci databázových tabulek a pro nastavení fulltextového indexu a katalogu.
  - Podadresář DocumentManagementSystem, který obsahuje zdrojové kódy celé aplikace pro prostředí Microsoft Visual Studio 2010.
  - Podadresář EA, který obsahuje UML diagramy v programu Enterprise Architect.

Součástí přiloženého média CD je i soubor readme.txt, který obsahuje informace o obsahu přiloženého CD.