VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

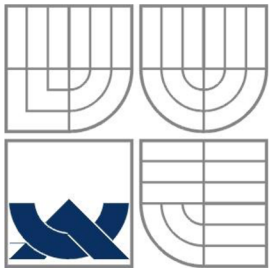
NÁSTROJ PRO URYCHLENÍ VÝVOJE APLIKACÍ V MS VISUAL STUDIU 2010

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

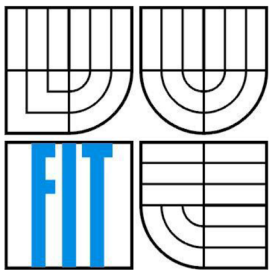
AUTOR PRÁCE
AUTHOR

JAN MACHALA

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

Nástroj pro urychlení vývoje aplikací v MS Visual Studiu 2010

PLUGIN FOR SIMPLIFACATION OF PROGRAMMING USING VISUAL STUDIO 2010

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN MACHALA
AUTHOR

VEDOUCÍ PRÁCE — Peter Jurnečka, Ing.
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá nástrojem na urychlení vývoje v novém programovacím jazyku pro nezkušené programátory. Aplikace napovídá ukázky zdrojového kódu v prostředí Visual Studia 2010 dle aktuálně používaného programovacího jazyku. Aplikace je typu klient-server, server zajišťuje vyhledávání ukázek zdrojových kódů.

Abstract

This thesis is focused on tool for simplification development in new programing language for non-experienced programmers. Application suggests sample source codes for current programming language in Visual Studio 2010. The application is a client-server, the server implements searching of the source code examples.

Klíčová slova

Visual Studio 2010 addin, WPF, WCF, C#, Webservices

Keywords

Visual Studio 2010 addin, WPF, WCF, C#,Webservices

Citace

Machala Jan: Nástroj pro urychlení vývoje aplikací v MS Visual Studiu 2010, bakalářská práce, Brno, FIT VUT v Brně, 2012

Nástroj pro urychlení vývoje aplikací v MS Visual Studiu 2010

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Peter Jurnečka
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Machala
15. 5. 2012

Poděkování

Děkuji svému vedoucímu Ing. Peteru Jurnečkovi za odbornou pomoc, kterou mi poskytl během celé práce na této bakalářské práci.

Obsah

1 Úvod

Vývoj aplikací se s každým novým programovacím jazykem zrychluje. To co programátorům stačilo znát před deseti lety už v současnosti nestačí. Podíváme-li se trochu do historie, tak v roce 1954 vznikl fortran, o čtyři roky později lisp, Smaltalk, C, C++, Perl, Java, PHP a tento rapidní vývoj jazyků nekončí. Od strojového kódu jsme se dostali ke komplexním jazykům, které se starají sami o alokaci paměti, načítání knihoven, kompilaci, linkování, typování, ukládání, načítání dat aj. Toto postupné zjednodušování, dle mého názoru, stále nekončí a bude pokračovat ještě spousty let. K rychlému vývoji nám také napomáhají stále vylepšované IDE[1]. IDE nám automaticky napovídají názvy metod, parametry metod, přidávají potřebné reference na knihovny, generují základy tříd se základními voláními, generují metody pro práci s privátními proměnnými z venku (tzv. gettery a settery), implementují rozhraní apod. Některé IDE nabízejí i vkládání chytrých bloků kódu, kde po zadání názvu objektu a vlastností vám automaticky vloží celý grafický prvek, například textové pole s vygenerovanými patřičnými metodami. Dále v refaktorování se mnohonásobně vylepšily a jednoduchá unixová kombinace sed, grep, vim, bash, už není nejlepší způsobem na refaktorování.

Práce se zabývá vývojem nástroje k urychlení učící křivky při učení nového programovacího jazyku. Z vlastní zkušenosti vím, že naučit se nový programovací jazyk je celkem jednoduché [9], naučíte se novou syntaxi jazyka, jaké používá konstrukce, programové vybavení, s čím můžeme a nemůžeme pracovat. Toto je bohužel jednoduchá část, další část je naučení se obrovského množství knihoven nutných k samotnému programování . Knihovny bývají většinou jiné a v každém programovacím jazyku se používají jinak (mnohojazyčných knihoven je málo). Můj nástroj urychluje především druhou fázi učení, kdy nenutí programátora k pročítání diskuzních fór, mnohaset stránkových knih a tutoriálů. Díky mému nástroji obdrží programátor relevantní ukázkový programový kód přímo do prostředí IDE.

[1] Integrated Development Environment – Integrované vývojové prostředí

2 Vývojové nástroje současnosti

Tato kapitola se zabývá současnými nástroji na urychlení vývoje. První podkapitola se zabývá základní navigací po souborech a v projektu. Další podkapitola popisuje současné způsoby automatického formátování zdrojového kódu. Podkapitola o zkratkách a šablonách nabízí pohled na časté psaní opakovaných částí kódu. Další kapitola probíhá velmi používanou autokompletaci názvů metod, tříd aj. Následující podkapitola se zabývá současným trendem v nápovědách ve zdrojových kódech. Refaktorování jako další podkapitola je nedílnou součástí programátora a způsobu urychlení vývoje. V poslední kapitole je představen inteligentní plugin do Visual Studia – ReSharper.

2.1 Navigace

Navigace je nedílnou součástí práce se zdrojovým kódem. Pro pohyb po zdrojovém kódu můžeme využít šipek na klávesnici a myš. Můžeme také využít vyhledávání ve zdrojovém kódu. To nakolik jsme schopni se pohybovat ve zdrojovém kódu, záleží na zvoleném editoru. Moderní IDE nám umožnují pohyb k definici metody, typu, symbolu, třídy, rozhraní aj. K navigaci můžeme také používat externí software, který nám rychle vyhledá daný textový řetěz či regulární výraz ve složce s projektem. Takovým zástupcem je například Agent Ransack [7]. Mnohé IDE umí také vyhledávání konkrétního textu ve vybrané metodě.

Problém je však navigace mezi soubory resp. navigace mezi třídami ve více souborech. Pokud chceme najít konkrétní třídu v neznámém souboru, máme možnost využít opět textové vyhledávání ve více souborech. To je při větších projektech značně zdlouhavé. Do Visual Studia 2010 existuje plugin, který umí vyhledávat v třídách pouhým zadáním pár znaků. Tento nástroj na zvýšení produktivity popíšu v sekci ReSharper.

Navigace ale nemusí být pouze na základě vyhledávaného textu. Je možné navigovat mezi předky či potomku třídy, hledat implementace dané třídy, nebo implementace vybraného rozhraní či abstraktní třídy. Tato navigace však dosud není nativní součástí Visual Studia 2010.

2.2 Formátování kódu

Kolem roku 2004 se na internetu často řešil spor mezi zastánci odsazení a pevných mezer v kódu. Jelikož už v té době se běžně pracovalo s verzovacími systémy typu CVS, bylo potřeba si sjednotit styl. V programovacích prostředích tedy vznikly nastavení a detektory aktuálního nastavení zdrojového kódu. Unixový editor *Vi* vyžadoval mít ve zdrojovém kódu speciální komentář s tímto nastavením.

Mezery a odsazení byl pouze začátek, posléze se vyvinulo spousty stylů odsazování kódu. Vzniklo okolo 10 stylů odsazování kódu [1]. Styly neřešily pouhé odsazení bloků kódu, řešily celou strukturu programu a jeho formátování. Moderní IDE dnes nabízí automatické formátování zdrojového kódu. Je možné nastavit i automatické formátování, při posílání nové verze kódu do CVS (aj.). Když porovnáváme dvě verze stejného souboru, je možné zapnout inteligentní porovnávání, které porovnává v daném jazyce – řídí se syntaxí jazyka, ne formátováním jako obyčejného textu.

Automatické formátování kódu v IDE nám nezaručí, že kód v CVS bude formátovaný stejně. Každý programátor si může toto výchozí nastavené přenastavit dle jeho preferencí. Proto se stává automatické formátování součástí procesu sestavení, resp. CI[2]. Mezi zástupce automatického formátování na této úrovni můžeme zařadit StyleCop [2] nebo FxCop [3].

2.3 Zkratky a šablony

Některé editory obsahují tzv. zkratky, šablony kódu nebo snippety. Ve své podstatě se jedná o funkčně podobné automaticky vkládané kusy kódu. Podíváme se na ně postupně.

Zkratky slouží k automatickému doplnění jednoho řádku kódu. Jako příklad bych mohl uvést zkratku *psf*, která byla po napsání automaticky nahrazena za *public static function*. Ve své době to bylo hojně využíváno.

Šablony nebo-li snippety byly následovníkem zkratek. Šablony se lišily především znalostí potřeb. Automaticky uměly přesunout kurzor na konkrétní část šablony, dále také uměly více řádků než jeden. Dokonce je možné nastavit i jednotlivé části, které je třeba doplnit. Nejlépe si to ukážeme na příkladu. Po napsání zkratky *cls* a stiknutí Tab se nám zobrazí kód na vytvoření třídy. Automaticky se nám nastaví kurzor na úpravu ClassName, po stisktnuí Tab pak na úpravu __construct a po dalším Tab se nastaví kurzor do těla konstruktoru.

```
1. class ClassName {
2.       function __construct() {
3.
4.       }
5. }
```

Kód 2.3-1 Expandovaná šablona pro zkratku *cls*

Snippety z Visual Studia 2010 fungují podobně, je možné je vyvolat pomocí vyvolat pomocí IntelliSense nebo po napsání zkratky stisknutím dvakrát klávesy Tab.

---

[2] CI – Continuos Integration česky Průběžná integrace, slouží k automatickému kompilování a testování příspěvků do verzovacího systému.

2.4 Autokompletace

Autokompletace se objevuje v IDE editorech již dlouho. Rozdíly v jednotlivých implementacích jsou značné. Některé editory, například PSPad už ze své podstaty nabízí všechny možné použitelné funkce bez ohledu na kontext. Visual Studio 2010 se svou implementaci IntelliSense nabízí kontextově orientované napovídání. V „nápovědě“ můžeme narazit na názvy lokálních či globálních proměnných, názvy typů a vlastních typů, použitelné metody a funkce jak již lokální, globální nebo i z ostatních tříd. Autokompletace nám pomáhá zvýšit produktivitu, ušetřit čas nutný k psaní a především si nemusíme pamatovat názvy jmen funkcí, tříd apod. Autokompletace se obvykle vyvolává klávesovou zkratkou Ctrl+mezerník.

Autokompletace nám je nápomocná i při psaní parametrů metody. Pokud přesně neznáme pořadí parametrů, nebo jakého datového typu jsou, či kterou přetíženou metodu vlastně chceme vybrat, autokompletace nám bude nápomocná. Obvykle jde vyvolat klávesou Ctrl + Shift + mezerník.

2.5 Nápovědy

Nápovědy, nebo-li Code inspections jsou nám při programování nápomocny. Díky nim není třeba přidávat reference na potřebnou knihovnu, nápověda nám ji nabídne a po potvrzení je reference automaticky přidána. Dalším typem nápovědy může být automatické přidání dokumentačního komentáře k metodě či třídě. Nápověda automaticky vygeneruje potřebný základ a programátor už jen doplní popis metody a parametrů.

Do nápovědy bychom mohli přidat i automatické vytváření deklarace proměnných, implementace metod z nadřazeného rozhraní, či je pouhé doporučení. Tím může být přehození pravdivostní podmínky větvení či cyklu.

Nápovědy jsou velmi dobře implementovány v nástroji do Visual Studia, o kterém budu mluvit v sekci ReSharper.

2.6 Refaktorování

Refaktorování je technika restrukturování kódu aplikace, která změní vnitřní chování aplikace, ale na venek zůstane aplikace stejná. Refaktorování je na rozdíl od restrukturalizace kódu bezpečné, neboť nemá zásadní dopad na ostatní součásti aplikace. Mezi základní techniky refaktorování bych uvedl přejmenování názvů, odstranění zbytečných podmínek typu If True, vytvoření rozhraní, přidáním nového parametru, přesunutím metody, třídy do jiného soubory, třídy apod. Martin Fowler uvádí na svých stránkách o refaktoringu okolo 80 technik refaktorování.

2.7 ReSharper

ReSharper je nástroj na zvýšení produktivity programátorů. ReSharper je naimplementovaný jako Addin do Visual Studia 2010, 2008 a 2005. Jedná se o placený addin, který však dokáže nesmírně urychlit vývoj a usnadnit práci. Akademická licence stojí 47€, osobní licence přijde na 189 €. Za tuto cenu však dostanete plnohodnotný nástroj. Kromě nativní implementace předešlých sekcí Navigace, Formátování kódu, Zkratky a šablony a Refaktorování umí také:

- Analýzu kódu – která odhalí chyby, možné problémy, doporučení, rychlé opravy aj.
- Nápovědy ke tvorbě lokalizovaných aplikací
- Generování kódu – implementace rozhraní, vytváření promenných/tříd/kontruktorů
- Uklizení kódu – odstraní nepotřebné volání, nepoužité proměnné, metody a některé techniky refaktorování
- Chytřejší šablony
- Integraci pro Unit Testing

3 Analýza

V předchozí kapitole byly představeny různé způsoby urychlení programování. Na základě denního používání zmíněných způsobů byla navrhnuta aplikace, která usnadní proces učení nového programovacího jazyka. V odborném článku od Jean C. Scholtze [9] jsou popsány obtíže při učení nového programovacího jazyka. Scholtz svou analýzu rozdělil na 5 oblastí: syntaxe, sémantika, plánování implementace, taktické plánování a strategické plánování za účelem zjištění, v které z těchto oblastí mají začínající programátoři největší problémy. Ze studie vyplývá, že programátoři mají největší problémy s plánováním implementace, když musí najít, jak se vyjádřit v novém programovacím jazyce nezávisle na algoritmu či konstrukcích. Programátoři také tíhli k implementaci v novém jazyku jako ve starém řádek po řádku. Na základě této studie byl navrhnut nástroj, který se snaží toto vyjádření v novém programovacím jazyce urychlit a usnadnit.

Aplikace na urychlení vývoje při učení nového programovacího jazyku bude implementována jako plugin do Visual Studia 2010. Bude obsahovat možnost vložit klíčové slova do vyhledávacího pole, které pak dané klíčové slova vyhledá ve svých zdrojích dat. Aplikace pak následně zobrazí takto nalezené výsledky. Výsledky budou ukázky zdrojového kódu v daném programovacím jazyce. Díky tomu bude mít začínající programátor možnost jednoduše vidět již funkční zdrojový kód, který používá dané klíčové slova – funkce, metody, třídy, typy aj. Výsledné data budou vhodně seřazeny, aby programátor našel mezi prvními výsledky hledanou ukázku zdrojového kódu. Na základě tohoto pluginu by si zkušený programátor začínající v novém programovacím jazyce mohl velmi rychle osvojit nový jazyk. Aplikace bude také náležitě otestována programátory, aby splňovala i jejich případné návrhy na vylepšení a opravené chyby.

3.1 Požadavky

Na cílovou aplikaci je kladeno několik požadavků. Požadavky se dělí na uživatelské a systémové. Požadavky na aplikaci byly vytvořeny na základě mých osobních preferencí a na požadavcích vedoucího práce.

3.1.1 Uživatelské požadavky

Z uživatelského pohledu jsou na aplikaci kladeny tyto požadavky:

- Klientská aplikace pro uživatele by měla být implementována a integrována v prostředí Visual Studia 2010.
- Po zadání požadovaného klíčového slova by aplikace měla vyhledat odpovídající výsledek ze svého zdroje informací.
- V případě nenalezení výsledku aplikace vhodně upozorní na tuto skutečnost.

- Klientská aplikace by měla být dostupná z menu Visual Studia a pomocí klávesové zkratky pro urychlení přístupu k ní.
- Po otevření aplikace uživatel může zadat vyhledávací dotaz do textového pole a ten pak stisknutím Enter odešle ke zpracování. Výsledek hledání je mu pak zobrazen v okně aplikace.
- Pokud uživatel bude mít kurzor na konkrétním řádku a vyvolá aplikaci, aplikace vyplní vyhledávací pole celým řádkem z editoru. Pokud uživatel bude mít vybranou konkrétní část textu, vyplní jí vyhledávací pole také.
- Aplikace by měla automaticky rozeznat programovací jazyk a vyhledávat právě v tomto zvoleném jazyce.
- Výsledek vyhledávání by mělo být možné zkopírovat do schránky.
- U výsledků musí být uveden zdroj informací.

3.1.2 Systémové požadavky

Z pohledu samotného systému jsou na aplikaci kladeny tyto požadavky:

- Aplikace by měla nalézt na požadovaný dotaz výsledek ze zdroje.
- V aplikaci by měl být implementován alespoň jeden zdroj dat.
- Aplikace by měla být schopna hledat ve více zdrojích, které se můžou chovat různě.
- Aplikace by měla být spustitelná na více počítačích.
- Základní části aplikace by měly mít jednotkové testy.

4 Návrh řešení

Tato kapitola je zaměřena na popis návrhu řešení celé aplikace. Zahajuje ji zamyšlením se nad možným řešením celé aplikace. Na základě výběru možné varianty řešení se pak odvíjí další podkapitoly této kapitoly. Následující podkapitola se věnuje podrobnějšímu návrhu architektury aplikace. Návrh architektury aplikace je doprovázen obrazovou dokumentací, která má za úkol přiblížit návrh celé architektury. Dále pak návrhem integrace pluginu. Následující podkapitola se zabývá návrhem klientské části aplikace a výběrem požité technologie. Následující kapitola se zabývá návrhem serverové části aplikace a její možnou implementací. V poslední kapitole je řešen výběr komunikačního rozhraní mezi klientskou a serverovou částí aplikace.

4.1 Zamyšlení nad architekturou aplikace

Na základě požadavků z předchozí kapitoly je možné dospět k dvěma možným variantám architektury výsledné aplikace. Z uživatelského pohledu je klientská část aplikace v obou variantách stejná. Ostatní části architektury se ale můžou lišit dle zvoleného řešení. Proto se budu těmito dvěma variantami dále zabývat s cílem nalezení nejoptimálnější architektury.

4.1.1 Varianta č. 1

Varianta č. 1 architektury aplikace spočívá v implementaci pluginu jako stand-alone aplikace tzn. samostatně fungující. Plugin by byl integrovaný do Visual Studia, grafické rozhraní by bylo stejné, rozdíl by však byl po odeslání klíčového slova ke zpracování. Aplikace by se přímo připojila k dostupným zdrojům, zadala vyhledávací dotaz a výsledky by vrátila zpátky do grafického rozhraní. Varianta je rychlá, není zapotřebí zbytečné komunikace s jinými aplikacemi, výsledky jsou vráceny dle délky trvání jednotlivých dotazů na zdroje dat. Výsledky a i řazení výsledků může být nekvalitní, neboť není možné implementovat vlastní řazení. Není také možné nabídnout cachované výsledky a tím urychlit načítání výsledků, dále také není možné vytvořit vlastní databázi výsledků a nabízet ji přednostně. Případně tuto vlastní databázi vylepšovat a tím i vylepšit výsledky.

4.1.2 Varianta č. 2

Varianta č. 2 spočívá v implementaci pluginu jako klientské aplikace napojující se na server pro získání potřebných dat. Server si sám o sobě zajistí veškerou potřebnou režii s vyhledáváním a zpracováním výsledků. Výsledek je pak vrácen zpátky uživateli do grafického rozhraní klientské aplikace. Varianta je pomalejší než varianta č. 1, neboť je zapotřebí komunikovat se serverem a ten pak komunikuje se zdroji dat podle potřeby. Na druhou stranu je ale možné dopravit uživateli

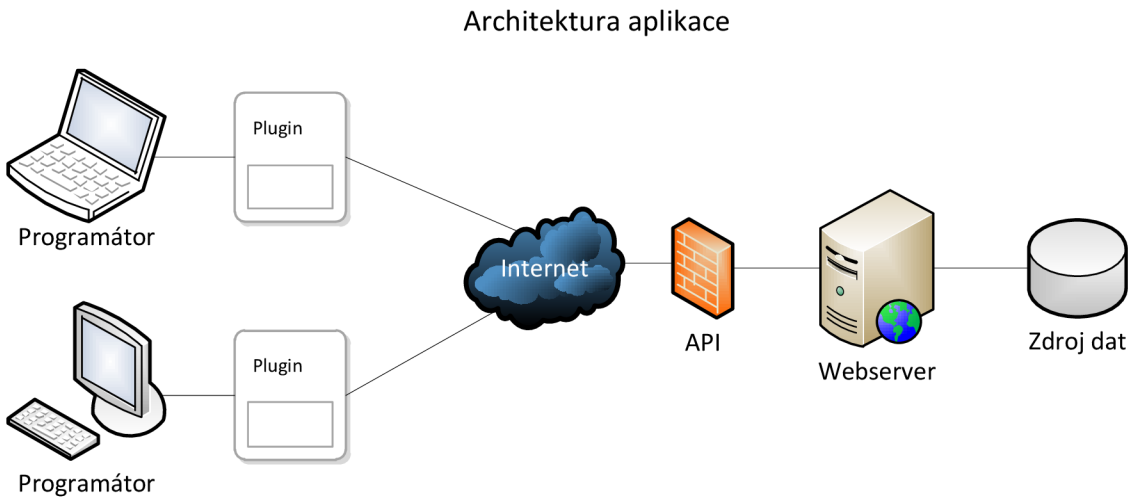
mnohem přesnější a vhodně seřazené výsledky. Můžeme také zapojit vyrovnávací paměť cache a tím urychlit dobu nutnou k čekání na výsledky.

4.2 Architektura aplikace

V předchozí podkapitole byly navrhnuty dvě varianty architektury aplikace. Toto rozhodnutí je pro práci stěžejní, nicméně byla vybrána varianta č. 2, tedy klient-server architektura s kompletní správou zdrojů na serveru.

Celá architektura aplikace je rozdělena do 4 logických sekcí, kterými se budeme zabývat podrobněji. První logická sekce bude návrh Pluginu, další sekce je návrh grafického rozhraní pro klientskou část aplikace, třetí pak je návrh serverové části aplikace a nakonec návrh komunikačního rozhraní mezi klientem a serverem.

Na obrázku vidíme návrh celé architektury aplikace. V levé části jsou počítače programátorů s nainstalovanou aplikací Visual Studio 2010. Programátoři v případě požadavku vyvolají Plugin. Jakmile programátoři zadají dotaz do vyhledávacího pole, je dotaz přeložen do formy, které rozumí klientské i serverové API a dotaz je pak vyslán přes internetovou síť k serveru. Server, resp. webserver, zpracuje dotaz, pokud odpovídá jeho API, zpracuje jej. Server pak nalezne potřebné informace ze zdroje dat a ty pak zpětnou cestou odešle do pluginu Visual Studia.



Obrázek 4.2-1 Architektura celé aplikace z pohledu návrhu

4.3 Plugin

Studiem tvorby pluginu (Add-in) do Visual Studia 2010 bylo zjištěno, že je třeba vytvořit projekt spojující Visual Studio a klientskou aplikaci. Jelikož bylo rozhodnuto, že plugin bude určený

pro nejnovější verzi Visual Studia 2010, řídil jsem se dokumentem [4]. Ve zmíněném dokumentu jsou popsány veškeré doporučení a postupy k implementaci pluginů do Visual Studia 2010. Microsoft tímto dokumentem směřuje programátory k udržování jednotnosti rozhraní a podobného chování pluginů. Dokument se zmiňuje o téměř každé části Visual Studia a jeho správného využití, mezi ně patří vizuální vzhled, klávesové zkratky, použité technologie na grafické rozhraní aj.

4.4 Klientská část aplikace

Klientská část aplikace bude zajišťovat grafické uživatelské rozhraní, zadávání dotazu a bude stát na začátku vyhledávacího procesu – zadávání dotazu a také na konci – přijímání výsledků. Z toho plyne, že bude obsahovat grafické uživatelské rozhraní.

Klientská část aplikace, bude volána z projektové části pluginu. Klientská část bude tenká klientská aplikace, která se připojí přes rozhraní k serveru. Server zpracuje dotaz a vrátí odpověď zpátky přes rozhraní do klientské části aplikace.

Klientská část aplikace může být implementována různými technologiemi a jazyky. Rozhraní by bylo možné implementovat v jazyce Python (WxWidgets), WinForms C#/Visual Basic (Microsoft .NET Framework 2.0), Windows Presentation Foundation (Microsoft .NET Framework 4.0) a jiné.

4.4.1 WinForms

WinForms jsou součástí Microsoft .NET Framework 3. WinForms [6] je grafické programové rozhraní nad Microsoft .NET Frameworkem, ten nám umožňuje přístup k nativnímu rozhraní prvků operačního systému Microsoft Windows. Některé prvky grafického rozhraní jsou mapovány přímo bez jakýchkoliv úprav. Mohlo by se zdát, že se jedná o chytřejší náhradu Microsoft Foundation Class Library, která byla předchůdcem WinForms, ale není tomu tak. WinForms nenabízí tolik moderní a oblíbenou architekturu aplikace model-view-controler (MVC). Programátorská komunita vytvořila MVC architekturu pro WinForms, nejedná se však o nativní podporu.

WinForms jsem si pro implementaci uživatelského rozhraní nevybral z následujících důvodů:

- Nepodporuje nativně MVC architekturu
- Pouhá nadstavba nad základními grafickými prvky Microsoft Windows
- Zastaralá technologie, která má moderního nástupce WPF

4.4.2 Windows Presentation Foundation

Windows Presentation Foundation (WPF), nabízí programátorům oproti WinForms unifikovaný programový model pro vytváření chytrých Windows aplikací s rychlým a chytrým uživatelským rozhraním použitelným jak pro grafické rozhraní, tak i pro dokumenty a média. WPF je všestranná knihovna [5].

WPF je součástí Microsoft .NET Framework 4. Jádro systému využívá grafické karty, jeho implementace díky tomu podporuje vlastní vykreslování komponent nezávislé na procesoru. Díky vektorové reprezentaci grafických komponent se aplikace přizpůsobuje zobrazovacímu zařízení a jeho rozlišení. WPF nabízí řadu předpřipravených komponent jako je Extensible Application Markup Language (XAML), data binding (automatické napojení grafického rozhraní na vnitřní proměnné), layouty, 2D a 3D grafiku, animace, styly, šablony, dokumenty, média, text a typografické prvky.

Microsoft také poskytuje řadu nástrojů na práci s WPF grafickými prvky rozhraní, které dávají tomuto složitému a obsáhlému Frameworku snadnější možnost tvorby aplikací. Microsoft Expression Blend 4 poskytuje uživateli komplexní grafické rozhraní pro návrh a implementaci jakéhokoliv grafického rozhraní. V aplikaci můžete nastavit jakýkoliv grafický prvek v menu. Jelikož je WPF založeno na značkovacím jazyku XAML, je možné také uživatelské rozhraní kompletně vytvářet v přiloženém textovém editoru, případně to kombinovat. XAML je podmnožinou jazyka XML a je tedy přehledný a vhodně strukturovaný.

Uživatelské rozhraní bude naimplementováno využitím Windows Presentation Foundation. WinForms z předchozího oddílu mají příliš nevýhod než výhod. Vybraní WPF pro implementaci bylo především díky těmto důvodům:

- Podpora nástrojů pro kompletní vytváření grafického rozhraní – Microsoft Expression Blend 4
- Nativní podpora MVC
- Využívání grafické karty k vykreslení grafického rozhraní
- XAML

4.5 Serverová část aplikace

Serverová část aplikace má jako hlavní úkol zpracovat připojení od klienta a obsloužit jej. Pod obsloužením si představme přijmutí a navázání spojení od klienta, dále pak přijetí a zpracování požadavku od klienta, vyřízení a dokončení požadavku a odeslání odpovědi zpět klientovi.

Serverová část aplikace může být taktéž implementována v různých jazycích a na různých operačních systémech, a proto bude vytvořeno API, které bude používat klientská aplikace a serverová část ji bude implementovat. Vytvořením API je možné implementovat server dle libosti a stejně tak klientskou aplikaci. Pro implementaci byly brány v úvahu tyto serverové programovací jazyky: C++, PHP, C# a F#. Níže naleznete rozbor jednotlivých možných implementací a jejich rozbor.

4.5.1 C/C++

Tento jazyk operuje na velmi nízké úrovni. Implementace serveru je proveditelná, byla by však velmi komplikovaná. Obzvláště pokud víme, že existují mnohem lepší možnosti implementace. V C/C++ by bylo nutné implementovat i samotný webserver pro příjem neblokujících konkurentních spojení od klientů.

4.5.2 PHP

PHP je velmi rozšířený programovací jazyk pro webservery. PHP v webserverové verzi nativně podporuje neblokující konkurentní server. To nám velmi ulehčuje práci a není třeba implementovat server na úrovni socketů. Mezi nejznámější webservery pro hostování PHP patří Apache (nebo IIS s PHP přes CGI, Sphinx). Implementace v jazyce PHP by přinesla oproti C/C++ navíc nativní implementaci webserveru. S PHP jazykem mám největší zkušenosti.

4.5.3 C#

C# je velká výzva, neboť jsem se s ním dosud nesetkal. Jazyk C# zastřešuje celá rodina programů, frameworků a pluginů. C# zastřešuje velkou kompaktní rodinu produktů, která se chová většinou podobně (má stejné ovládání) a má určitou kvalitu. C# je základním kamenem .NET Frameworku, Visual Studia a Expression Blenderu. Naproti tomu jazyk PHP, C/C++ nenabízí takto ucelené řešení a programátor je odkázaný na různé programy od různých autorů s různou kvalitou a možnostmi.

4.5.4 F#

F# je nový jazyk od společnosti Microsoft, nabízený prostředím .NET Frameworku. F# je multiparadigmatický programovací jazyk, který spojuje funkcionální a imperativně objektový přístup. Jazyk je ovlivněn programovacími jazyku Haskell, C# a LINQ. Jelikož je kompatibilní s .NET Frameworkem a Visual Studiem 2010, vypadá to na vítěze výběru programovacího jazyku pro server. F# také podporuje tzv. DSL, tedy Domain-Specific Language, tedy možnost implementovat si vlastní DSL jazyk pro své účely. WCF services se dají naprogramovat právě v F#, což by mohlo usnadnit práci s WCF.

4.6 API

API[3] slouží k určení specifického komunikačního rozhraní mezi dvěma stranami. V mém případě se jedná především o klientskou a serverovou část aplikace. Způsobů implementací je více, je však

[3] API - Application Programming Interface česky označuje rozhraní pro programování aplikací

možné v různých jazycích docílit stejné implementace, neboť síťová vrstva TCP je standardizovaná. Při rozhodování, jakou implementaci zvolit je důležité zohlednit podporu klientské a serverové části s danou implementací. Pokud by bylo nalezeno rozhraní, které je pro zvolený programovací jazyk vlastní, vývoj rozhraní by obsahoval minimu chyb (nebo žádné) a rychlost vývoje by byla také vyšší.

V případě PHP je jako běžná komunikace brána komunikace přes HTTP protokol. Data jsou v případě webových stránek ve formátu HTML, JSON aj. V případě C/C++ je implementace naprosto libovolná, nic naprogramovaného v základu není. C# nabízí Webservices pro ASP.NET a WCF.

4.6.1 Webservices pro ASP.NET

WebServices pro ASP.NET je komponenta, která je postavena na HTTP protokolu a umí na této vrstvě oboustranně komunikovat. WebServices od Microsoftu přicházejí s univerzálním multiplatformním protokolem, resp. přicházejí s jednoduchou koncepcí, která posílá XML zprávy přes HTTP protokol. Server a klient musí rozumět těmto zprávám. Zprávy jsou popsány pomocí WSDL[4] a XSL[5] Schema Definice (XSD) specifikace. Jakýkoliv server může implementovat dané specifikace. Pokud by bylo zapotřebí přepsat klientskou aplikaci do jiného programovacího jazyku, stačilo by pouze naprogramovat vhodné vytváření požadavků na server podle výše zmíněných schémat a specifikací.

Visual Studio 2010 podporuje WebServices pro ASP.NET, díky tomu jsou nám potřebné schémata schovány a není potřeba je ručně vytvářet (resp. Základní schémata jsou vytvořené, my už jen definujeme atributy a hodnoty v XML). Po vytvoření rozhraní a třídy implementující toto rozhraní je možné vygenerovat potřebný WSDL soubor a následně XSD soubor pro klientské rozhraní.

4.6.2 Windows Comunication Foundation - WCF

Windows Communication Foundation je nástupce WebServices pro ASP.NET, který je založený na .NET Framework verze 4.0 a opírá se o SOA[6]. SOA zobecňuje pomocí WSDL jazyka předchozí implementaci WebServices pro ASP.NET. WCF ale nejsou jen WebServices, součástí jsou i další služby jako WebServices-Discovery, WebServices-Adressing, WebServices-ReliableMessaging, WebServices-Security and RSS Syndication Services. Všechny zmíněné služby implementují rozhraní WSDL. Tato technologie je podporována Visual Studiem 2010 a 2008, což nám dává spousty výhod při implementaci.

WCF se řadí mezi moderní technologie pro komunikaci. Jedná o nejnovější technologii, která je kompatibilní se staršími Webservices pro ASP.NET, vybral jsem si toto API pro implementaci mého komunikačního rozhraní.

---

[4] WSDL – Web Service Definition Language (jazyk definující webovou službu)
[5] XSL – eXtensible Stylesheet Language (rozšiřitelný stylovací jazyk)
[6] SOA – Service-Oriented Architecture (Architektura orientovaná na služby)

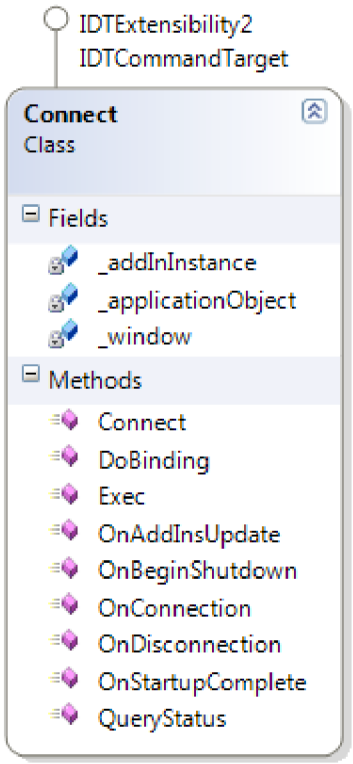
5 Implementace

Tato kapitola popisuje postupy a techniky, které byly použity při programování celé aplikace. V první podkapitole je popsána implementace integrace pluginu do Visual Studia 2010. Další podkapitola se zabývá implementací API rozhraní a jeho schopnostem při komunikaci. Další podkapitola podrobně popisuje implementaci grafického uživatelského rozhraní klientské aplikace za pomocí WPF. Předposlední podkapitola se zabývá implementací serverové části aplikace a implementací zdroje dat. Poslední podkapitola popisuje celkovou architekturu na úrovni sekvenčního diagramu.

5.1 Plugin

K integraci pluginu (add-in v žargonu Visual Studia) jsem využil předpřipraveného průvodce pro vytváření nového pluginu. Nalezneme jej v menu pod New Project/Other Project Types/Extensibility/Visual Studio Add-in. V průvodci můžete vybrat v jakém programovacím jazyce chcete plugin implementovat. Na výběr je z Visual C# (ten jsem zvolil já), Visual Basic, Visual C++ / CLR a Visual C++ / ATL. V dalším kroku je nutné vybrat místo užití pluginu, na výběr je z VS 2010 a VS 2010 se zapnutými makry. Dále pojmenujeme plugin a přidáme popisek pluginu. Další možné nastavení je pak oblast použití pluginu, tedy zda se bude jednat pouze o grafické rozhraní, nebo zda-li jej chceme pouze používat v příkazové řádce VS. Je také možné zapnout automatické spouštění pluginu při zapnutí resp. po načtení Visual Studia. V poslední kroku už jen můžeme nechat vygenerovat i „About box“, tedy česky „O aplikaci“.

Po dokončení průvodce se automaticky vytvoří projekt, který bude obsahovat základní třídy, předpřipravenou konfiguraci pro spouštění a pro debugovací režim, také je plugin automaticky přidán do Add-in Manageru, tedy správce pluginů.



Obrázek 5.1-1 zobrazuje class diagram třídy Connect. Třída slouží na spojení Add-in Manageru s klientskou částí aplikace

Základní třída `Connect` rozšiřuje třídu `IDTExtensibility2` a `IDTCommandTarget`. Třída `IDTExtensibility2` obsahuje základní metody, které vykonávají příkazy při spuštění Visual Studia (`onStartupComplete`), při načtení pluginu (`onAddInsUpdate`), při vypnutí pluginu (`onBeginShutdown`) a dvě metody volané při připojení (`onConnection`) nebo odpojení (`onDisconnection`) pluginu k Visual Studiu. Druhá implementující třída `IDTCommandTarget` slouží k umožnění volání pluginu pomocí příkazu *command*. Příkaz *command* se používá k volání pluginu z prostředí IDE. V mém případě bylo této funkce využito k napojení pluginu na klávesové zkratky Visual Studia.

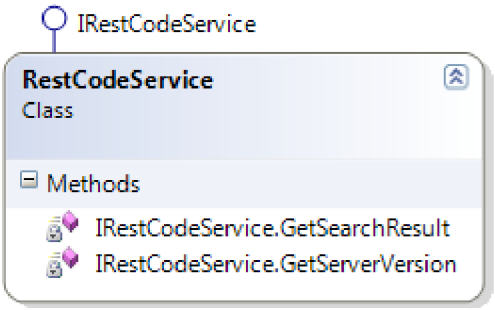
Průvodce nám také vytvořil definiční xml soubor pluginu, který slouží k automatickému načítání při spuštění Visual Studia 2010. Soubory pro plugin se musí nacházet ve složce Documents\Visual Studio 2010\Addins. Pokud tato složka ještě neexistuje, je třeba ji vytvořit. Soubor musí mít koncovku .addin.

Plugin je možné spustit z hlavního menu Visual Studia z Tools\Code Search. Plugin je namapovaný na klávesovou zkratku F6 a je možné jej vyvolat kdykoli. Pokud budeme mít otevřený

projekt a budeme mít vybraný konkrétní řádek, tento řádek se nám zkopíruje do vyhledávacího pole, pokud budeme mít vybraný konkrétní text, vloží se také.

5.2 API

Komunikační vrstva serveru s klientskou aplikací je implementována pomocí WebServices pro ASP.NET. K vytvoření implementace bylo potřeba z knihovny System.ServiceModel vytvořit rozhraní, s předepsanými pravidly z dokumentace. Na obrázku níže můžeme vidět class diagram API rozhraní.



Obrázek 5.2-1: Class diagram API rozhraní

Rozhraní musí mít potřebné atributy u metod, které mají být součástí API. Níže můžete vidět výsledné rozhraní pro mou aplikaci:

```
1. [ServiceContract]
2. public interface IRestCodeService {
3.  [OperationContract]
4.  String GetServerVersion();
5.
6.  [OperationContract]
7.  IEnumerable<ResultItem> GetSearchResult(string query, string
   language);
8. }
```

Kód 5.2-1: Komunikační rozhraní služby ve WCF

V kódu výše vidíme dvě základní metody, `GetServerVersion()` a `GetSearchResult(string query, string language)`. První zmíněná metoda vrátí aktuální verzi instalovaného kódu na serveru (ta se zobrazuje v pravém dolním rohu Grafického rozhraní). Druhá metoda hledá na základě parametrů ve zdrojích a vrací výsledek jako `IEnumerable<ResultItem>`. Tento návratový typ resp. rozhraní byl vybrán, neboť nejméně

svazuje programátorovi ruky při práci s výsledky. Díky používání `IEnumerable` na místo např.: `Collection`, `List`, aj. bylo možné využívat knihovny LINQ. Anotace `ServiseContract` označuje, definuje rozhraní služby v aplikaci. `OperationContract` pak označuje operace služby v aplikaci.

K plné funkčnosti API je nutné implementovat rozhraní `IRestCodeService` ve třídě `RestCodeService`. Implementace rozhraní vyžaduje specifikování dalších anotací k metodám implementujícím metodám z rozhraní.

```
1. [ServiceKnownType(typeof(IEnumerable<ResultItem>))]
2. [AspNetCompatibilityRequirements(RequirementsMode =
   AspNetCompatibilityRequirementsMode.Allowed)]
3. [ServiceBehavior(InstanceContextMode =
   InstanceContextMode.PerCall)]
4. public class RestCodeService : IRestCodeService {
5. [WebGet]
6. string IRestCodeService.GetServerVersion() {…}
7. [WebGet(UriTemplate = "{language}/{query}")]
8. IEnumerable<ResultItem> IRestCodeService.GetSearchResult(string
   query, string language) {…}
9. }
```

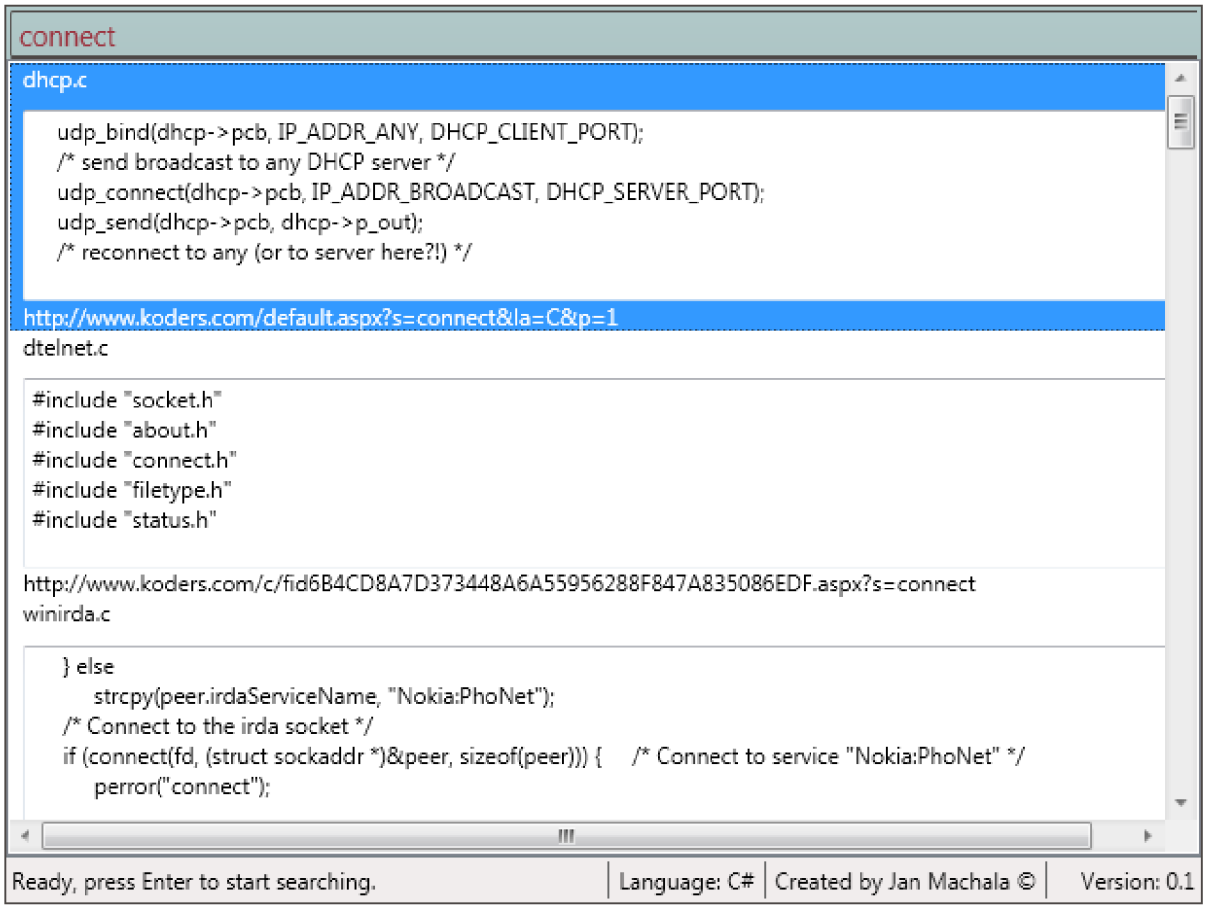
Kód 5.2-2 Třída RestCodeService implementující rozhraní IRestCodeService - popis anotací

Atribut `ServiceKnownType` definuje funkci, která vrací známé datové typy aplikace. V mém případě se jedná o funkci `typeof`. Atribut `AspNetCompatibilityRequirements` značí povolení kompatibility s novější verzí API služeb WCF. Atribut `ServiceBehavior` značí implementační vykonávání po zavolání této služby. V našem případě specifikujeme, že při volání je tato funkce vykonána při tomto požadavku. Atribute `WebGet` nám definuje rozhraní pro volání pro protokol REST. Pokud nadefinujeme UriTemplate, můžeme mít i vlastní cestu k požadované odpovědi. V mém případě by cesta mohla vypadat např. „C#/Regexp“.

5.3 WPF – Klientská část aplikace

Klientské rozhraní bylo naimplementováno dle uživatelských a systémových požadavků. K prvotní implementaci rozhraní bylo využito aplikace Microsoft Expression Blend 4 (dále jen EB). EB je výborná aplikace na návrh jakéhokoliv uživatelského rozhraní pro WPF. Aplikace z prvu vypadá složitě, ale při hlubším prozkoumání a pochopení principů WPF se stává ovládání aplikace logickým. Uživatelské rozhraní klientské části aplikace bylo načrtnuto v EB, posléze však bylo psáno přímo

jazyce XAML. EB byl používán pouze pro první návrh rozhraní po dobu platnosti 30 dní trial verze. Po vypršení licence byl návrh dolaďován pomocí jazyka XAML a Visual Studia 2010. Visual Studio 2010 umožňuje náhled na aplikace vytvořené v jazyce XAML. Pokud využíváme DataTemplate pro definici obsahu prvku, náhled ve Visual Studiu 2010 není dostupný. Jsme tak odkázáni na dokumentaci a teoretické chování prvků. Jelikož jazyk XAML obsahuje DTD, Visual Studio nám usnadňuje definování XAML za pomoci autokompletace – IntelliSense. IntelliSense zde umí napovídat i hodnoty atributů. Návrh aplikace před uživatelským testování si můžete prohlédnou na obrázku níže.



Obrázek 5.3-1: Klientská aplikace navržená v EB a připravená pro uživatelské testování

Na obrázku Obrázek 5.3-1 vidíme v horní části aplikace vyhledávací pole, které obsahuje klíčové slovo „connect“. Pod ním se nachází výpis výsledků vyhledávání. Jeden výsledek obsahuje název souboru – nadpis, ukázku zdrojového kódu a adresu ke zdroji. Ukázkový zdrojový kód je vložen do textového pole pro snazší kopírování.

Pokud chceme vytvořit libovolný výpis dat, můžeme k tomu využít prvek ListBox. Pro ListBox je třeba nadefinovat vlastní šablonu, která se pak bude využívat k vykreslení. V šabloně můžeme použít jakékoliv prvky včetně obrázků, v našem případě bylo využito prvků TextBlock a TextBox (a po uživatelském testování prvek Hyperlink). To celé bylo obaleno do mřížky Grid.

K ListBoxu je možné napojit odpovídající datový zdroj, který se pak automaticky stará o aktualizaci. Aktualizase grafického prvku je provedena v okamžiku k přistoupení přednastavené proměnné.

Dále je také možné přebírat atributy z nadřazených prvků a tak docílit dynamického resp. přizpůsobitelného vzhledu, který není definovaný napevno (přesně na pixel).

Pod výpisem výsledků můžeme vidět prvek StatusBar. StatusBar zobrazuje aktuální programovací jazyk, verzi kódu na serveru, stav aplikace a tvůrce aplikace.

Na verzi serveru se aplikace automaticky ptá při inicializaci aplikace, pokud se spojení na server nevydaří, verze se nenačte a na místo toho se zobrazí N/a, tedy nedostupné. V takovém případě je problém se serverem.

Programovací jazyk se automaticky detekuje podle jazyku otevřeného zdrojového kódu. Když tedy budeme mít otevřený C#, F# nebo cokoliv jiného, plugin si vezme při otevření právě tuto hodnotu a použije ji při vyhledávání. Pokud nebudeme mít otevřený zdrojový kód, použije se jako výchozí hodnota C#.

5.4 Serverová část aplikace

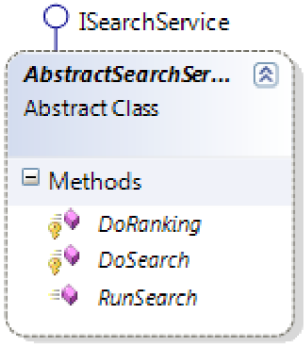
V kapitole návrh bylo spekulováno o implementačním jazyku serveru. Na implementaci byl vybrán jazyk C#. Při implementaci serveru byla implementace podřizována dvěma základním pravidlům DRY[7] a „Tell, Don't Ask[8]“. Mnohokrát byl již napsaný kód smazán a napsán znovu, aby bylo dosaženo základních myšlenek těchto pravidel. Implementace se řídila zásadami TDD[9], aby byl vytvořen testovatelný programový kód. K tomu byly vytvořeny jednotkové testy. Díky striktnímu dodržování zmíněných pravidel bylo docíleno jednoduchého, neopakujícího se a lehce pochopitelného zdrojového kódu.

V předchozí podkapitole jsme vytvořili rozhraní služby ve WCF, nyní budeme rozhraní `IRestCodeService` implementovat. Jak již bylo řečeno dříve `GetServerVersion` nám vrátí verzi serveru, která je uložena v kódu jako konstanta. Metoda `GetSearchResult` implementuje nadřazené rozhraní tak, že pro všechny dostupné vyhledávací zdroje provede metodu `RunSearch`. Každý zdroj ji může implementovat jinak, a tak tak bylo vytvořeno rozhraní (`ISearchService`) pro tyto účely a abstraktní třída (`AbstractSearchService`). `ISearchService` obsahuje metodu `RunSearch`.

[7] DRY – Don't Repeat Yourself – Je důležité se vyvarovat opakování stejného kódu v aplikaci.
[8] Tell, Don't Ask – Volně přeloženo znamená Dělej a neptej se. Tato technika v podstatě nařizuje, abychom radši psali řídící struktury uvnitř volaných metod na místo rozhodování se z návratové hodnoty, co budeme s daným stavem metody dělat dál.
[9] TDD – Test-Driven Development – Vývoj zaměřený na testování

5.4.1 Zdroje dat

Na základě návrhu v předchozí kapitole byl implementován jeden zdroj dat, a sice webová stránka Koders.com. Každý zdroj dat musí implementovat rozhraní `ISearchService`, které je vyobrazeno na obrázku níže.



Kód 5.4-2 Rozhraní, které musí vyhledávací služby (zdroje dat) implementovat

Koders.com

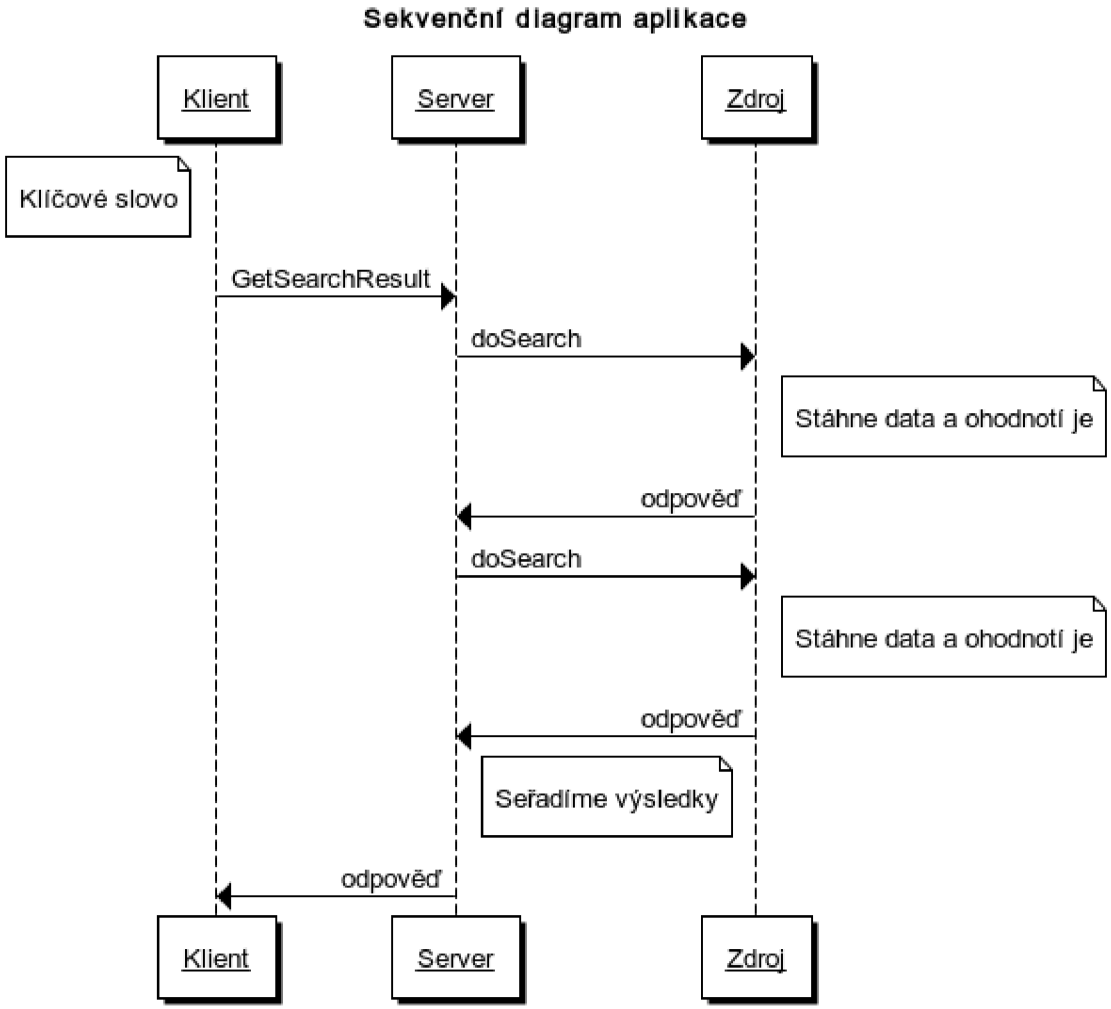
Koders.com poskytuje API ke svému vyhledávání, resp. poskytuje API, ale jen pokud přesně víte, co hledáte (knihovna, třída, metoda, rozhraní aj.), obecné vyhledávání toto API neposkytuje, výsledků tedy bylo docíleno dolováním dat přímo z výsledků vyhledávání. Ke stažení a zpracování výsledků bylo využito knihovny System.Net, System.IO a na zpracování System.Text.RegularExpressions. O stažení webové stránky se postarala statická třída `WebsiteFetcher`.

Další zdroje dat

Serverová část je navrhnuta velmi obecně. Je možné přidat jakýkoliv jiný zdroj dat na server. Mezi dalšími uvažovanými zdroji patřil projekt *www.google.com/codesearch*. Tento projekt však byl v průběhu návrhu aplikace uzavřen pro veřejnost a bylo nutné najít jiný zdroj informací. Mezi dalšími zdroji dat bych uvedl ještě webové stránky *searchco.de*, *krugle.org*, *grepcode.com*. Server je navržený tak, že by bylo možné využít i lokální (nebo externí) SQL databázi.

5.5 Architektura aplikace

V předchozích podkapitolách byly podrobně popsány jednotlivé logické celky celé architektura aplikace. Na sekvenčním diagramu níže můžeme vidět, jak probíhá běžné vyhledávání zadaného klíčového slova od klientské aplikace přes server až po získání dat ze zdroje:



Obrázek 5.5-1 Sekvenční diagram celé aplikace

Obrázek 5.5-1 ukazuje jednotlivé kroky v aplikaci, které je nutné vykonat při vyhledávání z klientské aplikace. Vstupním bodem je zde *klíčové slovo* umístěné vlevo nahoře na obrázku. Po zadání klíčového slova klientská aplikace volá funkci *GetSearchResult*. Funkce zpracuje požadavek a dotáže se na Serveru odpovědi. Server zavolá funkci *doSearch*, která vykoná samotné vyhledávání dat ve zdroji. Pokud je potřeba udělat více dotazů na zdroj dat, server tak učiní. Po zpracování výsledků jsou seřazené výsledky odeslány zpět do klientské aplikace.

V případě implementace více zdrojů dat se hledání doSearch opakuje pro další zdroje dat, dokud není celý výsledek hotový. Tato část serveru je v současném stavu velmi nevyhovující, neboť dotazy na zdroje jsou volány sériově a nikoli paralelně. Tato vlastnost může vyústit až ve zdlouhavé čekání na výsledek. Předpoklad se potvrdil při testování, je však při současném stavu počtu zdrojů udržitelný.

6 Testování

Tato kapitola popsuje postupy a způsoby testování prováděné aplikace. První podkapitola se zabývá testováním jednotek, další podkapitola se zabývá integračními testy aplikace.

6.1 Testy jednotek

Testy jednotek, nebo-li z anglického Unit Testing slouží k testování malých částí zdrojového kódu. Za takovou jednotku může být považována funkce nebo procedura. Při vývoji vzniklo několik jednotek, které mají i své testy. Mezi ně patřily tyto jednotkové testy:

- Vyhledávací modul pro zdroj Koders.com – *KodersSearchServicetTest*
- Fiktivní vyhledávací modul, který ověřuje základní funkčnost rozhraní pro vyhledávací moduly – *FakeSearchServiceTest*
- Jednotka pro stahování z internetu – *WebsiteFetchTest*

6.2 Integrační testy

Integrační testování je testování větších částí projektu, než jednotkové testy. Integrační testování se zaměřuje na celkovou funkčnost aplikace skládající se z menších komponent. Integrační testování většinou probíhá po velkých změnách v komponentách. Bývá také součástí testování před zveřejněním aplikace. Cílem integračního testování je zajištění stejné funkčnosti, jak před změnou.
V průběhu vývoje byly prováděny tyto integrační testy

- Klientská aplikace WPF a její správná funkčnost oproti serveru
- Plugin a jeho správná integrace do prostředí Visual Studia 2010

7 Uživatelské testování

Tato sekce popisuje průběh a výsledky uživatelského testování. První podkapitola se zabývá prvním uživatelským testováním s výsledky. Další podkapitola upravuje aplikaci na základě uživatelského testování. Další podkapitola pak provádí druhé uživatelské testování a poskytuje výsledky. Další podkapitola pak opravuje poslední objevené chyby při testování. Poslední podkapitola pak vyhodnocuje úspěšnost či neúspěšnost uživatelského testování.

7.1 Uživatelské testování č. 1

Aplikace byla předložena uživateli-programátorovi (dále jen tester) k otestování. Tester dostal za úkol vytvořit jednoduchou aplikaci, která se připojí na vzdálený webserver a stáhne libovolnou webovou stránku a uloží ji. Tester byl seznámen s ovládáním a požadavky [3.1] na plugin. Tester nikdy neprogramoval v programovacím jazyce C# avšak měl zkušenosti s jazykem PHP.

Na základě pozorování testera u práce se daný plugin v tomto stavu neosvědčil jako spolehlivý zdroj zdrojového kódu, tester poskytl řadu návrhů na vylepšení, které by tyto nedostatky měly vyřešit. K zobrazení nedostatků aplikace jsem vytvořil jednoduchou tabulku se seznamem chyb.

Tabulka 7.1-1: Seznam chyb nalezených při 1. uživatelském testování

Popis chyby	Následky chyby	Příčiny chyby	Doporučení pro opravu
Konkrétní výsledek obsahoval nedostatečné množství zdrojového kódu.	Programátor hledá možné řešení jinde. Ví alespoň, co má hledat.	Zdroj dat koders.com obsahuje pouze malý náhled na kód.	Stahovat ukázkové kódy přímo z uvedených zdrojů.
Nebylo možné kliknout na odkaz se zdrojem	Ruční hledání výsledků – nevyužití pluginu.	Grafické rozhraní po kliknutí na zdroj neotevře tento zdroj v internetovém prohlížeči	Po kliknutí na text otevřít internetový prohlížeč se zdrojem
Dlouhá časová prodleva čekání na výsledek	Ztráta času při hledání. Délka čekání cca 10s	Výsledek se vyhledává živě na internetu	Předpřipravit výsledky do vlastní databáze
„Version: N/a“	-	Informace se nestáhne ze serveru	Prověřit připojení na server v době inicializace pluginu
„Language: N/a“	Nejistota, jaký je aktuální programovací jazyk vyhledávání	Při inicializaci se nenastavil výchozí jazyk	Nastavit jazyk při inicializaci
Nejasné odlišení jednotlivých výsledků mezi sebou	Zmatení programátora	Nenastaveno visuální oddělení výsledků.	Přidat zvýraznění sudých výsledků

Duplicitní výsledky	Zbytečné pročítání stejných výsledků	Nekontrolování duplicitních výsledků ze zdrojů	Přidat detekování stejných výsledků

Tester dále nalezl několik výhod, které mu plugin přinesl. I přes všechny chyby byl tester schopen dokončit úkol avšak pracněji. Díky pluginu tester dostal povědomí o tom, co má hledat na internetu, nebo přímo v dokumentaci. Výsledky byly sice krátké, ale většinou obsahovaly podstatnou informaci o použité třídě nebo knihovně a tudíž bylo možné na základě této informace hledat dál.

Tester se dále zmínil o řazení výsledků. Jak bylo psáno již v úvodu, dobré řazení výsledků by tento plugin obrovsky zvýhodnilo. Současné řešení výsledky řadí ve stejném pořadí jako zdroj dat. I přes tento požadavek tester nalezl požadovaný výsledek do 1-10 místa.

Tester byl také spokojen s integrací do Visual Studia, na ní by nic neměnil. Pozitivně hodnotil automatické kopírování aktuálního řádku do vyhledávací oblasti a stejně tak i označeného textu. Rovněž detekce jazyka mu fungovala, když namátkou vyzkoušel vytvořit nový projekt v C++. Aplikaci v prostředí Visual Studia spouštěl pomocí klávesové zkratky F6 na místo položky v menu.

7.2 Úpravy aplikace

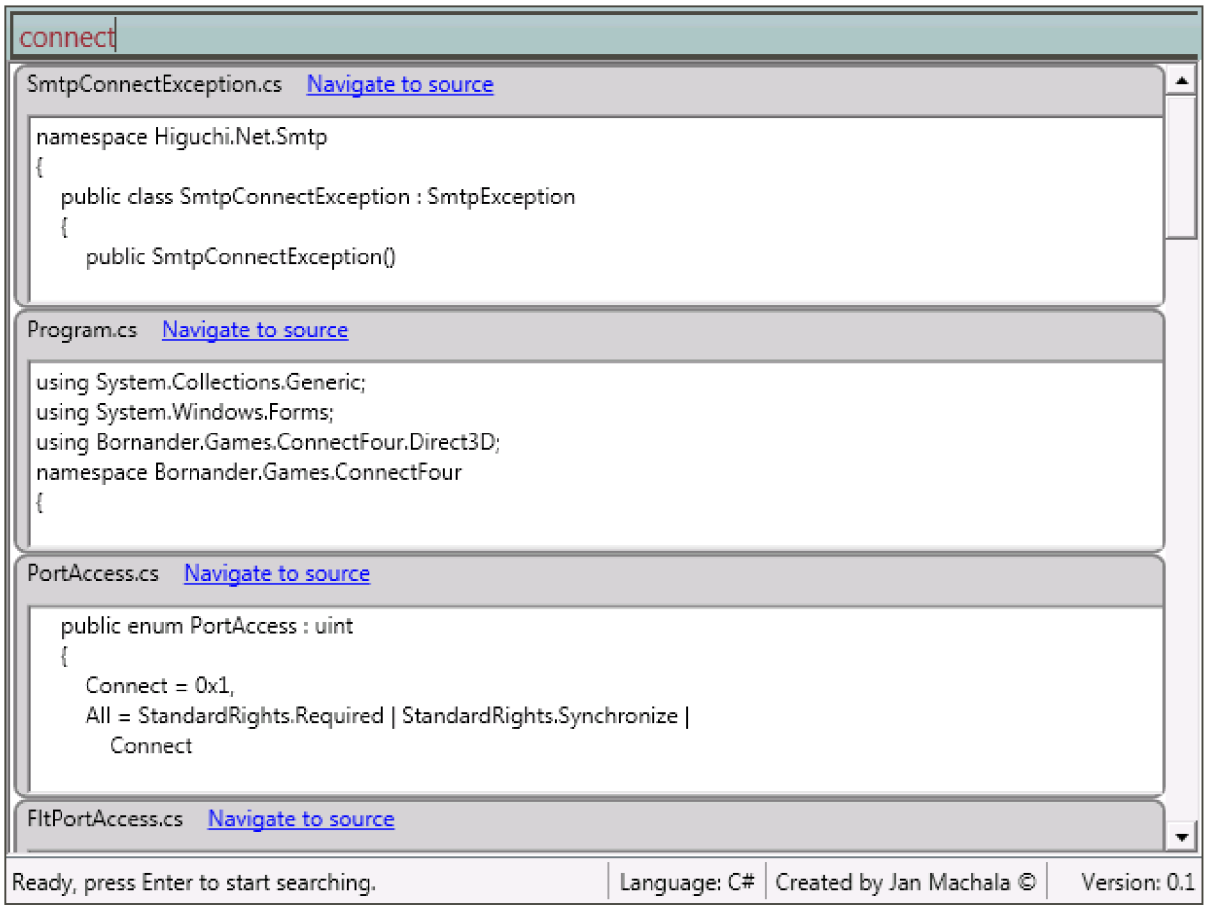
První uživatelské testování objevilo chyby v grafickém uživatelském rozhraní a ve zdroji dat. Připomínky grafického rozhraní byly očekávány, neboť nebylo ještě zcela doprogramované. V aplikaci byly opraveny následující chyby:

Tabulka 7.2-1: Opravené chyby z 1. uživatelského testování

Popis chyby	Způsob opravení
Nebylo možné kliknout na odkaz se zdrojem	Uri odkazu byla schována a přesunuta za titulek výsledku. Byla přidána akce pro otevření internetového prohlížeče.
„Version: N/a“	Pořadí inicializace bylo opraveno.
„Language: N/a“	Byla přidána defaultní hodnota pro spouštění mimo projektové okno.
Nejasné odlišení jednotlivých výsledků mezi sebou	Opravou chyby s neklikatelným odkazem došlo také ke zpřehlednění výpisu výsledků.
Duplicitní výsledky	Na serveru přibylo ověřování duplicitních výsledků.
Dlouhá časová prodleva čekání na výsledek	Na serverové části aplikace byla přidána cache na úroveň stahování obsahu ze zdroje – třída `WebsiteFetcher`. Aplikace se zrychlila o 50% na 5s.

Oprava chyby „Konkrétní výsledek obsahoval nedostatečné množství zdrojového kódu.“ Není při současném návrhu možná, především je problém v limitaci zdroje dat. Oprava by byla možná provést dodatečným dotazem a stažením cílového zdrojového ukázkového kódu, tato operace by však značně prodloužila dobu hledání a tudíž opravena nebude.

Na obrázku můžete vidět uživatelské rozhraní po opravě chyb:



Obrázek 7.2-1 Opravené uživatelské rozhraní po 1. uživatelském testování

7.3 Uživatelské testování č. 2

Druhé uživatelské testování probíhalo se dvěma testery. Přizval jsem tentokrát nového testera. Nový tester také neměl zkušenosti s C#. Zadání úkolu bylo stejné jako při předchozím testování.

Na základě testování bylo zjištěno, že předchozí opravené chyby byly opravdu opravené a testeři se změnami byli příjemně spokojeni. Chyba s nedostatkovým množstvím ukázkového kódu sice přetrvává, ale díky možnosti si prohlédnutí cílového zdrojového kódu je aplikace praktičtější. Pozitivně bylo také hodnoceno odstranění duplicitních výsledků. Čas načítání výsledků se zdál o cca 50% rychlejší jak v předchozí verzi (z původních 10s jsou výsledky načítány do 5s, někdy i rychleji). Níže uvádím tabulku s chybami z 2. uživatelského testování.

Tabulka 7.3-1: Seznam chyb nalezených při 2. uživatelském testování

Popis chyby	Následky chyby	Příčiny chyby	Doporučení pro opravu
Konkrétní výsledek obsahoval nedostatečné množství zdrojového kódu.	Programátor hledá možné řešení jinde. Ví alespoň, co má hledat.	Zdroj dat koders.com obsahuje pouze malý náhled na kód.	Stahovat ukázkové kódy přímo z uvedených zdrojů.
Kurzor přestane blikat, rozhraní zamrzne	Programátor neví, zda aplikace běží, nebo vyhledává.	Vyhledávání je ve stejném vlákně, jako grafické rozhraní	Vytvořit vyhledávací vlákno, které po zpracování pošle výsledek do Grafického rozhraní
Chybí informace o probíhajícím hledání	Programátor neví, zda aplikace vyhledává.	Grafické rozhraní nezobrazuje stav vyhledávání	Aktualizovat status bar při začátku hledání a při ukončení.

Testeři pozitivně hodnotili celou aplikaci. Pokud bych spravil problémy s grafickým rozhraním a přidal další zdroje dat a rozšířil výsledky, byli by s aplikací spokojeni.

7.4 Úpravy aplikace

Na základě nahlášených chyb od Testerů byly opraveny chyby. Zamrzání aplikace bylo vyřešeno vytvořením speciálního negrafického vlákna (`BackgroundWorker`), které na pozadí provede dotaz na server a poté vrátí data zpět do grafického rozhraní 32[10]. Díky této změně bylo potřeba trochu přehodnotit strukturu klientské aplikace. Samostatné vlákno totiž nemůže přistupovat k proměnným v hlavním vlákně, a proto bylo potřeba předávat veškeré potřebné hodnoty parametry. Doposud jsem se nesetkal s takovým řešením zamrzání aplikace, ale v podstatě mi přijde logické a funkční dle očekávání.

Chybějící informace o průběhu hledání byla, díky vytvoření speciálního vlákna na vyhledávání, jednoduše implementována nastavením příslušné hodnoty status baru.

7.5 Shrnutí uživatelského testování

Ze začátku jsem byl skeptický k provádění uživatelského testování, nicméně po prvním uživatelském testování jsem získal několik podmětných nápadů na vylepšení a nahlášení několika zásadních chyb. Po opravení jsem provedl ještě jednu iteraci uživatelského testování, která objevila posledních pár chyb. Můj osobní pocit z testování je skvělý. Díky testování jsem opravil i chyby, kterých bych si nevšiml, nebo bych je nepovažoval za důležité. Aplikace se tak posunula o velký krok kupředu. Aplikace nyní funguje tak, jak jsme zvyklí z normálních aplikací.

8 Rozšíření

V této kapitole se zamyslím nad možnými rozšířeními aplikace. První podkapitola se zabývá rozšířením o další zdroje dat pro aplikaci. Další podkapitola navrhuje výkonnostní vylepšení paralelizací získávání dat. Poslední podkapitola rozšiřuje možnosti navrhnuté architektury o vytvoření pluginu do dalšího IDE.

8.1 Zdroje dat

Při uživatelském testování bylo mimo jiné zjištěno také to, že aplikace by mohla obsahovat lepší zdroj dat, který by poskytoval větší ukázky kódu. Jak již bylo zmíněno v oddílu 3.1.2, aplikace byla navrhnuta primárně tak, aby při požadavku na přidání nového zdroje dat se nemusela aplikace přepisovat. Aplikace nabízí rozhraní `ISearchService`, a abstraktní třídu `AbstractSearchService`. Nejdůležitější by však bylo na přidání nového zdroje to, jak by se hodnotily jeho výsledky a jak by ovlivnily současné zdroje. Bylo by zapotřebí vytvořit potřebné váhové ohodnocení jednotlivých zdrojů a na jejím základě pak vypočítat hodnocení výsledků.

8.2 Paralelizace zdrojů dat

Datové zdroje jsou v současnosti volány sériově, jeden po druhém. To není rychlé řešení, neboť se čas na získání dat zvyšuje s narůstajícím počtem zdrojů lineárně. Tohoto chování by bylo možné docílit vytvořením pracovních vláken, které by po zpracování vrátili výsledky. Bylo by také možné po zpracování každého výsledku jej vracet zpět hlavnímu vláknu, které by pak mohlo odpovědět rychleji, při docílení požadovaného množství výsledků.

Pokud bychom našli zdroj dat, který by nám poskytl API rozhraní, Docílili bychom taktéž zrychlení.

8.3 Uživatelské rozhraní

Uživatelské rozhraní je v současném stavu jednoduché. S přibývajícími zdroji by bylo možné přidat více typů výsledků. V případě, že by byla vyhledávána konkrétní třída, mohl by být výsledek upraven a zobrazoval by pouze odkaz na zdroj, nebo by mohl vypsat dokumentaci k dané třídě. Dalším typem výsledku by mohl být vložený tutoriál i s ukázkovým kódem.

Další možností by mohlo být asistované vkládání výsledku do kódu. Po stisknutí zvolené klávesy by se vybraný výsledek zkopíroval do aktuálního dokumentu. S tímto kusem kódu by se také vytvořil komentář, který by v sobě uchoval odkaz na zdroj a použitý vyhledávací řetěz.

8.4 Klientská aplikace pro IDE Netbeans

Celá aplikace je již od základu navrhována tak abstraktně a obecně, že případné vytvoření další implementace klientské aplikace by neměl činit vetší problém (pomineme-li složitost implementace v daném jazyce). Aplikace má definované standardizované rozhraní WSDL jazyka, které nám umožňuje používat jej nezávisle na implementačním jazyce serveru.

9 Závěr

Bakalářská práce si kladla za cíl vytvořit nástroj na urychlení vývoje v MS Visual Studiu 2010. Nástroj byl úspěšně naprogramován a otestován. Z hodnocení testerů a i z mého osobního pohledu nástroj umožňuje urychlit seznamování se s novým jazykem začínajícím programátorům.

Při vytváření této aplikace jsem se naučil programovat v jazyce C#, seznámil jsem se také s principy a funkčností Webservices, Windows Presentation Foundation a Windows Communication Foundation. Nejvíce mne zaujala práce s WPF, neboť jsem se dosud nesetkal s návrhem grafického rozhraní pro Windows aplikace.

Nová také pro mě byla zkušenost s uživatelským testováním klientské aplikace. Na základě uživatelského testování bylo odstraněno několik chyb, testeři však také poskytli nápady na vylepšení, které mne nenapadly. Je velmi motivující vidět, jak aplikaci již někdo používá a jaké při tom zažívá problémy, pocity a reakce.

Literatura

[1] Brace and Indent Styles and Code Convention. In: *Cher* [online], 2011 [cit. 2012-05-01]. Dostupné z: http://www.riedquat.de/prog/style

[2] MSDN Archive: StyleCop. *StyleCop* [online]. 2008 [cit. 2012-05-09]. Dostupné z: http://archive.msdn.microsoft.com/sourceanalysis

[3] FxCop. *FxCop* [online]. 2012 [cit. 2012-05-09]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx

[4] Microsoft Visual Studio 2010 User Interface Guidelines. MICROSOFT CORPORATION. *Microsoft Visual Studio 2010 User Interface Guidelines* [online]. 2008 [cit. 2012-03-20]. Dostupné z: http://archive.msdn.microsoft.com/VS2010UX

[5] Introduction to WPF. MICROSOFT CORPORATION. *Introduction to WPF* [online]. 2012 [cit. 2012-03-23]. Dostupné z: http://msdn.microsoft.com/en-us/library/aa970268.aspx

[6] Windows Forms. *Windows Forms* [online]. 2012 [cit. 2012-04-22]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/dd30h2yb.aspx

[7] Agent Ransack. *MythicSoft* [online]. 2012 [cit. 2012-05-09]. Dostupné z: http://www.mythicsoft.com/page.aspx?type=agentransack&page=home

[8] Refactorings in Alphabetical Order. *Aplha List Of Refaktoring* [online]. 2012 [cit. 2012-05-09]. Dostupné z: http://www.refactoring.com/catalog/index.html

[9] SCHOLTZ, Jean C. A transfer of skill between programming languages. *SIGCHI Bulletin*. 1989, roč. 21, č. 1.

[10] C# BackgroundWorker. *C# BackgroundWorker* [online]. 2012 [cit. 2012-05-11]. Dostupné z: http://www.dotnetperls.com/backgroundworker

Seznam příloh

Příloha 1. Obsah CD/DVD

Přiložené CD/DVD obsahuje:

- Tento dokument ve formátu PDF a DOCX
- Zdrojový kód aplikace
- Spustitelná aplikace
- Návod na instalaci
- Návod k použití