VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

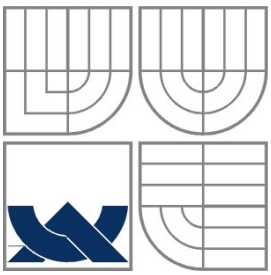
INTEGRACE VÝVOJOVÝCH PROSTŘEDKŮ AVG DO PROSTŘEDÍ VISUAL STUDIO 2005

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

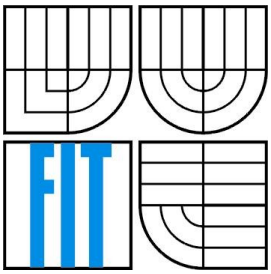
AUTOR PRÁCE
AUTHOR

JAKUB RAJMAN

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE VÝVOJOVÝCH PROSTŘEDKŮ AVG DO PROSTŘEDÍ VISUAL STUDIO 2005

INTEGRATION OF AVG DEVELOPMENT TOOLS TO VISUAL STUDIO 2005 ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB RAJMAN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2008

Zadání bakalářské práce/6764/2007/xrajma00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Rajman Jakub**
Obor: Informační technologie
Téma: **Integrace vývojových prostředků AVG do prostředí Visual Studio 2005**
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s vývojovým prostředím Microsoft Visual Studio 2005.
2. Prostudujte možnosti rozšíření a uživatelských úprav pro zvýšení produktivity práce, jako např. vizualizace abstraktních datových typů, podpora úryvků kódu (*code snippets*) apod.
3. Navrhněte a implementujte vizualizační moduly pro abstraktní datové typy systémových knihoven AVG, generování kódových úryvků pro typické konstrukce a vazbu na DoxyGen dokumentační systém.
4. Implementované moduly integrujte do repozitáře SVN.
5. Proveďte hodnocení vaší práce a diskutujte možnosti dalších rozšíření.

Literatura:

- dle zadání vedoucího

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

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

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

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

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

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

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jakub Rajman**
Id studenta: 78776
Bytem: Hlavní 28/10, 725 28 Ostrava
Narozen: 11. 02. 1986, Ostrava
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Integrace vývojových prostředků AVG do prostředí Visual Studio 2005
Vedoucí/školitel VŠKP: Kočí Radek, Ing., Ph.D.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Bakalářská práce pojednává o vytváření malé aplikace zvané add-in pro Visual Studio. Add-in poskytuje vývojovému prostředí několik nových vývojových prostředků AVG. První nástroj nabízí podporu vizualizérů pro debugger Visual Studia. Druhou schopností add-inu je Editor a Manažer Code Snippetů pro jazyk C++. A posledním nástrojem je integrace Programové dokumentace AVG jako help do prostředí Visual Studia. Všechny tyto nástroje spolupracují s firemním serverem.

Klíčová slova

AVG, Microsft, Visual Studio, vývojové prostředí, IDE, add-in, debugger, vizualizéry, Code Snippets, Doxygen, .NET, C#, Python

Abstract

Bachelor's thesis treats of creating small application called add-in for Visual Studio 2005. Add-in provides some new AVG development tools into IDE. First tool offers support for new Visual Studio Debugger's visualizers. Second add-in's ability is Editor and Manager C++ Code Snipets. And the last tool is integration AVG Program documentation as a help in Visual Studio. All these tools cooperate with firm's server.

Keywords

AVG, Microsoft, Visual Studio, development environment, IDE, add-in, debugger, visualizers, Code Snippets, Doxygen, .NET, C#, Python

Citace

Jakub Rajman: Integrace vývojových prostředků AVG do prostředí Visual Studio 2005, bakalářská práce, Brno, FIT VUT v Brně, 2008

Integrace vývojových prostředků AVG do prostředí Visual Studio 2005

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího.
Další informace mi poskytl David Makovský.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Rajman
1.5.2008

Poděkování

Rád bych poděkoval Davidu Makovskému za odbornou a technickou pomoc při specifikaci požadavků, návrhu a implementaci aplikace.
Dále bych rád poděkoval vedoucímu práce Ing. Radkovi Kočímu za poskytnuté rady pro vypracování bakalářské práce.

Obsah

Obsah....1
1 Úvod....3
2 Add-in pro Visual Studio 2005....4
2.1 Vývojové prostředí....4
2.2 Visual Studio 2005....5
2.3 Možnosti rozšíření VS 2005....6
2.4 Vývoj Add-inu....8
2.5 Návrh projektu....9
3 Struktura a návrh systému....10
3.1 .NET....11
3.2 C#....12
3.3 Python....12
4 Podpora Vizualizérů....14
4.1 Motivace....14
4.2 Implementace....14
4.2.1 Základní struktura pravidla....15
4.2.2 Příklad....16
4.3 Integrace do projektu....18
5 Podpora Code Snippets....20
5.1 Motivace....20
5.2 Implementace....21
5.2.1 Struktura XML souboru *.snippet....21
5.2.2 Příklad....22
5.3 Integrace do projektu....23
6 Dokumentace aplikace v Helpu....24
6.1 Motivace....24
6.2 Doxygen....25
6.2.1 JavaDoc....25
6.2.2 Psaní komentářů....25
6.3 Integrace do projektu....26
6.3.1 Doc-O-Matic Express....27
6.3.2 Generátor nápovědy na serveru....27
6.3.3 Klientská část v add-inu Visual Studia....27

7 Závěr....28
Literatura....29
Seznam příloh....30

1 Úvod

Programování je všedním chlebem každého IT specialisty. Ale tak jako hardwarová výbava běžných počítačů skočila mílovými kroky vývojem kus dopředu, tak i způsoby a styly programování ušly dlouhou cestu. Od prapůvodní tvorby programů v jazyce blízkém stroji, přes programování jednoduchých aplikací za použití textového editoru a překladače daného jazyka, až k vývoji složitých a náročných týmových projektů a softwarů.

Pryč jsou doby, kdy lví podíl na tvorbě nesl samotný proces programování, ale naopak se dnes klade mnohem větší důraz na samotnou přípravu, shromáždění potřebných informací a důmyslné rozvržení projektu. Jelikož se však ani dnes program sám nenapíše, probíhají stále čím dál větší snahy o usnadnění námahy při programovacím procesu. Počínaje vyšší mírou abstrakce nových jazyků, dostupností mnoha knihoven s již vytvořenými a použitelnými kusy kódu a zdokonalenými vývojovými prostředími, podporujícími mnoho užitečných funkcí konče. A právě touto problematikou se zabývá má práce.

Společnost AVG Technologies vyvíjí své aplikace zabezpečovacích systémů AVG v prostředí Microsoft Visual Studio 2005 a mou snahou bylo vnést do tohoto prostředí nové funkce usnadňující programování. Vytvořil jsem tudíž jednoduchou aplikaci Add-in, která se po nainstalování navázala přímo na Visual Studio a obohatila ho hned několika novými funkcemi.

První funkcí je vylepšená podpora vizualizérů, nástrojů, které zdokonalují schopnosti integrovaného debuggeru při ladícím procesu.

Neméně užitečným rozšířením je také podpora Code Snippets (úryvků kódů) pro programování v jazyce C++. Moderní přístupy k programování již nenutí vývojového inženýra psát složitě kód písmeno po písmeni či slovo po slově. Naopak umožňuje automatické doplňování rozepsaného slova či řádku. S mým rozšířením si však programátor při psaní jednoduchým způsobem vloží přímo hotový předvytvořený blok kódu, se kterým může nadále pracovat.

Poslední schopností aplikace v sobě skrývá integrování programové dokumentace, která je vygenerována z komentářů zdrojových kódů, přímo do nápovědy prostředí Visual Studio. Tento nástroj velmi usnadňuje týmovou práci, kdy programátor používající konstrukci vytvořenou jiným programátorem. Nemusí složitě zkoumat a luštit napsanou konstrukci v kódu, ale ihned a snadno vyčte v nápovědě její popis.

2 Add-in pro Visual Studio 2005

Add-in neboli rozšíření je malá aplikace, po jejímž spuštění či nainstalování dojde k obohacení většího a komplexnějšího softwaru o nové funkce a schopnosti. Mnoho softwarových firem vyvíjí své aplikace s možnostmi jejich následného rozšíření. Jedná se o důmyslný tah, který umožňuje nejen přidávat nové vlastnosti samotné aplikaci, ale i spolupracovat s jiným softwarem. Jako příklad můžeme nahlédnout do vlastností dnešních webových prohlížečů. Zde si mnohé multimediální programy integrují do prohlížeče nástroje pro přehrávání multimediálních souborů. Nebo naopak jiné síťové aplikace umožňují webový prohlížeč vybavit lepším nástrojem pro stahování. A dokonce i zabezpečovací software obohatí prohlížeč nástrojem pro automatickou kontrolu obsahu stahovaných dat.

2.1 Vývojové prostředí

Jistě jste již setkali s označením IDE, což je zkratka pro anglický výraz Integrated Development Environment, neboli vývojové prostředí (viz [3]). Jak již název napovídá, jedná se o software integrující v sobě vývojové prostředky pro tvorbu programů, ať už jednoduchých či náročných aplikací.

Základem IDE je editor zdrojového kódu, který disponuje schopností zvýrazňovat syntaxi daného jazyka. Při editaci tak hned snadno poznáte, kdy se jedná o jméno proměnné, makro, klíčové slovo, komentář nebo snad řetězec ohraničený uvozovkami. Zvýraznění syntaxe je přímo šité na míru danému jazyku. Mnohé editory umožňují i nastavení barev a pozadí dané syntaxe, a tak si můžete upravit editor tak, aby se jeho užívání pro vás stalo co nejpříjemnějším.

Vedle editoru je v IDE integrován rovnou i překladač či interpret daného jazyka. Pryč je tak chvíle, kdy program překládáte v příkazové řádce za použití složitých přepínačů pro danou kompilaci. V IDE si vše nastavíte předem a pak už jen jednoduchým stisknutím tlačítka, či klávesové zkratky kompilaci aktivujete.

Debugger je povětšinou další podstatnou součástí IDE. Jedná se o rozmanitý ladící nastroj, který slouží k nalézání chyb v programu. Umožňuje tak programátorovi po zkompilování aplikace krokovat její běh příkaz po příkazu. Vše přitom programátor sleduje ve zdrojovém kódu, a tudíž přesně ví, kde se běh jeho aplikace právě nachází. Pokročilejší debuggery umožňují zobrazit i obsah proměnných v danou chvíli, nastavovat záchytné body běhu a mnohé jiné schopnosti. Právě díky integraci debuggeru se stává IDE profesionálním nástrojem pro tvorbu programů a velmi schopným pomocníkem.

Složitější vývojová prostředí disponují i mnohými jinými funkcemi, jako například vizuálním návrhářem grafického rozhraní aplikace, či nástrojem pro tvorbu a správu projektů, který je velice schopným pomocníkem při týmové práci.

Vývojové prostředí může být určeno pro jeden konkrétní programovací jazyk, nebo může obsahovat nástroje pro více jazyků. Hlavní podstatou je rozhodně editor, umožňující spolupracovat s různými syntaxemi, nebo jistá řádka kompilátorů či interpretů. Jednou ze schopností kompilátorů je i podpora několika hardwarových architektur.

Vývojové prostředí obnáší vedle svých nespočetných schopností jednu zásadní nevýhodu, kterou je vysoká hardwarová náročnost. Tento zápor je jeden z hlavních důvodů sporu mezi přívrženci a odpůrci vývojových prostředí. Při tvorbě jednoduchého skriptu si jistě vystačíte třeba jen s poznámkovým blokem a překladačem v příkazové řádce, ale při vývoji náročného a složitého softwaru se pro vás vývojové prostředí stane nepostradatelným pomocníkem a rádcem.

Mezi nejznámější reprezentanty vývojích prostředí patří:

- Object Pascal: Delphi, Kylix
- Java: NetBeans, Eclipse, JBuilder
- C a C++: Visual Studio, C++ Builder, Turbo C, Code::Blocks, Dev-C++
- Specializované: CodeWarrior, DJGPP
- S podporou více jazyků: Visual Studio, Kdevelop, Eclipse

2.2 Visual Studio 2005

Jak už bylo zmíněno výše, jedná se o vývojové prostředí. Visual Studio (viz [4]) bylo původně určeno pro vývoj a tvorbu aplikací využívajících produktové řady Microsoft Windows. Původním podporovaným jazykem byl tehdy Visual Basic. Později přibyly jazyky Visual C++ a Visual J++.

Hlavní a stěžejní událostí pro Visual Studio bylo vytvoření platformy .NET framework a jejího zakomponování do tohoto prostředí. Tímto směrem se nadále začaly ubírat cesty vývoje. Visual Studio 2005 již podporuje hned několik jazyků, mezi které patří hlavně Visual C++, Visual C#, Visual J# a Visual Basic. Kvalitní podpora .NET frameworku byla zachována, ba dokonce obohacena o .NET 2.0 verzi. Podstatnou výhodou této podpory je propracovaná Intellisence, což je nástroj, který je schopen za běhu studovat napsaný kód a automaticky tak doplňovat a nabízet výběry při editaci. Programátor si už nemusí pamatovat všechny metody, které daná třída nabízí, ale ve chvíli, kdy je chce použít, editor mu je sám nabídne.

Visual Studio 2005 je interně označován jako Visual Studio 8. Osmičková verze však již má dnes svého následníka ve verzi Visual Studio 2008 (někdy také označovaná jako devítková verze). Nová verze přináší novinky ve formě podpory platformy .NET framework verze 3.5 a samozřejmě také mnohé pokročilé vývojové a ladící nástroje.

2.3 Možnosti rozšíření VS 2005

Visual studio je nejen světově uznávané vývojové prostředí, ale jde mnohem dál. Toto IDE je také otevřená a rozšířitelná platforma. Například v případě, že se vytvoří úplně nový programovací jazyk, tým vývojářů má možnost přidat podporu pro tento jazyk do prostředí Visual Studia. Tato podpora může zahrnovat jak schopnost editovat kód v daném jazyce (ať už prostřednicím textového nebo grafického designéru), tak samotný překlad a kompilaci i přesto, že Visual Studio ve výchozím stavu tento jazyk nezná. Hlavním faktorem umožňujícím vytváření rozšíření do již stávajícího prostředí je Visual Studio 2005 SDK.

Software development kit (SDK) je sadou vývojových nástrojů, umožňujících softwarovým inženýrům vytvářet aplikace pro daný konkrétní softwarový balíček, softwarový framework, hardwarovou platformu, operační systém, či třeba herní konzoli. Může se jednat buďto o jednoduché aplikační rozhraní či speciální programovací jazyk, nebo může SDK zahrnovat důmyslný hardware pro komunikaci s vestavěným systémem. Hlavní nástroje, které slouží pro tvorbu nových aplikací a zahrnující překladače či další případné nástroje, bývají obvykle integrovány do vývojového prostředí. Software development kit často také zahrnuje kvalitní dokumentaci, příklady kódů či jiné technické poznámky. Tento nástroj bývá obvykle k dostání u výrobců daného software či hardware, pro které je tento kit určen. Často bývá i ke stažení přímo na internetu. Při vývoji aplikací pomocí SDK je nutné předem řádně nastudovat licenční práva k danému softwaru.

Není divu, že i Microsoft přichází se svými Software development kity pro možná rozšíření svých aplikací. Jistý prim hraje samosebou řada operačních systémů Microsoft Windows. Tyto vývojové soupravy v sobě snoubí knihovny, příklady, dokumentaci a nástroje využívající aplikační programové rozhraní. Jedním z těchto nástrojů je právě i Visual Studio 2005 SDK, přinášející podporu rozšíření tohoto vývojového prostředí. Podrobnosti v [5].

Visual Studio 2005 SDK nabízí 3 úrovně rozšířitelnosti pro vývojové prostředí:

Makra jsou nejjednodušším způsobem, jak rozšířit prostředí Visual Studia. Vývojové prostředí zahrnuje přímo IDE pro makra, skrz která je možno vyvíjet makra napsaná v jazyce Visual Basic.NET. Prostředí pro tvorbu maker vychází přímo z prostředí Visual Studia a je mu tedy

velmi podobné. Pomocí napsaných maker dosáhneme vytváření podprogramů či funkcí, které lze třeba napojit na položky v menu nebo v nástrojích k tlačítkům Visual Studia. Programátorovi to tak pomůže vytvářet různé úlohy k automatizování běžných akcí ve Studiu. Ačkoliv je psaní maker velmi jednoduché, jejich rozsah použitelnosti je dosti limitovaný. Proto s nimi nelze přistupovat k interním činnostem Visual Studia. Nelze tedy například vytvářet nová okna nástrojů. Vytváření nových nástrojů formou maker s sebou nese ještě jednu nevýhodu. Všechny zdrojové kódy maker nejsou kompilovány, a tudíž distribujete-li makra, distribujete volně otevřené a přístupné kódy.

Add-iny jako další prostředek pro rozšíření funkcí Visual Studia mohou stejně jako makra také zautomatizovat některé akce vývojového prostředí. Výhoda Add-inů vzhledem k makrům spočívá v tom, že mohou být psány v libovolném jazyce a kompilovány. Ve výsledné fázi tedy distribuovány formou binárních souborů. Add-iny jdou však mnohem dál. Při programování add-inu získáváte přístup k objektu zvaném DTE, což je kořenový objekt reprezentující Visual Studio. Skrze tento objekt je možno přistupovat k jednotlivým částem studia a obohatit tak jeho funkčnost mnohem markantněji než v případě maker.

Visual Studio Packages (balíčky) jsou posledním možným způsobem rozšíření. Stejně jako add-iny jsou napsány v programovacím jazyce, následně kompilovány a distribuovány formou binárních dat. Použitelným jazykem pro tvorbu VS balíčků obvykle bývá C++.NET či C#.NET. Podporován je i VB.NET, avšak pro něj není vytvořen průvodce pro tvorbu balíčků. Na rozdíl od add-inů jsou balíčky integrovány přímo do prostředí Visual Studia a nahrávány jako jeho součást. Jejich výhoda spočívá v možném přístupu přímo k funkcím samotného jádra IDE, což vede k větším schopnostem a síle oproti použití add-inů.

Visual Studio 2005 SDK je primárně určeno pro tvorbu VS balíčků. Své uplatnění sehrává i při tvorbě Add-inů díky tomu, že přichází s tzv. Experimental Hive během, nebo-li možností spustit kopii Visual Studia v úrovni připravené pro experimentální účely. Vytvořený add-in lze tak snadno zkompilovat, spustit a ladit v této úrovni, aniž by došlo k zásahům přímo do samotného hlavního IDE. Experimentální běh je přesto naprosto totožný s hlavním studiem, přebírá od něj i veškeré uživatelovo nastavení, ale na rozdíl od hlavního IDE, lze v tomto módu všechny provedené změny navrátit zpět do původní podoby.

2.4 Vývoj Add-inu

Pro tvorbu add-inů obsahuje Visual Studio jednoduchého průvodce (viz [6]). Průvodce lze spustit standardním způsobem tak, jako bychom vytvářeli nový projekt.Zde však místo klasického projektu, jako např. konzolové aplikace v jazyce C++, lze vybrat z dalších možných projektů právě Visual Studio Add-in. Průvodce nás posléze provede stručnými volbami. Tady lze určit název aplikace, vytvořit doplňující informace k add-inu, které se budou zobrazovat v klasickém help menu Visual studia, a v neposlední řadě zde můžeme také vybrat jazyk, ve kterém lze add-in vytvářet. Z nabídky jsou nám doporučovány jazyky Visual C#, Visual J#, Visual Basic, Visual C++. Všechny tyto jazyky využívají framework .NET.

Po dokončení průvodce se již před programátorem objeví otevřený projekt nového add-inu, který lze ihned zkompilovat a spustit. Samozřejmě tento projekt neumí nic víc než pouhé připojení ke stávajícímu Visual Studiu, ale veškerá další funkcionalita je již na programátorovi. Základní strukturu add-inu tvoří třída Connect se svými metodami. Tělo těchto metod slouží k editaci a s jejich pomocí lze přidávat add-inu novou funkcionalitu. Jedná se o tyto metody:

OnConnection() - tato metoda se provede ve chvíli, kdy je addin nahrán
OnDisconection() - tato metoda se provede ve chvíli, kdy je addin uvolněn z paměti
OnAddInsUpdate() - tato metoda se provede ve chvíli, kdy dojde ke změně struktury add-inu
OnStartupComplete() - tato metoda se provede s každým spuštěním Visual Studia
OnBeginShutdown() - tato metoda se provede s vypnutím Visual Studia
QueryStatus() - tato metoda se provede s každou změnou dostupností interních příkazů
Exec() - tato metoda se provede s každým spuštěným příkazem

Nejpoužívanějšími metodami jsou určitě OnConnection(), ve které programátor vytvoří základní strukturu se všemi používanými příkazy, a metoda Exec(), kde naprogramuje samotnou obsluhu daných příkazů. Rozdíl mezi použitím metod OnConnection() a OnDisconection() oproti metodám OnStartupComplete() a OnBeginShutdown() je ve chvíli jejich volání. První dvě zmíněné se volají pouze v případě prvního použití add-inu, a pokud se zásadně nezmění struktura add-inu, jako například jeho úplné odstranění a znovupřipojení, nedojde k opětovnému volání těchto metod. Druhé dvě metody se naopak použijí přesně s každým spuštěním či skončením hostující aplikace, v našem případě Visual Studia.

Uvnitř třídy má programátor přístup ke dvěma podstatným objektům. Prvním z nich je _applicationObject. Tento objekt reprezentuje samotné Visual Studio. Díky němu získáváme přístup

k funkcím, objektům a informacím o hostujícím prostředí. Druhým objektem je _addinInstance, který představuje instanci samotného add-inu.

Po zkompilování add-inu dojde k vytvoření několika souborů. Soubor s příponou *.AddIn je soubor ve formátu xml informující hostující aplikaci o existenci add-inu. Další soubory již mají standardně používanou příponu *.dll a jedná se o binární data samotného add-inu. Jiné případné soubory mohou mít příponu *.pdb a obsahovat ladící informace. Tyto soubory jsou používány pouze během procesu vývoje a rozhodně nejsou nezbytné pro distribuci aplikace. Pro použití add-inu stačí zmíněné soubory zkopírovat do adresáře Addins umístěného v podadresářové struktuře uživatelovy Application Data.

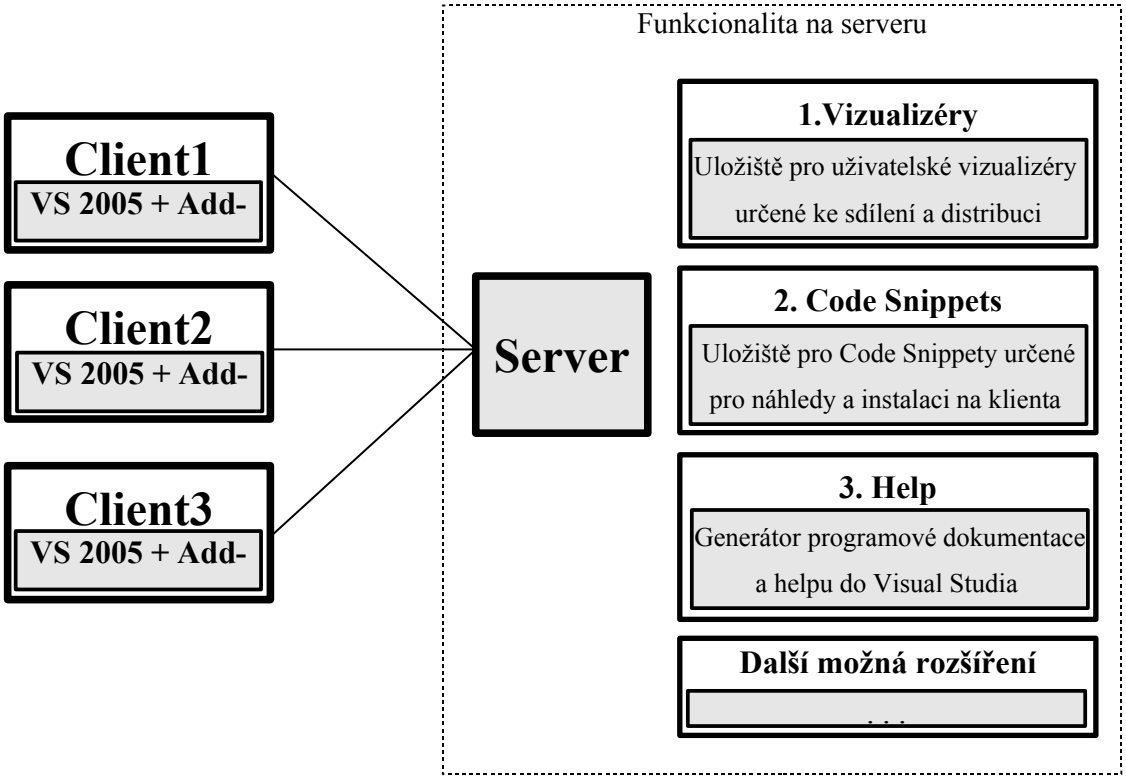
2.5 Návrh projektu

Pro tvorbu projektu byl vybrán způsob rozšíření Visual Studia metodou vytvoření add-inu, který přináší dostatek schopností pro firmou požadovaná rozšíření. Aplikace pro uživatele vypadá jako jednoduchý balíček, který uživatel standardním, pro něj přirozeným způsobem nainstaluje do svého prostředí. Ve Visual Studiu se objeví nová podnabídka v hlavní nabídce „Tools“ s nástroji, které add-in přináší. Základní nastavení těchto nástrojů lze provádět v nabídce „Options“ a zde v její podsekci věnované add-inu. Pomocí tlačítek nabídce „Tools“ je možno provádět buďto přímo integrované funkce, či otevírat nová okna v add-inu obsažená.

3 Struktura a návrh systému

Aplikace add-in ve Visual Studiu může do tohoto prostředí programátorovi přinášet mnohé užitečné funkce a schopnosti. Velice silným pomocníkem se však stává ve chvíli, kdy je umožněno programátorům tyto nové funkce a schopnosti sdílet v rámci firmy. A právě tímto směrem se ubírala má práce.

Vyvíjený sysém názorně vysvětluje následující obrázek.



Návrh celého systému jsem rozdělil na dva samostatně oddělené bloky. První část využívá firemní server. Na serveru jsou uloženy jednak instalační balíky aplikací sdílených ve firmě, či mnohé další používané nástroje. Avšak nejpodstatnější pro mne byla existence všech zdrojových kódů vyvíjené aplikace na tomto serveru. Tyto zdrojové kódy jsou používány v jedné z funkcí add-inu, která generuje programovou dokumentaci. Zde na serveru bude integrována základní funkcionalita následně použitelná v add-inu. Jedná se o již výše zmíněné generování heplu k vyvíjené aplikaci, či shromažďování a sdílení souborů s vizualizéry a Code Snippety. Bude-li add-in obohacen o případné další funkce, i jejich funkcionalita může být zaintegrována na serveru. Serverová část není implementována jako samostatná aplikace, ale jako sada jednoduchých skriptů využívajících

schopnosti serveru a obsluhující aplikace používané pro vytváření nástrojů pro add-in. O spouštění těchto skriptů se stará administrátor.

Druhý blok je tvořen add-inem. Tento add-in si může nainstalovat každý programátor na svou pracovní stanici. Po zadání přístupových cest k různým umístěním na serveru se může programátor dostat ke zdrojům používaných v add-inu. Bude moci tak například kdykoliv stáhnout nejaktuálnější nápovědu k vyvíjené aplikaci vygenerovanou na serveru a integrovat ji do svého vývojového prostředí na pracovní stanici.

Pro implementaci celého systému bylo použito několika technologií. Pro tvorbu samotného add-inu do Visual Studia bylo nutné zvolit programovací jazyk pracující s platformou .NET framework. Z nabízených možných .NET jazyků jsem si vybral C#, který byl svou syntaxí dosti podobný jazykům C a C++, se kterými jsem se do té doby setkal. Pro funkcionalitu pracující na serveru jsem použil skriptovací jazyk Python, jehož interpret byl již součástí aplikací nainstalovaných na serveru.

3.1 .NET

V roce 2000 byla oficiálně uvedena firmou Microsoft platforma .NET jako klíčový produkt, jehož rozvoj a propagace je součástí dlouhodobé strategie firmy. Platforma .NET byla primárně určená pro vývoj aplikací pro systémy Windows. K vývoji platformy .NET vedly tyto důvody:

- nekompatibilita programovacích jazyků a jejich obtížná spolupráce s knihovnami
- vysoká chybovost aplikací (chyby v konverzi datových typů, nestabilní práce s pamětí)
- problémy s verzemi knihoven (více verzí knihoven a jejich nekompatibilita)
- zastaralý a nepřehledný způsob vývoje dosavadních aplikací

Tyto problémy spolehlivě řeší platforma .NET a její výhody plynou z principu řízených běhových prostředí, na kterém je platforma založena.

Většina aplikací vytvořených v běžně používaných programovacích jazycích jako je C++, Delphi, Visual Basic jsou po kompilaci určeny pouze pro jednu danou architekturu. To je způsobeno tím, že zdrojové kódy jsou při kompilaci přímo převedeny do kódu stroje. Platforma .NET na to jde jiným způsobem. Zde se při kompilaci zdrojové kódy přeloží do tzv. mezikódu a až na výsledné architektuře, kde jsou spuštěny, dojde k překladu do strojového kódu. Stejného principu využívá také jazyk Java firmy Sun Microsystems.

Mezikód se nazývá MSIL (zkratka pro Microsoft Intermediate Language) a tento jazyk relativních adres je spouštěn klíčovou součástí .NET frameworku, pojmenovanou CLR (Common

Language Runtime neboli společné běhové prostředí). Oblíbenou součástí CLR u programátorů je existence Garbage Collectoru. Tato sada složitých algoritmů se stará o uvolňování nepotřebných programových objektů z paměti, o což se již vývojáři nemusejí starat, a tak odpadá riziko již zmíněné nekorektní práce s pamětí, která ve většině situací končí pádem aplikace.

Další důležitou vlastností platformy .NET framework je CLS (zkratka pro Common Language Specification neboli Společná jazyková specifikace) a s ní související i CTS (zkratka pro Common Type System neboli Společný typový systém). Výsledkem použití CLS a CTS je rovnocennost programovacích jazyků. To znamená, že pro vývoj aplikací v .NET frameworku je možné použít jeden z několika programovacích jazyků vyšší úrovně. Primárně byl pro tuto platformu vytvořen jazyk C# a Visual Basic.NET, ale později došlo k rozšíření na více jazyků, jako je J#, Managed C++, a jejich řady je možné stále rozšiřovat.

Platforma .NET framework je převážně určena pro vývoj menších aplikací. Pro její větší náročnost na hardware a nutnost mít pro spuštění vytvořeného programu na cílové architektuře nainstalován překladač CLR, je pro vývoj větších aplikací nevhodná. Zde se doporučuje použít jeden z běžných jazyků a optimalizovat vývoj cíleně na danou architekturu. V .NET frameworku lze vyvíjet běžné konzolové aplikace, ale daleko zajímavější jsou aplikace s využitím knihoven Windows.Forms, interně využívající Microsoft Win32 API. Dalším odvětvím jsou webové aplikace vytvořené v ASP .NET, který nahrazuje zastaralé ASP 2.0 a posouvá tak tvorbu dynamických webů o pořádný kus dále.

3.2 C#

C Sharp (viz [1]) je vysoce úrovňový objektově orientovaný programovací jazyk, spadající mezi jazyky platformy .NET framework. Tento jazyk byl postaven na základech jazyka C++ a Java. Lze jej využít pro vývoj jednoduchých konzolových aplikací a aplikací využívající grafické knihovny Windows. Dále je přizpůsoben k tvorbě databázových programů či webových aplikací. V případě vývoje aplikací v prostředí Visual Studio podporující intellisense nabízí vysoký komfort pro programování.

3.3 Python

Python (viz [2]) je dynamický interpretovaný jazyk. Vznikl v roce 1990 v Nizozemsku jako následník tehdejšího jazyka ABC. Jeho tvůrcem je Guido Van Rossum, který se i nadále podílí na jeho vývoji. Od té doby je však vyvíjen jako open source projekt, který je nabízen zdarma instalačními balíčky pro většinu platforem. Někdy bývá zařazován mezi takzvané skriptovací jazyky, ale jeho možnosti jsou větší. Python je navržen tak, aby umožňoval tvorbu rozsáhlých,

plnohodnotných aplikací (včetně grafického uživatelského rozhraní).Python je široce používán na mnoha místech, kde je potřeba přehledný a snadno spravovatelný kód. Je kupříkladu využíván organizací NASA pro uživatelské rozhraní systému řídícího lety raketoplánů, či aplikačním serverem Zope pro originální implementace protokolu BitTorrent.

Syntaxe pythonu je založena na odsazování. Nenajdeme tedy zde žádné značky otevírající či uzavírající blok kódu tak, jak jsme zvyklí z běžných jazyků. Tento způsob může být pro začátečníka dosti netradiční. Taktéž pro něj může být dosti matoucí, že se jedná o beztypový jazyk.

Python je hybridní jazyk, což znamená, že může využívat ať už objektové orientované paradigma, procedurální, či v omezené míře dokonce funkcionální paradigma. Jeho základní struktura je velmi blízká většině typický jazyků, a tak se lze s tímto jazykem sžít během několika málo hodin. Moduly jazyka napsané v Pythonu se tzv. kompilují. Ale v podstatě jde o kompresi zdrojového kódu do formy, která se rychleji načte do paměti.

Jak už bylo řečeno, Python je interpretovaný jazyk, což nepřináší pouze nevýhodu v podobě pomalejšího běhu, ale i mnohé výhody. Může se v podstatě jednat o předpoklad použitého objektového modelu a také nám vývoj velice usnadní možnost spustit interpret interaktivně. To znamená, že můžeme zadávat příkazy a ihned vidět jejich výsledky.

4 Podpora Vizualizérů

4.1 Motivace

Vývojové prostředí Visual Studio 2005 je vybaveno debuggerem pro ladění aplikací, který stejně jako mnohé jiné debuggery disponuje schopností zobrazovat obsahy proměnných. Tento nástroj je velmi užitečný zobrazujeme-li obsahy jednoduchých typů, struktur či tříd. Ale ve chvíli, kdy je struktura třídy rozmanitější a komplikovanější, stává se zobrazení obsahu její instance dosti nepřehledné.

Představme si pro příklad již standardní strukturu binárního stromu, tvořenou jednotlivými uzly s ukazateli na levý a pravý prvek. Tuto strukturu naplníme 100 uzly a poté budeme chtít obsah této struktury zobrazit debuggerem. Debugger nám nejdříve zobrazí kořenový uzel a poté pomocí roletové nabídky následující levý a pravý uzel. U těchto uzlů se celý proces zobrazení opakuje a při počtu 100 prvků se zobrazení těch nejspodnějších může stát noční můrou. Visual Studio však nabízí nástroj, kterým si celý tento proces můžeme ulehčit a zobrazit celý obsah stromu jako jednu přehlednou tabulku. Debugger pak sám projde celou strukturu metodou in-order (levý – aktuální – pravý) a obsah uzlů zobrazí do jedné úhledné tabulky. Nástroje pro zobrazení obsahu proměnných se nazývají Vizualizéry.

4.2 Implementace

Vizualizace, nebo-li zobrazování obsahu datových struktur, se řídí jednoduchými pravidly. Celá tato pravidla vydají na jeden samostatný skriptovací jazyk, který Microsoft používá. Základem pro tvorbu vizualizérů je tedy naučit se pravidla pro jejich zápis a vědomí, jak toto aplikovat pro prostředí Visual Studia.

V adresáři s nainstalovaným prostředím najdeme strukturu ..\Common7\Packages\Debugger, v níž se nachází soubor Autoexp.dat. Tento soubor obsahuje všechna pravidla, používaná Visual Studiem, a jen jeho editací dosáhneme nových Vizualizérů ve svém vývojovém prostředí. Tento textový soubor je načítán s každým spuštěním Studia, a je tudíž nutné po každé jeho editaci restartovat aplikaci, abychom uplatnili námi nově vytvořená pravidla. Nová pravidla se připisují vždy na konec, jelikož aplikace čte soubor Autoexp.dat od začátku, a nalezne-li pravidlo pro třídu, která již pravidlo má, použije toto novější pravidlo. V použití pravidel také platí jistá hiearchie. Nenajde-li Studio pro danou třídu její konkrétní pravidlo, použije obecnější. A dále platí, že jsou-li prvky třídy také typem, pro nějž existuje pravidlo vizualizéru, toto pravidlo se použije. Podrobnosti v [7].

4.2.1 Základní struktura pravidla

Pravidlo lze psát vždy pouze pro jednu, a to konkrétní strukturu či třídu. Nelze tedy aplikovat pravidla pro více stejných či podobných tříd. Zápis se totiž ztotožňuje se jménem třídy, nikoliv s jeho strukturou. Základní struktura pro popis jedné třídy se skládá ze čtyř částí definující, kde se dané pravidlo bude aplikovat.

```
Typename {
      preview
      {
            preview-string-expressinon
      }
      stringview
      {
            text-vizualizer-expression
      }
      children
      {
            expanded-contents-expressions
      }
}
```

Popis:

1. **Typename** – úvodní část definující typ či třídu, které se týká následující popis
2. **Preview** – část preview definuje jednořádkový výraz, který se zobrazí ve Visual Studiu v okně Watch, QuickWatch nebo Command Window
3. **Stringview** – část stringview definuje řetězec, který bude zobrazen v Text, XML nebo HTML vizualizéru
4. **Children** – část children definuje popis zobrazení hiearchicky strukturovaného vizualizéru.

V našem vlastním vizualizéru není nutno uvádět všechny tyto části. Podstatné je hlavně uvést v sekci typename jméno třídy, které se následující popis bude týkat, a pak už jen uvést tu část, jejíž vizualizaci budeme chtít upravit. Zbylé části je možno vynechat. Visual Studio si pak danou část doplní samo buďto implicitně, nebo z podřízených použitých tříd a struktur.

Do těchto jednotlivých bloků pak již zapisujeme daná pravidla zobrazení formou výrazů. Vizualizace v části preview se uplatňuje pouze jednořádkově, obvykle jako hlavička dané třídy, sloužící pro rychlou a stručnou informaci o typu. Naopak sekce children popisuje hiearchickou

strukturu obsahu dané struktury při detailnějším prozkoumávání obsahu třídy. Část stringview se příliš nepoužívá. Své uplatnění nachází pouze v případě, že je obsah proměnné ve struktuře XML či HTML.

4.2.2 Příklad

Pro ukázku, jaké má použití vizualizérů výhody, jsem vybral již výše uvedený příklad v kapitole „motivace“ se stromovou strukturou.

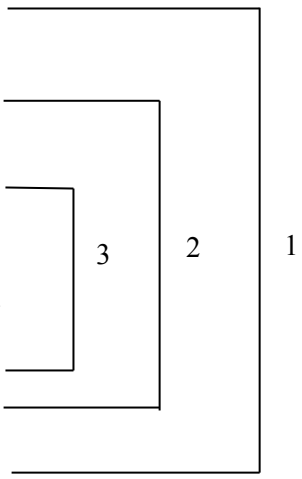
4.2.2.1 Kód struktury „strom“ v C++

```
struct TtreeNode
{
	TtreeNode *leftNode, *rightNode;
	int value;
};

struct Ttree
{
	TtreeNode *rootNode;
	int size;
};
```

4.2.2.2 Kód vizualizéru ukázkové struktury

```
Ttree {
	children
	(
		#tree
		(
			size : $c.size,
			left : leftNode,
			head : $c.rootNode,
			right : rightNode
		)
	)
}
```



Jak je na ukázkovém kódu vizualizéru vidět, vizualizér nese přímo jméno C++ struktury „Ttree“, kterou popisuje. Ohraničená část (1) znázorňuje popis části „children“, čili vizualizér obsahuje pravidlo pouze pro detailnější strukturované zobrazení obsahu proměnné. V sekci children se nachází pravidlo „#tree“ (2). Tento blok udává, že se bude jednat o zobrazení struktury strom. Uvnitř tohoto bloku jsou jednotlivé vlastnosti pravidla strom v závislosti na popisované C++ struktuře „Ttree“ (3). Řádek „size“ udává celkový počet prvků ve struktuře, řádek „head“ popisuje kořenový uzel, řádky „left“ a „right“ obsahují ukazatele na následující uzly.

Jak je možno vidět, v pravidle se několikrát objevuje zápis „$c“, který zastupuje C++ strukturu o které celé pravidlo pojednává. Při snaze vytvářet vlastní vizualizéry pro třídy AVG jsem došel k závěru, že syntaxe zápisu pravidel je dosti nešťástná.

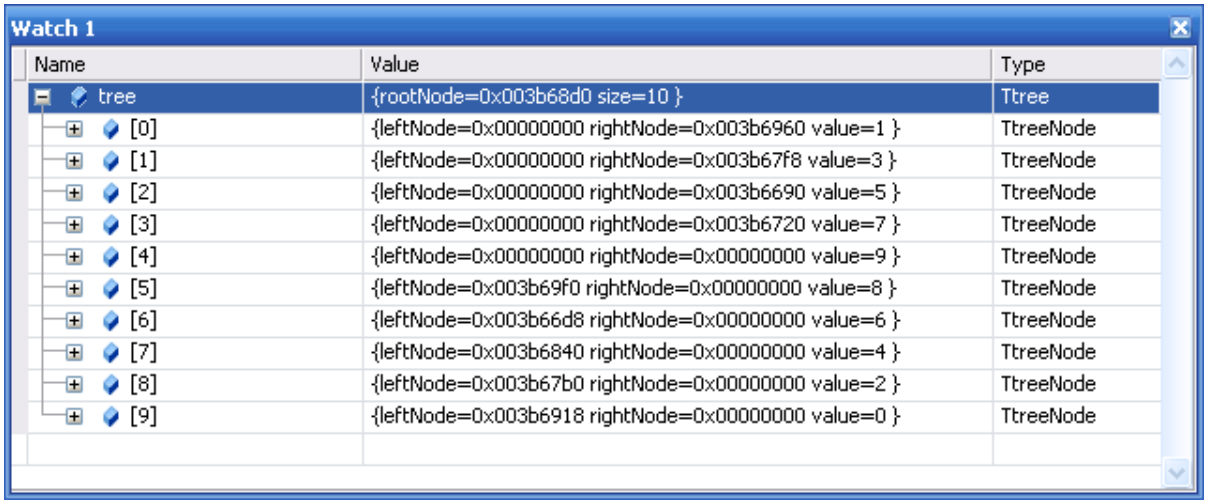
Strukturu v naší aplikaci naplníme hodnotami od 1 do 10 a provedeme náhled na obsah proměnné v okně „Watch Window“.

4.2.2.3 Náhled bez použití vizualizéru

Watch 1

Name	Value	Type
tree	{rootNode=0x003b68d0 size=10 }	Ttree
rootNode	0x003b68d0 {leftNode=0x003b6918 rightNode=0x00000000 value=0 }	TtreeNode *
leftNode	0x003b6918 {leftNode=0x00000000 rightNode=0x003b6960 value=1 }	TtreeNode *
leftNode	0x00000000 {leftNode=??? rightNode=??? value=??? }	TtreeNode *
rightNode	0x003b6960 {leftNode=0x003b67b0 rightNode=0x00000000 value=2 }	TtreeNode *
leftNode	0x003b67b0 {leftNode=0x00000000 rightNode=0x003b67f8 value=3 }	TtreeNode *
leftNode	0x00000000 {leftNode=??? rightNode=??? value=??? }	TtreeNode *
rightNode	0x003b67f8 {leftNode=0x003b6840 rightNode=0x00000000 value=4 }	TtreeNode *
leftNode	0x003b6840 {leftNode=0x00000000 rightNode=0x003b6690 value=5 }	TtreeNode *
rightNode	0x00000000 {leftNode=??? rightNode=??? value=??? }	TtreeNode *
value	4	int
value	3	int
rightNode	0x00000000 {leftNode=??? rightNode=??? value=??? }	TtreeNode *
value	2	int
value	1	int
rightNode	0x00000000 {leftNode=??? rightNode=??? value=??? }	TtreeNode *
value	0	int
size	10	int

4.2.2.4 Náhled s použitím vizualizéru



Jak je vidět na ukázkách, je náhled na strukturu „strom“ bez použití ukázkového vizualizéru velice nepřehledný a dosti komplikovaný.

4.3 Integrace do projektu

Ač jsou vizualizéry velmi mocnou zbraní, disponují jednou nevýhodou. Tato nevýhoda tkví v jejich neuniverzálnosti. Jak už bylo řečeno výše, vizualizér popisuje vždy jednu danou konkrétní třídu, která je identifikována svým jménem. Tvorba vizualizérů je tak tzv. „šitá na míru“ zdrojovým kódům vyvíjené aplikace. Po krátké úvaze dojdeme k závěru, že tvorba vizualizérů pro vývoj rozsáhlejších aplikací jednou osobou je téměř nemožná. Jedním z důvodů pro toto tvrzení je, že programátor by musel znát všechny zdrojové kódy vyvíjené aplikace, a to plně dopodrobna. Což je u náročných projektů, jakým je aplikace AVG, na kterých pracuje celý tým odborníků a kde je projekt rozdělen do několika modulů, které opečovává vždy někdo jiný, zcela nepředstavitelné. Druhý důvod spočívá v nestabilním kódu vyvíjené aplikace. S každou zásadnější změnou kdekoliv v kódu by bylo nutné přepsat i vizualizér třídy, které by se tato změna týkala.

Tyto okolnosti vyústily v jednoduché řešení. Každý programátor, autor a opatrovník svých zdrojových kódů a v nich obsažených tříd si bude své vizualizéry tvořit sám. Mým záměrem tedy bylo obeznámit programátory s tvorbou a použitelností vizualizérů, a jelikož jsem při svém studiu vizualizérů nenarazil nikde na přívětivý manuál, natožpak manuál v češtině, sepsal jsem jej sám.

Věřím, že ve chvíli, kdy se jednotlivci ve vývojovém týmu dozví o této schopnosti Visual Studia, několik svých vlastních vizualizérů si napíší. Tyto vizualizéry pak bude možné poslat na pracovní server administrátorovi, který je už utřídí a připraví ke stahování. Jednou ze schopností mého add-inu pro Visual Studio je samotné stahování uživatelských vizualizérů ze serveru a následné

integrování do spuštěného Studia na programátorově počítači. Programátor už se pak nebude muset zaobírat tím, kde má nové vizualizéry stáhnout, kde je má uložit a jak vše aplikovat tak, aby mu nové vizualizéry řádně pracovaly na jeho počítači. Nemusí dokonce ani vědět, jak se vizualizéry tvoří, pokud je nepotřebuje vytvářet, ale pro pouhé použití stačí pouze znát, k čemu jsou. A pak už jen vědět na které tlačítko kliknout ve svém Visual Studiu a jeho add-in už provede vše za něj.

5 Podpora Code Snippets

5.1 Motivace

Každý programátor jistě při své práci používá již jednou vytvořený hotový kód. Opomeneme-li řádné modulární programování s členěním kódu do funkcí, metod, tříd a objektů, občas je zkrátka třeba přímo zkopírovat blok kódu. Může se jednat buďto o několika řádkovou posloupnost příkazů či snad složitý algoritmus plný cyklů a větvení. V tuto chvíli musí programátor nejdříve najít ve změti souborů místo ve své aplikaci, kde tento blok kódu použil, následně jej zkopírovat na nové místo a posléze editovat a přizpůsobit ho takto pro nové použití.

A právě zde přichází Visual Studio se svou schopností podpory Code Snippets. Jedná se o malé útržky kódu, které poskytují určitou předpřipravenou funkčnost, ať už se jedná o vyřešení nějaké časté úlohy nebo o pokročilejší věci typu refaktoringu (i ten je realizován pomocí Code Snippetů).

Uživatel programátor pak může libovolně vyvolat kontextovou nabídku pravým tlačítkem myši a zvolit volbu „Insert Snippet“. Pak už jen z nabízených možností vybere blok kódu, který právě potřebuje. Celý tento proces může samozřejmě ovládat i pomocí klávesových zkratek a takto jednoduchým a rychlým způsobem může obohatit svůj kód hned o několik řádků kódu obsahujících v sobě požadovaný algoritmus.

Další výhodou při použití Code Snippetů na rozdíl od běžného kopírování je jeho připravenost na následnou editaci. Po běžném kopírování musí programátor procházet vložený kód řádek po řádku, důkladně jej číst a hledat změny, které je potřeba provést pro přizpůsobení ke svému kódu. Pokud se tedy například v daném kopírovaném bloku kódu vyskytuje desetkrát použití proměnné „foo“, která se však v přeneseném kódu jmenuje „fii“, je nutné toto použití přepsat na všech deseti místech. S použitím Code Snippetu se tomuto řešení vyhneme, jelikož je předpřipraven tak, že místa, která je nutno vždy editovat podle potřeby, jsou zvýrazněny a při vkládání nového Code Snippetu snadno přístupná přes klávesu TAB. Pokud se stejně jako ve výše popsaném případě bude totožný výraz vyskytovat na více místech, zde jej stačí změnit jen při výskytu prvním.

Nesmíme opomenout ani možnost vkládaní Code Snippetů tzv. „Surrounded“. To znamená, že nejdříve si programátor napíše kousek svého kódu, poté označí a následně obklopí Code Snippetem, který vloží blok kódu před a kousek kódu za zvýrazněný kód. Typickým příkladem může být ošetřování kódu na odchytávání výjimek. Programátor nejprve napíše „bezohledně“ svůj kód, který při daných chybách může vyvolat výjimku. Blok kódu poté označí a nakonec jej obklopí Code Snippetem Try-Catch-Finally.

5.2 Implementace

Visual Studio 2005 přináší podporu Code Snippets pro většinu jazyků, ve kterých se dá v tomto prostředí vyvíjet aplikace. Výjimku však tvoří jazyk C++. Pro tento jazyk je nutno stáhnout a poté doinstalovat plugin „VSCodeSnippetsC++“, avšak i přesto je obsažených nabízených Code Snippetů velice poskrovnu. Kouzlo a uplatnění tohoto nástroje a jemu podobných spočívá až ve chvíli, kdy má programátor možnost vytvářet nové vlastní Code Snippety.

Code Snippety jsou v systému reprezentovány formou XML souborů s příponou *.snippet, kde každému Code Snippetu je určen vždy jeden soubor. Tyto soubory je možné už hotové stáhnout či editovat a v neposlední řadě také vytvořit. Pak už stačí jen soubory nahrát do přesně určených adresářů Visual Studia a toto prostředí už samo rozpozná nové soubory a informace z nich integruje do svého editoru. Pro vytváření nových Code Snippetů je třeba znát strukturu XML souboru.

5.2.1 Struktura XML souboru *.snippet

Struktura XML souboru pro reprezentaci Code Snippetu není nijak složitá. Skládá se ze tří částí:

1. **Hlavička XML souboru** – obsahuje nejnutnější informaci o verzi XML, kódování, jmený prostor a formát Code Snippetu. Hlavička XML souboru je popsána tagy <?*xml*?>, <*CodeSnippets*>, <*CodeSnippet*>

2. **Hlavička Code Snippetu** – obsahuje nejnutnější informace popisující Code Snippet uzavřené mezi tagy <*head*>.
   Jedná se o tyto informace:
   - název
   - klávesovou zkratku
   - jednoduchý popis Code Snippetu informující o jeho činnosti
   - jméno autora
   - určení, zda se jedná jen o obyčejný vkládací Code Snippet, či Code Snippet typu „surronded“

3. **Struktura Code Snippetu** – blok uzavřený mezi tagy <*Snippet*> je nejpodstatnější část celého XML souboru. Obsahuje jednak samotný kód, který se vloží do editoru po použití Code Snippetu. V kódu jsou také použity značky, ohraničené z obou stran symbolem „$“, kterými se definují místa možná k následné editaci po použití snippetu. Tyto proměnné, označující místa pro editaci, jsou také popsány v této sekci a nechybí jim samozřejmě pojmenování, krátká nápověda a implicitní hodnota.

5.2.2 Příklad

Jako příklad uvádím ukázkový kód Code Snippetu pro algoritmus cyklu „while“ pro jazyk C++. Zde je možno vidět přehledně všechny výše jmenované části.

```
<?xml version="1.0" encoding="utf-8" ?>
<CodeSnippets xmlns="http://schemas.microsoft.com/..../CodeSnippet">
      <CodeSnippet Format="1.0.0">
            <Header>
                  <Title>while</Title>
                  <Shortcut>while</Shortcut>
                  <Description>Code snippet for while loop</Description>
                  <Author>Microsoft Corporation</Author>
                  <SnippetTypes>
                        <SnippetType>Expansion</SnippetType>
                        <SnippetType>SurroundsWith</SnippetType>
                  </SnippetTypes>
            </Header>
            <Snippet>
                  <Declarations>
                        <Literal>
                              <ID>expression</ID>
                              <ToolTip>Expression to evaluate</ToolTip>
                              <Default>true</Default>
                        </Literal>
                  </Declarations>
                  <Code Language="cpp">
                        <![CDATA[
                              while ($expression$)
                              {
                                    $selected$ $end$
                              }
                        ]]>
                  </Code>
            </Snippet>
      </CodeSnippet>
</CodeSnippets>
```

5.3 Integrace do projektu

Visual Studio 2005 disponuje schopností vkládat Code Snippety do kódu (s použitím pluginu „VSCodeSnippetsC++“ dokonce i pro jazyk C++). Neovládá však žádnou schopnost vytvářet nové Code Snippety. A právě tady nachází své uplatnění add-in Visual Studia, který přichází hned se dvěma nástroji.

Prvním nástrojem je editor Code Snippetů. Tento editor slouží ke snadnému vytváření a editování XML souborů *.snippet. Programátor si zde nejdříve sepíše nebo zkopíruje blok kódu, který chce použít jako Code Snippet, dopíše detaily o názvu, klávesové zkratce, autorovi a krátký informační popis. Nakonec může přidávat či odebírat proměnné označující místa pro editaci v případě použití Code Snippetu. Může jim také měnit název, implicitní hodnotu, či jejich krátký popis. Tyto všechny schopnosti editor přináší v příjemném uživatelském prostředí. Programátor se tak už nebude muset učit a studovat strukturu XML souborů, ale jejich obsah si doslova nakliká.

Po sepsání všech informací o Code Snippetu může programátor svůj výtvor uložit. Zde je mu automaticky nabídnut adresář, ze kterého vývojové prostředí Visual Studia čerpá při hledání použitelných Code Snippetů. Uloží-li programátor svůj Code Snippet kamkoliv jinam, nebude mít možnost tento svůj výtvor používat ve své aplikaci. K uložení je možné zvolit ještě jednu destinaci, a tou je adresář, do kterého si Visual Studio nahrává své vlastní, implicitní, základní Code Snippety. V tomto případě se již musí jednat o znalého programátora, avšak tento způsob ukládání není doporučován.

Editor Code Snippetů je koncipován jako nástroj pracující s Code Snippety pouze na jedné klientské stanici. Nepřináší tedy žádnou podporu sdílení či šíření Code Snippetů mezi více uživateli a mezi více pracovními stanicemi. Podporu této funkce přináší druhý nástroj Manažer Code Snippetů. Ten je zamýšlen tak, že soubory s užitečnými či oblíbenými a použitelnými Code Snippety se posbírají od jednotlivých programátorů a nahrají na firemní server. Manažer Code Snippetů integrovaný jako add-in do Visual Studia má pak schopnost listovat mezi uloženými Code Snippety na serveru, prohlížet si jejich obsah a nakonec jedním stisknutím tlačítka vše nahrát k sobě na svou pracovní stanici. Opět bylo snahou vytvořit co nejjednodušší a co nejintuitivnější ovládání.

Editory i Manažer Code Snippetu jsou určeny pouze pro práci s Code Snippety pro jazyk C++. Tento jazyk dostačuje požadavkům firmy AVG Technologies, jelikož její aplikace je vyvíjena převážně právě v tomto jazyce. Při další práci na tvorbě add-inu by bylo možné jeho schopnosti rozšířit i na jiné jazyky, samozřejmě však jen na ty, které jsou podporovány prostředímVisual Studia.

6 Dokumentace aplikace v Helpu

6.1 Motivace

Ke správným návykům každého programátora patří řádné zdokumentování své provedené práce. Dobře sepsaná dokumentace přináší hned několik výhod. Opustí-li programátor svou vytvořenou aplikaci na nějakou dobu a poté se k ní navrátí s úmyslem přidat či pozměnit její funkci, množství času mu pak zabere studium struktury aplikace i přesto, že byl jejím tvůrcem. Při použití dokumentace, která je vytvořená tak, aby vytvářela určitý abstrakt a byla blízko lidskému chápání, se čas potřebný k prostudování struktury aplikace zkrátí na minimum.

Dokumentace aplikace nemusí být užitečná jen autorovi aplikace, ale může být potřebná hlavně v týmové spolupráci. Sepíše-li autor dobře dokumentaci svého produktu, vyhne se tak častým dotazům o způsobu používání svého kódu jinými programátory. Při práci ve firmě o několika desítkách programátorů, kterým je AVG Technologies, je třeba způsob struktury dokumentace sjednotit.

Pro tyto účely se používá generovaná dokumentace z popisných komentářů ve zdrojovém kódu. Na soubory obsahující kód, popsaný komentáři, se aplikuje nástroj, který rozumí struktuře daného jazyka a struktuře zapisování komentářů. Tento nástroj vygeneruje nápovědu v požadovaném formátu, ať už určenému pro tisk, či k uchování v elektronické podobě. Podle psychologického náhledu je pro člověka příjemnější tištěná podoba pro vstřebávání nových informací, avšak elektronická podoba nabízí většinou jistou interaktivitu, která usnadňuje a zrychluje vyhledávání konkrétních potřebných informací. V tištěné podobě je potřeba vždy mezi stránkami listovat a složitě při vyhledávání přeskakovat. Elektronická podoba se svou podporou všemožných odkazů toto listování urychluje a spolu se schopností navrácení zpět dopomáhá k udržování kontextu.

Oblíbeným a uznávaným nástrojem pro tvorbu programové dokumentace je aplikace Doxygen a mezi generované formáty patří HTML, LaTeX či PDF. Vytvořený add-in do Visual Studia jde však ještě dále a se svým rozšířením přináší novou schopnost. Má-li programátor vygenerovanou dokumentaci v elektronické podobě, má ji stále externě oddělenou od svého vlastního projektu. Narazí-li tedy při svém programování na problém a potřebuje poradit, nezbývá mu nic jiného než si otevřít dokumentaci a pomocí klíčových slov začít vyhledávat. Add-in integruje tuto dokumentaci přímo do nápovědy Visual Studia. Programátor tak nemusí navíc otevírat dokumentaci zvlášť odděleně od svého projektu, ale přímo v prostředí Visual Studia spustit klávesovou zkratkou help, který jej automaticky navede přímo k dokumentaci řešeného problému. Tento postup řešení problému je uživatelsky velice příjemný a urychlí práci.

6.2 Doxygen

Doxygen (viz [8]) je nástroj pro generování dokumentace ze zdrojových kódů. Zdrojové kódy musí obsahovat komentáře, které daný kód popisují a které jsou hlavním zdrojem informací pro vytvoření dokumentace. Podporované jazyky jsou C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, C#. Doxygen pro tyto jazyky obsahuje kvalitní parser, a syntaktický analyzátor, díky čemuž je schopný vygenerovat dokumentaci popisující množství skriptů, funkcí, tříd a proměnných spolu s jejich vlastnostmi a parametry. Volitelně pak lze například graficky zobrazovat závislosti mezi jednotlivými moduly či částmi kódu. Výstupní formáty generované dokumentace jsou HTML, LaTeX, RTF, XML, a manuálové stránky. Doxygen je multiplatformní a lze ho používat pod unixovými systémy, taktéž i pod Windows a Mac OS. Komentáře pro Doxygen musí dodržovat styl JavaDoc.

Doxygen je volně stažitelná aplikace a její licence spadá pod GNU General Public. Z pohledu uživatele se jedná o jednoduchou aplikaci, běžící v příkazové řádce bez grafického rozhraní. Ovládání je snadné. Stačí si jen vygenerovat šablonu konfiguračního souboru. Zde pak řádně nastavit námi požadované vlastnosti a tento soubor použít jako parametr příkazu spouštějící generátor. Mezi typické nastavitelné vlastnosti patří například cesta k souborům se zdrojovými kódy, přípona vybraných souborů, formát generované dokumentace spolu s jeho vlastnostmi a mnoho dalších.

6.2.1 JavaDoc

Sun Microsystems vydal softwarovou pomůcku (utilitu) pro automatické generování dokumentace programu. Co se velikosti týče, jde o malou konzolovou aplikaci, která umí procházet zdrojový kód programu v Jazyce Java, vyhledávat uvozené komentáře a ty posílat na výstup současně generovaného HTML souboru. Pojem JavaDoc neoznačuje jen utilitu pro generování dokumentace, ale také pravidla pro zápis komentářů. Tyto pravidla využívají i jiné generátory dokumentace, zejména pak Doxygen. Další informace v [9].

6.2.2 Psaní komentářů

Samotný zápis instrukcí pro JavaDoc, které se mají zpracovat, se provádí do speciálního komentáře začínající ho /** a uzavřeného */. Tento zápis se umisťuje před deklaraci třídy, metody, funkce, proměnné a dalších.

Uvnitř tohoto komentáře se pak nacházejí příkazy, nebo-li tagy, označující důležité popisy. Je možné použít dvojí způsob zápisu tagů. Buďto se zpětným lomítkem „\“, nebo se symbolem zavináče „@“ před názvem tagu. Po vybrání jednoho ze dvou zápisů je však doporučeno dodržovat tento jeden zvolený po celou dobu dokumentace. Některé příkazy jsou doplněny jedním nebo více argumenty.

Pro jejich použití slouží zápisy „<param>“ v případě jednoho parametru, či „(param)“, „{param}“, „[param]“ v případě více parametrů.

6.2.2.1 Nejpoužívanější tagy

\brief – tag označuje, že se jedná o základní informativní popis následného objektu
\class – tag se používá pro zvýraznění a označení názvu třídy
\code – uvnitř tohoto tagu je umístěn blok kódu
\param – označení parametru funkce či metody se provátí právě tímto tagem
\return – pro návratový typ metody či funkce slouží tento tag
\struct – tag pro zvýraznění, že se jedná o popis struktury obdobně jako u tagu \class
\typedef – tag pro označení popisu části „typedef“
\union – tag označuje popis struktury „union“
\var – tag pro označení popisu části „var“

Pro psaní komentářů podle pravidel dokumentace pro Doxygen existují desítky dalších použitelných tagů. Tyto tagy lze konzultovat v dokumentaci pro systém Doxygen. Zde jsem uvedl pouze nejpoužívanější tagy v projektech aplikace AVG.

6.3 Integrace do projektu

Po vytvoření programové dokumentace nástrojem Doxygen je třeba zajistit, aby tato dokumentace byla zaintegrovaná jako help do prostředí Visual Studia. Visual Studio 2005 však již používá jako nápovědu dokumenty ve verzi Microsoft Help 2.x. Tento formát byl vyvinut Microsoftem v roce 2001 primárně pro aplikaci Visual Studio.NET a knihovny MSDN. A u těchto systémů jeho uplatnění skončilo, jelikož i ve Windows XP se nadále používá formát Microsoft Help 1.x. Tento fakt vedl jistě k důvodu nízké podpory formátu Help 2.x, se kterým jsem se setkal.

Jelikož mezi podporované výstupní formáty dokumentace pomocí aplikace Doxygen formát Microsoft Help 2.x nepatří, bylo nutné zjistit, jak tohoto formátu dosáhnout. Použitelné formáty pro generování do podoby nápovědy byly pouze HTML a XML. Ale ani pro tyto formáty jsem nenašel žádný možný způsob přímé konverze do hledaného formátu. Jedinou realizovatelnou cestou se ukázala konverze nejprve z formátu HTML do formátu Microsoft Help 1.x a z ní posléze do verze Help 2.x. Nakonec však Visual Studio nepřinášelo podporu vyhledávání v této generované nápovědě, a tudíž se tato nápověda stala opět pouhou dokumentací, zabudovanou do vývojového prostředí. Po dalším možném hledání se východiskem a řešením celého problému stala aplikace Doc-O-Matic verze 6 v provedení Express určeném k bezplatnému užívání.

6.3.1 Doc-O-Matic Express

Doc-O-Matic (viz [10]) je profesionální komerční aplikace, vycházející z dokumentačních nástrojů typu Doxygen. Nabízí grafické uživatelské prostředí a podporu více vstupních i výstupních formátů. Aplikace Doc-O-Matic je volně a bezplatně ke stažení ve verzi Express. Ač je ochuzena o množství funkcí své plné verze, stále podporuje jako výstupní formát Microsoft Help 2.x a zároveň zvládá vytvořenou nápovědu registrovat do systému tak, aby byla ihned použitelná v prostředí Visual Studia. Tento přesvědčující fakt byl výchozí pro uplatnění v mém projektu.

6.3.2 Generátor nápovědy na serveru

S použitím nástroje Doc-O-Matic je nyní programátor schopen si vytvořit ze zdrojových kódů dokumentaci a zaintegrovat si ji automaticky jako help. Avšak pro toto řešení potřebuje mít na svém počítači nainstalovánu tuto aplikaci, musí znát její ovládání a konfiguraci a v neposlední řadě musí mít na své pracovní stanici celý kompletní projekt vyvíjené aplikace včetně všech souborů se zdrojovými kódy. K těmto záporům patří také fakt, že generování a registrování nápovědy do systému je časově náročné a je nutno jej provádět s každou změnou zdrojových souborů, aby nápověda zachovávala svou aktuálnost.

Toto vedlo k nápadu přesunout veškerou generující činnost na server, kde je jistota nejaktuálnějších stabilních zdrojových souborů kompletní aplikace a kde je také zaručen výpočetní výkon a neobtěžuje tak žádného z programátorů. Pro potřeby serveru však bylo nutné celý proces zautomatizovat. Program Doc-O-Matic byl prozkoumán, osekán o grafické uživatelské prostředí a bylo využito jeho schopnosti pracovat jako konzolová aplikace v příkazové řádce. Dále byl vytvořen skript naprogramovaný v jazyce python, který je schopný sepsat všechny existující zdrojové soubory aplikace a tento seznam předat programu Doc-O-Matic. Administrátor teď po nainstalování aplikace generátoru na server při prvním použití pouze nastaví základní konfiguraci s uvedením cest k potřebným zdrojům a celý generátor spustí. Při další potřebě vygenerovat aktuální nápovědu už stačí jen tento program, tvořený mým skriptem, spustit. V případě potřeby si tak může administrátor práci ulehčit a nastavit si automatické spouštění generátoru v časových intervalech.

6.3.3 Klientská část v add-inu Visual Studia

Poté, co je zajištěna aktuálnost kompletní verze programové dokumentace ve formě helpu na serveru pomocí pravidelného generování, je nyní možné tuto nápovědu jednoduše ze serveru přenést do pracovní stanice. S touto funkcí přichází můj add-in do Visual Studia. Programátorovi nyní stačí v nastavení add-inu uvést cestu k serveru, kde je dokumentace uložena, jedním tlačítkem celou dokumentaci stáhnout a zaintegrovat do svého prostředí Visual Studia 2005.

7 Závěr

Práce na celém projektu byla velice zajímavá. Zpočátku byly mé představy o řešení zadané problematiky dosti rozdílné od výsledné práce. V průběhu vývoje jsem však docházel k názorům, že jistými způsoby navržené implementace bude projekt nerealizovatelný a bylo nutné nacházet jiné možnosti. Pro vytváření nápovědy jsem musel upustit od původně zamýšleného generátoru Doxygen, jelikož nenabízel žádné výstupní formáty, ze kterých by bylo možné vytvořit help integrovatelný do prostředí Visual Studia. Vizualizéry zase nedostačovaly svou schopností požadavkům programátorů ve firmě. Musel jsem tedy navrhovat jiné způsoby zobrazení vybraných tříd. Visual Studio v původní instalaci nepodporovalo Code Snippety jazyka C++. Zde naštěstí stačilo stáhnout nový nástroj, který tuto podporu přinášel.

Při tvorbě projektu jsem se obohatil o nové znalosti. Seznámil jsem se s jazyky C# a Python, naučil jsem se pro vývoj aplikací používat vývojové prostředí Visual Studio, ke kterému jsem byl dříve dosti skeptický, a v neposlední řadě jsem se seznámil s technologiemi pro rozšiřování funkčnosti aplikací, převážně tedy Visual Studia.

Vedle znalostí mi byly velkým přínosem i zkušenosti, získané při samotném programování. Snažil jsem se provést návrh a implementaci tak, aby byla přiměřeně náročná na výpočetní prostředky. Aby byl kód čitelný, řádně zdokumentovaný a aby byl rozšiřitelný a použitelný při další práci. Nejvíc mne obohatila i samotná realizace projektu. Počínaje zadáním, kterému jsem nerozuměl a musel jsem tedy získávat a hledat nové vědomosti na jeho řešení. A konče samotnou implementací, která musela splňovat požadavky zadavatelů.

Ač je v projektu splněno vše, co bylo původně zadáno, pracuji na něm stále dál. Možností rozšíření je nepřeberně. Jako příklad bych uvedl podporu interních firemních chybových kódů či nástroj pro generování šablony pro nápovědy. Další mou snahou je rozšířit add-in také na novou verzi prostředí Visual Studio 2008. Ani tímto však má práce nekončí. Rád bych dovedl aplikaci do dokonalosti z hlediska implementace. S novými znalostmi používaných implementačních nástrojů přichází i nové nápady, jak tento projekt přepracovat jinak a lépe s ohledem na hardware a čitelnost kódu.

Literatura

[1] Nagel, Christian. *C# 2005:Programujeme profesionálně*. Brno, Computer Press, 2007

[2] Harms, Daryl, McDonald, Kenneth. *Začínáme programovat v jazyce Python.* Brno, Computer Press, 2003

[3] *Wikipedia: The Free Encyclopedia: Integrated development environment* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://en.wikipedia.org/wiki/Integrated_development_environment>.

[4] *Wikipedia: The free encyclopedia: Microsoft Visual Studio* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://en.wikipedia.org/wiki/Microsoft_Visual_Studio>.

[5] *MSDN Library: Tutorial 1: Getting Started with Visual Studio Extensibility* [online].
[cit. 2008-05-06].
Dostupné z WWW <http://msdn.microsoft.com/cs-cz/library/bb330853(en-us,vs.80).aspx>.

[6] *MSDN Library: How to: Create an Add-in* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://msdn.microsoft.com/en-us/library/80493a3w(VS.80).aspx>.

[7] *Virtualdub.orq: Writing custom visualizers for Visual Studio 2005* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://www.virtualdub.org/blog/pivot/entry.php?id=120>.

[8] *Doxygen: Source code documentation generator tool* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://www.stack.nl/~dimitri/doxygen/>.

[9] *Sun Developer Network: Javadoc* [online]. [cit. 2008-05-06].
Dostupné z WWW <http://java.sun.com/j2se/javadoc/>.

[10] *Doc-O-Matic: Source Code Documentation and Help Authoring Tool* [online].
[cit. 2008-05-06]. Dostupné z WWW <http://www.doc-o-matic.com/>.

Seznam příloh

Příloha 1. CD obsahující zdrojové kódy a instalátor aplikace, testovací soubory, programy důležité pro běh aplikace a programovou dokumentaci

Obsah CD

Adresářová struktura přiloženého CD je následovná:

- Adresář Projekt:
  - AvgAddin – adresář projektu add-inu
    - Binary_Files – instalátor add-inu
    - Source_Solution – zdrojové soubory
  - HelpCreator – adresář zdrojových skriptů pro generování nápovědy
  - ManualVizualizers – adresář s manuály pro tvorby vizualizérů
    - manual_cs – manuály v češtině
    - manual_en – manuály v angličtině
    - Vizualizers.zip – ukázkové příklady
  - TestFiles – Adresář s testovacími soubory
    - Vizualizers – zdrojové soubory příkladů v C++
    - autoex.dat_add – zdrojový soubor s vizualizéry
  - ProgramDocumentation - Programová dokumentace Add-inu
- Adresář Programy:
  - DocOmatic – Generátor dokumentace
  - HtmlHelpWorkshop – Generátor HTML helpu
  - Java – Java interpret
  - Python – Python interpret
  - VS_SDK – Visual Studio SDK
  - VSCodeSnippetsC++ – Plugin pro VS 2005
- Soubor BakalarskaPrace.pdf – technická zpráva