

Obrazový informačný systém pre prístrojovú techniku

A Pictorial Information System for Technical Appliances

Bc. Tomáš Kutlák

Diplomová práce
2012



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ABSTRAKT

Táto diplomová práca sa zaoberá vývojom informačného systému pre diagnostiku. Najprv je popísaný modelovací jazyk UML, pomocou ktorého bol informačný systém navrhnutý. V kapitolách sú ďalej popísané návrhové vzory observer a MVC, vysvetlené niektoré časti .NET použité v projekte. Celá intranetová časť systému bola zostavená technológiami: ASP .NET s použitím MySQL databáze.

Kľúčové slová: Diagnostika, C#, .NET, MVC, LINQ, ASP, UML, Model, Návrh

ABSTRACT

This thesis describes the development of an information system for the diagnosis. At first is described the modeling language UML, with which the information system was designed. In the Chapters are then described the Observer design patterns and the MVC, explained certain parts of .NET used in the project. The entire part of intranet system was constructed with technologies: ASP .NET using the MySQL database.

Keywords: Diagnostics, C#, .NET, MVC, LINQ, ASP, UML, Model, Design

Chcem poďakovať pánovi Petrovi Neumannovi za jeho odbornú podporu behom písania práce a všetkým, ktorí mi behom štúdia pomáhali.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST.....	10
1 VÝVOJ INFORMAČNÝCH SYSTÉMOV.....	11
1.1 VODOPÁDOVÝ MODEL	11
1.2 PRÍRASTKOVÉ MODELY	11
1.2.1 RUP	11
1.2.2 Plan driven metodika.....	12
1.2.3 Scrum	12
1.2.4 Extrémne programovanie	12
1.3 ZNOVUPOUŽITIE	12
2 NÁVRH	13
2.1 METÓDA CRC	13
2.2 JAZYK UML.....	13
2.2.1 Model	13
2.2.2 Požiadavky	14
2.2.3 Use case model a model Aktivít.....	15
2.2.4 Model tried a model komponentov	16
2.2.5 Implementácia triedy a rozhrania v C#	18
3 NÁVRHOVÉ VZORY	19
3.1.1 Návrhový vzor observer	19
3.1.2 Model View Controller (MVC)	20
4 VÝVOJOVÉ PROSTRIEDKY	22
4.1 C# A PLATFORMA .NET.....	22
4.1.1 C#	22
4.1.2 .NET framework	22
4.1.3 Microsoft Visual Studio	23
4.1.4 ASP .NET.....	24
4.1.5 ADO .NET Entity Framework	24
4.1.6 Prúdový vstup a výstup	24
4.1.7 Reflexia	26
4.1.8 XML a serializácia	27
4.1.9 Lambda výrazy	28
4.1.10 Delegáti a anonymné metódy	28
4.1.11 LINQ	29
4.1.12 GUID.....	32
4.1.13 Anonymné typy	32
4.1.14 Rozhranie IEnumerable	32
4.1.15 Generiká	32
4.2 JAVASCRIPT.....	33
4.3 MYSQL	33
II PRAKTICKÁ ČÁST.....	34
5 NÁVRH	35

5.1	POŽIADAVKY	35
5.1.1	Popis požiadaviek.....	36
5.2	DEPLOYMENT MODEL.....	38
5.2.1	Popis deployment modelu	38
5.3	KOMPONENTOVÝ MODEL	39
5.4	USE CASE MODEL.....	40
5.5	STATICKÝ MODEL (DIAGRAM TRIED).....	43
6	WEBOVÝ PROJEKT ASP .NET MVC 3.....	44
6.1	ADO .NET MODEL.....	44
6.2	NÁHLAD NA PROJEKT	47
6.2.1	Popis systému.....	47
6.3	IMPLEMENTÁCIA AUTENTIZÁCIE A BEZPEČNOSTI.....	49
6.3.1	Prihlásenie do systému	49
6.3.2	Registrácia.....	50
6.3.3	Zmena hesla	50
6.3.4	Role	51
6.4	IMPLEMENTÁCIA UKLADANIA OBRÁZKOV A PRÍLOH	51
6.5	IMPLEMENTÁCIA DYNAMICKÝCH TABULIEK	52
6.6	IMPLEMENTÁCIA ZAŠKRTÁVACIEHO POLÍČKA FORMULÁRA.....	53
6.7	IMPLEMENTÁCIA ROZBAĽOVACIEHO ZOZNAMU	53
6.8	VÝBER STROJA	54
6.9	ZOBRAZENIE SÚČIASTOK STROJA NA URČITEJ ÚROVNI.....	55
6.9.1	Popis funkčnosti z používateľského hladiska	55
6.9.2	Model	56
6.9.3	Logika	56
6.9.4	Pohľad a klientsky skript.....	57
6.10	EDITÁCIA SÚČIASTKY NA URČITEJ ÚROVNI.....	59
6.11	VÝBER TYPU ČASTI A VÝBER OBRÁZKU	60
	ZÁVER	62
	ZOZNAM POUŽITEJ LITERATURY.....	63
	ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK.....	65
	ZOZNAM OBRÁZKOV	67

ÚVOD

Niekedy sa stáva, že zariadenie prestane fungovať. Môže to byť spôsobené rôznymi vplyvmi, medzi ktoré patria aj mechanické poškodenie, skrat, poškodenie elektrostatickým výbojom, ale aj opotrebenie súčiastky. Pretože obsluha zariadenia nemusí a z pravidla nevie dobrou terminológiou popísať poruchu, musí technik absolvovať výjazd na miesto poruchy. Dochádza tak k zbytočným zmätkom i stresu. Najviac ak sa od problematického zariadenia očakáva, že má pracovať 24 hodín a 365 dní ročne, vznikajú finančné a časové straty na oboch stranách.

Riešením je vytvorenie informačného systému, ktorý dokáže narábať so súčiastkami stroja na určitej úrovni a kde bude údržba pridávať skúsenosti k zariadeniu. Taký systém musí tiež zvládnuť poskytovať informácie o poruchách a ich riešení v potrebnom čase.

Diplomová práca sa zaoberá najmä implementáciou intranetovej časti informačného systému. V teoretickej časti práce popisujem návrhový jazyk UML a programovacie techniky, dneska v praxi často používané návrhové vzory a samotnú webovú aplikáciu v praktickej časti. Miesto SQL schémy sa tu nachádza schéma modelu v ADO.NET.

V práci nevyšlo miesto na detailné popisovanie programovacích jazykov, preto je jazyk C#, .NET a SQL popisovaný minimálne. Jednotlivé menné priestory, ktoré sú aplikované sú zas popísané viac detailnejšie.

Diplomová práca obsahuje i priložené CD, kde nájdete webovú aplikáciu, diplomovú prácu v elektronickej podobe a ďalšie prílohy.

I. TEORETICKÁ ČÁST

1 VÝVOJ INFORMAČNÝCH SYSTÉMOV

V súčasnosti existuje veľa prístupov vo vývoji IS. Výber metódy vývoja pritom závisí od aplikácii IS v praxi. Rozdielny prístup zvolím pri vývoji IS štátnej správy, internetového obchodu, spravodajského webu, vývoju krabicového programu, embedded systému, či realtime systému. Výber závisí aj na veľkosti vývojového tímu (počtu zamestnancov vo firme zaoberajúcej sa vývojom IS) a počtu koncových užívateľov, ktorý budú systém používať.

Je ťažké, alebo často nie je ani možné nájsť tak dobrú metódu vývoja, ktorá by fungovala pre každý vývojový proces.

Všetky IS majú podobný postup vytvorenia: špecifikácia, vývoj, overenie a prispôsobenie podľa budúcich zmien. Tieto procesy obsahujú ďalšie podprocesy.

Vodopádový model, prírastkový model a SW inžinierstvo znovupoužitia komponentov sú najznámejšie procesné modely. Pri vývoji sa najčastejšie používa ich kombinácia.

1.1 Vodopádový model

Vodopádový model je najznámejší a určitú dobu bol aj najpoužívanejší model. Skladá sa zo 7 častí: špecifikácia požiadavkov, návrh, implementácia, integrácia, testovanie, inštalácia a údržba. Výhodou tohto modelu sú jednoznačne za sebou idúce fáze, z ktorých sa vytvára dokumentácia. Nevýhodou je nízka flexibilita pri veľkých projektoch a komunikácia so zákazníkom (neumožňuje sa prispôsobiť meniacim sa požiadavkám a projekt sa môže predražiť). [5]

1.2 Prírastkové modely

Medzi prírastkové modely patria agilné metodiky: Rational Unified process (RUP), Plánové (plan-driven), extrémne programovanie (XP) a metóda Scrum.

1.2.1 RUP

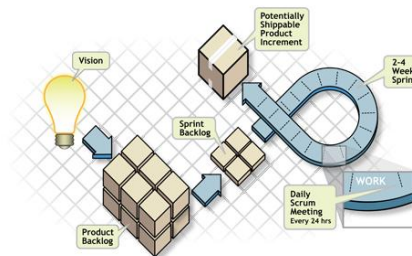
Cieľom RUP je zvýšenie frekvencie a skrátenie doby tvorby prírastkov. RUP má výhody v oblastiach riadenia požiadaviek, obsahuje komponentovú architektúru (vhodný prístup pre znovupoužitie), vizuálne modelovanie s použitím UML, zvýšená kontrola kvality, kontrola a akceptácia pribúdajúcich sa zmien (zmenové požiadavky).

1.2.2 Plan driven metodika

Plan driven metodiku najviac zaujímajú odpovede na otázky: Ako dlho bude projekt trvať? Koľko bude projekt stáť? Je zamerané na risk a dokáže ho dopredu predpovedať.

1.2.3 Scrum

Scrum metodika začína ideou a vytvorením tzv. backlogu produktu. Backlog produktu obsahuje zoznam prianí usporiadaných podľa priority. Vývojový tím si vezme menšiu časť z backlogu a navrhne spôsob implementácie. Potom začne vývojový sprint a na záver vznikne potencionálne odoslateľná práca. Progres na vývojovom sprinte sa každý deň kontroluje a ScrumMaster usmerňuje tím na cieľ. Každý sprint má svoj výstup. [7]



Obrázok 1 - Scrum

1.2.4 Extrémne programovanie

XP rieši ideálne programátorské prostredie a dobrú komunikáciu medzi programátormi a zákazníkom. Iterácie u tejto metódy sú veľmi rýchle (až denné). Sústreďuje sa na vývoj a testovanie. Z pohľadu návrhu nepoužíva UML, ale skôr jednoduchšie CRC (Class Responsibility Collaborators).

1.3 Znovupoužitie

Pri vývoji môžeme postupovať tak, že sa rozhodnem pre zostavenie IS z komponentov, miesto vyčerpávajúceho vývoja menších častí IS. Niektoré komponenty je možné kúpiť na internete, ktorým často konkurujú komponenty k stiahnutiu zadarmo.

2 NÁVRH

Návrh IS je rovnako dôležitý ako samotná implementácia. Treba aspoň navrhnuť základné kamene a štruktúru stavby.

Postup návrhu:

1. Kontext a externé interakcie so systémom: Kontextový model a to Use case model, alebo model aktivít.
2. Systémová architektúra: V projekte bude použitá architektúra MVC a Klient - Server. Veľmi dobrá a dnes často používaná architektúra vývoja je MVC, ktoré je popísané v kapitole 3.1.2 Klient – Server architektúra je znázornená deployment diagramom v praktickej časti DP.
3. Identifikácia Objektov: Návrh tried a k nim príslušné metódy a atribúty.
4. Identifikácia Rozhraní: Metódy tried, ktoré musia byť implementované.

Návrh bude uskutočnený v jazyku UML. Na identifikáciu tried bude čiastočne použitý CRC.

2.1 Metóda CRC

Metóda CRC je vhodná na zjednodušenie tvorby tried. Tím programátorov sa zide na brainstormingu a identifikujú triedy podľa slovníku problému. Každú identifikovanú triedu zapíšu na kartu. Karta obsahuje názov triedy, triedy potomkov, rodičovské triedy, zodpovednosti a kolaborantov. Kolaboranti sú objekty, ktoré pracujú s triedou. Zodpovednosť triedy sa určí pomocou scenáru.

2.2 Jazyk UML

UML je štandardizovaný jazyk. O štandardizáciu je zodpovedné OMG a RTF. Štandard jazyka nepopisuje ako zaviesť diagramy, ani spôsob ako ich implementovať. UML považujeme za modelovací jazyk, ktorý obsahuje modely so statickou štruktúrou a dynamickým správaním.

2.2.1 Model

Model je zobrazenie, ktoré prevádza prvky a vlastnosti pôvodného modelovaného systému na prvky a vlastnosti modelujúceho systému. [19] Nie je to však celý systém, pretože

model UML je len popisom štruktúry a významu systému. Model je vždy len abstrakciou na určitej úrovni.

Model UML obsahuje sémantickú informáciu a vizuálnu prezentáciu. Sémantická informácia nás informuje, o čom model je (trieda, use case...). Vizuálna prezentácia zobrazuje obsah modelu v správnom zoskupení a so správnymi väzbami.

2.2.2 Požiadavky

Požiadavky zistíme buď dialógom - rozhovorom so zadávateľom, alebo pomocou scenárov.

Preto, aby sme mohli systém použiť, musíme vedieť ako sa s ním manipuluje a prečo sa práve s ním pracuje takým spôsobom. Tieto vzťahy v systéme popisuje zdroj požiadaviek business procesy. Napríklad zadanie PSČ na formulári, jeho overenie a výpis chybového hlásenia. V prípade jeho správneho overenia nasleduje odoslanie formulára.

Obmedzenia sú ďalším zdrojom požiadaviek. Obmedzenie sa týka celej hierarchii požiadaviek. Príkladom je obmedzenie veľkosti prílohy.

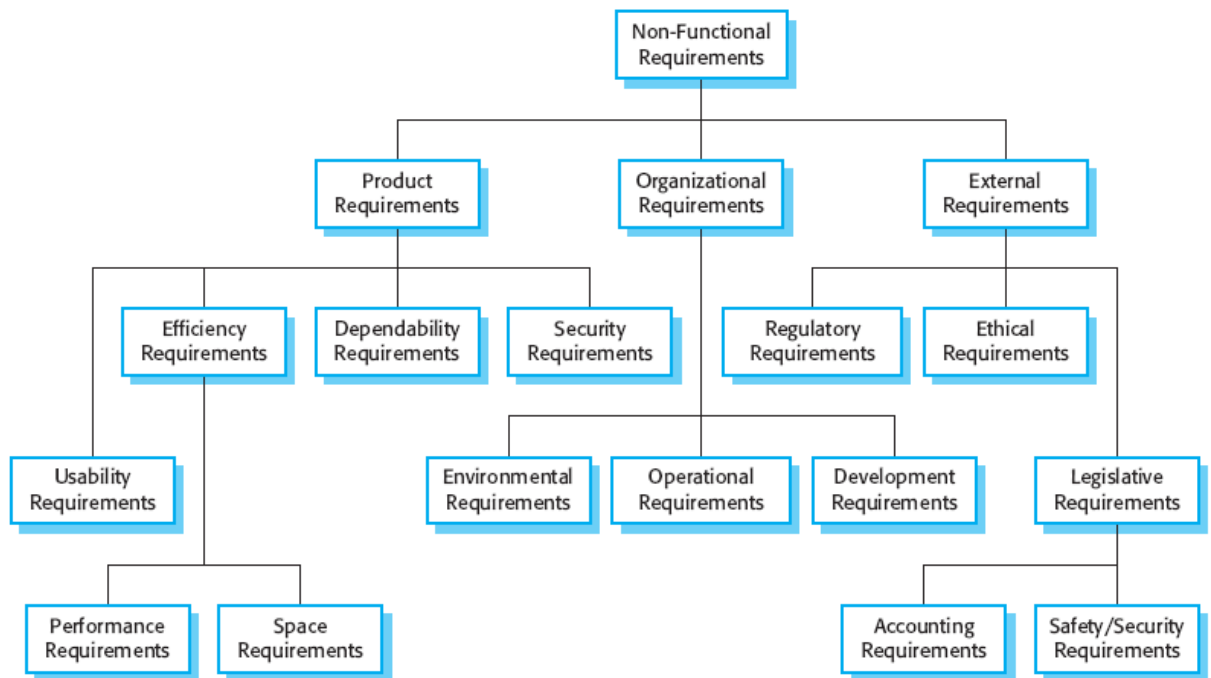
Neodmysliteľným zdrojom požiadaviek sú pravidlá. Pravidlá sú požadované a musia byť splnené. Napríklad: Formulár bude obsahovať 3 zaškrŕavacie políčka, ktoré obsahujú oprávnenia čítať, zapisovať a vymazávať na určitom formulári.

Posledným zdrojom požiadaviek je výkonnosť. Sem môžem zaradiť napríklad prenos po sieti, odozva databázového príkazu, atď. Požiadavky výkonnosti nájdem na každej úrovni návrhu.

Identifikáciu požiadaviek je vhodné urobiť v rôznych perspektívach. Dobrými perspektívami sú používatelia, zdroje a funkcionalita.

Získané požiadavky potom rozdelíme na dvojice funkčné, nefunkčné a užívateľské, systémové. Nefunkčné požiadavky sa týkajú: Produktu, Organizácie, alebo môžu byť externého charakteru. Ku kvantifikácii nefunkčných požiadaviek možno použiť atribúty: rýchlosť, veľkosť, jednoduchosť použitia, spoľahlivosť, robustnosť a prenositeľnosť.

Popis požiadavku obsahuje: funkciu, popis, vstup, zdroj, výstup, cieľ, akciu, požiadavok, podmienku vstupu, podmienku výstupu, vedľajšie pôsobenie.



Obrázok 2 - Nefunkčné požiadavky [17]

Scenár obsahuje: popis, činnosť, ostatné aktivity, stav systému pri ukončení.

Problémami pri identifikácii požiadaviek okrem komunikácie návrhára sú predpoklady, ktoré vychádzajú z toho, že zákazník predpokladal niečo a návrhár sa ho na to nespýtal. Ďalším z problémov je, že používatelia systému sú ľudia s odlišnými názormi. Jeden sa drží procesu, ďalší potrebuje veľa vecí zmeniť, ďalší odmieta dosiahnuť súhlasu, atď...

V praxi sú požiadavky pomerne nestále. Zákazníci stále prichádzajú s novými požiadavkami, alebo chcú zmeniť už definované požiadavky. Podľa štúdie firmy IBM pripadá v priemere 25% zmien v požiadavkách, čo je približne 70 - 85% prepisovania projektu. [9]

2.2.3 Use case model a model Aktivít

Use case model a model aktivít sa pri návrhu používajú často spoločne. Možno s nimi namodelovať funkčné ciele IS spoločne so zákazníkom.

Use case model popisuje vlastnosti, ktoré používatelia od systému očakávajú. Popisom modelu sú používatelia a ostatné prvky systému a ich vzájomná interakcia s cieľovou / kľúčovou vlastnosťou systému. Z jednej takejto interakcií môžeme vytvoriť niekoľko use case scenárov.

V use case modeli môžem použiť tieto väzby: asociácia, rozšírenie, generalizácia a vloženie. Asociácia sa používa pri komunikácii aktor - use case. Ostatné väzby sa používajú medzi jednotlivými use case.

Use case obsahuje: Systém, use case, aktori, assumpcie, vstupné podmienky, init, data, popis, odozva, ukončenie, výstupné podmienky a komentár.

Systém, use case a aktor sú základnými identifikačnými blokmi dôležité pre ďalší popis. Assumpcie obsahujú predpoklad o aktorovi, ktorý bude používať use case. Vstupné podmienky sú nutné k tomu, aby mohol aktor use case použiť. Init znamená spôsob, ako bude use case naštartovaná. Popis zaznamenáva výmeny správ medzi aktorm a use case. Ukončenie obsahuje zoznam akcií, ktoré ukončia use case. Výstupné podmienky určujú pre každé ukončenie podmienku, ktorá musí byť splnená. Komentár je nepovinné pole: Obsahuje ďalšie informácie, ktoré neboli spomenuté v predchádzajúcich poliach.

Model aktivít je podobný stavovému modelu. Na rozdiel od stavového modelu má model aktivít tzv. aktivity, pomocou ktorých namodelujeme workflow, use cases, business procesy, alebo vykonanie určitej operácie. Model obsahuje aktivačné stavy a prechody. Aktivačný stav čaká na dokončenie operácie. Pre rozdeleniu toku používam rozhodovania (decissions), na ktorý pripojím prechody s podmienkou. Do modelu aktivít môžem zahrnúť tzv. branching, mergeing, forking a joining, ktoré sa používajú pre prácu s viacerými vláknami. Aktivity je vhodné rozdeliť to swimlanes.

2.2.4 Model tried a model komponentov

Model tried obsahuje modelovacie elementy, ktorými popisujem veci. Medzi tieto modelovacie elementy patria triedy, rozhrania a dátové typy. Správanie entít môžeme vyjadriť týmito modelovacími elementmi: use case, aktor a komponenta.

Trieda je diskretným konceptom – popisom množiny objektov s podobnou štruktúrou, správaním a vzťahmi. Rozdiel medzi triedou a objektom možno popísať:

“The class defines the rules; the objects express the facts. The class defines what can be; the object describes what is.” [10]

Elementy triedy obsahuje tri časti: mennú časť, atribúty a operácie. Atribúty a operácie triedy obsahujú viditeľnosť pre iné elementy modelu tried. Viditeľnosť môže byť verejná (+), súkromná (-), chránená (#) a package (balíček) (~). Najextrémnejšie viditeľnosti sú verejné a súkromné. Chránené viditeľnosti sú verejnými viditeľnosťami, ale len pre

podtriedy nadradenej triedy, pre ostatné triedy sú súkromnými viditeľnosťami. Balíčky obmedzujú viditeľnosť pre elementy rovnakého balíčka. Popri viditeľnosti môžu atribúty triedy a operácie obsahovať statickosť a obmedzenia.

Atribúty triedy majú povinný dátový typ. Medzi dátové typy radíme čísla, reťazce a enumeračné dátové typy. Atribúty a Operácie majú i nepovinné vlastnosti: Atribúty môžu obsahovať znak ododenia (/) a predvolenú hodnotu. Operácie triedy okrem viditeľnosti obsahujú návratový dátový typ a parametre.

Rozhranie popisuje objekty. Obsahuje len operácie. Niekoľko tried, prípadne komponentov sa podieľa na realizácii rozhrania.

Najdôležitejšími vzájomnými vzťahmi medzi elementmi modelu sú asociácia, závislosť, tok, generalizácia, realizácia a použitie.

Asociácia definuje pravidlá, podľa ktorých budú objekty spojené. Určenie asociácie môže byť špecifikované jej názvom. Konce asociácie obsahujú násobnosť, viditeľnosť a názov rolí.

Asociácia môže obsahovať aj atribúty. Ak prvý objekt vyberá pomocou atribútu asociácie práve druhý objekt z množiny podobných objektov, potom sa jedná o kvalifikátor atribútu. V ostatných prípadoch sa atribúty asociácie zhrnú do asociačnej triedy.

Existuje i reflexívna asociácia, ktorá spája rovnaké objekty. Môžem ju použiť na modelovanie hierarchií.

Špeciálnym typom asociácii je agregácia a kompozícia. Agregácia mi umožňuje mať kontrolu nad určitou množinou objektov. Objekty - časti agregácie sú spojené s agregátom a spoločne tvoria skupinu závislých objektov. Silnejšou agregáciou je kompozícia, u nej od kompozitu závisia jej časti a to vrátane vzniku a zániku kompozitu.

Generalizácia udáva vzťah medzi nadradenými elementmi a podradenými elementmi. Špecifickejšie podradené elementy dedia vlastnosti zo všeobecných nadradených elementov. Pri dedičnosti určím atribút diskriminátora k identifikácii podelementov. Podelement môže dediť z dvoch nadelementov, potom píšem o viacnásobnej dedičnosti.

Generalizácia mi povolí polymorfizmus: v podtriedach môžem mať odlišnú implementáciu vybraných operácií. V nadradenej triede potom definujem abstraktnú operáciu, ktorá nemá implementáciu.

Komponentný model je vhodný k modelovaniu rozsiahlejšieho systému. Obsahuje komponenty a rozhrania a vzájomné väzby medzi nimi.

UML ešte obsahuje Sekvenčný model, Stavový model a Kolaboračný model, ktorými sa táto práca už nezaobera.

2.2.5 Implementácia triedy a rozhrania v C#

Triedu v C# vytvorím pomocou kľúčového slova **class**. Verejné atribúty sa označia kľúčovým slovom **public**, chránené majú **protected** a súkromné **private**.

Rozhranie vytvorím pomocou kľúčového slova **interface**. Rozhranie je možné vytvoriť refaktorovaním. Refaktorovanie ako nástroj Visual Studia popisujem v kapitole 4.1.3.

Asociáciou (tiež kompozíciou a agregáciou) zlepším celkovú abstrakciu triedy, pretože definujem asociovanú triedu ako atribút druhej asociovanej triedy.

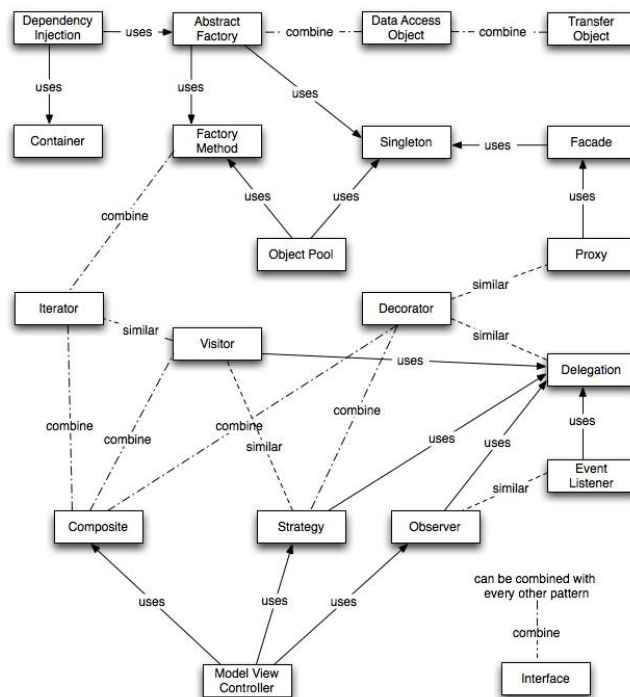
Generalizácia sa v C# prejaví odvođením podradenej triedy od nadradenej triedy. Názov podradenej triedy má tvar: **class** meno_podradenej_triedy : meno_nadradenej_triedy. V C# nie je možné implementovať viacnásobnú dedičnosť.

3 NÁVRHOVÉ VZORY

Návrhový vzor je znovupoužiteľné, opakované riešenie podobných problémov v návrhu a vo vývoji. Je to nápad, ako riešiť určitý problém, ktorý sa môže vyskytnúť vo veľkom počte odlišných situácií. Pred použitím návrhového vzoru sa musí správne rozhodnúť, či skutočne návrhový vzor pomôže daný problém riešiť. Návrhový vzor je treba správne premietnuť do návrhu, z návrhu potom do zdrojového kódu.

Návrhové vzory prišli z architektúry a antropológie.

Popis návrhového vzoru sa skladá z: Názvu návrhového vzoru, popisu významu návrhového vzoru (aký problém rieši) a ako dosiahneme riešenia, obmedzenia a hrozby, ktoré musíme zväžiť pred dosahovaním riešenia.



Obrázok 3 – Návrhové vzory [14]

Obrázok 3 ukazuje najznámejšie návrhové vzory a v nasledujúcej sekcii vám popíšem návrhový vzor observer.

3.1.1 Návrhový vzor observer

Návrhový vzor observer je vhodné použiť keď chceme oddeliť zodpovednosť objektov. V koncepcii návrhového vzoru observer sa nachádzajú dve strany: Subjekt na jednej strane a skupina Observerov na strane druhej. Observer sa definuje ako rozhranie, alebo

abstraktná trieda, ktorá ma len jednu operáciu – update. Rozhranie observera môže implementovať niekoľko tried, v ktorých bude implementácia operácie update.

Subjekt má skupinu observerov zaregistrovanú, pričom je vždy možné zaregistrovať, alebo odregistrovať observerov zo skupiny. Subject obsahuje jednu operáciu, nazvem ju notify, ktorá v každom observerovi zavolá operáciu update. Po zavoľaní sa každý observer informuje o údajoch subjektu, ktoré použije pre svoj ďalší pracovný postup.

Observer je vhodný návrhový vzor pre realizáciu MVC popísaného v kapitole 3.1.2.

3.1.2 Model View Controller (MVC)

Model View Controller je architektonický návrhový vzor, ktorý oddeľuje aplikačnú logiku od používateľského rozhrania.

MVC bol objavený v roku 1979 na projekte Smalltalku, ktorého názov je Xerox PARC. Autor projektu je Nórsky vedec a profesor Trygve Reenskaug. Jednalo sa o nástroje a obrazovky v prostredí Smalltalku.

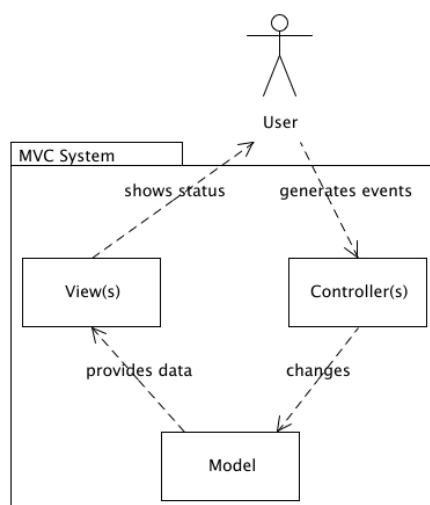
Každá časť MVC je separovaná od inej časti MVC a má určenú svoju zodpovednosť:

Model predstavuje štruktúru údajov. Model je zodpovedný za dáta a operácie s dátovým úložiskom. Model nesmie vykonávať operácie, a to generovať pohľady, alebo spracovávať klientské požiadavky. Model upozorňuje observera na zmeny údajov.

Kontroler obsluhuje požiadavky, ktoré prídu od klienta. Generuje pohľady, ktoré sú potrebné v danom momente. Ak je treba, vykoná operácie na modeli. Kontroler v žiadnom prípade nemá na starosti prácu s dátami a zobrazovanie pohľadov.

Pohľad vykreslí vhodný formulár, s ktorým bude používateľ interagovať. Môže existovať viacero pohľadov pre jeden model. Pohľad je automaticky upozornený modelom, kedy sa má obnoviť. Pohľad nie je zodpovedný za dáta a nesmie spracovávať požiadavky.

Výhodou je lepšia orientácia v projekte, možnosť zmeniť len jednu časť MVC bez modifikácie iných častí MVC a možnosť pridávať a generovať pohľady pre ďalšie (mobilné) zariadenia.



Obrázok 4 – MVC [14]

4 VÝVOJOVÉ PROSTRIEDKY

Na vývoj webovej aplikácie diagnostiky som vybral jazyk C# a platformu .NET od Microsoftu na strane serveru, Javascript na strane klienta a databázové úložisko MySQL.

4.1 C# a platforma .NET

Pod C# a .NET už programujem dlhší čas a za tú dobu som získal veľa skúseností. Preto bol pre mňa .NET jasné rozhodnutie. O OOP v jazyku C# nájdete v sekcii 2.2.5. V ďalších podkapitolách sú popísané jednotlivé programovacie techniky použité v projekte.

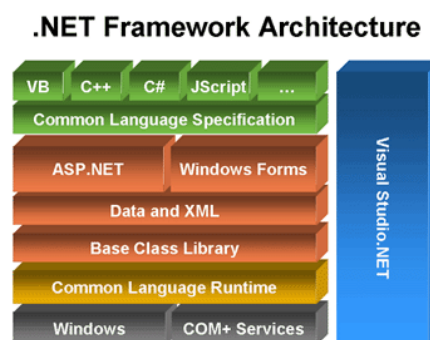
4.1.1 C#

C# je objektovo orientovaný programovací jazyk podobný C++. Hlavný architekt jazyka je Anders Hejlsberg, ktorý je tiež tvorca programovacieho jazyka Turbo Pascal a architekt programovacieho jazyka Delphi.

C# bol vytvorený primárne pre produkty operačného systému Microsoft Windows, ale môže multiplatformovo bežať na Linuxe a MAC OS. Umožní to projekt Mono, ktorý má vlastný prekladač a prostredie.

Platí, že v programoch C# sa nedealokuje pamäť. O uvoľnenie pamäte sa stará GC, ktorý nepoužívané objekty po určitom čase uvoľní.

4.1.2 .NET framework



Obrázok 5 – Architektúra .NET

.NET framework je prostredie pre vývoj, nasadenie a beh aplikácií. Programovanie je založené na objektovo orientovanom prístupe. .NET umožňuje programovať vo veľkom množstve programovacích jazykoch (VB, C++, C#, ...). Spoločné vlastnosti týchto jazykov definuje CLS. Každá verzia .NET frameworku má základnú knižnicu (Base Class Library) na ktorú hierarchicky nadväzujú ďalšie technológie. Od verzie .NET frameworku

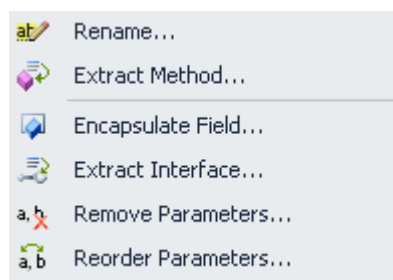
sa odvíja aj celkový počet technológií. Každá technológia má svoje menné priestory, ktoré sa do projektov .NETu pridávajú pomocou kľúčového slova **using**.

Nasleduje JIT kompilácia do byte kódu. CLR umožní beh aplikácie a ďalšie nevyhnutné služby, ako GC, typovú bezpečnosť a spracovanie výnimiek.

4.1.3 Microsoft Visual Studio

Microsoft Visual Studio je vynikajúci editor zdrojového kódu. Používa Intellisense - zvýrazňuje kľúčové slová a pri písaní zdrojového kódu zobrazuje pomôcky formou tooltipu (napr. pri písaní argumentov funkcií). K rýchlemu písaniu kódu pomáha veľký počet klávesových skratiek. Chyby a varovania sú podčiarkované už behom písania zdrojového kódu.

Visual Studio má v možnosť refaktoringu.



Obrázok 6 - Refaktoring

Rename umožní premenovať symbol na viacerých miestach v projekte, ale ešte pred premenovaním ukáže náhľad zmien k schváleniu programátorom. Vyberiem si časť kódu, zvolím extract method, a Visual Studio presunie vybranú časť kódu do statickej metódy. Pri presúvaní zistí premenné, ktoré sa vo zvýraznenom kóde nachádzajú a zadefinuje ich ako argumenty metódy. Encapsulate field vytvorí verejný člen vlastnosti (property) z verejného člena triedy, ktorý hneď po vytvorení vlastnosti zmení verejný atribut na súkromný (zapuzdrí ho v triede). Do tejto vlastnosti môžem napísať overovací zdrojový kód, prípadne obmedziť „getter / setter“ vlastnosti atribútu len na čítanie. Extract interface vytvorí rozhranie. Z dialogového okna vyberiem metódy z tried, ktoré má obsahovať rozhranie. Remove parameters a Reorder parameters umožnia narábať s parametrami metódy, pred usporiadaním (zmazaním) ukáže Visual Studio náhľad zmien.

Projekty Visual Studia môžu byť napojené na verzovacie systémy (Visual SourceSafe, SVN, TFS).

Debugger Visual Studio umožňuje krokované v kódě, náhled na hodnotu proměnné v danom okamžiku, zmenu proměnné při ladení, procházení prvků kolekcí, generování výsledku LINQ dotazů. Při spuštění ladenia webovej aplikácii sa dá do činnosti integrovaný webový server, ktorý umožní pracovať s webovou aplikáciou takmer ako na reálnom webovom serveri.

4.1.4 ASP.NET

ASP.NET je technológiou .NET frameworku pre generovanie webových stránok. Na webovom serveri sa vytvorí webová stránka a tá sa v čistom HTML odošle na klienta. ASP.NET je spojením starších ASP s princípmi .NET frameworku. Základným prvkom je Web Form, ktorá obsahuje (X)HTML prvky a tzv. placeholders v <% %>. Každá ASPX stránka obsahuje kód v pozadí napísaný v C#.

4.1.5 ADO.NET Entity Framework

ADO.NET entity framework mapuje kolekciu objektov na databázové tabuľky. Umožní previesť metódy na objekte na sériu SQL príkazov. Za spracovanie mapovania je zodpovedný Entity Client Data Provider. S jednotlivými druhmi databáz (MS SQL, Oracle, MySQL, ...) komunikuje ADO.NET dáta provider. MySQL má dáta provider pre ADO.NET k stiahnutiu na svojich webových stránkach.

Je lepšie pracovať s kolekciou objektov, ako skladať SQL príkazy a posielať ich oproti databázy.

Príklad použitia ADO.NET entity frameworku:

```
User user = new User();
user.Name = "Tomas";
db.users.AddObject(user);
db.SaveChanges();
```

Je zrozumiteľnejší, ako jeho ekvivalent v klasickom ADO.NET (excerpt kódu):

```
SqlCommand sq = new SqlCommand( "INSERT INTO users(Name) VALUES('Tomas')",
connection);
```

4.1.6 Prúdový vstup a výstup

Pri ukladaní veľkého množstva dát sa často volí medzi ukladaním do databáze (BLOB) a ukladaním na pevný disk na aplikačnom serveri. Častejšie býva uprednostnená databáza, pretože sú dobre vyriešené scenáre s mnohopoužívateľským prístupom. Ak si zvolím

ukladanie do súborového systému, musím zaistiť, aby sa súbor na server uložil vždy pod iným menom. Predídem tak konfliktom pri nahrávaní súboru na server.

Súbormi a prúdmi sa v .NET zaoberá menný priestor System.IO. V projekte som použil triedy DriveInfo, DirectoryInfo, FileInfo, File a Path.

Triedy File a FileInfo pracujú so súbormi. DirectoryInfo a FileInfo obsahujú i podobné vlastnosti a metódy. DirectoryInfo pracuje s adresármi.

Triedy FileInfo (DirectoryInfo) dokážu:

- zmeniť atribúty súboru (adresáru),
- zistiť dátumy, kedy boli vytvorené, alebo zmenené súbory (adresáre),
- zistiť existenciu súboru (adresáru),
- získať meno, alebo príponu súboru (adresáru),
- získať súbor (adresár) s kompletnou cestou,
- zmazanie, vytvorenie, premiestnenie, alebo obnovenie súboru (adresára).
Obnovenie súboru znamená zistenie aktuálneho stavu (používateľ mohol urobiť nejakú zmenu v prehliadači súborov).

Trieda DirectoryInfo najviac dokáže:

- zistiť koreňový adresár, alebo rodičovský adresár,
- vytvoriť podadresár,
- zistiť súbory a adresáre na určitej ceste.

FileInfo dokáže zistiť:

- adresár, v ktorom súbor leží,
- veľkosť súboru v bytoch,

FileInfo môže skopírovať súbor a uskutočniť prúdové operácie: Otvoriť pre čítanie, otvoriť pre zápis.

Hlavný rozdiel medzi triedami File a FileInfo je v tom, že File má statické metódy, ktoré FileInfo nemá. To znamená, že metódy File môžem volať bez toho, aby som musel vytvoriť instanciu triedy.

Trieda DriveInfo poskytuje informácie o pevných diskoch a iných médiách nainštalovaných na PC. Trieda DriveInfo mi zistí:

- veľkosť disku,
- voľné miesto na disku,
- formát disku (NTFS, FAT32...),
- typ disku (pevný, CD-ROM),
- či je disk pripravený na zápis,
- popisok disku (label),
- koreňový adresár disku,
- všetky dostupné jednotky na danom PC.

Path narába s cestou k súboru (adresáru):

- Dokáže správne poskladať časti cesty do celej cesty,
- zo vstupnej cesty zistiť názov súboru, názov adresára, alebo koreňový adresár.
- Odpovie na otázku, či cesta je absolútna, alebo relatívna, alebo či cesta končí príponou súboru.

Ukážka práce so System.IO je v praktickej časti.

4.1.7 Reflexia

Reflexia znamená prezeranie, alebo zásah do metadát preloženého kódu za behu aplikácie. Menný priestor .NETu je System.Reflection. Vnútro preloženého kódu obsahuje podobné informácie zdrojového kódu, ale skrátene o množstvo nepotrebných informácií, ktoré strojové spracovanie nepotrebuje. Spomenuté informácie sú hlavne komentáre a názvy premenných.

S reflexiou dokážem napríklad zavolať dynamicky vytvorenú metódu, prípadne dekompilovať strojový kód na zdrojový kód, atď.

Zameriam sa na zistenie a zmenu metadát uložených v skúmanom objekte, ktorú používam na vytvorenie dynamických tabuliek.

Pomocou GetType() dostanem referenciu na object Type a z neho zistím kolekciu vlastností (property) objektu, pomocou GetProperty() metódy. Postupným prechádzaním prvkov kolekcie dostanem meno a dátový typ každého prvku kolekcie. Týmto spôsobom zistím, aké má trieda implementované členy (metódy, vlastnosti a členy).

Ak mám zistenú vlastnosť cez Type.GetType(), môžem vzdialene načítať hodnotu vlastnosti z objektu (instancie triedy), ktorá danú vlastnosť obsahuje. Dosiachnem toho

zavolaním `GetValue()` metódy, do argumentu metódy uvediem priamo objekt, ktorý je instanciou triedy a metóda vráti požadovanú hodnotu. Takmer analogicky zmením property s použitím `SetValue`, kde sa pripisuje navyše jeden argument s hodnotou, ktorú chcem nastaviť.

4.1.8 XML a serializácia

XML je značkový jazyk pre štruktúrované elektronické dokumenty. Je to otvorený štandard vydaný W3C. Má veľmi podobnú syntax, ako jazyk HTML. Na rozdiel však od HTML XML používa vlastné značky (tiež tagy, elementy) navrhnuté tvorcom XML dokumentu. XML má striktnú syntax, tj. musí mať správny tvar (well formed) a musí byť správny (valid). Well formed znamená, že všetky tagy sú uzavreté a uzavretia sú správne usporiadané. XML je Valid vtedy, ak spĺňa všetky platné pravidlá pre XML (XSLT, DTD, DOM, XHTML, HTML, SGML, SMIL a XML-RPC) [20].

XML je v dnešnej dobe naozaj veľmi používané. Je to formát súboru vhodný pre import určitej časti systému z časti iného systému. Formáty Open Office, či novšie Microsoft Office (DOCX, XLSX...), sú komprimovanými štruktúrami XML dokumentov (ZIP). Používa sa ako výmenný formát medzi webovými službami.

Uvediem možnosti ako v .NETe spracovať XML súbor: LINQ pre XML, použitie tried `XmlDocument(XmlReader, XmlWriter)` a XML serializácia. Vystačím si s XML serializáciou, pretože v projekte budem používať jednoduché XML, ktoré vytvorím z kolekcie objektov.

Serializácia transformuje celý živý objekt do prúdu a tento prúd uloží do súboru na médium. Táto transformácia je obojstranná. Zo súboru môžem dostať prúd a z prúdu živý objekt.

Postup serializácie / deserializácie:

- Vytvoriť triedu k serializácii;
- Vytvoriť instanciu triedy `XmlSerializer` a do konštruktoru dať typ serializovanej triedy (prípadne typ kolekcie serializovanej triedy);
- Vytvoriť prúd pre čítanie/zápis;
- Zavolať metódu `Serialize/Deserialize`, ktoré pracujú s prúdom a serializovaným objektom.

XML Serializácia je zahrnutá v System.Xml.Serialization.XmlSerializer. Každý objekt má svoj predpis - triedu. Ak chceme serializovať objekt, musíme triedu označiť atribútom [Serializable]. Predvolene serializer serializuje vlastnosti objektu ako XML element. Môžem však použiť [XmlAttribute] atribut pre XML atribut, alebo [XmlText] pre XML Text (to je obsah medzi root elementami). Ďalšie atributy sú [XmlElement], [XmlAttribute], [XmlRoot] a [XmlType].

4.1.9 Lambda výrazy

V .NET verzií 3.0 boli predstavené Lambda výrazy. Lambda výraz má tvar: (parametre) =>(cieľ delegáta) (možno čítať parametre do cieľu delegáta). Pre jednoduché lambda výrazy je možné vynechať zátvorky. Príklad:

```
var z = 1;
Func<int, int, int> b = (x, y) => (x * y * z);
z = 0;
var d = b(2, 3);
```

Lambda výraz v príklade je $(x, y) \Rightarrow (x * y * z)$. Aká bude hodnota premennej d po zavolaní b(2, 3)? Odpoveď na otázku nájdete v kapitole 4.1.10.

4.1.10 Delegáti a anonymné metódy

K pochopeniu Lambda výrazov potrebujem poznať Delegáta. Delegát je objekt, ktorý odkazuje na (jednu/viac) cieľových metód. Delegát je možné porovnávať s callbackom, ktorý v .NET stále existuje, nie je však typovo bezpečný. Callback umožňuje v aplikáciách reagovať na rôzne udalosti, napríklad používateľské rozhranie aplikácie. Majme:

```
static string Ciel(int cislo) { return cislo.ToString(); }
delegate string Presmerovac(int cislo);
Presmerovac p = Ciel;
String vysledok = p(3);
```

V tomto príklade vidieť, akým spôsobom funguje delegát a jeho odkazovanie. Pri zavolaní p(3) sa nakoniec zavolá Ciel(3), pretože p je typu Presmerovac a ten ukazuje na metódu Ciel.

Predošlý príklad môžem zapísať s použitím anonymnej metódy, vynechávam Ciel:

```
delegate string Presmerovac(int cislo);
Presmerovac p = delegate(int cislo) { return cislo.ToString(); }
```

Teraz prepíšem Lambda výraz $\text{Func}<\text{int}, \text{int}, \text{int}> b = (x, y) \Rightarrow (x * y * z)$;

z predošlej kapitoly do výrazu obsahujúceho delegáta:

```
var z = 1;
Func<int, int, int> b = delegate(int x, int y) { return(x * y * z); };
z = 0;
var d = b(2, 3);
```

Pointa je v tom, že pri zavolaní `b(2, 3)` sa zavolá telo anonymnej metódy, ktorá je zároveň cieľovou metódou delegátu. Premenná `z` je globálnou premennou pre anonymnú metódu a výsledok `b(2, 3)` je v tomto príklade 0.

Anonymnú metódu s Presmerovačom môžem zapísať aj v tvare: `(int cislo) => { return cislo.ToString(); }`

`Func<int, int, int>` udáva špeciálny typ delegáta, kde v špicatých zátvorkách sú uvedené generiká (generics).

4.1.11 LINQ

Skratka LINQ znamená Language Integrated Query. V projekte obrazového systému sa LINQ používa pri práci s databázou a pri zobrazovaní jednotlivých vrstiev súčiastok stroja.

Vnútorňou hybnou silou LINQu sú už spomenuté delegáti a často sa používajú aj lambda výrazy. LINQ pracuje hlavne s kolekciami. Tieto kolekcie modifikuje pomocou tzv. objektovými dotazmi (Object queries) a rozširujúcimi metódami (Extension methods). Každá kolekcia, ktorá je implementáciou rozhrania `IEnumerable` obsahuje rozširujúce metódy LINQu.

Syntax objektových dotazov je podobná SQL, ale funkcionálna LINQu sa blíži funkcionálnemu programovaniu.

Jednotlivé sekcie kódu, kde sa evidentne rozrastá LINQ, komunikujú medzi sebou cez rozhrania `IEnumerable` a `IQueryable`. Výsledný objektový dotaz musím previesť na kolekciu a tak dostanem konkrétne výsledky.

Bez LINQu by som musel prácne prehľadávať záznamy v iteráciách a celková čitateľnosť kódu by sa znížila. Zvýšila by sa aj zložitosť kódu na pochopenie pre budúcich programátorov, ktorí po mne kód prevezmú.

Na nasledujúcom obrázku vidieť prehľad LINQ. Príkazy LINQu sa radia do skupín (Projekcia, Triedenie, Filtrácia, Zoskupovanie, Agregácia, ...). Každá skupina obsahuje maximálne 4 body. V prvom bode je znázornený vstup a výstup operácie. V druhom bode je objektový dotaz. Tretí a až e bod obsahuje rozširujúce metódy a v poslednom bode sú niektoré vhodné metódy použiteľné k filtrácií.

Objektový dotaz, i doplňujúce operácie majú v svojich definíciách rozhranie IEnumerable s generikom (popísaný v kapitolách 4.1.15), typy TSource, TResult, TKey, bool, int, TElement. Treba dosadiť vždy objekt správneho typu do 2. alebo 3. bodu a dostanem správny LINQ dotaz/rozširujúcu metódu.

TInner, TOuter a TInnerKey a TOuterKey sú podobné TSource a TKey. Používajú sa pri spojení dvoch objektov rozhrania IEnumerable, pričom sa spoja pomocou kľúčových vlastností zdrojových objektov (vnútorného a vonkajšieho).

TSource je typ zdrojového objektu na ktorý aplikujem LINQ. TKey je typ objektu, podľa ktorého sa bude triediť, zoskupovať, alebo spájať. TSource môže byť napríklad objekt Zákazník a TKey meno zákazníka.

TElement a TResult sú výsledkom projekcie. TElement je uložený do určitej skupiny pri zoskupovaní. TResult je typu objekt, ktorý vznikne použitím selektora. Môže byť rovnaký typ, ako TSource, ale nemusí. TResult môže byť typom vlastnosti objektu TSource, alebo úplne vytvorený nový objekt, ktorý má spoločné/podobné vlastnosti s objektom typu TSource.

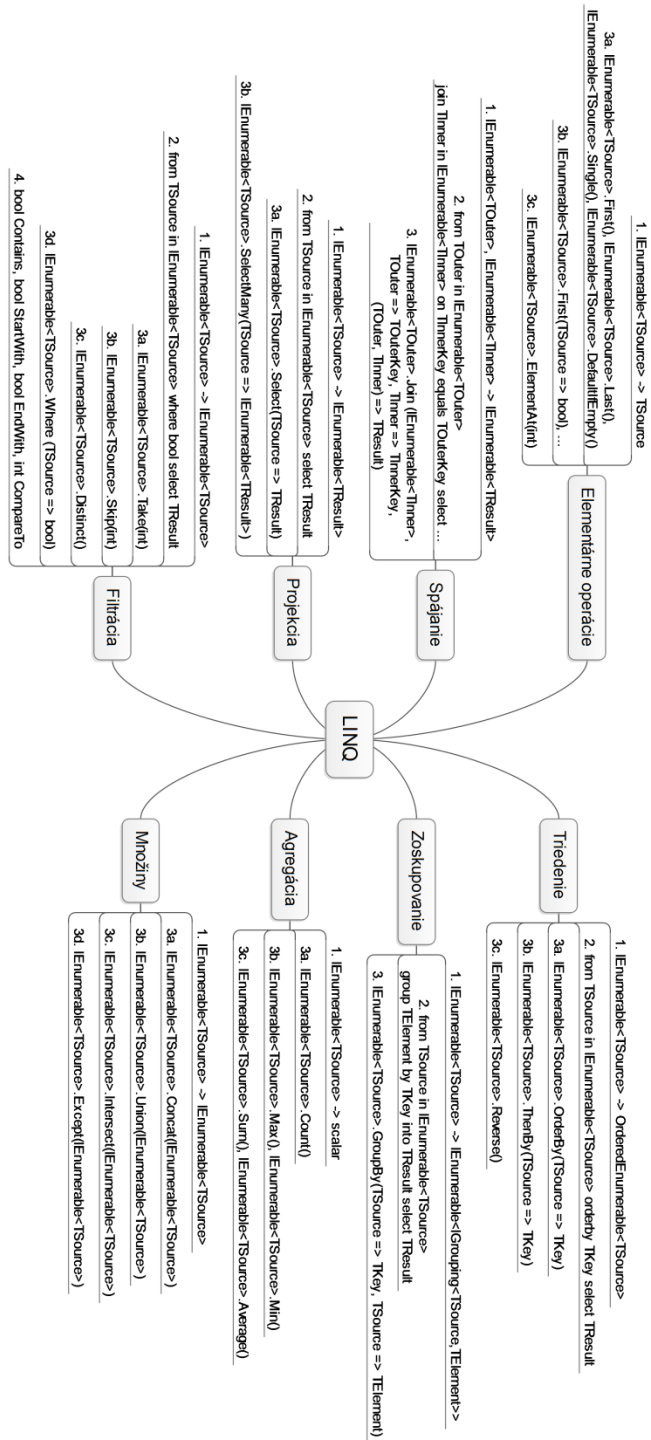
Rozširujúca metóda where vo filtrácii potrebuje objekt typu bool, alebo podmienku. Často sa používa podmienka, spadajúca pod typ TSource. Majme:

```
class Zakaznik { string name {get; set; }}
List<Zakaznik> zakaznici = new List<Zakaznik>();
zakaznik.Add(new Zakaznik() { name="Mrkvicka" });
var zakaznik = from z in zakaznici where z.name.EndsWith("a") select z.name;
```

V tomto príklade je typ Zakaznik chápaný všeobecne ako TSource. Premenná z vo **from .. in .. select** klauzuly je typu Zákazník. Za **where** kľúčovým slovom nasleduje podmienka, ktorá vyfiltruje kolekciu (vstupný List) zákazníkov. Za kľúčovým slovom **select** definujem, čo konkrétne chcem mať v premenne zákazník. Výsledok dotazu bude typu **string** a to znamená všeobecný typ TResult.

Príklad môžem zapísať aj ako rozširujúcu metódu s použitím lamda výrazu:

```
var zakaznik = zakaznici
.Where(z => z.name.EndsWith("a"))
.Select(z => z.name);
```



Obrázok 7 - LINQ

4.1.12 GUID

GUID je náhodne generovaný a je (skoro isto [2]) unikátny na celom svete. Generuje sa zavolaním: `System.Guid.NewGuid().ToString();`

4.1.13 Anonymné typy

Anonymný typ je jednoduchý dátový objekt na uloženie údajov. Vytvorím ho bez toho, aby som musel definovať triedu. Anonymný typ definujem výlučne pomocou kľúčového slova **var** a **new**.

Napríklad:

```
var software=new { company="Microsoft", title="Windows", version=8 };
```

Po deklarácii objektu anonymného typu `software` môžem s ním pracovať úplne tým istým spôsobom, ako by bol vytvorený inštanciou triedy.

Anonymný typ prekladač preloží on-the-fly do internej triedy (internal class).

4.1.14 Rozhranie IEnumerable

Rozhranie `IEnumerable` obsahuje len jednu metódu `IEnumerator GetEnumerator()`. Ak objekt nie je implementovaný rozhraním `IEnumerable`, nemôže byť použitý ako výčtová kolekcia v príkaze **foreach** a neobsahuje ani rozširujúce metódy. Kľúčové slovo **foreach** znamená iteráciu, kde pre každý prvok vstupnej kolekcie sa časť kódu uzavretá vnútri iterácie vykoná práve jeden krát. Majme:

```
int [] ipAddress = new int [] { 127, 0, 0, 0 };
StringBuilder sb = new StringBuilder();
foreach(int byte in ipAddress) sb.AppendFormat("%s.", byte);
```

Iteráciu `foreach` môžem prepísať pomocou `GetEnumerator()`:

```
using (var enumerator = ipAddress.GetEnumerator())
while (enumerator.MoveNext()) sb.AppendFormat("%s.", enumerator.Current);
```

Rozhranie `IEnumerator` má okrem členov `MoveNext()` a `Current` ešte aj metódu `Reset()`, ktorá vráti počítanie enumerátora na začiatok.

4.1.15 Generiká

“C# provides generics to remove the need for casting, improve type safety, reduce the amount of boxing required, and make it easier to create generalized classes and methods.”

[16]

V špicatých zátvorkách sa napíše typ, aký má generická trieda akceptovať, napr.:

```
List<int> ListCisel; // kolekcia (trieda) List pracujúca s int  
List<Zakaznik> zakaznici; // kolekcia (trieda) List pracujúca so Zákazníkom
```

Výhoda generík je v tom, že pri volaní metód triedy Listu sa nemusí spraviť boxing premenných vstupujúcich do argumentov metód. Podobne kompilér vie lepšie rozoznať, aké typy sa v zdrojovom kóde používajú a nedochádza k chybám pri pretypovaní.

4.2 Javascript

Webový server vygeneruje a pošle klientovi webovú stránku. Obsah vygenerovanej webovej stránky spracuje webový prehliadač, ktorý má používateľ nainštalovaný na svojom zariadení. Môžu to byť zariadenia od PC po TV až po mobilný telefón. Vygenerovaná webová stránka môže obsahovať aj skript v interpretovanom programovacom jazyku, s ktorým dokáže webový prehliadač pracovať. Príkladom takéhoto programovacieho jazyka je Javascript, AJAX, VB script... V práci sa zameriam na Javascript.

Javascript je interpretovaný, klientsky, multiplatformný a veľmi rozšírený skriptovací jazyk. Kód napísaný v Javascripte beží na Windowse, na Linuxe aj na MACu.

Programovanie v Javascripte je pomerne náročné. Vystačím si s webovým prehliadačom a textovým editorom. Textový editor by mal rozoznať syntax a zvýrazniť kľúčové slová. Napísaný program nie je treba prekladať.

Kód napísaný v JS väčšinou pozostáva z funkcií, ktoré sú volané z udalostí na webovej stránke, alebo z iného miesta v JS.

K HTML tagom, ktoré potrebujem riadiť pomocou JS, priradím jednoznačné identifikátory pomocou id atribútu. Aby som mohol JS použiť, definujem udalosť vhodnému HTML tagu. Pri vyvolaní udalosti používateľom sa zavolá funkcia napísaná v JS.

Jednou z najužitočnejších funkcií JS považujem getElementById(id), ktorá mi vráti JS DOM objekt, ktorý potom môžem modifikovať.

4.3 MySQL

MySQL je relačný databázový systém pod GNU licenciou. Je jeden z najpoužívanejších databázových systémov. Používa ho wikipedia, facebook, google, twitter. Pre prácu s údajmi uloženými v databáze je konštruovaný jazyk SQL. Jazykom SQL a databázou MySQL sa viac zaoberal Daniel Guryča v DP, na ktorú nadväzujem.

II. PRAKTICKÁ ČÁST

5 NÁVRH

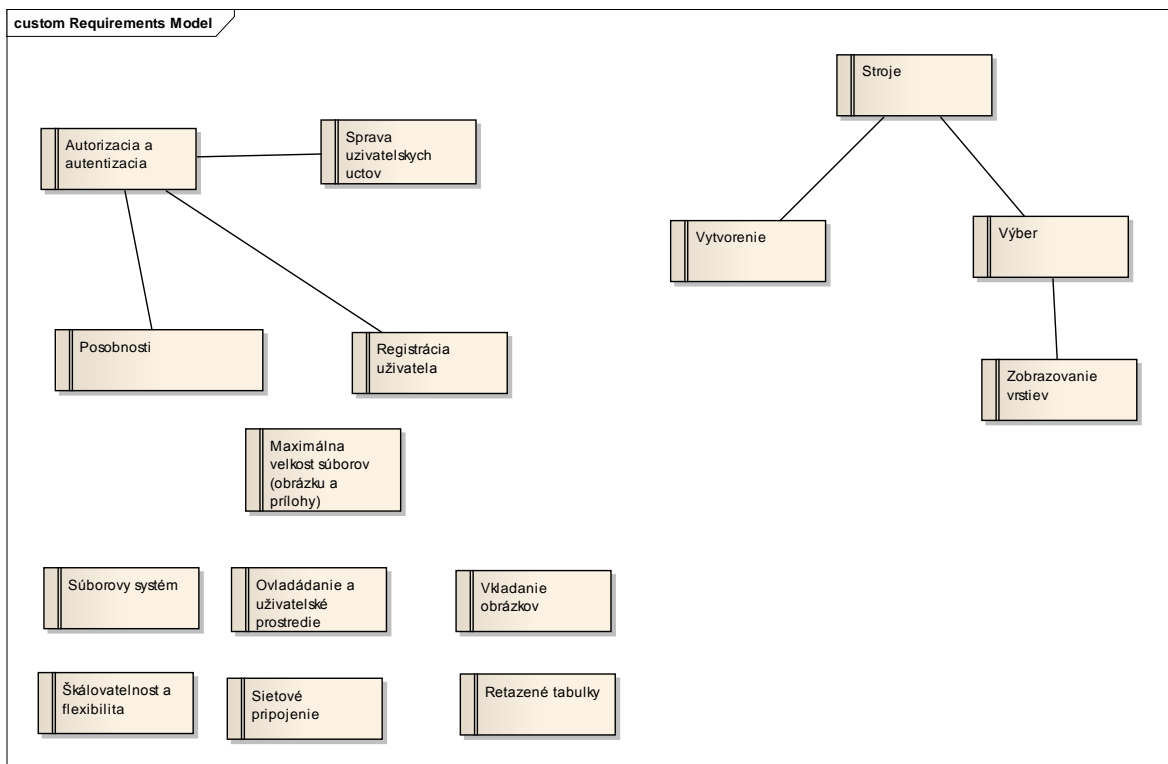
Pre návrh som zvolil jazyk UML. V praktickej časti je sú už navrhnuté konkrétne modely: Deployment model, Komponentový model, Model Požiadavkov, Use case model a Statický (triedy) model obrazového informačného systému pre diagnostiku.

V práci sa uplatňuje najviac vodopádový a inkrementálny model. Inkrementálnosť je vyjadrená neustálym vyvíjaním prírastkov projektu (aplikácia, návrh, dokumentácia, práca) a následné uloženie do SVN. V ostatnom je práca výsledkom vodopádového modelu.

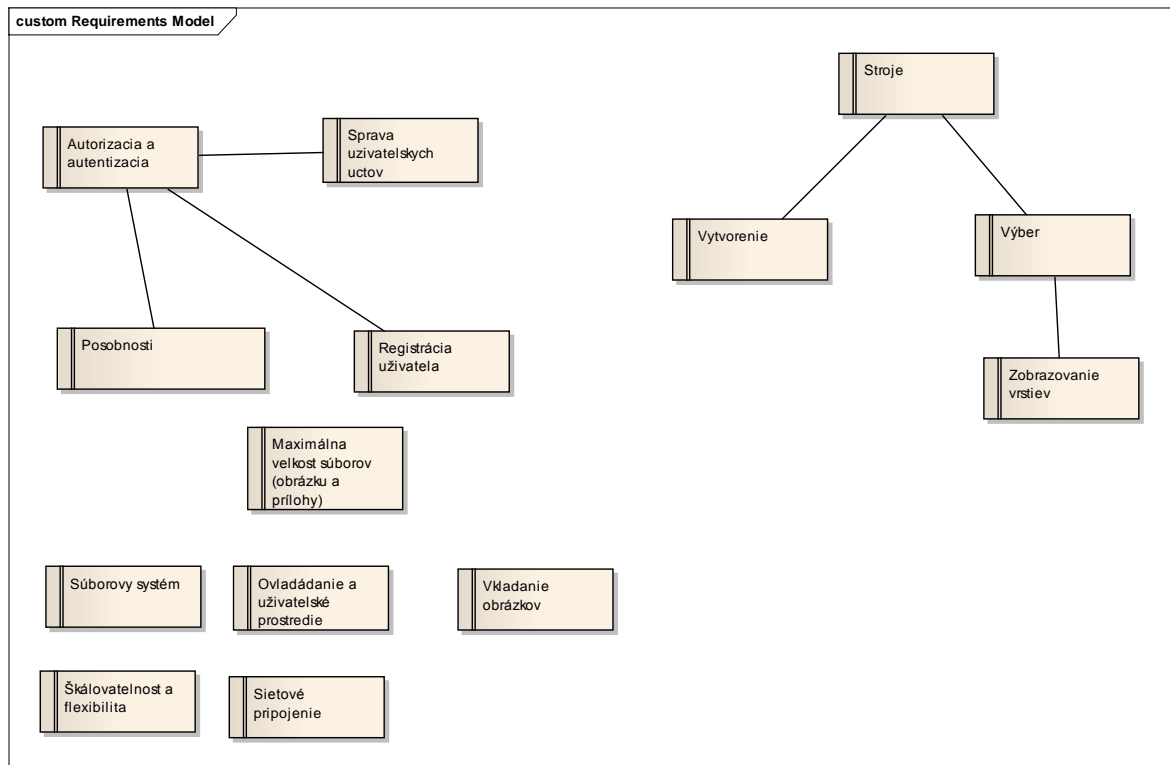
Znovupoužiteľnosť je zaistená použitím architektonického návrhu MVC. Je oddelená prezentačná časť od dátovej časti.

Základným popisom jazyka UML a modelov sa venujem v teoretickej časti práce.

5.1 Požiadavky



Obrázok 8 – Model požiadavkov 1



Obrázok 9 – Model požiadavkov 2

Požiadavky udávajú, čo zákazník očakáva od systému.

5.1.1 Popis požiadaviek

Autorizácia a autentizácia – *Funkčný, Používateľský* - Bude prístupná len časť systému pre oprávnené osoby a firmy.

Editácia formulárov - *Funkčný, Používateľský* – Je možnosť editovať formuláre používateľom oprávneného k editácii formulára. Každý formulár bude validovať vstupné hodnoty validátorom, ktorý užívateľa upozorní na zadanie nesprávnej hodnoty.

Maximálna veľkosť súborov (obrázku a prílohy) - *Nefunkčný, Používateľský, Produkt* – Bude najviac 2MB.

Ovládanie a používateľské prostredie - *Funkčný, Používateľský* - Obsahuje: Stroje, moduly, súčiastky, prílohy, skúsenosti, obrázky.

Pôsobnosti – *Funkčný, Používateľský* - Možnosť definovania používateľských rolí a náležitých oprávnení.

Prehľad aktuálnych dát (strojov) v systéme - *Funkčný, Používateľský* - Zobrazenie strojov a iných dát uložených v databáze a systéme.

Prílohy – *Funkčný, Používateľský* - K súčiastkam bude možné nahrať súbory.

Riešenie pre jednotlivé moduly – *Funkčný, Používateľský* - Diagnostik bude môcť pridať / zmeniť riešenie poruchy na danom module.

Schopnosť vetviť – *Funkčný, Používateľský* - Vetvenie súčiastok stroju do vrstiev.

Sieťové pripojenie – *Nefunkčný, Používateľský, Produkt* - Z dôvodu vysokého dátového toku medzi klientom a serverom (práca s obrázkami a prijímanie/odosielanie dát z klientskych počítačov).

Správa používateľských účtov - *Funkčný, Používateľský* - Administrátor môže zobrazit' a zmeniť účty používateľov.

Súborový systém - *Funkčný, Používateľský* - Vhodné použiť výkonný súborový systém pre ukladanie obrázkov.

Zobrazovanie vrstiev - *Funkčný, Používateľský* - Po kliknutí na obrázok vyššej vrstvy server pošle klientovi obrázok vyššej vrstvy. Bude možné sa vrátiť nazad na nižšiu vrstvu.

Zobrazovanie výsledkov - *Funkčný, Používateľský* - Možnosť zobrazit' nadradený formulár používateľom oprávneného k zobrazeniu formulára.

Škálovateľnosť a flexibilita - *Nefunkčný, Používateľský, Produkt* –

Systém bude ďalej obsahovať tieto súčasti: **Firmy, Inštalácie, Nástroje a príslušenstvo, Poruchy, Registrácia používateľa, Správa používateľských účtov, Stroje, Vkladanie obrázkov, Vytvorenie a výber stroja** - *Funkčné, Uživatelské*.

5.2 Deployment model

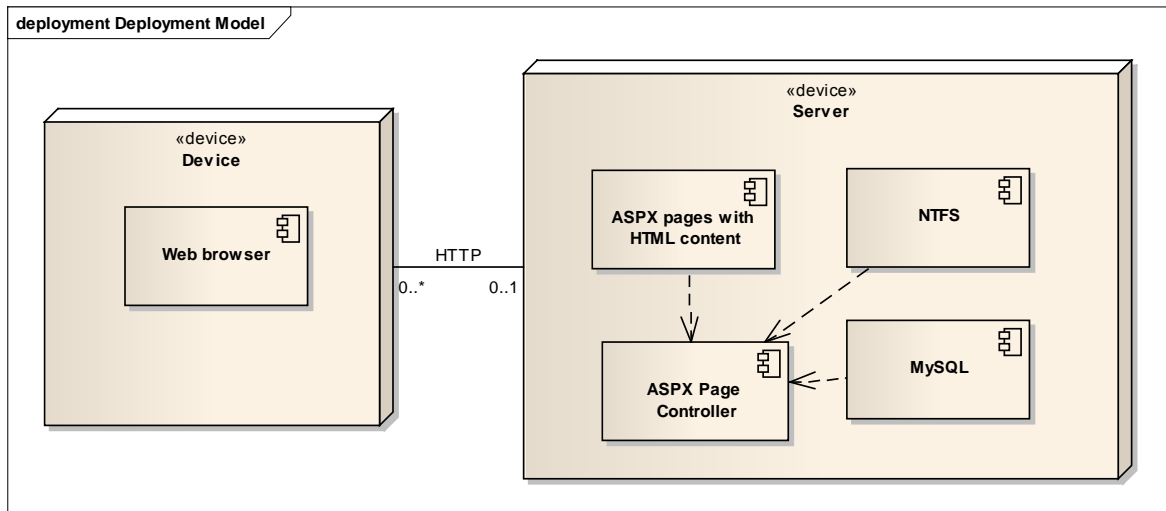


Figure 10 – Deployment model

Deployment model ukazuje prostredia, kde bude aplikácia pôsobiť (bude aplikácia zavedená).

5.2.1 Popis deployment modelu

- Device – Je každé zariadenie, ktoré má webový prehliadač (Web browser) a podporu javascriptu s možnosťou ukladania cookies. Príkladom zariadenia je počítač, mobilné zariadenie, tablet, atď... Toto zariadenie komunikuje so serverom cez protokol HTTP.
- Server – Vzdialený počítač, kde je zavedená webová aplikácia, súborový systém a databázový systém. Server zaisťuje všetky služby a pripojené zariadenia sú na ňom závislé. Pri výpadku serveru (pri zrútení serveru), sa prepojenie preruší a klienti nebudú môcť s IS pracovať.
- ASPX Page Controller - Zaisťuje komunikáciu s klientom pomocou udalostí (events). Udalosťou rozumiem interakciu klienta s určitým prvkom webovej stránky. Klient napríklad po stisnutí tlačítka na webovom formulári vyvolá udalosť.
- ASPX pages with HTML content – Predstavuje vygenerované webové stránky posielané na klienta.
- MySQL – Databázový systém, kde sa získavajú, alebo ukladajú dáta.
- NTFS – Súborový systém na serveri pre prácu s obrázkami, prílohami a XML.

5.3 Komponentový model

Cieľom komponentového modelu je znázorniť jednotlivé moduly a ich vzájomné vzťahy. Jednotlivé moduly závisia jeden na druhom. Komponenty môžu byť spustené, ako spustiteľné súbory (EXE), alebo obsiahnuté ako zdrojové súbory v projekte za kompilácie.

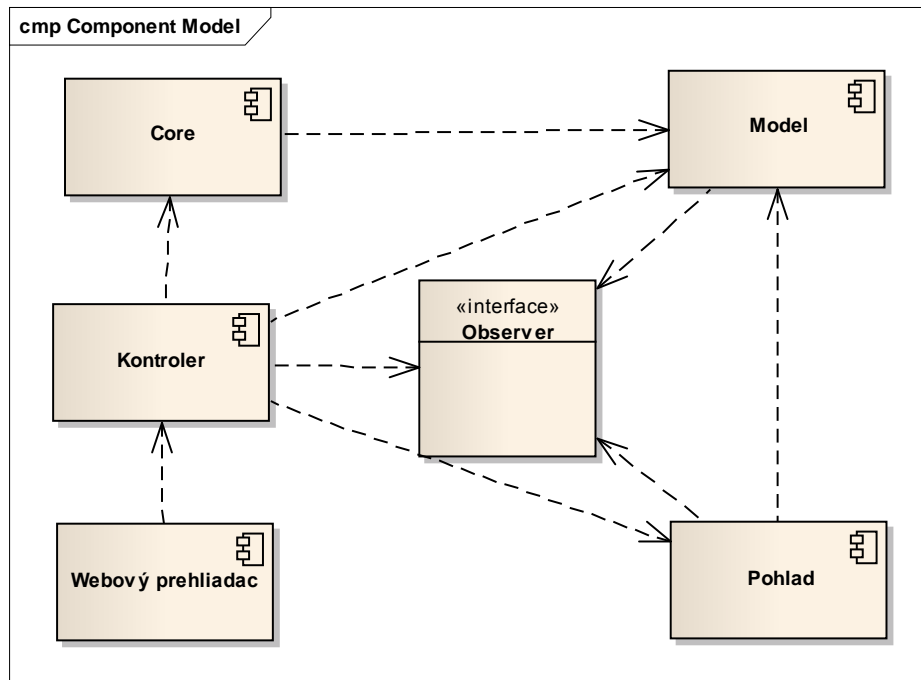
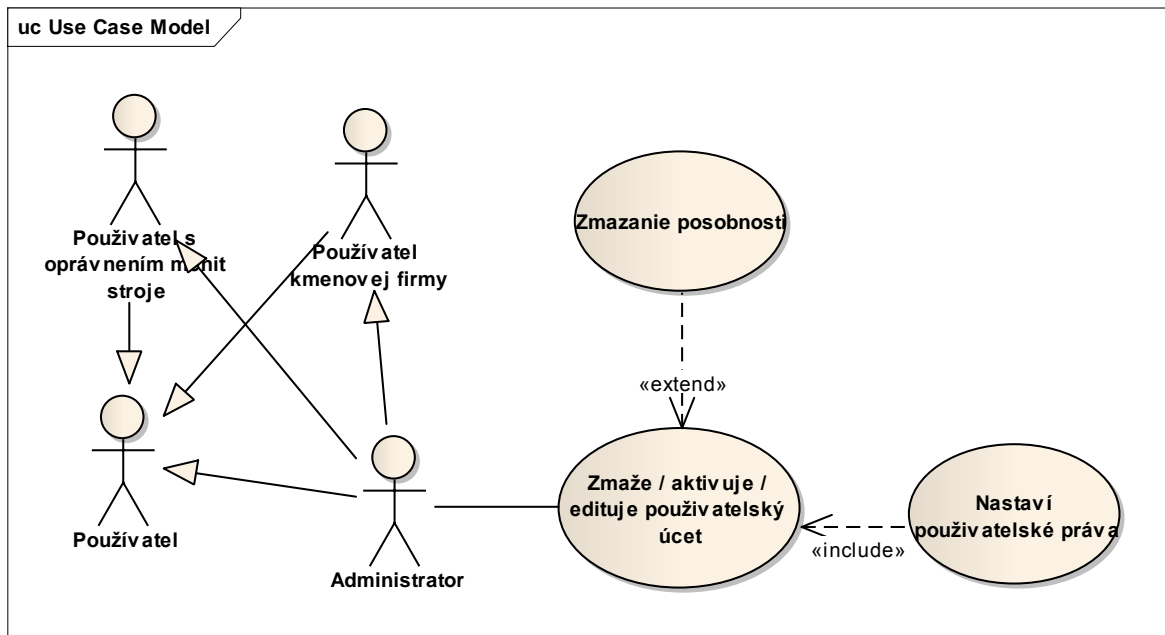


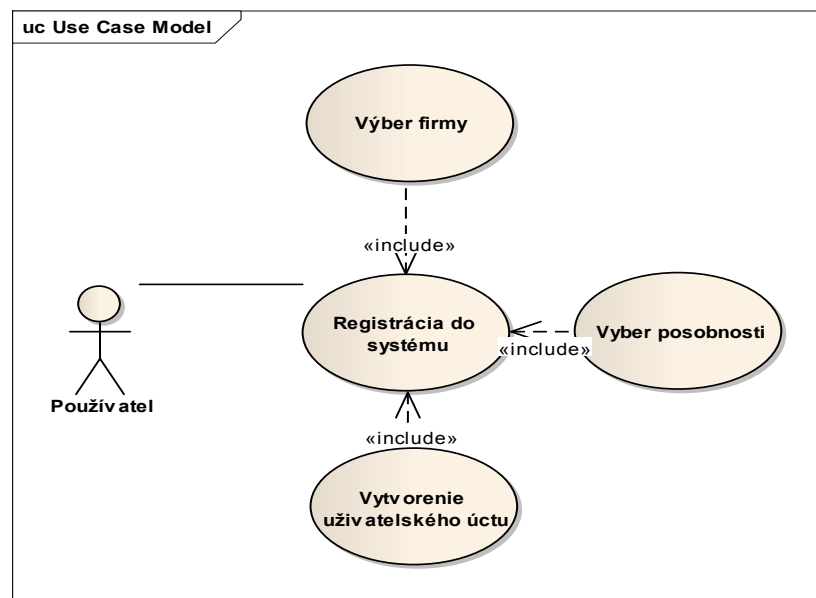
Figure 11 – Komponentový model

Komponentový model predstavuje architektonický návrhový vzor Model View Controller popísaný v kapitole 3 – Návrhové vzory.

5.4 Use Case model



Obrázok 12 – Use Case z pohľadu administrátora

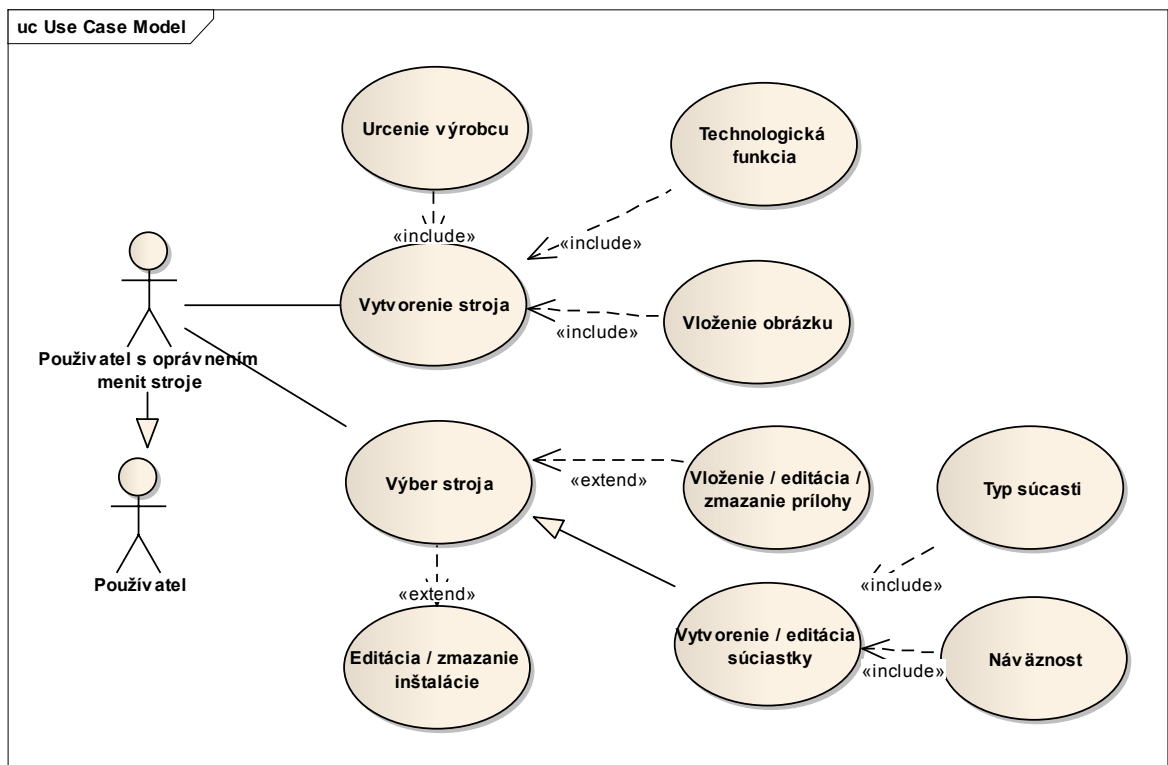


Obrázok 13 – Use Case z pohľadu používateľa

Pomocou Use Case modelu znázorním prípady použitia systému jednotlivými užívateľmi. Use Case model mimo iné pomôže aj pri identifikácii tried. Môžem potom vytvoriť tieto scenáre:

Popis aktivácie užívateľského účtu administrátorom:

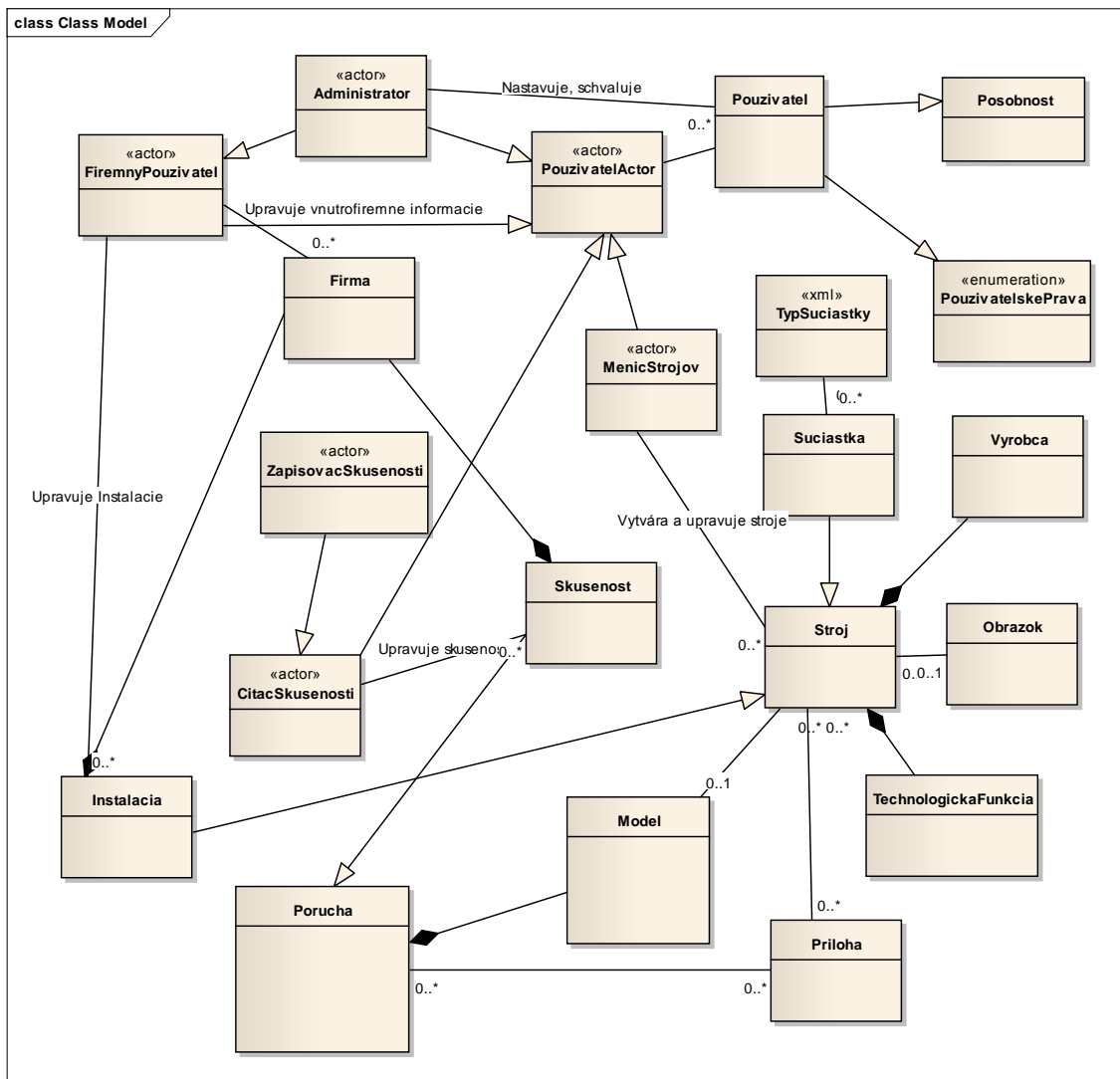
- Systém: Web
- Actors: Administrator
- Use Cases: Login, Zmaže / aktivuje / edituje používateľský účet, Nastaví používateľské práva
- Assumptions: Platný administrátor systém, ktorý má oprávnenie na vykonanie zmien.
- PreConditions: Používateľ v systéme nebol doteraz aktivovaný. Používateľ má právo systém používať.
- Init: Naštartuje administrátor. Systém zobrazí registrovaných používateľov k aktivácii.
- Dáta: Zoznam používateľov, stav účtu používateľa a dostupné oprávnenia v systéme.
- Popis: Administrátora najprv zaloguje, potom vyberie, aktivuje používateľský účet, pri nastavení účtu musí nastaviť práva. Ďalej so systémom pracuje a generuje ďalšie scenáre, alebo ukončí prácu so systémom.
- Response: Používateľ bol úspešne aktivovaný. Používateľ má nastavené právo XYZ.
- Termination: Operácia prebehla úspešne, došlo k výnimke, používateľ je už aktivovaný, používateľ neexistuje v systéme.
- PostConditions:
 - Operácia prebehla úspešne – v prípade, že používateľ existuje a nebol aktivovaný
 - Používateľ neexistuje v systéme – v tabuľke users sa používateľ nenachádza
- Komentáre: žiaden



Obrázok 14 – Use case z pohľadu používateľa s oprávnením meniť stroje

5.5 Statický model (diagram tried)

Diagram tried priamo zosynchronizujem so zdrojovým kódom. Enterprise Architect obsahuje možnosti synchronizácie kódu do programovacieho jazyka C#.



Obrázok 15 – Diagram tried

V diagrame vidieť hlavné triedy stroj a súčiastka. Stroju priradujem práve jeden model, práve jednu technologickú funkciu, práve jeden obrázok a môžu nahrat' niekoľko príloh. Od stroja ďalej berú vlastnosti inštalácia, súčiastka a porucha. V diagrame nie sú uvedené operácie a atribúty, ale je možné ich obnoviť zo zdrojového kódu. [10]

6 WEBOVÝ PROJEKT ASP .NET MVC 3

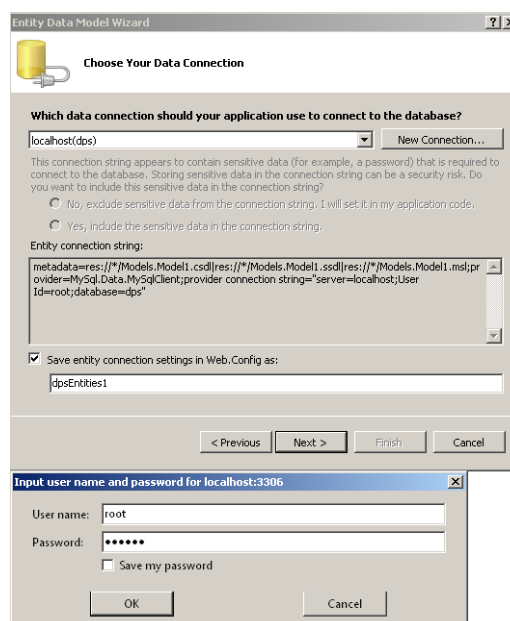
Vývoj bude uskutočnený v jazyku C#, pod platformou .NET 4.0, webový generátor bude ASP .NET, projekt MVC 3. Použijem MySQL databázový stroj.

6.1 ADO .NET Model

Pred vytvorením nového projektu je treba zo stránok Microsoftu stiahnuť AspNetMvc3Tools Setup a pre prácu s ADO.NET sa odporúča stiahnuť EF4FeatureCTP5. Dobré je tiež nechať Windows, aby sa aktualizoval automatickými update-ami.

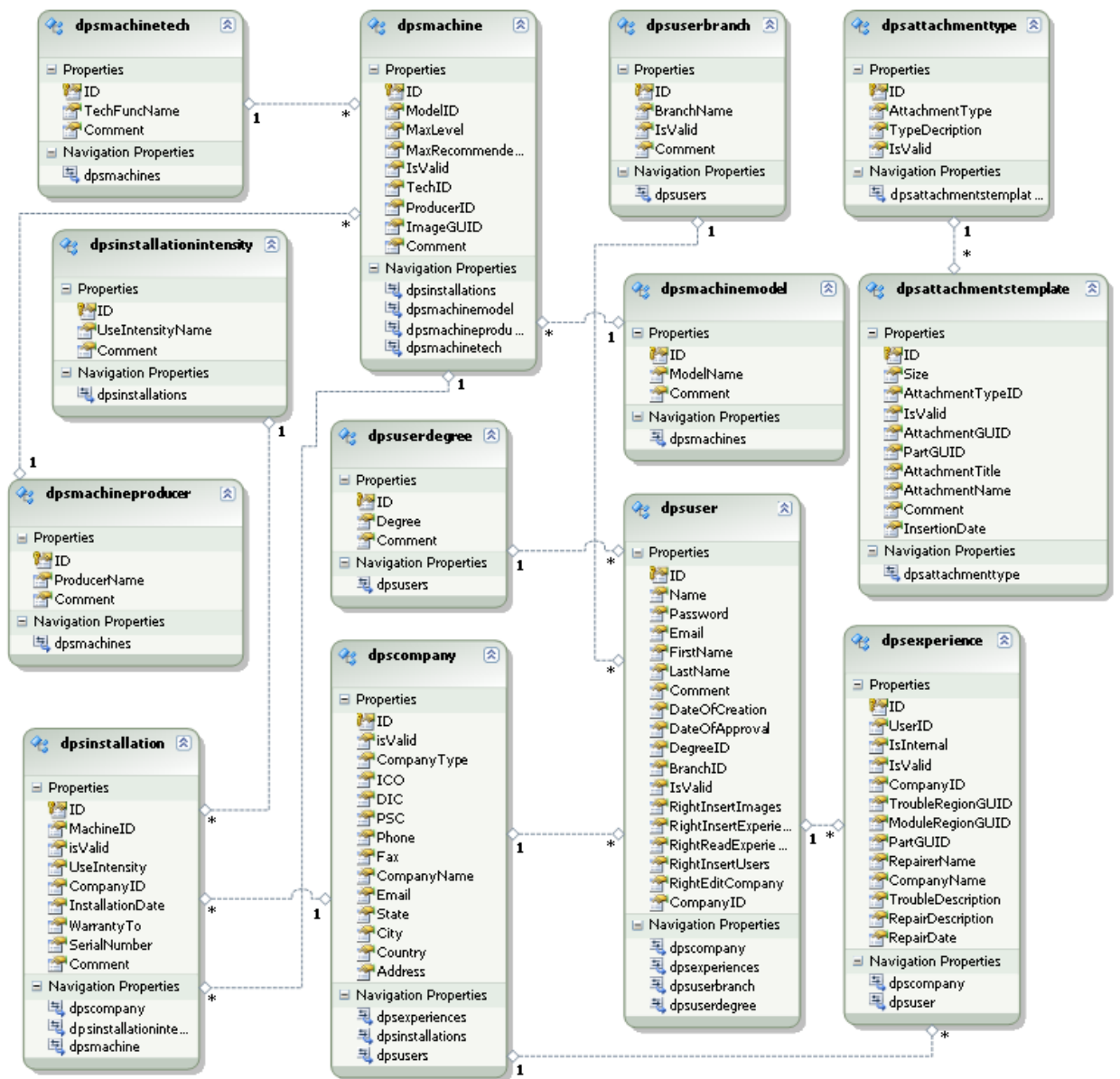
Aby som mohol projekt pripojiť na MySQL databázu, potrebujem stiahnuť a nainštalovať MySQL connector pre ADO .NET. Connector sa nachádza na webových stránkach MySql.

Po vytvorení nového projektu pridám nový ADO .NET model. Tento model je potrebné importovať už z existujúcej MySQL databáze, ktorú som prevzal a trochu upravil z [6]. Vybral som DB pripojenie.

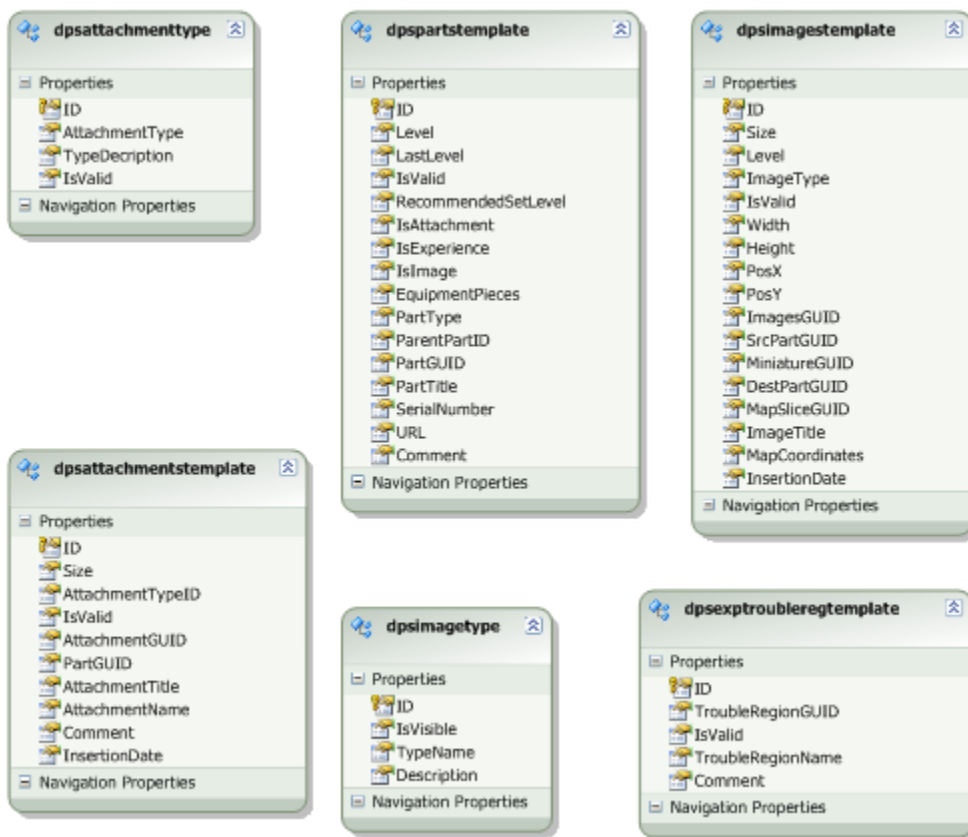


Obrázok 16 - Vytvorenie ADO .NET modelu

Na obrázku je vidieť, že je možnosť uložiť citlivú časť connection stringu do web.config súboru, ktorý je uložený na serveri, alebo citlivú informáciu nastaviť až v aplikačnom režime. Pre zjednodušenie som dal uloženie citlivých informácií do web.configu. Server by mal byť zabezpečený, tak aby sa nikto k web.configu nemohol dostať. Po výbere pripojenia nasleduje overenie oprávnenia prístupu do DB. Posledný krok sprievodcu je o výbere databázových objektov. Zaškrtol som DB tabuľky, ktoré som prevzal.



Obrázok 17 - ADO.NET model schéma 1 – Statické tabuľky

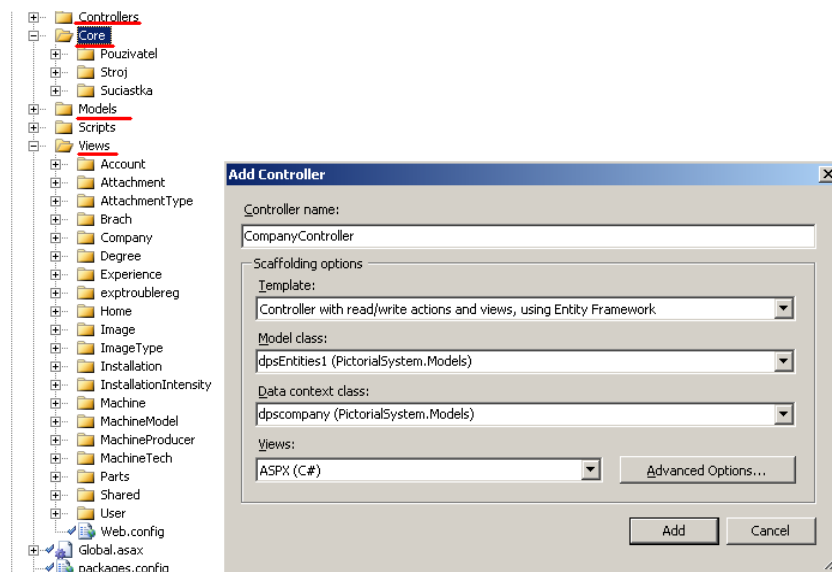


Obrázok 18 - ADO.NET model schéma 2 - Dynamické tabuľky

Dynamické tabuľky na rozdiel od statických budú vytvárať nové tabuľky pri vložení stroja (napríklad dpspart1 pre stroj s ID 1...).

Pre bližší popis tabuliek odporúčam pozrieť [6].

6.2 Náhľad na projekt



Obrázok 19 - Štruktúra projektu, vytvorenie kontroleru

Projekt sa skladá z 3 hlavných častí – Modely (Models), Pohľady (Views) and Kontrolery (Controllers) (MVC). Ku kontroleru je pridaná logická časť tzv. jadro (Core), ktorá obsahuje všetky pridané triedy navrhnuté v 5 kapitole.

Visual Studio umožňuje automatické vytvorenie MVC časti. Musím mať pridaný model v projekte, ktorý spolupracuje s databázou MySQL. Stačí potom pridať kontroler do projektu, vybrať si triedu modelu a triedu data contextu a Visual Studio vygeneruje kontroler a 5 pohľadov: Create.aspx, Delete.aspx, Details.aspx, Edit.aspx a Index.aspx. Po pridaní a kompilácii časti MVC spoločne komunikujú a so systémom je možné pracovať. Problém je v tom, že vygenerované pohľady obsahujú len editačné polia a ostatné ovládacie prvky sa musia naprogramovať.

6.2.1 Popis systému



Obrázok 20 – Klikacie menu

Po prihlásení do systému sa zobrazí klikacie menu. Ak má používateľ administrátorské práva, zobrazí sa mu rovnaké hlavné menu, ako je zobrazené na obrázku. Systém umožňuje:

- Práca so strojmi: Používateľ s administrátorským oprávnením a s oprávnením vkladania obrázkov bude mať v hlavnom menu zobrazený odkaz Machines a bude môcť pracovať s MVC časťou stroje (Machines). MVC časť stroje obsahuje 6 pohľadov: Vytvoriť (Create), Zmazať (Delete), Detaily (Details), Zmeniť (Edit), Index a Vybrať (Select). Po vytvorení stroja používatelia systému budú môcť vybrať stroj v pohľade Select a pridávať ďalšie súčiastky hierarchicky usporiadané vo vrstvách, alebo skúsenosti. Používateľ musí mať majú k tomu oprávnenie. Pri zmazení stroja dôjde k odstráneniu všetkých dynamických databázových tabuliek, ktoré prináležia stroju.
- Práca s používateľmi: Používateľ s administrátorským oprávnením bude mať v hlavnom menu zobrazený odkaz Users a bude môcť pracovať s MVC časťou používateľa (Users).
- Práca s firmami a inštalovanými zariadeniami: Používateľ s administrátorským oprávnením a s oprávnením editácií firemných dát bude mať v hlavnom menu zobrazený odkaz Companies, Installations a bude môcť pracovať s MVC časťou firmy a inštalácie (Companies, Installations).
- Práca so skúsenosťami: Po výbere stroja môže používateľ podľa oprávnení narábať s MVC časťou skúsenosti. Oprávnenie k Index, Details majú používatelia s právom čítať skúsenosti a zapisovať skúsenosti. Oprávnenie k Edit a Delete pohľadom majú používatelia s oprávnením zápis skúsenosti.
- Práca so súčiastkami: Pracovať s MVC časťou súčiastky môže len používateľ s pridelenými rolami administrátor, alebo s rolou vkladanie obrázkov.

Všetky tieto MVC časti obsahujú štandardne 5 pohľadov: Vytvoriť (Create), Zmazať (Delete), Detaily (Details), Zmeniť (Edit) a Index.

Na detaily stroja s ID 23 je možné sa dostať pomocou zadania cesty do prehliadača:
<http://../Machine/Details/23>

Na index používateľov je možné prejsť zadaním do adresného riadku:
<http://../Machine/User>.

V prípade, že nemá používateľ dostatočné oprávnenie k niektorej z častí MVC, bude presmerovaný na prihlasovací formulár.

6.3 Implementácia autentizácie a bezpečnosti

6.3.1 Prihlásenie do systému

Používateľ sa prihlasuje do systému pomocou prihlasovacieho formulára. Používateľ zadá meno a heslo a odošle formulár. Formulárové dáta obdrží kontroler a aplikačná logika spracuje dáta.

Heslo zahashujem MD5 algoritmom:

```
// convert to byte[]  
byte[] tmpdata = System.Text.Encoding.UTF8.GetBytes(source);  
// compute MD5  
byte[] hashpassword = MD5.Create().ComputeHash(tmpdata);  
// convert back to string  
var password = System.Text.Encoding.UTF8.GetString(hashpassword);
```

Potom porovnam dáta, či sú zhodné s databázou. Pokiaľ je overenie úspešné, vytvorím autentizačný cookie. Vypršanie platnosti cookie je nastavené na 20 minút a ešte do autentizačného cookie pridávam role. Nakoniec autentizačný cookie zašifrujem a pridám ho do HttpContextu.

Komunikácia medzi klientom a serverom nie je dobre zabezpečená a pri odposlúchaní (napr. na miestnej LAN) je heslo možné jednoducho získať. Podobne IS nie je dobre zabezpečený proti útokom man-in-the-middle. Riešením je vybranie zašifrovanej komunikácie medzi klientom a serverom, najlepšie pomocou SSL a získať certifikát od niektorej z certifikačných autorít.

6.3.2 Registrácia

Ak používateľ v systéme neexistuje, musí sa zaregistrovať pre prácu so systémom.

Create a New Account

Use the form below to create a new account.

Passwords are required to be a minimum of 6 characters in length.

Account Information

User Name

Email

Password

Confirm password

First Name

Last Name

Degree
 Degrees

Branch
 Branches

Company
 Companies

Obrázok 21 - Vytvorenie používateľského účtu

Používateľ vyplní registračný formulár a zaregistruje sa. Heslo prevediem do MD5 Hashu a vytvorí sa nová položka v db tabuľke users. Zaregistrovaný používateľ sa môže prihlásiť až po schválení registrácie administrátorom systému.

6.3.3 Zmena hesla

V prípade, že používateľ je prihlásený, môže si zmeniť heslo. Musí zadať predošlé heslo, nové heslo a potvrdiť nové heslo.

Postup zmeny hesla je podobný predošlým postupom. Používateľské meno mám, pretože používateľ je prihlásený. Overím heslo prevedením na MD5 tvar. Porovnam s databázou. Ak je heslo rovnaké, tak spravím update na db s novým MD5 hashom.

6.3.4 Role

Použivateľ môže mať v systéme tieto role:

```
public enum PouzivatelskePrava : int
{
    InsertImages, // Vkladanie obrazkov
    InsertExperiences, // Vkladanie skusenosti so sucastami
    ReadExperiences, // Zobrazenie skusenosti so sucastami
    InsertUsers, // Pokrocile operacie s pouzivatelmi
    EditCompany // Upravovat firemne informacie
}
```

Pre ďalšie spracovanie používateľských rolí potrebujem previesť hore uvedený enum na reťazec. Napísal som na to túto jednoduchú metódu:

```
public static void AddRoleFromRight(StringBuilder roles, byte RightType,
PouzivatelskePrava pp)
{
    if (RightType == 1)
    {
        if (roles.Length > 1) roles.Append(",");
        roles.Append(pp.ToString());
    }
}
```

Prvý parameter typu `StringBuilder` skladá výsledný reťazec pre oprávnenia. Druhý parameter je hodnota z databáze typu `byte`. Ak používateľ právo má, hodnota `RightType` je 1, v opačnom prípade je 0. Tretí parameter udáva typ práva, ktorý sa má pridať. Príklad volania operácie:

```
AddRoleFromRight(roles, dpu.RightEditCompany, PouzivatelskePrava.EditCompany);
```

Toto volanie pridá do objektu `roles` hodnotu "EditCompany".

6.4 Implementácia ukladania obrázkov a príloh

Pri práci so strojmi a súčiastkami je možné pridať obrázok, alebo prílohu. Vytvorením obrázku a jeho nahratím na server sa pre tento súbor vygeneruje jednoznačný identifikátor – GUID. Týmto identifikátorom sa súbor premenuje, uloží na server a na webových formulároch sa pod ním odkazuje.

V projekte sa nachádza pomocná trieda `FileHelper`, ktorá je zodpovedná pre prácu s obrázkami a prílohami. Najdôležitejšou metódou triedy je `UploadToServer`. Postupujem v nej tak, že najprv zistím, či je dostatok miesta na disku a či príloha nepresahuje maximálnu veľkosť. Potom rozdelím vstupný súbor na príponu a meno. Pomocou metódy `Path.Combine` vytvorím celú cestu pre uloženie súboru do adresára na disk serveru. Do `Path.Combine` vkladám: `Server.MapPath`, adresár `Uploads`, GUID súboru a príponu súboru.

Metóda `Server.MapPath` vráti fyzický adresár webovej aplikácie. Ak súbor na server už existuje, vygenerujem iný GUID. Nakoniec súbor uložím.

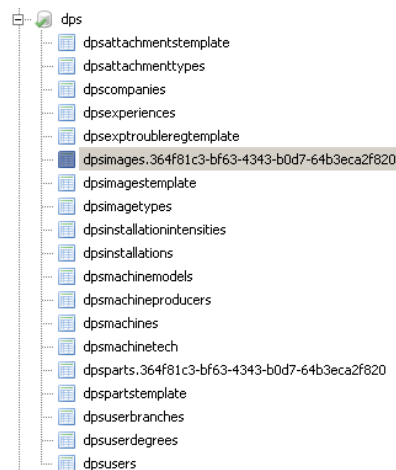
Prikladám výňatok kódu:

```
DriveInfo di = new DriveInfo(Path.GetPathRoot(ServerPath));
if (file.ContentLength > 0 && file.ContentLength < 2 * 1024 * 1024 &&
di.AvailableFreeSpace > file.ContentLength * 2)

// extract only the filename extension
var fileNameExtension = Path.GetExtension(file.FileName);
var path = Path.Combine(ServerPath, String.Format("{0}{1}", FileGuid,
fileNameExtension));
if (File.Exists(path))
{
FileGuid = Guid.NewGuid().ToString();
path = Path.Combine(ServerPath, String.Format("{0}{1}", FileGuid,
fileNameExtension));
}
file.SaveAs(path);
}
}
return FileGuid;
}
```

6.5 Implementácia dynamických tabuliek

Pre každý stroj sa vytvorí nová databázová tabuľka súčiastok, obrázkov, príloh a oblasti porúch. Pri každom vytvorení novej tabuľky sa za názov tabuľky pridá jednoznačný identifikátor GUID. Deje sa tak z dôvodu, že musím od seba odlíšiť tabuľky jednotlivých strojov.



Obrázok 22 - Dynamické tabuľky s GUIDom

Implementácia sa nachádza v triede `DbHelper`. Operácie triedy vytvárajú SQL dotazy `CREATE TABLE`, `DROP TABLE`, `DELETE`, `UPDATE` a `INSERT`. Do týchto operácií vstupuje GUID stroja, názov a objekt šablónovej tabuľky naplnený dátami. Pretože takmer

všetky operácie obsahujú rovnaké vstupné argumenty, rozhodol som sa ich presunúť do definície samotnej triedy.

Princíp je v tom, že sa pomocou reflexie prechádza vstupný objekt a z neho vyberiem názvy atribútov, ktoré sú zhodné s názvom stĺpcov tabuľky. Napríklad SQL dotaz UPDATE poskladám spojením: UPDATE názov šablónovej tabuľky a množina dvojíc názvu atribútu a jeho hodnoty.

U príkazu SELECT sa reflexia nepoužíva, len sa nahradí šablónová tabuľka tabuľkou s GUIDom. Vytvorím potom ObjectQuery a ďalej sa už v programe dotazujem pomocou LINQu.

6.6 Implementácia zaškrťavacieho políčka formulára

Ako som spomenul, ASP.NET MVC nevytvorí zaškrťavacie políčko (checkbox) automaticky. V pohľadoch preto treba nahradiť editačné pole IsValid zaškrťavacím políčkom, ktoré reprezentuje HTML tag:

```
<input type="checkbox" name="chkIsValid" value="Valid" <%= Model != null?Model.IsValid==1?"checked":String.Empty:String.Empty %> />
```

V špicatých zátvorkách sa nachádzajú dve C# inline podmienky, ktoré podľa hodnoty Model.IsValid (ne)zaškrtnú políčko. Potom v kontrolery musím zistiť, či je políčko zaškrtnuté, alebo nie. Ak používateľ nezaškrtnie zaškrťavacie políčko, chkIsValid sa v POSTe vôbec nenachádza.

```
if (Request["chkIsValid"] != null) dpsimagestemplate.IsValid = 1; else  
dpsimagestemplate.IsValid = 0;
```

6.7 Implementácia rozbaľovacieho zoznamu

V predošlej kapitole som ukázal, ako sú implementované zaškrťavacie políčka. Zložitejší problém je s rozbaľovacími zoznamami. Rozbaľovací zoznam musím zdefinovať do modelu:

```
public IEnumerable<SelectedItem> ImageSizeCombo { get; set; }
```

Týmto zápisom definujem, že budem v kontroleri a v pohľadoch používať takú kolekciu, ktorá obsahuje SelectedListItem a implementuje rozhranie IEnumerable (tj. každý SelectedListItem (až na posledný) v kolekcii má definovaného svojho následníka). IEnumerable a generiká sú popísané v kapitolách 4.1.14 a 4.1.15.

V kontroleri naplním rozbaľovací záznam hodnotami. Excerpt kódu:

```

List<SelectListItem> ss = newList<SelectListItem>();
bool FirstSelected = true;

foreach (var dpi in de.dpsimagetypes.ToList())
{
ss.Add(new SelectListItem { Value = dpi.ID.ToString(), Text = dpi.TypeName, Selected
= FirstSelected });
FirstSelected = false;
}
// Model.ImageSizeCombo = ss.ToArray();

```

dpsimagetypes je ADO.NET objekt priamo previazaný s databázovou tabuľkou dpsimagetypes. Pre každý prvok kolekcie dpsimagetypes vyberiem ID a TypeName a ich hodnoty priradím členom nového SelectListItem. Prvý SelectListItem bude v rozbalovacom zozname vybraný.

Do pohľadu uvediem metódu C#, ktorá vytvorí rozbalovací zoznam:

```

<%:Html.DropDownListFor(x => x.ImageSizeText, new SelectList(Model.ImageSizeCombo,
"Value", "Text"))%>

```

6.8 Výber stroja

Po výbere stroja z roletového menu a kliknutí na Select sa používateľovi zobrazí celý stroj. Používateľ môže pokračovať editáciou súčiastok, alebo modifikáciou skúseností (ak má k tomu oprávnenie). Pre zobrazenie časti stroja na vyšších úrovniach používateľ klikne na obrázok stroja.



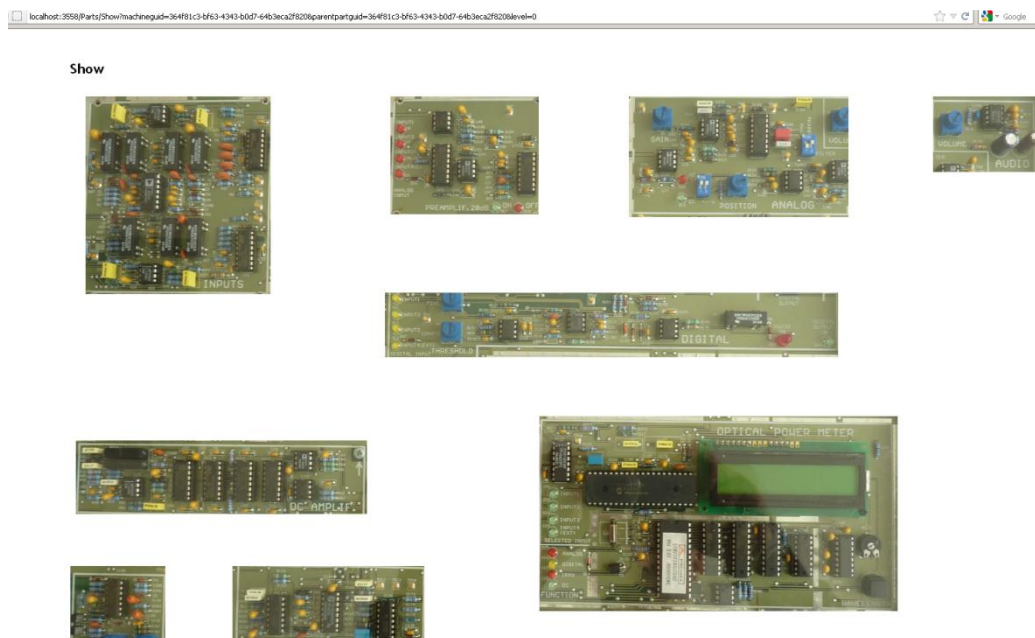
Obrázok 23 - Výber stroja

Obrázok stroja je opatrený odkazom na prvú úroveň súčiastok. Tvar odkazu: `http://../Parts/Show?machineguid={guid1}&parentpartguid={guid1}&level=0` znamená, že pošlem požiadavku kontroleru Parts na vygenerovanie novej úrovne súčiastok. Musím uviesť identifikačné parametre stroja a zdrojovej súčiastky. Pretože zdrojová súčiastka je zároveň strojom sú parametre `machineguid` a `parentpartguid` zhodné. Na ďalších úrovniach budú tieto parametre odlišné. Posledným parametrom je úroveň.

6.9 Zobrazenie súčiastok stroja na určitej úrovni

6.9.1 Popis funkčnosti z používateľského hľadiska

Pri kliknutí na obrázkový odkaz v kapitole 6.8 sa zobrazí prvá vrstva súčiastok. Používateľ môže ďalej vyberať súčasti kliknutím myši a takto sa posúvať nižšie a nižšie v hierarchickom strome súčiastok. Uvediem odkaz z prvého obrázku: `http://../Parts/Show?machineguid={guid1}&parentpartguid={guid2}&level=1` Parameter `machineguid` musím mať vždy k dispozícii pre dynamické tabuľky. `Parentpartguid` a `level` potrebujem, aby som mohol zistiť súčiastky, ktoré nadväzujú na používateľom vybranú súčiastku.



Obrázok 24 - Zobrazenie súčiastok na prvej úrovni

6.9.2 Model

Najprv vytvorím modelové triedy ShowPartModel a PartImage.

PartImage musí obsahovať všetko potrebné pre odkaz, samotný obrázok a identifikáciu obrázku. Obrázok obsahuje pozíciu, text a veľkosť. Odkaz vygenerujem z úrovne a identifikácií stroja a súčiastky.

ShowPartPanel obsahuje všetky obrázky, ktoré chcem poslať do náhľadu, a štatistické informácie spoločné pre všetky obrázky. Kontroler odošle celú instanciu ShowPartModel na klientsky počítač. Prehliadač klientskeho počítača vypočíta aktuálne súradnice obrázkov.

```
public class PartImage
{
    public Point position;
    public string machineguid;
    public string partguid;
    public string parentguid;
    public string imageguid;
    public string alttext;
    public Size size;
    public int level;
}

public class ShowPartModel
{
    public PartImage[] partimage { get; set; }
    public Point maxposition { get; set; }
    public Point minposition { get; set; }
    public Size minsize { get; set; }
    public Size maxsize { get; set; }
}
```

6.9.3 Logika

Dole uvedený zdrojový kód nájde v databáze všetky súčiastky, ktoré chcem vygenerovať do pohľadu.

```
var AllMachineParts = from x in part.GetPartsEnumerable(machineguid)
select x;

var FilteredParts = from x in AllMachineParts where x.ParentPartID == parentpartguid
&& x.Level == (level + 1)
select x;

var FilteredPartsAndImages = from x in FilteredParts
join y in part.m_Image.GetImagesEnumerable(machineguid) on x.PartGUID equals
y.SrcPartGUID
select new PartImage {
    imageguid=y.ImagesGUID,
    alttext = x.PartTitle,
    level = x.Level,
    machineguid = machineguid,
```

```

    parentguid = x.ParentPartID,
    partguid = y.SrcPartGUID,
    position = new Point(y.PosX, y.PosY),
    size = new Size(y.Width, y.Height)
};

maxposition = new Point(FilteredPartsAndImages.Max(xx => xx.position.X),
FilteredPartsAndImages.Max(xx => xx.position.Y));
maxsize = new Size(FilteredPartsAndImages.Max(xx => xx.size.Width),
FilteredPartsAndImages.Max(xx => xx.size.Height));
minposition= new Point(FilteredPartsAndImages.Min(xx => xx.position.X),
FilteredPartsAndImages.Min(xx => xx.position.Y));
minsize = new Size(FilteredPartsAndImages.Min(xx => xx.size.Width),
FilteredPartsAndImages.Min(xx => xx.size.Height));
PartImagesArray = FilteredPartsAndImages.ToArray();

```

Keď trochu nadsadím, môžem tvrdiť, že celá metóda je aplikáciou LINQ pre SQL na naplnenie instancií objektov tried ShowPartModel a PartImage. Riadky kódu skladajú svoj objektový dotaz (ObjectQuery), ktorý sa pošle do databáze až pri zavolaní metódy FilteredPartsAndImages.ToArray(). V tomto bode sa vytvoria a naplnia všetky instance potrebné pre generovanie náhľadu Show. Predchádzajúce príkazy stále pracujú s typom IEnumerable. Skladanie ObjectQuery zahrňuje postup:

- Získam súčiastky stroja;
- Vyfiltrujem súčiastky podľa guidu súčiastky a úrovne;
- Na časti naviažem obrázky;
- Vyberiem si parametre, ktoré ma zaujímajú.

6.9.4 Pohľad a klientsky skript

Kód v javascripte:

```

window.onload = function() { generate(); };

function generate() {
    <% foreach (var img in Model.PartImagesArray)
    { %>

        computeElements("<%= img.imageguid %>", <%= img.position.X %>, <%= img.position.Y %>, <%= img.size.Width %>, <%= img.size.Height %>);

    <%} %>
}

```

Hneď na začiatku zaregistrujem onload udalosť a zavolá sa funkcia generate. Vo vnútri špicatých zátvoriek s percentami (<% %>) definujem, čo má generátor pohľadu vygenerovať pomocou jazyka C#. Pri <%= %> sa jedná len o C# výraz. Pre každý obrázok

vytvorený v Core sekcii generátor pohľadov vygeneruje funkciu `computeElements` pre javascript. Takže napríklad pre 9 obrázkov bude v javascript funkcii `generate` 9 volaní funkcii `computeElements`. Do každej z týchto funkcií musí generátor náhľadov doplniť správne argumenty: GUID obrázku, pozíciu a veľkosť.

Nasleduje skrátený výpis funkcie `computeElements`:

```
function computeElements(imageguid, x, y, szx, szy)
{
var maxX=<%: Model.maxposition.X %>;
...
varx1 = document.getElementById(imageguid+"D");
...
x1.style.position="absolute";

x=((x-minX)*screen.width)/maxX;
y=((y-minY)*screen.height)/maxY;

x1.style.top=y+"px";
x1.style.left=x+"px";
...
}
```

Cieľom funkcie je prispôbiť súradnice a veľkosť zobrazovaciemu zariadeniu na klientskom počítači. Do začiatku funkcie obsahuje štatistiky vytvorené v Core časti. Potom zmením absolútnu pozíciu `<div>` HTML elementu vzhľadom k zobrazovacím možnostiam zariadenia.

Zostáva ukázať generovanie trojice `<div>`, `<a>` a `` (musia byť opatrené `foreach` iteráciou):

```
<div id="<%: img.imageguid %>D" style="position: absolute; left: 0px; top: 0px; ">
<a href="<%: Url.Action("Show", "Parts", new { machineguid = img.machineguid,
parentpartguid=img.partguid, level=img.level } ) %>">
<img id="<%: img.imageguid %>I" alt="<%: img.alttext %>" src="/uploads/<%:
PictorialSystem.Core.Machine.FileHelper.GetFile(img.imageguid,
Server.MapPath("~/uploads")) %>"/>
</a></div>
```

Ako HTML identifikátor je GUID veľmi vhodný, pretože je vždy unikátny. Za týmto GUIDom ešte doplňujem znaky D a I, aby som odlišil `img` a `div` HTML elementy. Štýl pozicionovania musí byť `absolute`. `Url.Action` je metóda ASP.NET MVC, ktorá mi poskladá link vo vhodnom formáte pre komunikáciu s kontrolerom.

Metóda `GetFile` vráti upraví vstupný GUID parameter a nájde správnu príponu súboru umiestneného na serveri. Menná časť súboru je podobná GUIDu ale nemá znaky pomlčky

“-“. Server.GetPath vráti cestu virtuálneho adresára, ktorý sa nachádza na webovom serveri. Do tohto adresára + “/uploads/” sú nahrané súbory.

6.10 Editácia súčiastky na určitej úrovni

Po výbere stroja môže používateľ vytvoriť, alebo editovať súčiastky na určitej úrovni. Pre každú súčiastku je možné pridať obrázky.

Is Valid? <input type="checkbox"/>	
Recommended Set Level 1: <input type="checkbox"/> 2: <input type="checkbox"/> 3: <input type="checkbox"/> 4: <input type="checkbox"/> 5: <input type="checkbox"/>	
Is Equipment? <input type="checkbox"/>	
Equipment Pieces 0	
Part Type Part	Level 2
Part Title B1	X 150
Serial Number B1	Y 67
URL B1	Image Type 1 Type
Comment B1	Is Valid? <input checked="" type="checkbox"/>
Select Parent Part B9 - 1	Image Size 640x480
Images Add/Edit Images	Title C13
Attachments Add/Edit Attachments	Upload File (left blank for no change) Procházet...
Experiences Add/Edit Experiences	
Save	Save

Obrázok 25 - Zmeniť súčiastku a zmeniť obrázok

Dôležité je vybrať si rodičovskú súčiastku, alebo stroj v roletovom menu Select Part a vyplniť jednotlivé editačné prvky (Part Title, Serial Number, URL, Comment). Po kliknutí na Add / Edit Images sa otvorí nový tab na správu obrázkov k danej súčasti. Tu je už možné nastaviť súradnice, veľkosť obrázku a nahrať obrázok na server.

Kompletným popisom pohľadov sa zaoberá diplomová práca D. Guryčky, podľa ktorej som náhľady naprogramoval a sú súčasťou priloženého CD.

6.11 Výber typu časti a výber obrázku

Z predošlej kapitoly chcem ukázať naplnenie rozbaľovacích zoznamov:



Obrázok 26 - Výber typu časti a výber veľkosti obrázku

Zoznamy sa plnia z XML súborov umiestených na webovom serveri. Súbory su nazvané PartTypes.xml a ImageSizes.xml. Uvediem výpis ImageSizes.xml:

```
<?xml version="1.0"?>
<ArrayOfXmlImageSizes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <XmlImageSizes>
    <SizeTitle>640x480</SizeTitle>
    <Width>480</Width>
    <Height>640</Height>
  </XmlImageSizes>
  <XmlImageSizes>
    <SizeTitle>800x600</SizeTitle>
    <Width>600</Width>
    <Height>800</Height>
  </XmlImageSizes>
</ArrayOfXmlImageSizes>
```

XML vzniklo serializáciou objektu triedy:

```
[Serializable()]
public class XmlImageSizes
{
  public string SizeTitle { get; set; }
  public short Width { get; set; }
  public short Height { get; set; }
  public XmlImageSizes() { }
}
```

XML Serializáciou je možné zaistiť neskoršiu zmenu údajov bez zásahu programátora (stačí zmeniť XML súbor na serveri), či kompilácie webovej aplikácie. XML v budúcnosti môžu použiť aj iné webové aplikácie. Aby získal objekt s údajmi, je možné vytvoriť napríklad metódu:

```
public static object DeserializeXml(Type XmlSerializableType, string path, string
XmlFileName)
{
    XmlSerializer xsz = new XmlSerializer(XmlSerializableType);
```

```
string xmlfile = Path.Combine(GetServerPath(path), XmlFileName);  
using (Stream fStream = new FileStream(xmlfile, FileMode.Open,  
FileAccess.Read, FileShare.None))  
return xsz.Deserialize(fStream);  
}
```

Do `XmlSerializableType` je treba dosadiť pole, do `path` cestu k súboru umiestneného na serveri a konečne samotný názov súboru. Túto metódu je možné použiť všeobecne na serializáciu nielen `XmlImageSizes`, ale akéhokoľvek objektu. Táto istá metóda je použitá aj na `Part Type`.

ZÁVER

Webová aplikácia pre technickú diagnostiku je ukončená a beží na HTML, ASP.NET MVC a MySQL. Informačný systém bol otestovaný na zariadení EF-970 od španielskej firmy Promax v prehliadačoch Internet Explorer, Mozilla Firefox, Opera a Google Chrome a neboli zistené žiadne nedostatky.

Informačný systém je možné zaviesť na ktoromkoľvek IIS v operačnom systéme Windows, webové stránky pobežia v spomínaných prehliadačoch.

Ako bolo spomenuté v úvode, je hotová len intranetová časť:

- Autentizácia a autorizácia používateľov.
- MVC, komunikácia aplikácie s databázou vrátane správy dynamických tabuliek.
- MVC časti nutné pre prácu s informačným systémom.
- Ukladanie obrázkov a príloh.
- Prehľad vrstiev a hierarchickej štruktúry.

Pre celkovú funkčnosť doporučujem v budúcnosti dotvoriť:

- Zmeniť CSS štýly aplikácie.
- Zlepšiť prácu na popredí aplikácie: umožniť rezy súčiastok a prácu so súčiastkami na jednotlivých vrstvách v prehľade vrstiev.
- Umožniť pohľad Part\index.aspx pracovať na konkrétnej vrstve.
- Pridať vyhľadávanie.
- Pridať pohľady na správu skladu.

ZOZNAM POUŽITEJ LITERATURY

- [1] AcademicTutorials.com [online]. c2012 [cit. 20.05.2012]. Dostupný na World Wide Web: <http://www.academictutorials.com/microsoft.net/dotnet-framework.asp>
- [2] ALBAHARI J., ALBAHARI B. C# 4.0 in a Nutshell. O'Reilly Media Inc, 2010. 1033p. ISBN 978-0-596-80095-6.
- [3] DIMITROV D. Populate a select in asp net mvc 3 [online]. c2012 [cit. 20.05.2012]. Dostupný na World Wide Web: <http://stackoverflow.com/questions/5729375/populate-a-select-in-asp-net-mvc3-with-data-in-the-database-using-nhibernate>
- [4] FREEMAN A. SANDERSON S. Pro ASP.NET MVC3 Framework. Apress, 2011. 852p. ISBN 978-1-4302-3404-3.
- [5] FUJTÍK O. Produktová dokumentace v agilním přístupu vývoje [online]. c2011 [cit. 20.05.2012]. Dostupný na World Wide Web: is.muni.cz/th/256345/fi_b/Bachelor_thesis.txt
- [6] GURYČA D. Programový informační a zobrazovací systém pro technickou diagnostiku. Univerzita Tomáše Bati ve Zlíně, 2004.
- [7] LACEY M. Scrum Framework [online]. c2012 [cit. 20.05.2012]. Dostupný na World Wide Web: http://www.scrumalliance.org/pages/what_is_scrum
- [8] MACDONALD M., FREEMAN A., SZPUSZTA M. Pro ASP.NET 4 in C# 2010. apress, 2010. 1575p. ISBN 978-1-4302-2529-4.
- [9] MCCONNEL S. Dokonalý kód. Computer Press, 2006. 894p. ISBN 80-251-0849-X.
- [10] PENDER A. T. UML Weekend Crash Course, 2002. 385p.. ISBN 0-7645-4910-3.
- [11] PLÁŠEK P. Programový informační a zobrazovací systém pro technickou diagnostiku. Univerzita Tomáše Bati ve Zlíně, 2004.
- [12] PURDY D., RICHTER J. Exploring the Observer Design Pattern [online]. c2002 [cit. 20.05.2012]. Dostupný na World Wide Web: <http://msdn.microsoft.com/en-us/library/ee817669.aspx>
- [13] RUMBAUGH J., JACOBSON I., BOOCH G. The Unified Modeling Language reference manual. Addison-Wesley, 1999. 550p. ISBN 0-201-30998-X

- [14] SCHATTEN A., DEMOLSKY M., GOSTISCHA E. F. Best practice software engineering [online]. c2012 [cit. 20.05.2012]. Dostupný na World Wide Web: <<http://best-practice-software-engineering.ifs.tuwien.ac.at/>>
- [15] SHALLOWAY A., TROTT J. Design patterns explained.
- [16] SHARP J. Microsoft® Visual C#® 2010 Step by Step, 2010. 781p.
- [17] SOMMERVILLE I. Software engineering. Addison-Wesley, 2011. 773p. ISBN 978-13-703515-1.
- [18] TROELSEN A. Pro C# 2010 and the .NET 4 Platform Fifth Edition. apress, 2010. 1753p. ISBN 978-1-4302-2550-8.
- [19] VAŠEK, V., VAŠEK, L. Simulace systémů. Zlín. FT VUT, 1991. 136p.
- [20] Wikipedia [online]. c2012 [cit. 20.05.2012]. Dostupný na World Wide Web: <<http://www.wikipedia.org/>>

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

ADO	ActiveX Data Objects
AJAX	Asynchronous JavaScript and XML
ASP	Active Server Pages
BLOB	Binary Large Object
CD	Compact Disc
CLR	Common Language Runtime
CRC	Class-Responsibility-Collaboration card
CSS	Cascade Style Sheet
DOM	Document Object Model
DTD	Document Type Definition
GC	Garbage Collector
GUID	Globally Unique Identifier
HTML	Hyper Text Markup Language
IBM	International Business Machines Corporation
ID	IDentifier
IO	Input / Output
JIT	Just In Time compiler
JS	JavaScript
LAN	Local Area Network
LINQ	Language Integrated Query
MAC	Macintos
MD5	Message Digest 5
MVC	Model View Controller
NTFS	NT File System

OOP	Object Oriented Programming
PC	Personal Computer
RUP	Rational Unified Process
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SSL	Secure Socked Layer
SVN	Subversion (Repository)
TFS	Team Foundation Server
TV	TeleVision
UML	Unified Modeling Language
XP	Extreme programming
XML	Extensible Markup Language

ZOZNAM OBRÁZKOV

Obrázok 1 - Scrum.....	12
Obrázok 2 - Nefunkčné požiadavky [17].....	15
Obrázok 3 – Návrhové vzory [14]	19
Obrázok 4 – MVC [14].....	21
Obrázok 5 – Architektúra .NET.....	22
Obrázok 6 - Refaktoring	23
Obrázok 7 - LINQ.....	31
Obrázok 8 – Model požiadávok 1	35
Obrázok 9 – Model požiadávok 2.....	36
Figure 10 – Deployment model	38
Figure 11 – Komponentový model	39
Obrázok 12 – Use Case z pohľadu administrátora	40
Obrázok 13 – Use Case z pohľadu používateľa.....	40
Obrázok 14 – Use case z pohľadu používateľa s oprávnením meniť stroje	42
Obrázok 15 – Diagram tried	43
Obrázok 16 - Vytvorenie ADO .NET modelu	44
Obrázok 17 - ADO.NET model schéma 1 – Statické tabuľky	45
Obrázok 18 - ADO.NET model schéma 2 - Dynamické tabuľky.....	46
Obrázok 19 - Štruktúra projektu, vytvorenie kontroleru	47
Obrázok 20 – Klikacie menu	47
Obrázok 21 - Vytvorenie používateľského účtu.....	50
Obrázok 22 - Dynamické tabuľky s GUIDom.....	52
Obrázok 23 - Výber stroja	54
Obrázok 24 - Zobrazenie súčiastok na prvej úrovni	55
Obrázok 25 - Zmeniť súčiastku a zmeniť obrázok	59
Obrázok 26 - Výber typu časti a výber veľkosti obrázku	60