VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2012 Bronislav Draška

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Bronislav Draška**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: KVADOS, a.s.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek, Ph.D.**

Konzultant bakalářské práce: Mgr. Pavel Ullmann

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2012 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 .....

Tímto bych chtěl poděkovat vedení společnosti KVADOS za možnost vykonat bakalářskou praxi právě v této společnosti. Dále bych chtěl velmi poděkovat mému konzultantovi Mgr. Pavlu Ullmannovi za velmi ochotný a přátelský přístup, mnoho cenných rad a za profesionální vedení mé bakalářské praxe. Nakonec bych rád poděkoval mému vedoucímu bakalářské práce Ing. Marku Běhálkovi, Ph.D. za rady při tvorbě této práce.

Abstrakt

Tato bakalářská práce popisuje průběh mé odborné praxe ve společnosti KVADOS, kde jsem působil na pozici stážisty. V první polovině této praxe jsem se zabýval zkoumáním možností v oblasti automatické vizualizace dat, převážně za využití produktu Microsoft Visio 2010 a nástrojů VSTO. Završením projektu byla implementace demonstračních aplikací. Na základě těchto aplikací a nabytých poznatků pak vedení společnosti rozhodovalo o dalším průběhu tohto projektu. Následně poté jsem se podílel na rozšiřování doplňkové aplikace pro Microsoft Word pod názvem myDOC. Tato aplikace slouží pro interní použití a má za úkol zefektivnit tvorbu analýz a sjednotit jejich strukturu. Zde bylo mým úkolem jak upravovaní stávajících funkcionalit, tak doplňování funkcionalit nových.

Klíčová slova: C#, .NET, KVADOS a.s., MS Office Visio, MS Office Word, Visual Basic for Applications, Visual Studio Tools for Office, Visual Studio, add-in, Nevron

Abstract

This bachelor thesis describes the course of my professional practice in KVADOS where I worked at a trainee position. In the first half of this practice I have engaged in exploration of opportunities in the field of automatic data visualization, primarily using Microsoft Visio 2010 and VSTO. Culmination of the project was the implementation of demonstration applications. On the basis of this applications and the gained knowledge company's management made decisions about further course of this project. Thereafter, I participated in the expansion of Add-in application for Microsoft Word called myDOC. This application is for internal use and is designed to make the creation of analyzes more effective and unify their structure. Here was my role to editing of existing functionality and adding new functionalities.

Keywords: C#, .NET, KVADOS a.s., MS Office Visio, MS Office Word, Visual Basic for Applications, Visual Studio Tools for Office, Visual Studio, add-in, Nevron

Seznam použitých zkratek a symbolů

VBA	–	Visual Basic For Applications
VSTO	–	Visual Studio Tools for Office
SQL	–	Structured Query Language
XML	–	Extensible Markup Language
MS	–	Microsoft
MSDN	–	Microsoft Development Network
TFS	–	Microsoft Team Foundation Server
LINQ	–	Language Integrated Quer
API	–	Application Programming Interface

Obsah

1 Úvod

Možnost absolvovat bakalářskou praxi mě zaujala již na dni otevřených dveří. Líbila se mi možnost získat nové dovednosti a zkušenosti z praxe a zároveň o průběhu této praxe napsat bakalářskou práci. Nikdy předtím jsem však v žádné firmě z oboru IT nepracoval, ani jsem se nepodílel na žádném projektu. Pouze jsem napsal pár maker pro MS Excel, když jsem pracoval na pozici pasportizačního pracovníka. Tato makra sloužila pro usnadnění práce při tvorbě pasportů budov a pro úzký okruh tehdejších spolupracovníků. Naprostá většina mých schopností tedy pocházela pouze z činností spojených se školním studiem.
Společnost KVADOS jsem si vybral ze seznamu umístěného na stránkách fakulty. Zaujala mě pozice pro programovaní doplňků pro Microsoft Office, s čímž jsem měl alespoň malé zkušenosti. Po písemném kontaktu a zaslání životopisu jsem byl pozván na ústní pohovor. Na základě tohoto pohovoru jsem byl následně přijat na pozici stážisty.

V první kapitole této práce se věnuji popisu odborného zaměření společnosti a mému pracovnímu zařazení. V druhé kapitole popisuji zadané úkoly a jejich řešení v rámci práce na automatické vizualizaci dat. Třetí kapitola pak popisuje mou práci na produktu s názvem myDOC. Ve čtvrté kapitole se věnuji získaným a scházejícím dovednostem. V závěrečné kapitole shrnuji celkový průběh této praxe, hodnotím její přínos a popisuji svůj celkový dojem.

2 Popis odborného zaměření firmy a pracovního zařazení studenta

2.1 Popis odborného zaměření společnosti

Společnost KVADOS je významným středoevropským tvůrcem a dodavatelem vlastních softwarových řešení. Zaměřuje se především na klientelu z oblasti obchodu a služeb. Působí v 11 zemích Evropy a to prostřednictvím vlastních obchodních zastoupení a rozsáhlé sítě obchodních partnerů. Společnost KVADOS je jednou z mála IT společností, které mají své vnitřní procesy certifikovány dle všech pěti norem ISO využitelných v této oblasti. Je také trojnásobným vítězem Microsoft Industry Awards a dvojnásobným vítězem Microsoft Technology Awards.

2.2 Popis mého pracovního zařazení

Začátkem července 2011 jsem se účastnil přijímacího pohovoru, kde jsem popisoval své dosavadní znalosti. Po absolvování tohoto pohovoru jsem byl pozván na schůzku s Technology Managerem panem Mgr. Pavlem Ullmannem, mým pozdějším konzultantem. Zde mi byly představeny projekty, ze kterých jsem si mohl vybrat. Všechny se týkaly programování v oblasti MS VSTO, nejvíce mě však zaujala právě vizualizace dat v aplikaci MS Visio.

V září 2011 jsem tedy nastoupil na pozici stážisty a byl jsem zařazen do výrobního oddělení Ventus. Byly mi představeny firemní zvyklosti, interní procesy a také podrobnější znění mého projektu. Pracovní název tohoto projektu byl myVVV (VentusVisioVisualizer)

Po skončení práce na projektu myVVV jsem byl přeřazen na projekt s názvem myDOC. Zde jsem prováděl větší množství úkolů různého charakteru. Jednalo se o vytváření nových funkcionalit, úpravu funkcionalit stávajících a v případě potřeby jsem opravoval nahlášené chyby. U myDOCu stojí za zmínku to, že jej před rokem vytvářel stážista ze stejné fakulty a také v rámci své bakalářské praxe.

3 Práce na projektu myVVV

Projekt myVVV si kladl za cíl prozkoumat možnosti vizualizace dat v aplikaci MS Visio 2010 a následně vytvořit doplňkovou aplikaci, která by byla schopna vizualizovat interní data. Tyto vizualizace měly převážně sloužit jako podklad pro diskuze programátorů. Na tomto projektu jsem pracoval pouze s mým konzultantem.

3.1 Ověřování možností vizualizace dat v aplikaci MS Visio

Cílem tohoto úkolu bylo seznámení se se samotnou aplikací MS Visio 2010 a to jak po uživatelské stránce, tak po stránce programového přístupu. Měl jsem také zjistit, ke kterým funkcím této aplikace je programový přístup a zda se nám takový přístup bude později hodit. Výstupem měla být dokumentace, kde jsem shrnul všechny své poznatky a tuto dokumentaci jsem pak využíval při plnění dalších úkolů.

3.1.1 Shrnuté poznatky

S aplikací MS Visio jsem se nikdy předtím nesetkal, nicméně její ovládání je velmi intuitivní a sžil jsem se s ní velmi rychle. Většina funkcí je dostupná přes Ribbon, stejně jako v ostatních aplikacích balíku MS Office od verze 2007. Výkres se vytváří a následně ukládá do Visio dokumentu. Obrazce do tohoto dokumentu můžeme vkládat buď z rozsáhlé nabídky předdefinovaných vzorníků, z uživatelsky vytvořených vzorníků a nebo z výše zmiňovaného Ribbonu pokud jde o základní prvky. Takové obrazce lze seskupovat a tyto skupiny pak také vkládat do vlastních vzorníků, čímž vznikají nové obrazce. Další možností jsou šablony. Ty umožňují otevřít dokument spolu s předdefinovanými vzorníky a uživatelským nastavením. Jsou zde také základní šablony, při jejímž otevření se spolu s novým dokumentem zobrazí v Ribbonu nová kategorie, která přísluší kategorii zvolené šablony. Pro mé pozdější potřeby se nejvíce nabízela šablona pro práci s databází. Spolu s ní se v Ribbonu objevuje funkce s názvem *Reverse Engineer*. Tato funkce slouží k automatické vizualizaci prvků databáze a jejich vlastností.

Při zkoumání programového přístupu jsem však narazil hned na několik problémů. Prvním z nich byla nekompletní dokumentace a to jak na stránkách MSDN[1] tak v *IntelliSense*. Dalším problémem byla velmi malá základna programátorů z dané oblasti. Tyto problémy jsem z velké části řešil pomocí nahrávání maker. Funkci, kterou jsem chtěl provést programově, jsem provedl v uživatelském rozhraní a nahrál do makra. Výsledný kód nebo jeho části jsem pak přepsal z jazyka VBA do C#. Některé úkony se však do makra vůbec nezaznamenávají a při zapnutí nahrávání makra se některé funkce v uživatelském rozhraní dokonce znepřístupní. Při delším pátrání jsem narazil na informaci o tom, že Microsoft k některým API u Visia znemožňuje přístup. Od Visa 2003 mezi tato API patří i to pro vizualizaci databáze a s tím tedy i výše zmiňovaná funkce *Reverse Engeneer*.

V grafickém rozhraní se dají veškeré vlastnosti obrazců nastavovat v okně *ShapeSheet*. Vlastností je zde nepřeberné množství a jsou přehledně rozděleny podle kategorií do tabulek. Práce s vlastnostmi v ShapeSeetu je velmi podobná práci s buňkami v sešitu aplikace MS Excel. Programový přístup k těmto vlastnostem je však ve stejném stylu. Tedy přes tři souřadnice, z nichž první představuje kategorii dané vlastnosti, druhá samotnou vlastnost a třetí představuje jeden z parametrů dané vlastnosti. Jednotlivé souřadnice lze získat pomocí enumerátorů, avšak díky absenci dokumentace je tato práce velmi obtížná.

Práce se samotnými obrazci je o něco intuitivnější. Pokud vznikne nějaký obrazec seskupením několika podobrazců, lze k němu pak přistupovat jako k objektu, který obsahuje další podobjekty. Navíc tyto podobjekty (podobrazce) mohou obsahovat další podobrazce a tak dále. Obrazce mají také tzv. konektory neboli spojovací body. Na tyto konektory lze napojit spojovací křivky. Tyto křivky mohou mít různé tvary koncových bodů nebo styly čar a slouží k vizuálnímu propojení jednotlivých obrazců. Toto propojení lze vyvolat i programově a to buď za pomoci určení konkrétních konektorů nebo automaticky, kdy si zvolí nejvhodnější konektory Visio. Takto propojené objekty lze také automaticky rozložit podle několika schémat a to jak z grafického rozhraní, tak programově.

```
    // Vybereme 2 objekty, které chceme propojit
visioPage.Shapes.ItemFromID[1].AutoConnect(visioPage.Shapes.ItemFromID[2], Visio.
    VisAutoConnectDir.visAutoConnectDirDown);

// nastavíme styl automatického rozložení (konkrétní hodnotu jsem získal díky nahrání makra)
visioPage.PageSheet.CellsSRC[(short)Visio.VisSectionIndices.visSectionObject, (short)Visio
    .VisRowIndices.visRowPageLayout, (short)Visio.VisCellIndices.visPLORouteStyle].
    FormulaForceU = "17";
```

Výpis 1: Automatické propojení dvou obrazců a rozložení na stránku

3.2 Vizualizace procesů softwarových agentů v MS Visio

Jak prozrazuje název kapitoly, cílem tohoto úkolu bylo vizualizovat procesy softwarových agentů. Softwarový agent je stavební prvek systému VENTUS® obsahující programový kód, vlastní menu, sestavy, tabulky a sloupce databáze a další konfigurace systému. V současné době existuje v IS VENTUS® více než tisíc těchto softwarových agentů. Vzhledem k tomu, že výstupem měla být pouze malá demonstrační aplikace, nepoužíval jsem reálný zdroj dat.

3.2.1 Zvolený postup řešení

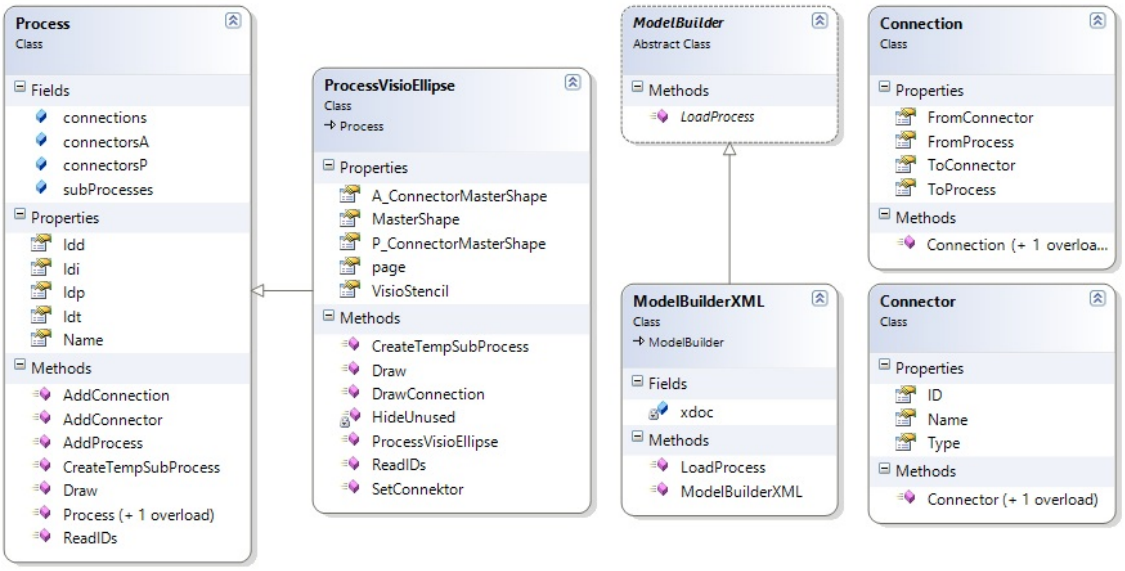
Jako alternativní zdroj dat jsem použil vlastní XML dokument. Ten jsem naplnil daty podle předloh, které mi poskytl můj konzultant. S XML dokumentem jsem pracoval pomocí LINQ to XML. LINQ jsem předtím neznal, ale velmi mě zaujal a jeho základní princip jsem pochopil během pár minut.

Pro vizuální podobu jednotlivých vlastností byla zvolena následující pravidla:

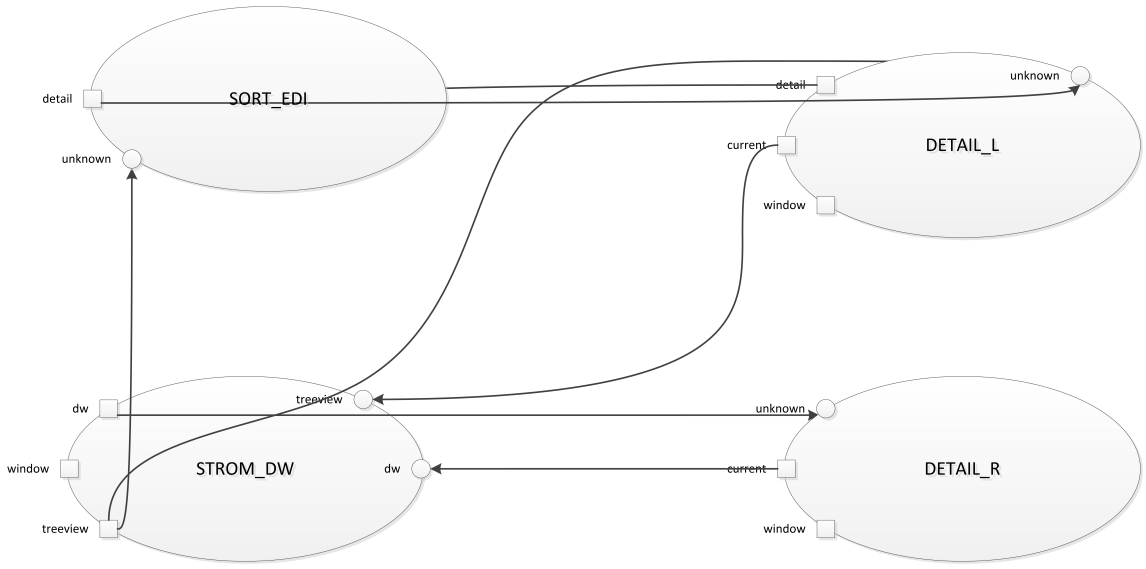
- Každý softwarový agent bude zobrazován jako elipsa
- Uprostřed této elipsy bude samotný název agenta
- Konektory budou rozmísťovány na krajích elipsy a jejich názvy se budou nacházet vedle nich
- Aktivační konektory budou zobrazovány jako čtverce a procesní jako kruhy
- Propojení budou zobrazována pomocí čar zakončených šipkou

S mým konzultantem jsme po krátké analýze rozhodli, že jednotlivé obrazce nebudu vytvářet programově, ale ručně. Vzhledem k časové náročnosti to bylo vhodnější řešení. Vytvořil jsem tedy obrazec, který představoval jednoho softwarového agenta. Konektory jsem vytvářel pomocí seskupeného a vystředěného čtverce s kruhem. Podle typu konektoru jsem pak byl schopen programově jeden z těchto obrazců skrývat. Stejně tak jsem pak skrýval i o nepoužité konektory. Ty jsem rovnoměrně rozmístil po obvodu elipsy. Bylo však nutno zachovat pořadí při vkládání jednotlivých konektorů. Pomocí pořadí vkládání obrazců se nastavují i jejich identifikátory, díky čemuž jsem pak byl schopen programově přistupovat ke konektorům na konkrétní pozici.

Načtené informace jsem nejprve převedl do objektového modelu a ten jsem následně vizualizoval. Při samotném vykreslování jsem využíval poznatky získané v prvním úkolu. Stačilo tedy vložit obrazce z vlastní šablony do dokumentu, nevyužité konektory poskrývat a následně konektory podle pravidel pospojovat. Nakonec jsem nad dokumentem zavolal metody pro vycentrování grafu a zvolil jeden z algoritmů pro automatické rozložení grafu.



Obrázek 1: Diagram tříd pro vizualizaci softwarových agentů



Obrázek 2: Ukázka vizualizace procesů softwarových agentů

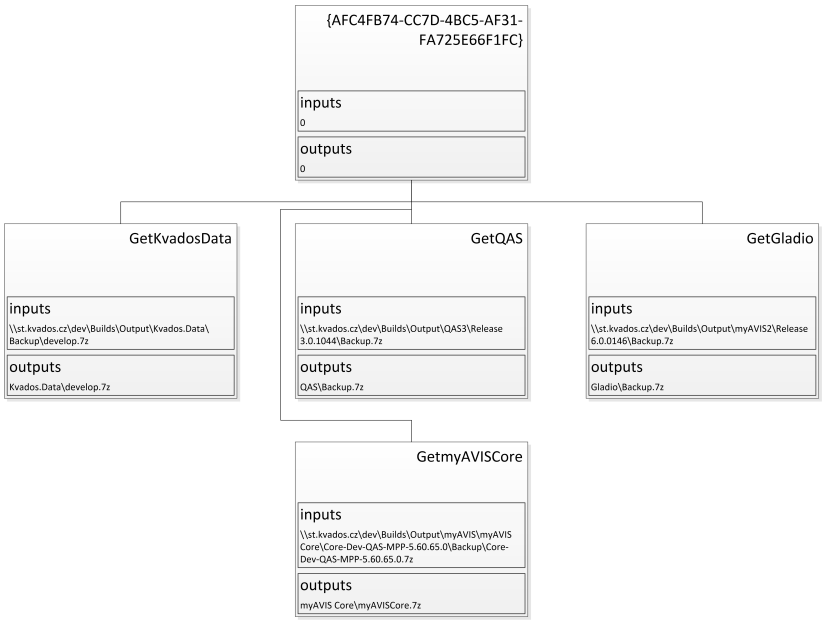
3.3 Vizualizace vazeb modulů

Cílem tohoto úkolu bylo vizualizovat závislosti projektu na jednotlivých modulech. Jako zdroj těchto informací jsem měl použít soubory *GetReferences.proj*. Tyto soubory jsou předpisem pro automatický *build* a používají se tehdy, když chce programátor vyvíjet nad konkrétní verzí. Podle tohoto souboru se mu na lokální disk stáhnou správné verze těchto modulů. Ve skutečnosti se nejedná o nic jiného, než XML dokumenty. Získání potřebných informací pouhým čtením obsahu tohoto dokumentu je však složité a časově náročné. Obzvláště při získávání zdrojových a cílových adresářů jednotlivých modulů, která jsou řešena pomocí skládání proměnných.

3.3.1 Popis mého řešení

Pro práci s XML dokumentem jsem použil LINQ to XML. Při vizualizaci jsem opět použil předvytvořený obrazec z vlastní šablony, ve kterém jsem zobrazoval název modulu, popřípadě ID projektu a absolutní zdrojové a cílové adresáře. Vše jsem podle načtených závislostí propojil, výsledný graf jsem vycentroval a uspořádal podle jednoho z dostupných algoritmů.

V tomto úkolu jsem nepoužíval na pozadí žádný objektový model. Vystačil jsem si pouze s jednou třídou a pár metodami. Tento postup jsem zvolil hned na začátku, neboť mi na daný úkol přišel dostatečný. Dnes bych však z důvodů přehlednosti jednoduchý objektový model použil.



Obrázek 3: Ukázka vizualizace vazeb modulů

3.4 Porovnání MS Visio s jinými produkty

V rámci tohoto úkolu jsem měl nalézt a porovnat aplikace nebo knihovny, které by pro nás byly vhodnou alternativou aplikace Visio. Pokud šlo o komerční produkt, měl jsem zjistit i cenu licence.

3.4.1 Dosažené výsledky

Většina produktů, na které jsem při svém hledání narazil, pracovaly buď se statickým obrázkem a nebo měly špatnou dokumentaci. Během hledání jsem však dostal tip na produkt s názvem Nevron. Jednalo se o sadu knihoven a prvků grafického rozhraní. Ukázkou dovedností těchto knihoven je aplikace pod názvem Nevron DiagramDesigner. Principiálně se jedná o stejnou aplikaci, jakou je Visio. To se týka především práce v uživatelském rozhraní a dostupných funkcí. Nicméně po stránce dokumentace a programového přístupu byla práce s Nevronem daleko příjemnější a intuitivnější než s Visiem. Aplikace se vytváří formou formulářové aplikace s využitím několika vizuálních komponent Nevronu, které značně usnadňují práci. Spolu s několika základními předlohami jsem se s programovým přístupem sžíval daleko rychleji, než v případě Visia. Na jeho bližší prozkoumání však už nezbyl čas.

3.5 Ukončení práce na projektu myVVV

Po dokončení posledního zmiňovaného úkolu následovala prezentace nabytých poznatků a demonstračních aplikací před vedením společnosti. Vedení se však rozhodlo, že se za daných okolností nebude v projektu dále pokračovat a byla mi nabídnuta práce na myDOCu. Všechny doposud získané poznatky jsem však shrnul alespoň do jednoduché dokumentace a předal svému konzultantovi pro archivaci. Společnost KVADOS má pro tyto účely zřízen vlastní portál s názvem kvadopedie[3].

4 Práce na projektu myDOC

Aplikace myDOC je dopňkovou (Add-in) aplikací pro MS Word, sloužící pro interní použití. Tuto aplikaci před rokem vytvářel také stážista v rámci své bakalalářské práce a od té doby se stal myDOC oficiálním interním projektem. Hlavní myšlenkou při vytváření myDOCu bylo usnadnit a sjednotit vytváření analýz. Usnadnění spočívá v použití stavebních bloků. Ty jsou rozděleny podle kategorií a jazykových mutací do jednotlivých šablon. Sjednocení je zajištěno tak, že existuje pouze jedna aktuální verze každé šablony. Ty jsou umístěny na serveru Sharepointu, odkud se automaticky stahují na lokální uložiště při spuštění Wordu. Pokud někdo provede změny v těchto lokálních šablonách a ty pak vypublikuje na server, automaticky začne server poskytovat tyto novější verze. Předchozí verze se přitom archivují. Při samotném vytváření analýz je možné vyhledávat a využívat informace přímo z databáze a tyto informace lze přes různá dynamická pole vkládat do dokumentu. Stejně tak lze pomocí obsahu v dokumentu získávat informace z databáze a to vše aniž by uživatel opustil prostředí aplikace Word. Všechny tyto funkce jsou přístupné přes Ribbon, kde má myDOC vlastní záložku.

4.1 Zobrazení informací z personalistiky

Cílem tohoto úkolu bylo vytvořit postranní panel, ve kterém by se zobrazovaly informace z personalistiky náležící danému úkolu. Tato akce má být vyvolatelná tlačítkem z Ribbonu nebo z kontextové nabídky. Pokud je označen text, má se napřed kontrolovat, zda označený text odpovídá masce kódu úkolu. V případě, že se kurzor nachází uprostřed slova nebo na jeho okraji a nebo byla označena pouze jeho část, má se zbytek slova dooznačit. Pokud formát odpovídá, má být zaslán dotaz do databáze. Pokud tento formát neodpovídá nebo nebyl v databázi daný kód úkolu nalezen, uživateli se zobrazí formulář pro možnost opravy. V případě, že byl záznam v databázi nalezen, v aktivním dokumentu se zobrazí postranní panel s požadovanými informacemi. Na tomto panelu se má také nacházet tlačítko, které zobrazí daný úkol přímo v modulu personalistiky.

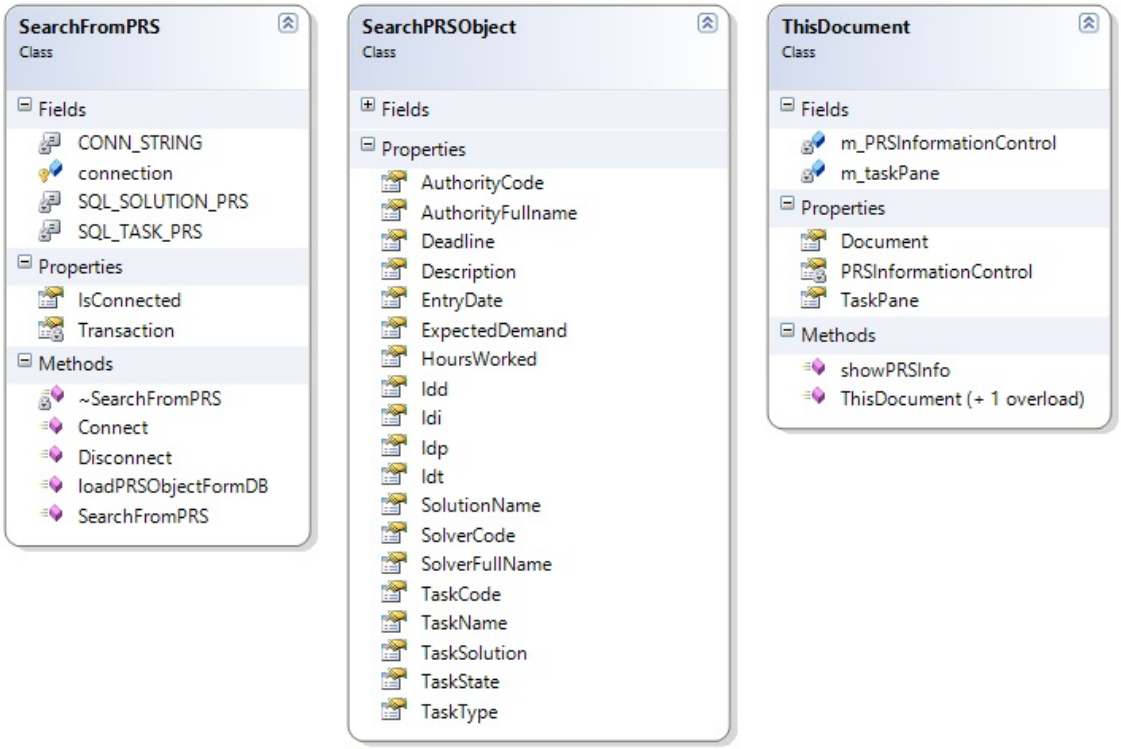
4.1.1 Zvolený postup řešení

Nejprve jsem začal s vytvářením postranního panelu. Vytvořil jsem si *UserControl* panel, do kterého jsem naskládal vizuální komponenty tak, abych zde byl později schopen zobrazit všechny informace dle zadání. u *customTaskPane* objektů jsem narazil na to, že vznikají při jejich prvním zobrazení, ale pokud jej uživatel zavře, pouze se změní hodnota vlastnosti *Visible* na *false*, s čímž bylo nutno počítat při opakovaném vyvolání. Do Ribbonu myDOCu jsem přidal tlačítko, které volá stejnou metodu z třídy *ThisAddin* jako tlačítko z kontextové nabídky. Později jsem tuto metodu rozšířil o parametr odesílatele, neboť vznikl požadavek na různé chování při vyvolání funkce z různých míst. V třídě *ThisAddin* jsem měl napsány také všechny metody potřebné pro řízení chodu této funkcionality.

Pro připojení k databázi jsem nemohl použít stávající připojení myDOCu. To využívalo

pro připojení do databáze uživatelský účet aktuálně přihlášeného uživatele systému. Účet, který byl přiřazen mi, však tento přístup neumožňoval. Dostal jsem tedy parametry pro připojení k databázi pomocí uživatelského jména a hesla. Pro toto připojení jsem vytvořil vlastní třídu *SearchFromPRS*, ve které jsem měl vše potřebné pro práci s databází. Pro získání informací z databáze jsem používal vlastní ORM a transakce s nejvyšší úrovní izolace. Pro přenesení získaných informací z databáze jsem si vytvořil třídu *SearchPRSObject*. Při čtení dat z databáze jsem si vytvořil instanci této třídy a namapoval na ni získané informace. Její obsah jsem pak zobrazil v postranním panelu.

Pro to, abych byl schopen zobrazovat postranní panel pouze u dokumentu, ze kterého byl vyvolán, jsem si vytvořil třídu *ThisDocument*. Ta má i do budoucna sloužit ke všem funkcionalitám, které se vztahují ke konkrétnímu dokumentu. Díky této třídě je možné mít otevřeno několik dokumentů najednou a zajistit při tom, aby každý dokument měl například svůj postranní panel se svými informacemi a neovlivňoval chod postranních panelů u ostatních dokumentů.



Obrázek 4: Potřebné třídy pro zobrazení informací z personalistiky

4.2 Testy pro podporu kalkulací

Zde jsem měl vytvořit testovací aplikaci, která by měla umět základní funkce plánované funkcionality pro podporu kalkulací. Jednalo se především o vyhledání všech identifikátorů v celém dokumentu analýzy včetně komentářů a jejich zobrazení v postranním panelu. Tento identifikátor se skládá z kódu úkolu, kódu funkcionálního požadavku a kódu kalkulace, kde každý má svou specifickou masku. Kódy jsou navíc odděleny implicitně středníkem a celý takovýto identifikátor je ohraničen závorkami. Pořadí těchto kódů v indentifikátoru však není pevné a některé kódy dokonce mohou chybět. Nakonec jsem měl otestovat rychlost na velkém, až pěti set stránkovém, dokumentu za využití vyhledávání pomocí regulárních výrazů a pomocí vlastního jednoznačného prefixu.

4.2.1 Zvolený postup a získané poznatky

Při vyhledávání v dokumentu jsem programově využíval standartní funkci Wordu pro vyhledávání. Při samotném vyhledávání se ukázalo, že čas vyhledávání pomocí regulárního výrazu je téměř shodný s časem při použití prefixu. Jako další zajímavost se ukázalo to, že s rostoucím počtem stran dokumentu a stejně rostoucím počtem výskytů hledaného výrazu se doba pro jejich nalezení zvyšovala daleko rychleji.

	100 výskytů/100 stran		250 výskytů/250 stran		500 výskytů/500 stran	
	prefix	regulární v.	prefix	regulární v.	prefix	regulární v.
komentáře	4 s	4,2 s	13 s	13,6 s	52,6 s	51,2 s
text	0,7 s	0,9 s	6,2	6,8	37 s	38 s

Tabulka 1: Naměřené časy při vyhledávání indentifikátorů v kometářích a v textu

Analýzy sice doposud měly maximálně kolem tří set stran, nicméně i tak jde o uživatelsky nepřívětivě dlouhou dobu. Při svém hledání jsem se pokoušel najít způsob, jak tuto práci zrychlit a co ji vlastně brzdí. Jediná směrodatná informace (kromě toho, že jde o vlastnost aplikace Word), kterou jsem k tomuto našel, spočívala v problému kompatibility s hardware. Jednalo se o zkušenosti uživatelů, kteří vypozorovali to, že na stanici (serveru), která je osazena dvěma procesory a oba jsou využívány, pracuje Word pomaleji než když má k dispozici procesor pouze jeden. Vzhledem k tomu, že jsem také pracoval skrze vzdálenou plochu na vzdáleném serveru, mohly být moje výsledky ovlivněny právě tímto. Nicméně způsob práce přes vzdálenou plochu využívá velké množství zaměstnanců, tudíž bylo nutno s tímto omezením počítat.
U finální podoby této funkcionality se budou idetifikátory načítat v novém vlákně, aby nedošlo k omezení práce vůči uživateli. S tímto byl spojen další problém, týkající ukládání pozic indentifikátorů do objektu typu *Range*. Tento objekt má v dokumentu přesně vymezenou počátční a koncovou pozici, které se aktualizuji, pokud dojde ke změnám v obsahu. *Range* tedy neustále zapouzdřuje stejný obsah, jaký mu byl přiřazen na začátku. Tato aktualizace však neprobíhá korektně, pokud uživatel použije funkci *zpět*. Tento problém jsem vyřešil pomocí záložek, neboť jejich pozice, ke kterým se vztahují, se aktualizuje správně ve všech případech.

4.3 Definice pořadí prvků v menu

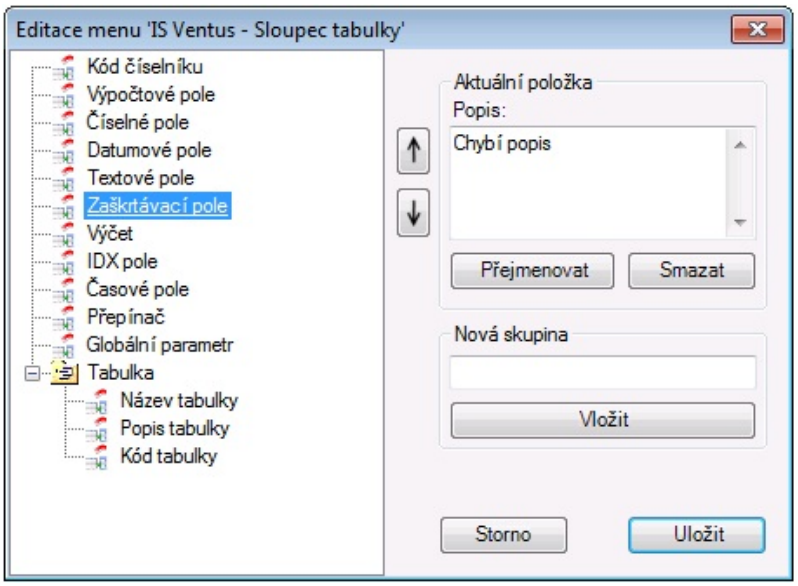
Tento úkol se týkal možnosti personalizace menu, do kterých se načítají stavební bloky. Doposud se do jednotlivých kategorií v Ribbonu myDOCu načítaly stavební bloky podle pořadí, ve kterém byly načítány z šablon. To se dělo při události prvního otevření jednoho z těchto menu. Mým úkolem bylo umožnit to, aby si uživatel mohl položky v menu libovolně přeskupovat, přidávat podmenu, měnit popisky stavebních bloků nebo rovnou jejich názvy, popřípadě prvky i odstraňovat. Změny, které není možné uložit standardně v aplikaci Word, se mají ukládat do XML dokumentu, přičemž každá kategorie i jazyková mutace má mít svůj vlastní. Tyto XML dokumenty se mají publikovat na server Sharepointu a stahovat z něj stejně jako doposud šablony stavebních bloků.

4.3.1 Zvolený postup řešení

Jako nejvhodnější formu řešení jsem zvolil samostatný formulář a prvky z menu jsem zobrazoval v komponentě *TreeView*. Při načítání prvků z menu do *TreeView*, při ukládání struktury uzlů *TreeView* do XML dokumentu a při naplňovaní menu podle struktury v XML dokumentu jsem vždy používal rekurzi. S XML dokumenty jsem opět pracoval pomocí LINQ.

Při načítání prvků do menu jsem se snažil využít maximum toho, co už bylo hotovo dříve. Přidal jsem tedy do těla původní metody pro naplnění menu vlastní kolekci. Místo do menu jsem načetl všechna tlačítka do této kolekce. Tím jsem získal všechna tlačítka, která se mají v menu nacházet. Podle XML dokumentu jsem teprve začal jednotlivá tlačítka vkládat do menu, popřípadě vytvářet podsložky. Pokud XML dokument neexistoval nebo se jej nepodařilo otevřít, naplnilo se menu tlačítky v původním pořadí.

V samotném *TreeView* jsem chtěl použít funkci *Drag&Drop*. Překvapilo mě, jakou pracnost použití této funkce v *TreeView* obnáší, ale na druhou stranu jsem měl větší prostor pro její přizpůsobení dle mých potřeb. Například objekty uzlů v TreeView není možné přesouvat na jinou pozici, byť ve stejné úrovni. Přesunutí uzlu jsem řešil tak, že jsem si vytvořil jeho kopii pomocí metody *Clone()*, tu vložil na určitou pozici a původní uzel jsem smazal. Dále bylo třeba rozlišit uzly, které představují stavební bloky, od těch, které představují složky. To jsem realizoval vizuálně pomocí ikon a programově pomocí vlastnosti *Tag* daného uzlu. Při samotném přetahování jsem získával přetahovanou položku podle pozice kurzoru. Přetahovaný uzel jsem pak vkládal před uzel, nad který byl přetahovaný uzel položen. Pokud došlo k přetažení na uzel představující složku, vložil se přetahovaný uzel jako poduzel.



Obrázek 5: Formulář pro definici pořadí prvků v menu

Rozpoznání změn, které uživatel prováděl, jsem řešil několika způsoby. Pro změnu názvu jsem si původní název ukládal do vlastnosti *Tag* (v případě menu je pod touto vlastností uložena informace o tom, že jde o menu). Díky tomu jsem byl schopen při ukládání indentifikovat jeho změnu u konkrétního bloku a tu pak uložit. Pro smazání bloků jsem použil kolekci, do které jsem si ukládal původní názvy jednotlivých bloků, které mají být smazány. U popisků bloků jsem změny neodhaloval, pouze jsem je znovu přenastavoval. Pokud však existoval stavební blok, který neměl žádný popis a pokoušel jsem se mu popis nastavit, došlo k pádu celé aplikace Word a to bez možnosti odchytit nějakou vyjímku. Jako dočasné řešení jsem použil podmínku, která ověřovala, zda daný blok má nastavený popisek a až poté jsem popisek přiřazoval. Později jsem opravil podmínku přímo ve formuláři, ve kterém se nový stavební blok do myDOCu doposud přidával. Po potvrzení se uložily požadované změny do šablon a do XML dokumentu a následně na to se zavolala metoda pro naplnění menu. Ke stavebním blokům je důležité přistupovat přes jejich šablonu, typ, kategorii a název. Ne pouze přes šablonu a název bloku, protože takto nemusíme získat konkrétní stavební blok, který požadujeme. Této chyby jsem se z počátku sám dopustil.

```
// takto sice získáme stavební blok s názvem, dle kterého jej hledáme
Word.BuildingBlock block = template.BuildingBlocks.Item("nazev_bloku");

// nicméně pouze tento přístup je jednoznačný
Word.BuildingBlock block = template.BuildingBlockTypes.Item(Word.WdBuildingBlockTypes.
    wdTypeCustom1).Categories.Item("nazev_kategorie").BuildingBlocks.Item("nazev_bloku");
```

Výpis 2: Ukázka špatného a správného přístupu ke stavebnímu bloku

5 Scházející a získané znalosti a dovednosti

5.1 Návrh objektového modelu

S návrhem objektového modelu jsem měl problémy hlavně z počátku této praxe. Jednak jsem měl spoustu špatných návyků, ale také jsem měl velmi málo zkušeností s touto problematikou. Chyběla mi také znalost problematiky návrhových vzorů. V tomto ohledu mi velmi pomohly předměty jako Vývoj informačních systémů a Databázové a informační systémy. V průběhu studia těchto předmětů jsem tak měl možnost využívat čerstvě nabyté znalosti ihned v praxi, což působilo velmi motivačně.

5.2 Systematický postup při řešení úkolů

Často se mi stávalo, že jsem předem neprodiskutoval svůj způsob řešení daného problému. Výsledkem bylo pak to, že jsem musel svou práci předělávat, což se samožřejmě projevovalo na čase, který jsem k úkolům vykazoval. Nejasnosti v bodech zadání úkolu jsem z počátku řešil až postupně v průběhu plnění celého úkolu, čímž jsem nadměrně časově zatěžoval i svého konzultanta. Občas se mi stávalo, že jsem nějakou část zadání pochopil špatně a tak ji také naimplementoval. Tato chyba se většinou projevila až při předvádění dané části úkolu.

6 Závěr

Během psaní této práce jsem se podílel na dalším úkolu, který jsem však nestihl dokončit před dopsáním této práce. Cílem tohoto úkolu bylo vytvoření funkcionality pro nastavovaní povinností kapitol u šablon analýz. Podle těchto šablon a další funkcionality pak bude možno jednoduše přidávat nebo odstraňovat kapitoly z aktuálního dokumentu analýzy. Dále jsem strávil nějaký čas děláním menších úprav a opravami nahlášených chyb. Ty se týkaly jak mnou vytvořených částí aplikace, tak částí vytvořených již dříve. Po skončení bakalářské praxe bych se měl věnovat již zmiňované funkcionalitě pro podporu kalkulací.

Od této praxe jsem očekával, že získám cenné zkušenosti a přehled a o tom, jak to chodí ve firmě z oboru IT. Tato očekávání byla naplněna beze zbytku, ba dokonce mnohem lépe, než jsem čekal. Po celou praxi nebylo dne, kdy bych se nenaučil něco nového. Navíc jsem měl možnost pracovat ve velmi profesionálním a příjemném kolektivu, což nebývá samozřejmostí a velmi si toho vážím. Dnes, s odstupem času, vnímám rozhodnutí zvolit si bakalářskou praxi, jako jedno z mých nejdůležitějších dosavadních rozhodnutí.

Bronislav Draška

7 Reference

[1] Microsoft, *Microsoft Development Network*, [ONLINE].
URL: http://msdn.microsoft.com

[2] KVADOS, *Stránky společnosti KVADOS, a.s.*, [ONLINE].
URL: http://www.kvados.cz

[3] KVADOS, *Portál kvadopedie*, [ONLINE].
URL: http://kvadopedie.kvados.cz

8 Hodiny strávené na jednotlivých úkolech

Úkol	**počet hodin**
Ověřování možností vizualizace dat v aplikaci MS Visio	66,25
Vizualizace procesů softwarových agentů	32,5
Vizualizace vazeb modulů	27,5
Porovnání MS Visio s jinými produkty	26,75
Prezentace a její příprava	9
Tvorba dokumentace pro archivaci	5,75
Celkem:	**167,75**

Tabulka 2: Hodiny strávené na projektu myVVV

Úkol	**počet hodin**
Zobrazení informací z personalistiky	42,25
Testy pro podporu kalkulací	47
Definice pořadí prvků v menu	47,25
Opravy nahlášených chyb	11
Volba kapitol dokumentu	84,75
Celkem:	**232,25**

Tabulka 3: Hodiny strávené na projektu myDOC

9 Seznam příloh na DVD

GetReferences.proj
- zdrojový soubor pro vizualizaci vazeb modulů

team_sort.xml
- zdrojový soubor pro vizualizaci procesů softwarových agentů

Muj_vzornik.vss
- mnou vytvořený vzorník aplikace Visio použitý při práci na projektu myVVV

GetReferencesDiagram.vsd
- výsledný Visio dokument vizualizace vazeb modulů

team_sort.vsd
- výsledný Visio dokument vizualizace procesů softwarových agentů