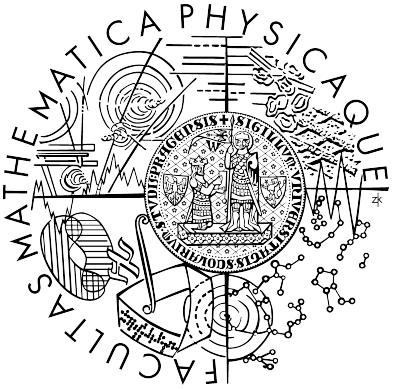
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Hudeček

Software pro česko-čínský a čínsko-český slovník

Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Petr Homola
Studijní program: Obecná informatika

2006

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu bakalářské práce panu Mgr. Petru Homolovi za podnětné připomínky a vedení. Dále bych chtěl poděkovat studentu sinologie Martinu Křížovi za odborné konzultace v oblasti čínštiny a zadání ukázkových dat.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 10.8.2006. Jan Hudeček

Obsah

Název práce: Software pro česko-čínský a čínsko-český slovník
Autor: Jan Hudeček
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Mgr. Petr Homola
e-mail vedoucího: homola@ufal.mff.cuni.cz

Abstrakt: Česko-čínský a čínsko český slovník je elektronický slovník použitelný jak pro začátečníka, tak pro zkušeného překladatele. Obsahuje podporu pro obousměrné vyhledávání slov i fulltextové prohledávání slovníku pro výskyt daného výrazu. Přístup k datům je hybridní – pokud je k dispozici databáze, použije ji, pokud není, načte datový soubor. Metodu přístupu k datům je možné za běhu programu měnit. Nad datovým souborem se budují indexy – implementované jako hashovací tabulky nebo binární stromy. V rámci zpříjemnění uživatelského prostředí bylo použito asynchronní více-vláknové načítání dat. Implementace na platformě .NET a MS SQL 2000 umožňuje snadné rozšiřování – například ve formě webové aplikace. Zároveň by měl být návrh architektury dostatečně pružný, aby v budoucnu dovolil editaci dat slovníku.

Klíčová slova: slovník, datové struktury, .NET

Title: Software for a Czech-Chinese and Chinese-Czech dictionary
Author: Jan Hudeček
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Petr Homola
Supervisor's e-mail address: homola@ufal.mff.cuni.cz

Abstract: Czech-Chinese and Chinese-Czech dictionary is an electronic dictionary which can be used both by a beginner or a seasoned translator. It allows searching in both directions and a fulltext search for given expression. Data access is hybrid – the program checks if it can access the database – if it fails it reads the data files. Moreover users can change the data source at run-time. The program builds indexes on the data file speeding searches up considerably. Indexes can be hashtables or binary trees. Asynchronous multi-threaded IO was implemented to enhance the comfort of the GUI. The .NET framework and MS SQL Server as a platform guarantees rapid development, deployment and scalability – for example adding a web application to the project would be quite easy. At the same time the design of the system allows for future improvements – for instance editing the dictionary from the GUI.

Keywords: dictionary, data structures, .NET

Kapitola 1 – Úvod

1.1 Cíl

Cílem této bakalářské práce je vytvořit elektronický česko-čínský a čínsko-český slovník, který bude použitelný jak pro úplného začátečníka tak i pro zkušeného sinologa. Slovník musí být snadno použitelný – mělo by být přímočaré najít ekvivalent pro konkrétní slovo, ale zároveň zobrazovat i příklady použití a gramatické kategorie daných slov. Návrh slovníku by měl umožňovat snadné rozšiřování, měl by mít pružný přístup k datům, ale zároveň pracovat v reálném čase. Nepovinným cílem je i editace dat slovníku a/nebo vytváření vlastního uživatelského slovníku.

1.2 Motivace

Cíl bakalářské práce vychází z nedostatku elektronických česko-čínských slovníků (v době vydání práce nebyl autoru znám žádný). Rovněž lexikografická kvalita tištěných slovníků značně pokulhávala za požadavky odborné veřejnosti. V době zvyšující se kulturní a ekonomické důležitosti Číny je to zásadní nedostatek. Slovník byl tedy navrhován s tím, aby poskytoval co nejvíce informací, co nejpřehlednější formou a aby obsahoval i informace o slohovém zařazení slova, eventuelně oboru do jakého spadá – pokud se jedná o odborný výraz.

1.3 Přehled dalších kapitol

Druhá kapitola se zabývá uživatelskou analýzou a návrhem aplikace. Kapitola obsahuje funkční a obecné požadavky s odůvodněním, způsoby užití a plány vývoje.

Třetí kapitola obsahuje designérskou analýzu a mohla by sloužit i jako programátorská příručka. Jsou zde popsány jak datový model tak i dynamické procesy a klíčová rozhodnutí při vývoji aplikace. Konkrétní rozhodnutí důležitá pro výkon aplikace jsou podložena měřeními.

Závěrečná kapitola hodnotí splnění cíle bakalářské práce a zkušenosti získané při vývoji.

Kapitola 2 – Analýza a návrh

2.1 Stávající elektronické slovníky

Před samotným popisem postupu vývoje je třeba zmínit stávající elektronické slovníky, které sloužily jako inspirace. Především je to Collins Cobuild – Lingea Lexicon, anglický výkladový slovník. Tento slovník ovlivnil autora při návrhu jak se budou nalezená hesla zobrazovat a v rozsahu uchovávaných dat. Dalším slovníkem, který významně ovlivnil návrh, je anglicko-čínský Kingsoft Powerword 2003. Z tohoto slovníku vycházel návrh práce s transliterací pin yin.

2.2 Návrh

2.2.1 Datová struktura

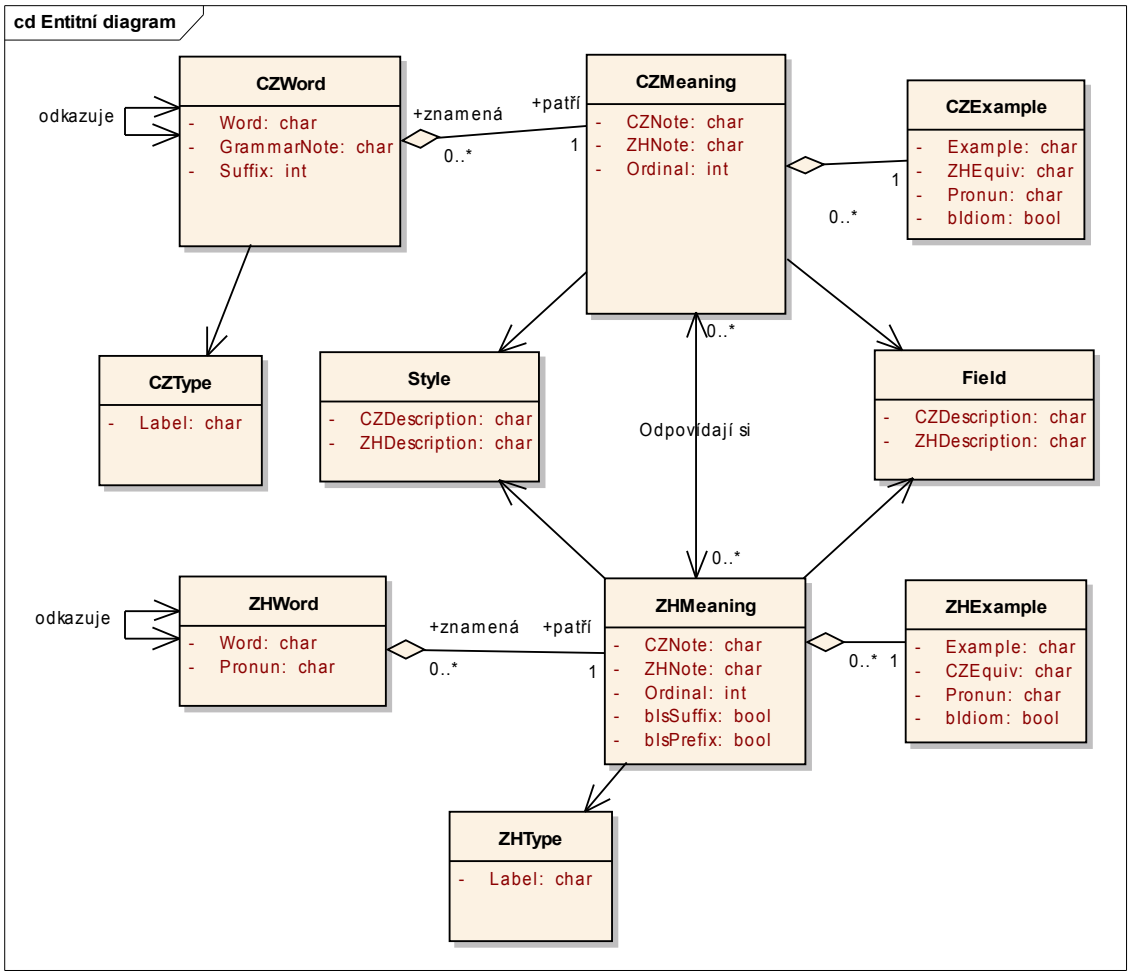
Z konzultací se sinology a z odborné literatury byly získány základní požadavky na datovou strukturu:

- základní linii slovníku bude tvořit vztah:

  české slovo – jeho významy – významy čínských slov – čínská slova
- příklady se musí pojit k významům slov
- je třeba rozlišovat mezi českými a čínskými slovními druhy
- bude vhodné rozdělit české slovo na kořen a koncovku
- u příkladů i u slov by měla být uvedena výslovnost
- datová struktura musí obsahovat místo pro poznámky popisující nepravidelnosti
- mezi slovy by mělo být možné vytvářet odkazy pro propojení slov, které spolu logicky nebo gramaticky souvisí
- u významů je třeba určit pořadí, v jakém se budou vypisovat
- u čínského významu by mělo být specifikováno zda se jedná o samostatné slovo nebo o předponu či příponu
- jelikož v čínštině jsou jednotlivé významy slov rozmanitější, měl by být slovní druh na čínské straně přiřazen až k významu
- u příkladů by mělo být uvedeno, že se jedná o idiom

Tyto požadavky vedly autora k datové struktuře se schématem viz obr.1

Obrázek 1 Entitní diagram



Hlavní vztah mezi slovy je v této struktuře CZWord – CZMeaning – ZHMeaning ZHWord. Bylo zvoleno anglické názvosloví, pro případ, že by projekt čínského slovníku byl zveřejněn i se zdrojovými kódy. Zkratka ZH vychází z čínského zhongguo de - čínský. Povšimněte si, že mezi CZMeaning a ZHMeaning je vztah n:m.

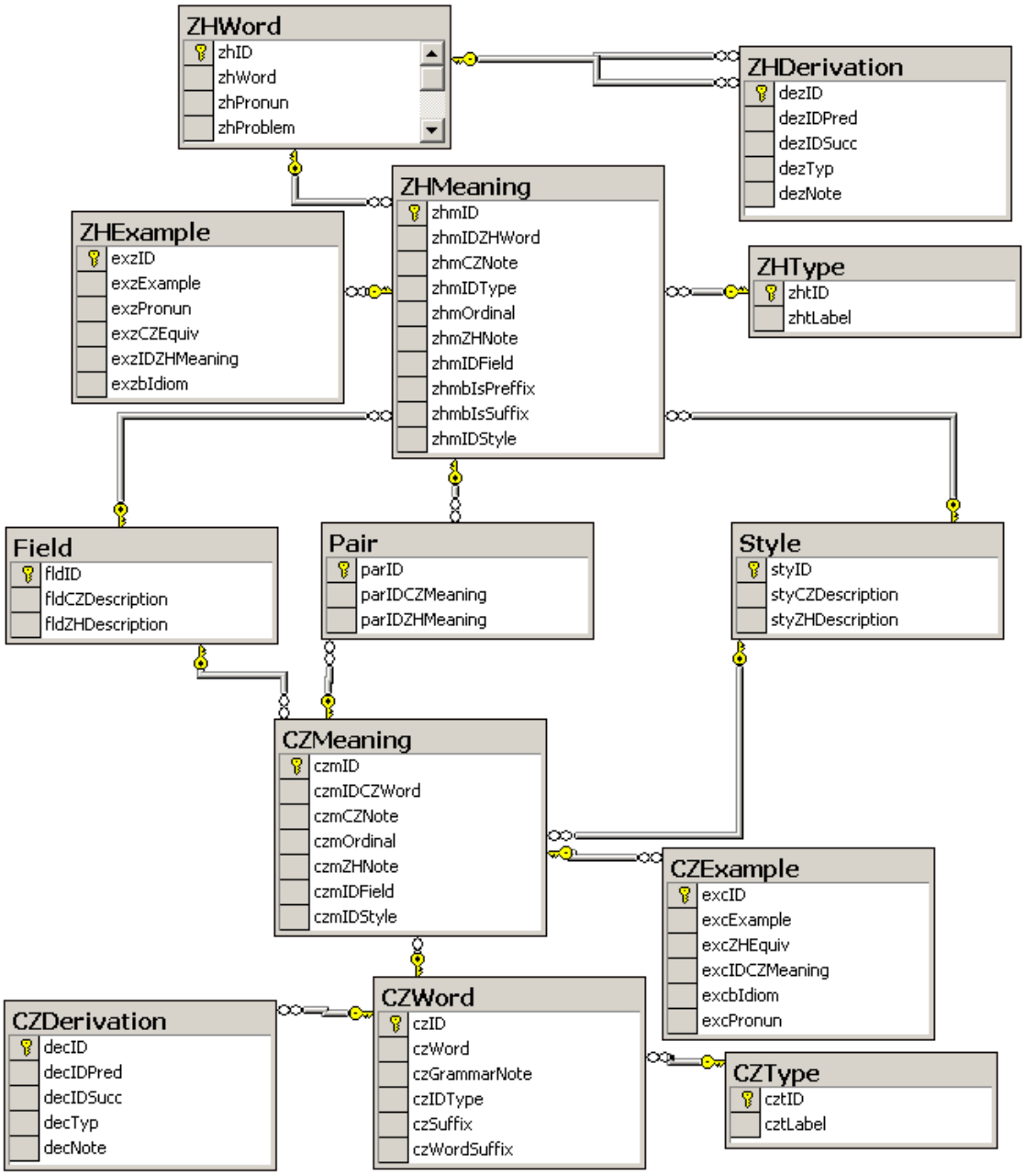
Toto schéma vzniklo postupným přidáváním atributů. Během konzultací se sinology bylo uznáno jako použitelné a bylo zahájeno zkoumání technických možností jeho implementace nad relačním databázovým strojem. K funkčním požadavkům na databázi se přidaly požadavky a specifikace technického rázu:

- jednotlivé entity budou reprezentovány jako tabulky a jednotlivé atributy a vazby jako sloupce
- do každé tabulky se přidá sloupec s automatickým číslem – identity
- všem sloupcům v tabulce bude přidán dvou- až tří- písmenný prefix jednoznačně určený tabulkou
- vazbu CZMeaning – ZHMeaning bude třeba realizovat vazebnou tabulkou Pair (slovníkový pár)

- vazbu odkazu mezi slovy bude třeba realizovat vazebnou tabulkou CZDerivation resp. ZHDerivation. Tabulka bude obsahovat dva sloupce odkazující na slovo. Z hlediska vazby budou oba sloupce rovnocenné
- do tabulky CZWord se přidá vypočítaný sloupec czWordSuffix – zřetězení czWord a czSufix pro snazší vyhledávání

Po zvážení všech požadavků bylo navržena DB se schématem viz obr.2.

Obrázek 2 Schéma databáze



Bylo rozhodnuto nepřidávat v této fázi žádné pomocné tabulky pro fulltextové vyhledávání, rozhodnutí o jejich případné přidání bylo odsunuto do testovací fáze. Během testovací fáze pak bylo zjištěno, že výkon fulltextového vyhledávání je

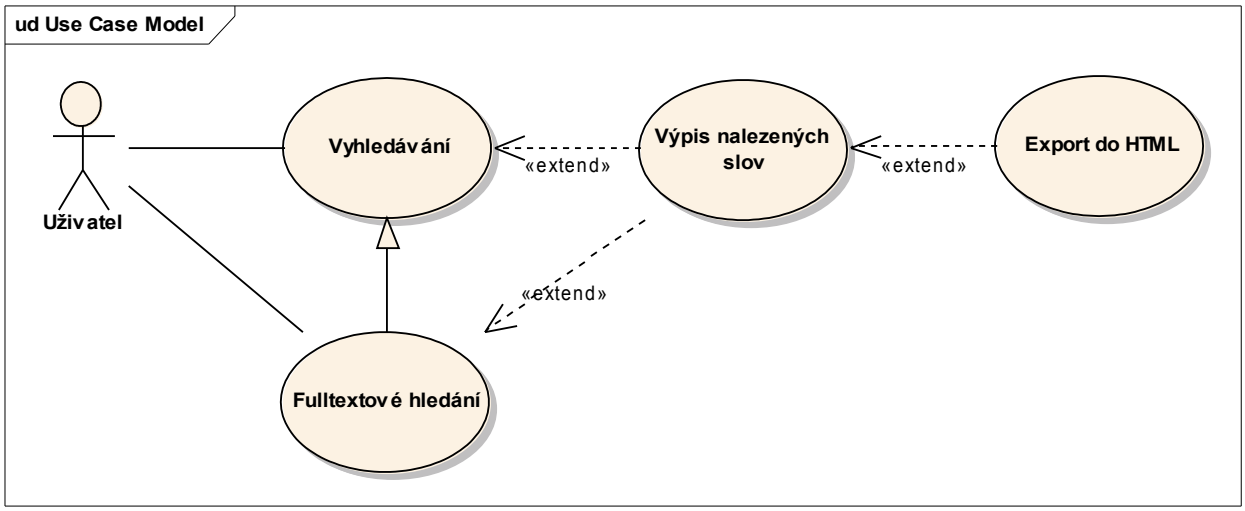
dostatečný i bez pomocných tabulek, navíc aplikace fulltextového vyhledávání na čínský text byla problematická. Byly zvažovány i další varianty datového schématu. Jednou z nich byla změna významu entity ZHMeaning resp. CZMeaning na jakýsi univerzální význam, který by měl se slovy vazbu n:m. Toto však bylo zavrženo z důvodu obtíží pro potenciální zadavatele – jednotlivý lidé by se nemuseli vždy shodnout na ekvivalenci jednotlivých významů. Zůstalo tedy u této varianty se silnější vazbou mezi slovem a jeho významem. Další zvažovanou variantou bylo navázat tabulky příkladů na tabulky významů vazbou n:m. To by umožňovalo sdílení příkladů mezí různými významy/slovy. Při zkušební implementaci však bylo zjištěno, že takové sdílení je využívané poměrně zřídka. Hlavní nevýhodou tohoto řešení pak bylo prodloužení dotazu na příklad, které bylo způsobeno přidanou vazebnou tabulkou.

Schéma datového souboru je stejné jako schéma databáze. Nejprve byla zvažována varianta, že by datový soubor obsahoval pouze výpis hesel pro všechna slova ve slovníku. Toto řešení by však způsobovalo značné komplikace při jeho aktualizaci. Vytvoření takového datového souboru by mohlo trvat i několik hodin, protože databázový server by musel pro každé slovo v obou částech provést několik komplikovaných dotazů. Pokud však bude mít datový soubor stejné schéma jako databáze, je jeho vytvoření otázkou maximálně několika desítek sekund. Přináší to s sebou sice komplikace v jeho obsluze, je to však přijatelná cena za zrychlení. Navíc takové řešení umožňuje uvažovaná rozšíření – přidání editace hesel. Je rovněž důležité i pro nasazení v uvažovaném systému: zadávání slovníku ve webové aplikaci - export datového souboru – prohlížení slovníku.

2.3 Způsob použití

Česko-čínský a čínsko český slovník se bude používat jako standardní elektronický slovník podobný příkladům uvedeným v odstavci o stávajících slovnících. Uživatel zadá slovo nebo výraz (bude povoleno použití ? jako zástupného znaku za jakýkoli znak a * jako zástupného znaku za jakýkoli počet zástupných znaků) a zvolí zda se má hledat fulltextově. Systém na to prohledá datové struktury a zobrazí nalezené výsledky. Uživatel bude moci stránku s výsledky exportovat do HMTL souboru.

Obrázek 3 Use case model

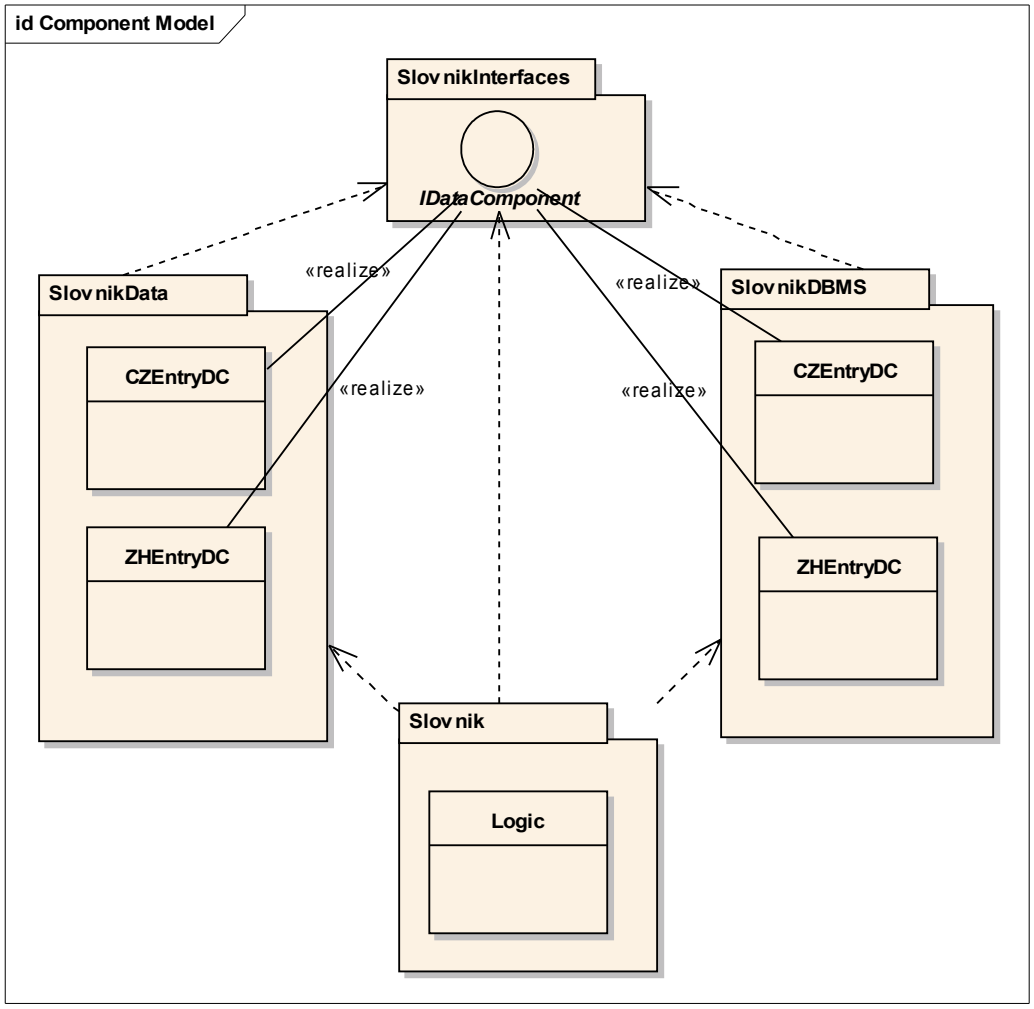


2.4 Celková architektura

2.4.1 Moduly

Vzhledem tomu, že mezi cíly byla možnost dvojího přístupu k datům, bylo poměrně záhy během vývoje rozhodnuto, že se projekt bude skládat ze samostatného modulu uživatelského rozhraní a základní logiky **Slovnik**, ze dvou modulů přístupu k datům **SlovnikData** pro přístup k databázi a **SlovnikDBMS** pro práci nad datovým souborem. Dalším modulem bude **SlovnikInterfaces**, který bude obsahovat společná rozhraní využívaná ve všech modulech.

2.4.2 Základní objekty



Obrázek 4 Diagram modulů

Rozhraní datového zdroje bude definovat IDataComponent, které bude umístěno v **SlovnikInterfaces.** Toto rozhraní budou implementovat dvě třídy z obou modulů pro přístup k datům – CZEntry a ZHEntry pro česko-čínský resp. čínsko-český směr používání slovníku. V modulu **Slovnik** bude třída Logic obsluhovat základní logiku dat.

2.4.3 Organizace datového souboru

Data slovníku budou uchovávána v lineárním datovém souboru. Původně bylo zamýšleno, aby vedle dat byly uloženy i indexy. Testovací provoz však ukázal, že na platformě .NET je deserializace (tj. načtení kompletního grafu objektů ze streamu dat) příliš pomalá a paradoxně se vyplatí si indexy budovat vždy znovu. Detaily viz kapitola 3.5. Z datového souboru se deserializací sestaví DataSet, který bude obsahovat objekty třídy odvozené od DataTable se seznamem indexů pro sloupce z tabulky. Indexy budou tří druhů – index na primárním klíči bude realizován logikou DataTable. Další indexy

budou vytvářené programem – pro sloupce typu řetězec binární strom a pro sloupce typu číslo hashovací tabulka.

2.4.4 Přístup k datům na SQL Serveru

Přístup k datům bude probíhat výhradně přes uložené procedury. I když by to u takto jednoduché (z hlediska práce s daty) aplikace nebylo nutné, používání uložených procedur nemá žádné závažné nevýhody, a proto bylo vybráno. Připojení na SQL Server bude autentifikováno pomocí NTLM Trusted Authentication

2.4.5 Uživatelské rozhraní

Uživatelské rozhraní by mělo být rychlé, snadno použitelné, ale zároveň pokrývat všechny funkce. Hlavní obrazovka slovníku se bude dělit na panel tlačítek, zadávací řádek a zobrazovací oblast. Pro implementaci zobrazovací oblasti byly zvažovány tyto možnosti:

- komponenta WebBrowser
- komponenta RichTextBox
- generování komponent Label na plochu formuláře

Žádné řešení nebylo bez chyby – s WebBrowserem se obtížně komunikovalo a nebylo možné zachycovat některé události, například klepnutí myší. RichTextBox používá pro zobrazení formátovaného textu RTF. Tento formát, jak se ukázalo během pokusné implementace, má zásadní potíže se zobrazováním českých písmen, čínských znaků a značek pro zápis s transliterací pin yin v jednom dokumentu. Vkládání obrázků do dokumentu je sice možné, nebylo však možné uzamčení jejich velikosti – uživatel by ji tedy mohl náhodou změnit a narušit tak členění stránky. Generování komponent Label se ukázalo jako velmi pomalé. Kód, který komponenty vytvářel a umísťoval byl navíc neúměrně složitý a náchylný k chybám. Nakonec bylo vybráno řešení založené na komponentě WebBrowser z těchto důvodů: velice snadno se do ní vkládá obsah – ve formátu HTML, má dobrou funkčnost – umožňuje používat některé funkce Internet Exploreru (například vyhledávání na stránce), ale i vypnutí jiných, například funkčních kláves nebo kontextového menu, je relativně rychlý (pro rozumně malé dokumenty) a COMová komponenta, kterou WebBrowser zaobaluje, umožňuje vše co je nutné pro příjemnou práci se slovníkem.

Pro větší přehlednost jak HTML kódu, tak i kódu, který ho generoval, bylo rozhodnuto použít HTML (bez snahy o jeho validnost – zobrazování bude probíhat

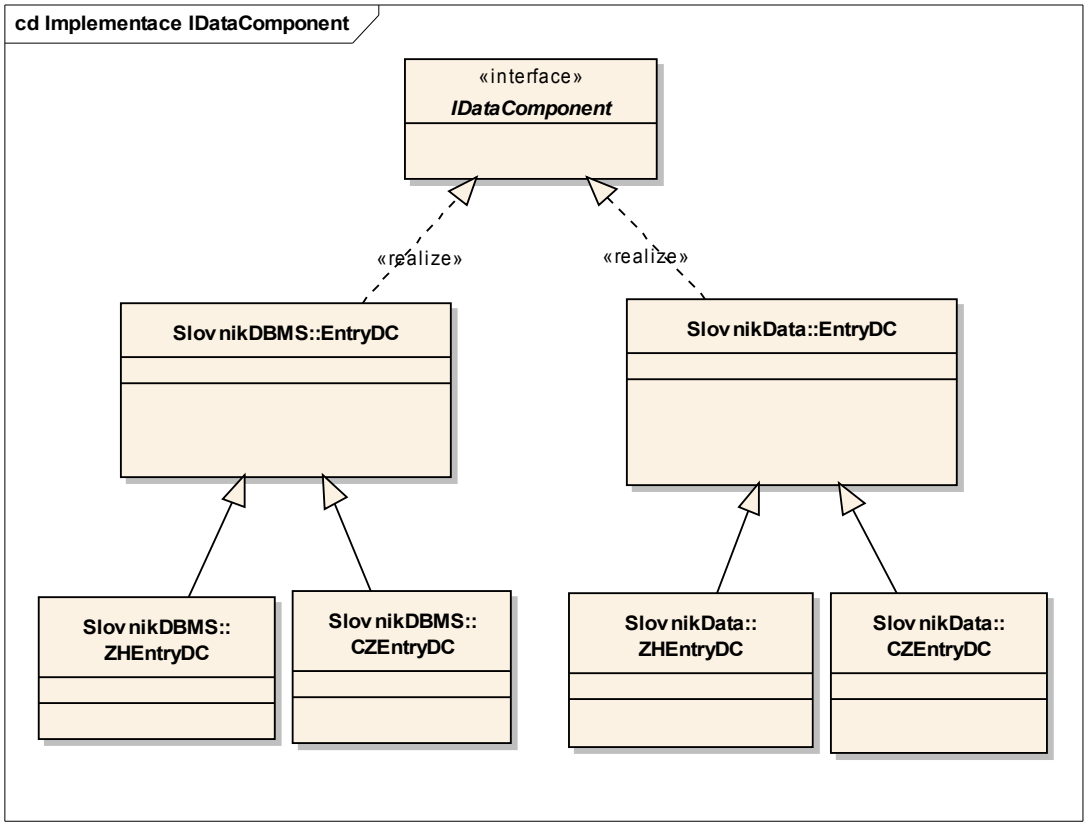
exkluzivně na IE) s CSS. Text hesla byl rozdělen do tagů `<span>` s třídami, jejichž názvy odpovídaly názvům sloupců, které obsahovaly.

Kapitola 3 – Design

3.1 SlovnikInterfaces – sdílené definice

Modul SlovnikInterfaces obsahuje rozhraní IDataComponent, které určuje jaké metody musí implementovat třída datového zdroje pro jeden směr překladu. Určuje tak i způsob přístupu k datům. Před použitím datového zdroje bude nutné zavolat metodu Init(), která připraví data. Hledání samotné bude probíhat pomocí metody Search(), která vyhledá daný výraz s přihlédnutím k parametru fulltext v datech slovníku a vrátí kolekci řádků obsahujících primární klíče nalezených slov. Následně se bude pro každý tento řádek volat metoda GetEntry(), která bude vracet typový dataset obsahující dané heslo. V hlavičce metody je uveden netypový dataset, aby mohla být použita stejná definice pro oba směry překladu. Dále rozhraní obsahuje metody pro zjišťování popisů slovních druhů, oborů a stylů. Třídy datového zdroje toto rozhraní implementují viz obr.5.

Obrázek 5 Implementace IDataComponent



Ke třídám datového zdroje je ještě třeba uvést, že jsou implementovány jako singletony-jedináčci. Třídy obsahují statickou položku m_instance, která ukazuje na

jejich jedinou instanci a statickou metodu GetInstance(), která inicilializuje m_instance, pokud to je potřeba, a vrátí ji. Důvody jsou zřejmé – ve slovníku nebude třeba přistupovat k datovému zdroji přes různé objekty.

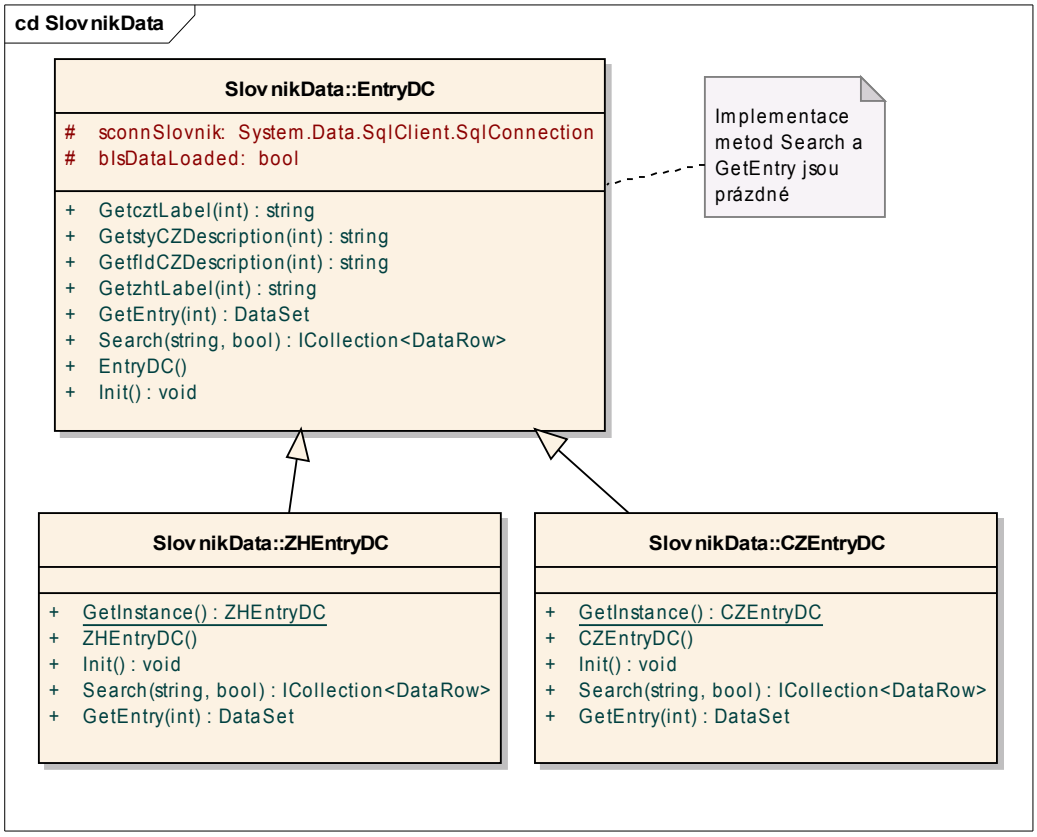
Dále SlovnikInterfaces obsahuje definice typových datasetů ZHEntry a CZEntry. Datasety obsahují všechna data potřebná k zobrazení odpovídajícího hesla – ZHEntry čínského hesla a CZEntry českého hesla. Tyto definice byly vygenerovány pomocí Visual Studia 2003 protože nová verze má potíže s generováním typových datasetů pro dotazy, které vracejí více result setů – v terminologii SQL serveru více tabulek.

3.2 SlovnikData – datový přístup SQL

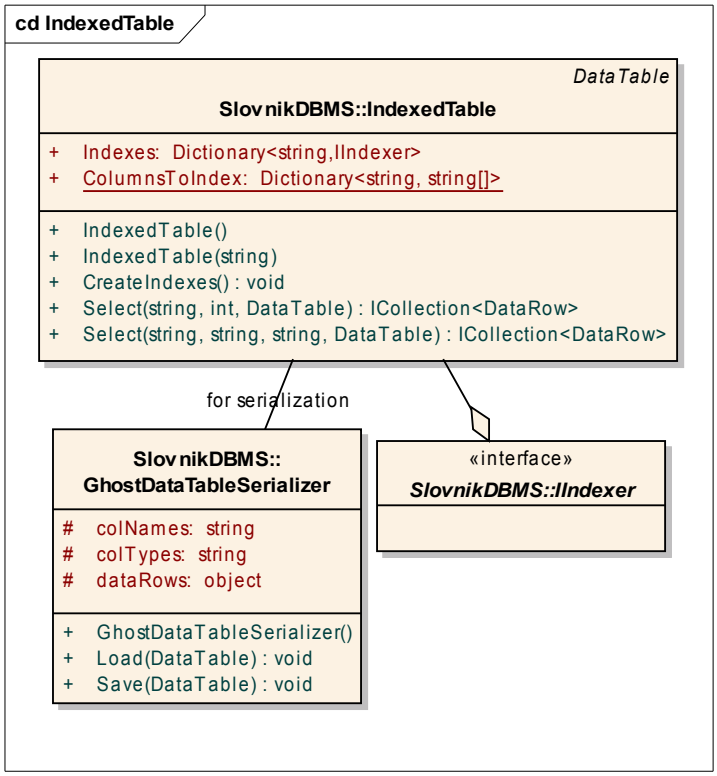
Modul SlovnikData obsahuje třídy poskytující přístup k datům uloženým na SQL Serveru. Rozhraní IDataComponent implementuje třída EntryDC, která je zodpovědná za implementaci společných metod IDataComponent - metod pro zjišťování popisů slovní druhů, oborů a stylů. Třída rovněž inicializuje připojení k databázi. To probíhá následovně: pokusí se připojit k databázi s velmi krátkým časovým limitem pro spojení (Connection Timeout). To se buď vydaří a pak připojení proběhne normálně, nebo dojde k chybě. Tato chyba se pak zachytí v modulu Slovník a dojde k rozhodnutí použít namísto databáze datový soubor. Použití kratšího limitu výrazně zkrátí startovací dobu. Také při výpadku spojení se serverem není třeba čekat výchozích 30 sekund než program začne používat datový soubor. Je pravda, že někdy může být 1 sekunda nedostatečná pro připojení k vzdálenému serveru, ale celá podpora pro přístup k SQL serveru je stavěna pro použití na lokální síti (například NTLM autorizace) a tam je 1 sekunda bohatě dostačující.

Od třídy EntryDC dědí česká resp.čínská datová komponenta CZEntryDC resp. ZHEntryDC. Tyto komponenty potlačují implementace metod Search() a GetEntry(), které byly ve třídě EntryDC ponechány prázdné. Tyto metody jsou v podstatě přímočarým voláním uložených procedur na SQL Serveru, které tak řeší logiku vyhledávání.

Obrázek 6. model tříd SlovnikData



3.3 SlovnikDBMS – obsluha datového souboru



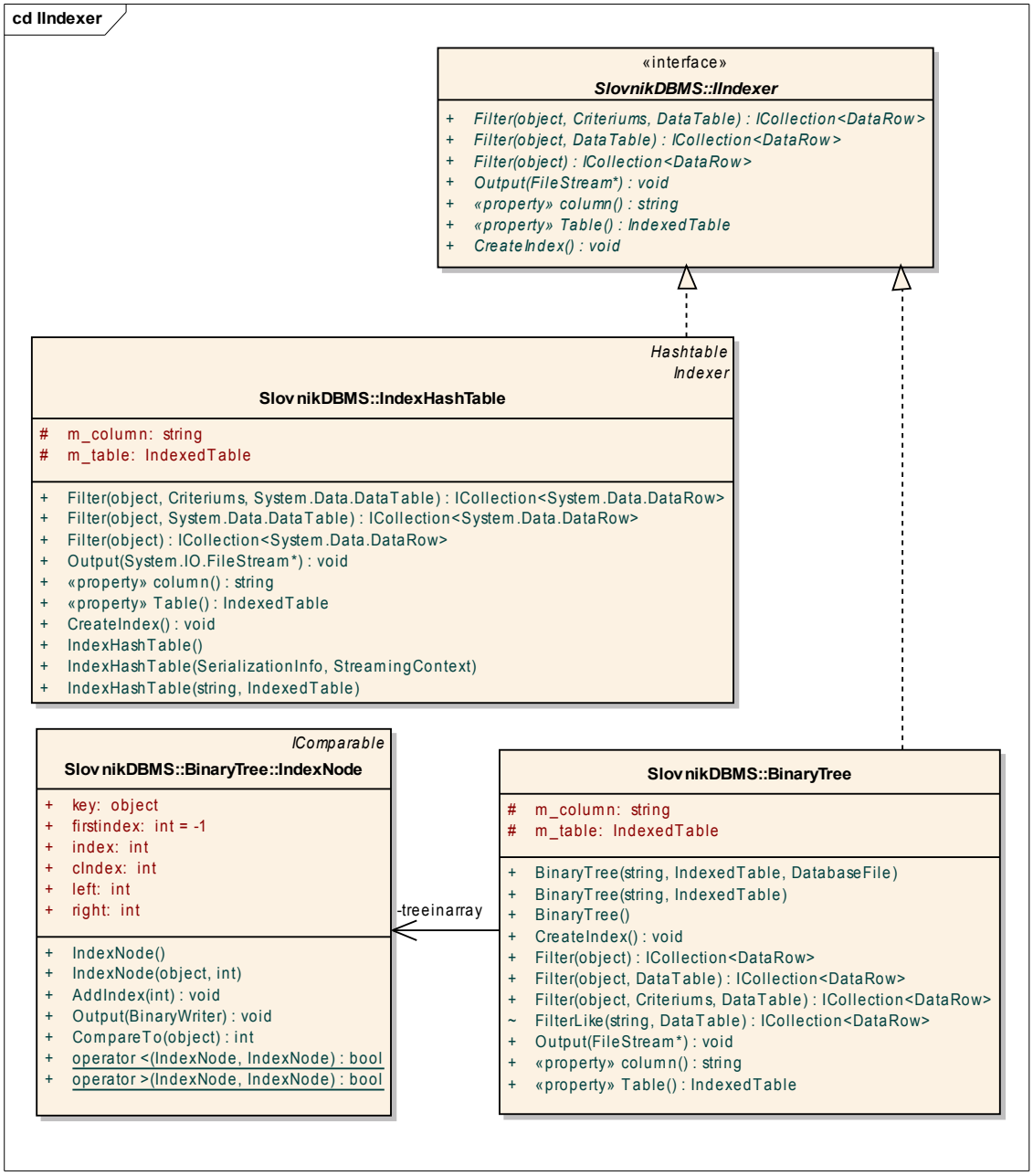
Obrázek 7 třída IndexedTable

Modul SlovnikDBMS obsahuje vedle tříd datového zdroje, jejichž veřejné rozhraní je obdobné tomu v modulu SlovnikData, třídy pro serializaci/deserializaci a pro datové operace nad tabulkami.

3.3.1 Serializace/deserializace

Datový soubor obsahuje binární serializace (pomocí BinaryFormatter) pomocné datové třídy GhostDataTableSerializer. V této třídě jsou pouze pole polí objektů – jedná se o řádky pomyslné tabulky. Jde tedy o odlehčenou verzi DataTable, jejíž hlavní výhodou oproti plné DataTable je výrazně rychlejší serializace/deserializace. Tato metoda uchovávání dat byla popsána v [2] a autorem upravena pro ještě lepší výkon – viz kapitola 3.5.

3.3.2 Indexy



Obrázek 8 IIndexer

Data ze souboru se načtou do objektů třídy IndexedTable, která je odvozena od DataTable, obsahuje však navíc logiku pro práci s vlastními indexy. Indexy implementují rozhraní IIndexer. Bylo rozhodnuto použít dva typy indexů – binární stromy (třída BinaryTree) pro indexování textových sloupců a třídu IndexHashTable (poděděnou od Hashtable) pro indexování čísel. Do indexu se zapíše klíč a číslo řádku, kde se klíč nachází v dané tabulce. Nepoužívají se přímo kolekce z .NET frameworku, protože ty si vynucují unikátnost klíčů. Větší flexibilitu nabízejí vlastní poděděné třídy.

Indexy se nepoužívají pro dotazy na podobnost slov (SQL predikát LIKE vyjma hledání testu začínajícího na daný výraz). Místo nich se data vyhledají výchozí metodou třídy DataTable Select(), která má překvapivě dobrý výkon.

Binární strom

Binární stromy byly zvoleny pro jednoduchost implementace i pro dobré praktické zkušenosti. Aby se zlepšila jejich výkonnost v nejhorších podmínkách byl zvolen zvláštní postup jejich konstrukce. Aby se udržela dobrá lokalita referencí, všechny uzly stromu (reprezentovány třídou IndexNode) jsou uloženy v poli, které je alokováno na maximální velikost počtu všech prvků – počet řádků v tabulce. Uzly IndexNode uchovávají pozice prvků majících daný klíč dvěma způsoby. První takový klíč je v položce firstindex, další se pak přidávají do pole indexů. Toto pole se podle potřeby zvětšuje, vždy na 2x větší velikost. Počet obsazených prvků tohoto pole je uložen v položce cIndex.

Vytvoření indexu probíhá následovně. Nejprve se vytvoří instance generické třídy List parametrizovaná třídou IndexNode. Tyto uzly se inicializují daty z tabulky, pak se setřídí podle klíče. Následně se znovu projde celý seznam a uzly, které mají stejnou hodnotu klíče (po setřídění musí ležet takové uzly v seznamu vedle sebe) se spojí. Setříděním se také docílí toho, že výsledný strom bude mít logaritmickou maximální hloubku. Následně se seznamu zkopíruje do výsledného pole. Nad polem se spustí rekurzivní metoda CreateSubtree(), která teprve vytvoří strom – spojí jednotlivé uzly. Metoda pracuje od středu a principem rozděl a panuj sestrojí celý strom. Pro další zvažované varianty a jejich výsledky viz kapitola 3.5.

Vyhledávání je v binárním stromu prostým procházením stromové struktury. Binární strom umožňuje efektivní vyhledávání nejen výrazů odpovídajících přesně, ale i slov začínajících na daný výraz. To je implementováno metodou FilterLike(), která projde strom a pošle metodě Pass() ukazatel (index do pole uzlů) na vrchol podstromu, jehož klíč začíná na hledané slovo a obsahuje tedy všechny klíče začínající na daný výraz.

Hashovací Tabulka

Hashovací tabulka byla zvolena pro výborný výkon i při velkém počtu prvků. Indexy tohoto typu se používají na sloupce čísel. Aby bylo snazší budovat indexy i nad opakujícími se prvky a aby bylo možné držet indexy v typové kolekci IIndexerů, bylo

přistoupeno k podědění třídy Hashtable. Třída IndexHashTable je hashovací tabulka, kde klíčem je buďto int nebo string a hodnoty implementují rozhraní IEnumerable parametrizované typem int. S tímto uspořádáním se snadno pracuje a testy ukázaly, že není až tak neefektivní, jak by se dalo čekat

3.4 Slovnik – uživatelské rozhraní

Modul Slovnik obsahuje uživatelské rozhraní, formátování výstupu a základní logiku zpracování požadavku na vyhledávání. Rozhodnutí, jaký datový zdroj se má používat, je na třídě DataComponentFactory, která je instancí návrhového vzoru Abstract Factory. Vytváří instance (pomocí GetInstance()) třídy datového zdroje podle nastavení položky useSQL.

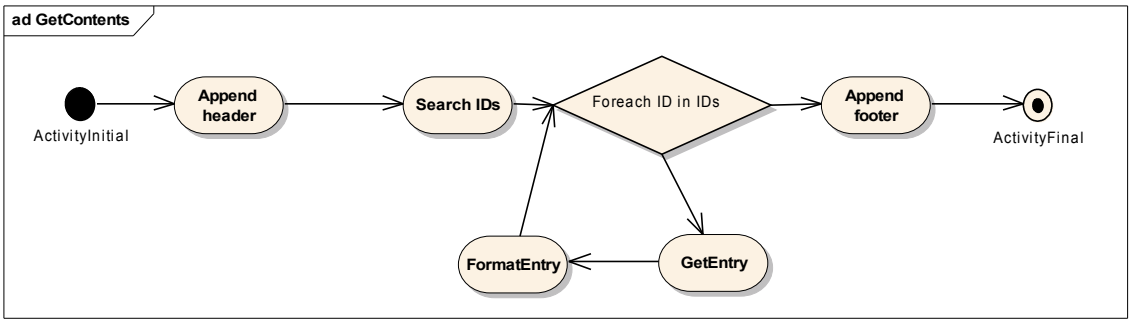
3.4.1 Uživatelské rozhraní

Uživatelské rozhraní je jednoduché a přehledné. Uživatel může přepínat mezi online a offline režimem, povolovat fulltextové vyhledávání a vyhledávání podobných slov. Volba vyhledávání podobných slov jen přidá znak * na začátek a konec hledaného výrazu. Pokud probíhá nějaká operace na popředí, zobrazí se okno s nápisem „Prosím čekejte“. Délku provádění operace může uživatel zjistit ze stavového řádku.

Inicializace datových zdrojů probíhá asynchronně na jednom z vláken ThreadPoolu. Zatímco uživatel píše hledaný výraz a nastavuje parametry hledání proběhne významná část načítání na pozadí. Synchronizaci zajišťuje objekt AsyncResult, který implementuje rozhraní IAsyncResult, a slouží především jako zaobalení třídy ManualResetEvent. Výsledky inicializace – úspěch nebo případné chyby zobrazuje opět primární vlákno formuláře/aplikace ve stavovém řádku.

3.4.2 Formátování výstupu

Za formátování výstupu je zodpovědná třída Formatting. Zveřejňuje obsah hlavičky a patičky výstupního souboru HTML (soubory header.htm a footer.htm v zdrojích aplikace) a metody pro zformátování slovníkového hesla FormatEntryCZ() a FormatEntryZH(). Postup vytváření výstupu – postupné přidávání HTML kódu do StringBuilderu – je obdobný jako při generování HTML kódu uživatelského ovládacího prvku v ASP.NET. Tento postup byl zvolen úmyslně, aby usnadnil vytváření potenciální webové aplikace Slovníku.



Obrázek 9 Postup vytváření hesla

3.5 Měření

Po implementaci základní funkčnosti začalo testování jednotlivých rozhodnutí – měření výkonu aplikace při nasazení různých alternativ.

3.5.1 Metodika testů

Testy byly prováděny v konkrétní fázi vývoje – nemohou tedy sloužit k absolutnímu porovnání komponent za všech podmínek, pouze k relativnímu porovnání alternativ. Testy byly prováděny vždy 5x – jako výsledek se bral aritmetický průměr. Měření bylo prováděno komponentou QueryPerformanceCounter. Jako testovací data byla zvolena původní skupina dat namnožená tak, aby slovník obsahoval cca 70 000 slovníkových párů a 35 000 slov na každé straně.

3.5.2 Měření indexů

Jelikož už při prvním spuštění aplikace se ukázalo, že hlavní její nedostatek z hlediska výkonu je inicializační čas, bylo zahájeno zkoumání kde se ztrácí nejvíc času. Načítání dat ze souboru bylo ihned zavrženo (načtení 19MB souboru trvalo asi 0,7s), další důvody viz [1]. Dalším kandidátem bylo vytváření indexů.

Tabulka 1 Vytváření všech indexů podle datové struktury

s	vytvoření	se zápisem na disk	**načtení z disku**
BinaryTree	4,61	5,29	**6,9s**
HashTable	5,23	5,98	**7,58**
SortedList	5,24	5,99	**7,60**
SortedDictionary	5,33	6,60	**7,92**

Výsledky byly velice překvapivé u všech uvažovaných datových struktur bylo rychlejší je znovu vytvořit než je načíst – deserializovat z binárního souboru! Důvod lze najít např. v [2]. Autory navrhované řešení: implementovat rozhraní ISerializable a řídit serializaci/deserializaci vlastním kódem se ukázalo jako nevyhovující – zlepšení nebylo

měřitelné. Pomalé načítání znamenalo mimo jiné i to, že inicializaci programu nebude možné nějak výrazně urychlit – vždy se budou muset vytvořit všechny indexy.

Přistoupilo se tedy k optimalizacím vytváření indexů. Vytváření indexu ze třídy IndexHashTable nebylo možné urychlit, důležitý kód je skryt v jádře .NET frameworku. Pozornost byla tedy zaměřena na vlastní třídu BinaryTree. Aby se více projevil výsledek optimalizací bylo použito výhradně indexů BinaryTree pro celou strukturu (tedy i pro sloupce typu číslo) a byla měřena doba jejich vytvoření. První měření proběhlo na struktuře stejné jako bylo popsáno výše až na položku uzlu firstindex, jeho výsledkem bylo 5,63s. Prvním pokusem jak zkrátit tuto dobu bylo implementovat uzly stromu (IndexNode) nikoli jako třídy, ale jako struktury. Tím by se teoreticky měla odstranit jedna dereference při přístupu k položkám uzlu a zlepšit lokalita referencí v poli uzlů. Výsledky však tuto domněnku vyvrátily, načítání se prodloužilo na 7,98s. Změny byly vráceny zpět a přidala se položka firstindex sloužící pro uložení prvního indexu – ušetřil se tím tedy přístup do pole a testování mezí. Doba se zkrátila na 5,02s.

Další zlepšení výkonu mělo nastat úpravou vytváření indexu. Ani změna algoritmu z inicializace seznamu uzlu, setřídění na inicializace setříděného seznamu (SortedList), ani použití parametrizované šablony setříděného seznamu, však nepomohlo, naopak 2x prodloužily načítání. Nejpravděpodobnějším důvodem bylo, že implementace setříděného seznamu v .NET frameworku není dostatečně efektivní.

3.5.3 Načítání tabulek

Pozornost byla tedy obrácena k načítání datových tabulek. Data se načítala pomocí původní GhostDataTable jak je k nalezení u daného článku, po dobu 13,6s. První myšlenkou na zrychlení bylo uzavření načítání dat mezi volání metod BeginLoadData() a EndLoadData() na objektech DataTable. To zkrátilo dobu načítání na 10,8s. Dalším krokem bylo nezapisovat sloupce, ale pouze datové řádky tabulky. To však zrychlilo načítání v průměru o 0,1s a bylo to daleko citlivější na změny struktury dat. Naopak změna položky třídy, které obsahovaly řádky z ArrayList na object[,] zrychlila načítání téměř dvojnásobně na 5,8s. Důvodem bylo pravděpodobně odstranění typových konverzí a pomalá serializace/deserializace samotné třídy ArrayList.

Otázka optimalizace načítání tabulek se řešila podrobněji. Proběhlo měření načítání samotné tabulky CZWord s 35 000 záznamy. V tabulce 2 je vylepšenou verzí rozuměna GhostDataTable po výše zmíněné úpravě.

Tabulka 2. Načítání tabulky CZWord

	Původní verze	Vylepšená verze
Načtení tabulky	1,74s	0,73s
Z toho		
Deserializace	1,40s	0,35s
Načtení	0,37s	0,37s

Z detailního měření vyplývá, že

a) deserializace samotná byla zrychlena 3x(!)

b) deserializace GhostDataTable i načítání dat do IndexedTable trvá stejně dlouho

Z bodu b) bylo usouzeno, že již byla dosažena hranice, na kterou se lze dostat optimalizacemi v .NET frameworku a od dalších optimalizací v této oblasti se upustilo.

3.5.4 Dotazy

Dále bylo zkoumáno, není-li zapotřebí optimalizovat i dotazy na slovník. Dotaz na vyhledání slova však trval méně než 0,1s a dotaz na fulltextové hledání mezi 0,8s a 2s, což je i pro uživatele poměrně přijatelná doba. Značnou dobu program strávil také u generování HTML kódu, zvláště pro více složitých hesel. Proti tomu však nebylo nalezeno žádné účinné opatření.

Kapitola 4 – Závěr

4.1 Splnění cíle

Česko-čínský a Čínsko-český slovník byl vytvořen a jeho použití je skutečně jednoduché. Slovník funguje i při větším množství dat než jaké je v současné době k dispozici, s rostoucím objemem těchto dat však roste doba potřebná pro jejich načtení.

Architektura systému by měla být dostatečně pružná pro plánovaná rozšíření. Implementace webové aplikace by měla být obzvláště snadná – uložené procedury a datová část může být sdílená, formát výstupu se dá použít obdobný.

Nepovinný cíl editace slovníku sice nebyl splněn, ale jeho pozdější implementace by také neměla být komplikovaná – tabulky jsou v datovém souboru uspořádány stejně jako v databázi, jsou plně pod kontrolou programu, proces použitý při deserializaci lze i obrátit a data zapsat zpět. Vytváření vlastních rozšíření slovníku by mohlo být realizováno obdobně.

4.2 Zkušenosti získané při vývoji

Nejdůležitější zkušenosti byly získány při analýze programu. Vytvoření analýzy a detailní propracování návrhu ušetřilo mnoho času, který by byl stráven opravou chyb v kooperaci tříd nebo logickými chybami návrhu. Na dobrou analýzu je však třeba detailně znát použité technologie, jejich možnosti a omezení.

Autor se musel detailně seznámit s mnoha jemu do té doby neznámými technologiemi – např. RTF nebo fulltextové vyhledávání a indexování na SQL Serveru 2000, které se do konečné podoby díla ani nedostaly.

Zajímavá byla práce na zrychlování a ladění výkonu aplikace. Při této měřící fázi byla odhalena nejen slabá místa v programu, ale i drobné chyby a zkrátila se tak doba potřebná pro detailní testování.

Literatura

[1] Kukol, P.,Gray,J.: Sequential File Programming Patterns and Performance with .NET, Microsoft Research, *Technical Report MSR-TR-2004-136,* Redmond, 2004, ftp://ftp.research.microsoft.com/pub/tr/tr-2004-136.pdf

[2] Richter, J.:Runtime Serialization, Part 2, *MSDN Magazine 2002/7,* Redmond, 2002*,* http://msdn.microsoft.com/msdnmag/issues/02/07/net/

[3] Čermák, F., Blatná, R.: Manuál lexikografie. *H&H*. Praha, 1995

[4] Cormen, T.,Leiserson, C., Rivest, R.,Stein, C.: Introduction to Algorithms. *The MIT Press*. Cambridge, 2001

[5] Microsoft Corp.: The SQL Server Books Online. *World Wide Web*, http://www.microsoft.com/sql/techinfo/productdoc/2000/books.asp

[6] DevX.com. *World Wide Web*, http://www.devx.com/