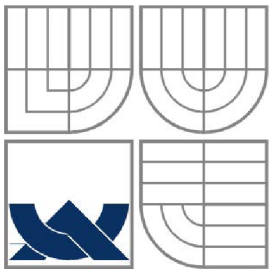
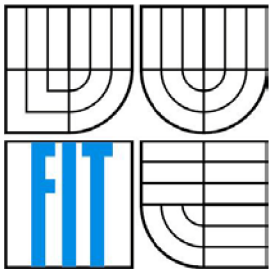


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

APLIKACE PRO MĚŘENÍ STABILITY STROMŮ

APPLICATION FOR TREE STABILITY MEASUREMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
JAKUB ŠOLTÉS

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. LUKÁŠ POLOK

BRNO 2013

Abstrakt

Práce se zabývá problematikou stability stromů při zatížení větrem. Definuje, co je to stabilita, jak se měří a proč je potřebné ji znát. Popisuje vývojový proces tvorby aplikace, která zkoumá stabilitu stromů. Analyzuje prostředky dostupné pro vývoj, zabývá se návrhem a implementací aplikace napsané v jazyku C# v prostředí .NET Frameworku obohaceném o grafický podsystém Windows Presentation Foundation a s využitím návrhového vzoru Model View ViewModel. Výsledkem práce je desktopová aplikace pro operační systém Windows, která na základě známých údajů o geometrii stromu a okolitém prostředí spočte koeficient stability stromu. Tato hodnota se následně využívá při hodnocení provozní bezpečnosti stromu.

Abstract

This thesis is concerned with the subject of tree stability under a wind load. It defines, what the tree stability is, which techniques are used for its measurement and why it is important to determine it. It describes development process of application, that examines tree stability. Furthermore it analyses available the development tools, discusses a design and an implementation of the application written in C# language. Application has been created using .NET Framework along with graphical subsystem Windows Presentation Foundation following the design pattern Model View ViewModel. The result of this thesis is a desktop application aimed on Windows operating system. It can compute stability coefficient of the tree using information about trees‘ geometry and its environment. The value of stability coefficient is commonly used for tree safety inspection.

Klíčová slova

strom, stabilita, koeficient stability, dendrometrie, C#, .NET, WPF, MVVM

Keywords

tree, stability, stability coefficient, dendrometry, C#, .NET, WPF, MVVM

Citace

Šoltés Jakub: Aplikácia pre meranie stability stromov, bakalářská práce, Brno, FIT VUT v Brně, 2013

Aplikácia pre meranie stability stromov

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka. Další informace mi poskytli Ing. Luděk Praus Ph.D. a Ing. Jan Čepela. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Šoltés
1.5.2013

Poděkování

Chcem poďakovať Ing. Lukášovi Polokovi, vedúcemu bakalárskej práce, za cenné rady a pomoc, ktoré mi poskytol pri tvorbe práce. Ďalej ďakujem Ing. Luďkovi Prausovi Ph.D. za poskytnutie dôležitých informácií v oblasti arboristiky a prístup k ďalším informačným zdrojom.

Obsah

1 Úvod

Jedným zo základných kritérií pre hodnotenie stromov je ich bezpečnosť. Prevádzková bezpečnosť stromu sa odvíja od mechanických vlastností dreva, geometrie stromu, zložitosti koreňového systému, podmienok okolitého prostredia a ďalších faktorov. Tie určujú výslednú stabilitu stromu – schopnosť zniesť statickú a dynamickú záťaž kladenú okolím, a to najmä vetrom. Stabilitu stromu je teda dôležité pravidelne sledovať. V prípade odchýliek od noriem je nutné uskutočniť adekvátne opatrenia, v extrémnych prípadoch končiace výrubom stromu. Takto sa zaistí, že pád stromu neohrozí zdravie človeka, prípadne nepoškodí cudzí majetok. Väčšina stromov v Českej republike je pravidelne kontrolovaná a vedú sa záznamy o ich stave v priebehu rastu.

Keďže hodnotenie stability stromu je pomerne časté a strom je počas jeho života niekoľkokrát podrobený meraniu, rozhodla sa Lesnická a dřevařská fakulta Mendelovy Univerzity v Brně založiť výskumne-vývojový projekt, ktorého súčasťou je vytvorenie aplikácie, ktorá uľahčí výpočet stability stromu. Táto aplikácia bude následne slúžiť ako nástroj pri posudzovaní prevádzkovej bezpečnosti a taktiež ako výuková pomôcka pre študentov arboristiky[1].

Cieľom tejto práce je vytvoriť pilotnú verziu tejto aplikácie - počítačový program, ktorý na základe fotografie stromu, známych údajov o geometrii stromu, okolitom prostredí a materiálových vlastnostiach skúmaného dreva vypočíta koeficient stability stromu[2]. Ten potom vystupuje ako dôležitý faktor pri ďalšom hodnotení bezpečnosti stromu.

Nasledujúca kapitola popisuje teoretický základ nutný k pochopeniu domény projektu. Tretia kapitola analyzuje problém a skúma prostriedky dostupné pre vývoj. Štvrtá kapitola sa zaoberá návrhom aplikácie a vzájomného prepojenia jej súčastí. Piata kapitola následne prezentuje fázu implementácie a testovania. Záver práce obsahuje zhrnutie, zhodnotenie výsledkov a diskutuje možnosti ďalšieho vývoja.

[1] arboristika – odbor zaoberajúci sa komplexnou starostlivosťou o dreviny
[2] koeficient stability stromu – dôležitý parameter pri hodnotení prevádzkovej bezpečnosti (bližšie informácie v kapitole 2)

2 Teoretický základ

Cieľom tejto kapitoly je uviesť čitateľa do problematiky stability stromov. Obsahuje informácie o tom, ako sa stabilita stromov meria, aké prostriedky sú dostupné pre jej hodnotenie a aký význam majú dosiahnuté výsledky.

Mechanickú stabilitu stromu môžno definovať ako schopnosť prenášať mechanické namáhanie bez zlyhania stromu. Zlyhaním sa rozumie zlom alebo vyvrátenie a dôjde k nemu vtedy, keď zaťaženie prekročí pevnosť materiálu, v tomto prípade pevnosť dreva. Odolnosť voči zlomu berie do úvahy predovšetkým mechanické vlastnosti dreva, zatiaľ čo odolnosť voči vyvráteniu zohľadňuje aj pevnosť ukotvenia stromu – koreňový systém v kombinácii s typom pôdy. Pre bežné meranie stability je však zohľadňovanie podzemnej časti stromu príliš náročné (preskúmanie celého koreňového systému vyžaduje špeciálne vybavenie alebo je úplne nemožné) a preto v nasledujúcom texte uvažujeme iba nadzemnú časť stromu.

2.1 Metódy hodnotenia

V súčasnosti existuje viacero prístupov a metód [1], ktorými možno stabilitu stromu hodnotiť. Najkomplexnejšou metódou hodnotenia je ťahová skúška, označovaná ako *Static Integrated Method* [2]. V porovnaní s inými postupmi umožňuje ťahová skúška meranie odolnosti nielen voči zlomeniu ale aj vyvráteniu, keďže sleduje aj silu ukotvenia a tým poskytuje dostatok informácií pre posúdenie stability. Simuluje sa zaťaženie vetrom, ktorému môže byť strom vystavený a následne sa skúmajú prejavy tohto zaťaženia na strom. Využívajú sa špecializované prístroje, ktoré majú za úlohu zhodnotiť kvalitu dreva a jeho mechanické vlastnosti. Detekujú trhliny a dutiny v dreve a prispievajú k spresneniu metódy. Tieto informácie sú dôležité pre vyjadrenie schopnosti prenášať mechanické napätie spôsobené statickou či dynamickou záťažou kladenou na strom. Výsledkom je exaktné ohodnotenie statického stavu stromu. Výhodou je možnosť simulovať rôzne veľkosti záťaže, nevýhodou zasa náročnosť a finančná nákladnosť tohto postupu. Preto sa využíva iba pri stromoch, ktorých hodnota či význam prevyšujú náklady na meranie.

Najbežnejším prístupom je vizuálne zhodnotenie charakteristík konkrétneho stromu a jeho aktuálneho statického stavu v kombinácii s podmienkami prostredia, v ktorom rastie. Zohľadňujú sa pritom dendrometrické[3] parametre stromu a typ dreva ako aj fyzikálne vlastnosti prostredia, najmä veterný profil. Takýto postup vyžaduje rozsiahle skúsenosti hodnotiteľa a jeho orientáciu v doméne. Manuálne posudzovanie stability je však časovo náročné a preto vznikla požiadavka tento proces aspoň čiastočne zautomatizovať. Všetky spomenuté parametre v skutočnosti reprezentujú vstupy matematického modelu, ktorého výstupom je koeficient stability stromu.

2.2 Koeficient stability

Hodnota koeficientu reprezentuje pomer pevnosti konkrétneho typu dreva a mechanického napätia vznikajúceho v kmeni stromu pri statickej záťaži (najčastejšie spôsobenej vetrom). Na výslednú hodnotu má vplyv viacero činiteľov. Je možné ju vypočítať na základe nasledujúcich údajov:

- výška stromu
- výška nasadenia[4]
- tvar koruny a veľkosť jej plochy
- priemer kmeňa v prsnej výške[5]
- mechanické vlastnosti stromu (tabuľkové hodnoty pre zvolený druh stromu)
  - koeficient aerodynamického odporu
  - pevnosť dreva
- veterný profil, pre ktorého výpočet potrebujeme poznať
  - kategóriu terénu podľa normy ČSN EN 1991-1-4 [3] a súčiniteľ drsnosti terénu
  - referenčnú rýchlosť vetra
  - nadmorskú výšku lokácie, v ktorej strom rastie
  - priemernú teplotu vzduchu v okolitom prostredí

[3] dendrometria – súbor geometrických vlastností stromu
[4] výška nasadenia – výška spodnej hranice koruny stromu
[5] prsná výška – arboristický štandard, konštanta s hodnotou 1.3m

Vzhľadom k frekvencii je najčastejším zdrojom zaťaženia vietor [4]. Toto zaťaženie je distribuované na celú plochu koruny a kmeňa. Určitá časť energie je spotrebovaná tlmením kmitov stromu odporom vzduchu. Zároveň dochádza k natočeniu listov v smere prúdenia vzduchu a zmenšuje sa tak náporová plocha. Aby bolo možné vypočítať napätie v kmeni stromu pri záťaži [5], je nutné poznať veľkosť tejto záťaže – silové pôsobenie vetra na strom. Vo väčšine prípadov sa sila vetra spočíta podľa vzťahu, ktorý sa nazýva *Newtonov zákon odporu* [6].

$$F = 0.5 * C_w * A * \rho_{air} * v^2 \tag{2.1}$$

kde **F** je vznikajúca sila [N], $\mathbf{C_w}$ je koeficient aerodynamického odporu stromu, **A** je náporová plocha koruny [m^2], $\boldsymbol{\rho_{air}}$ je hustota vzduchu [kg/m^3] a **v** je rýchlosť prúdenia vzduchu [m/s]. Tento spôsob výpočtu má svoje limity, pretože uvažuje laminárne prúdenie vzduchu[6]. Napriek tomu je takýto výpočet pôsobiacej sily všeobecne používaný a vo väčšine prípadov dostačujúci. V rovnici vystupujú dve neznáme: $\boldsymbol{\rho_{air}}$ a **v**.

1) Hustota vzduchu sa určuje na základe nadmorskej výšky a teploty vzduchu.

$$\rho_{air} = \frac{\rho_0 * P_h}{((1+ \gamma * T_{air}) * P_0)} \tag{2.2}$$

kde $\boldsymbol{\rho_0}$ je hustota vzduchu pri 0 °C [1.276 kg/m^3], $\mathbf{P_h}$ je atmosférický tlak pri danej nadmorskej výške [kPa], **γ** je tepelná konštanta [0.00366 1/K], $\mathbf{T_{air}}$ je teplota vzduchu [°C] a $\mathbf{P_0}$ je tlaková konštanta [100 kPa]. $\mathbf{P_h}$ sa spočíta ako:

$$P_h = P_0 * \left(1 - \frac{DtDh * H_{alt}}{T_{sea} * T_0}\right)^{5.225} \tag{2.3}$$

kde **DtDh** je teplotný gradient [0.0065 K/m], $\mathbf{H_{alt}}$ je nadmorská výška [m], $\mathbf{T_{sea}}$ je odpočítateľná hodnota teploty vzduchu pri mori [20 °C] a $\mathbf{T_0}$ je absolútna nula [273.15 K].

[6] laminárne prúdenie – prúdenie, pri ktorom sú prúdnice rovnobežné a nemiešajú sa

2) Rýchlosť vzduchu sa mení s výškou, v ktorej chceme výslednú silu počítať. Pre jej určenie je potrebné použiť rovnicu veterného profilu.

$$v = C_r * v_{ref} \tag{2.4}$$

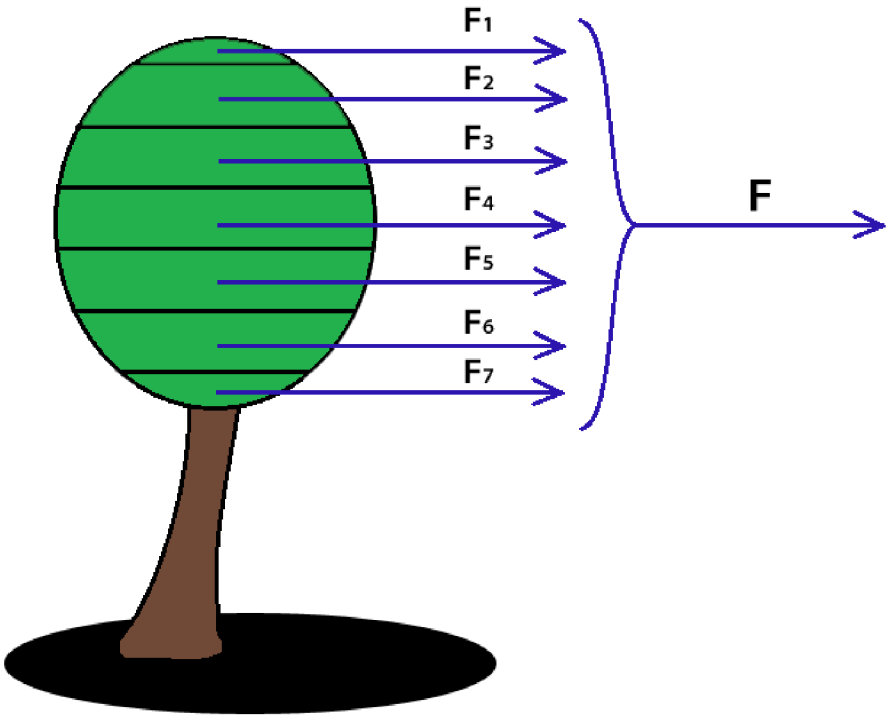
kde $\mathbf{C_r}$ je súčiniteľ drsnosti terénu a určuje sa podľa kategórie terénu nasledovne:

$$C_r = 0.19 * \left(\frac{Z_0}{0.05}\right)^{0.07} * \ln\left(\frac{Z}{Z_0}\right) \tag{2.5}$$

pričom Z reprezentuje maximum zo Z_{min} a aktuálnej výšky nad zemou, v ktorej rýchlosť vetra počítame. Hodnoty Z_0 a Z_{min} sa menia spolu s typom krajiny, v ktorej sa strom nachádza. Norma ČSN EN 1991-1-4 [7] definuje 5 kategórií terénu:

Kategória terénu	Z_0 [m]	Z_{min} [m]
More alebo pobrežné oblasti vystavené otvorenému moru.	0.003	1
Jazerá alebo vodorovné oblasti so zanedbateľnou vegetáciou a bez prekážok.	0.01	1
Oblasti s nízkou vegetáciou ako je tráva a s izolovanými prekážkami (stromy, budovy), ktorých vzdialenosť je maximálne 20-násobok výšky prekážok.	0.05	2
Oblasti rovnomerne pokryté vegetáciou alebo budovami, prípadne izolovanými prekážkami, ktorých vzdialenosť je maximálne 20-násobok výšky prekážok (ako napr. dediny, predmestský terén, súvislý les).	0.3	5
Oblasti, v ktorých je najmenej 15% povrchu pokrytého pozemnými stavbami, ktorých priemerná výška je väčšia ako 15m.	1.0	10

V momente keď určíme kategóriu terénu, sme schopný vypočítať pôsobiacu silu podľa vzorca (2.1) pre rôznu nadzemnú výšku. Následne sa využije integrálny výpočet, kedy postupne integrujeme cez celú plochu koruny a vždy berieme do úvahy plochu daného segmentu a výšku, v ktorej sa nachádza ťažisko tejto plochy. Súčet čiastkových síl vyjadruje výslednú silu pôsobiacu na korunu stromu. Situáciu ilustruje obrázok (2.1).



obrázok 2.1 – celková sila pôsobiaca na korunu stromu

Následne spočítame ohybový moment v prsnej výške (1.3 m).

$$M_O = F * (h_C - 1.3) \tag{2.6}$$

kde **F** je celková sila [N] a $\mathbf{h_C}$ je výška ťažiska koruny stromu [m]. Pre výpočet napätia v kmeni stromu v prsnej výške však potrebujeme poznať ešte hodnotu prierezového modulu. Pre kmeň s kruhovým prierezom platí vzťah:

$$W = \frac{\pi * d^3}{32} \tag{2.7}$$

kde **W** je prierezový modul [m^3] **d** je priemer kruhového prierezu [m]. Pre nepravidelné plochy sa prierezový modul počíta integračnou metódou popísanou v [4]. Napätie sa následne vyjadrí ako:

$$\sigma = \frac{M_O}{W} \tag{2.8}$$

Pomer pevnosti dreva a celkového napätia vznikajúceho v kmeni stromu v prsnej výške vo výsledku reprezentuje koeficient stability stromu. Hodnoty koeficientu v rozmedzí (0;1) sú považované za kritické a strom je klasifikovaný ako nestabilný. Hodnoty v rozmedzí <1;1.5> naznačujú, že strom je relatívne stabilný ale mala by sa mu venovať zvýšená pozornosť. Hodnoty nad 1.5 sú považované za uspokojivé a strom klasifikovaný ako stabilný.

Takýto matematický model je vhodný pre implementáciu vo forme počítačového programu, kde vstupné premenné definuje užívateľ. Výsledný koeficient stability je na týchto vstupoch závislý a mení sa spolu so vstupnými parametrami.

3 Analýza

Nasledujúca kapitola prezentuje priebeh softwarovej analýzy. Má za cieľ definovať účel vývoja, získať a špecifikovať požiadavky na software a podrobiť ich rozboru. Analyzovať prostriedky, ktorými bude aplikácia vytváraná a navrhnúť spôsob akým bude vývoj pokračovať.

Zadávateľ projektu jasne definoval cieľ vývoja. Vytvoriť počítačový program, ktorý na základe fotografie stromu, vložených informácií o geometrii stromu, type dreva a okolitých podmienkach spočíta koeficient stability stromu. K čomu sa táto hodnota používa a ako sa počíta popisuje kapitola 2. Vo fáze analýzy a špecifikácie požiadavok je nutné od klienta získať ďalšie informácie, ktoré pomôžu presne vymedziť ako sa má aplikácia správať, alebo naopak, ako sa správať nemá.

3.1 Definícia vstupov a výstupov

Na začiatku bolo nutné presne určiť všetky údaje, ktoré by mali byť výstupom programu. Nasledujúci zoznam je zhrnutím všetkých požadovaných informácií:

1) Textová reprezentácia (číselné vyjadrenie hodnôt)

- meno majiteľa stromu
- GPS súradnice lokácie
- druh stromu (taxonomické[7] zaradenie)
- výška stromu
- priemer kmeňa v prsnej výške
- výška nasadenia koruny
- rozmery koruny (výška a šírka)
- plocha koruny
- výška ťažiska koruny od zeme
- vzdialenosť ťažiska koruny od centrálnej línie stromu[8]
- kategória terénu
- referenčná rýchlosť vetra
- nadmorská výška lokácie
- priemerná teplota vzduchu v okolí
- celková sila pôsobiaca na korunu
- ohybový moment v prsnej výške
- ohybové napätie v prsnej výške
- koeficient stability

[7] taxonómia – vedná disciplína, ktorá definuje príslušnosť organizmu k určitej skupine organizmov so spoločnými znakmi a charakteristikami a túto skupinu pomenúva
[8] centrálna línia stromu – pomyselná priamka, ktorá spája najnižší a najvyšší bod kmeňa

2) Grafická reprezentácia (označenie na fotografii stromu)

- najnižší bod stromu
- najvyšší bod stromu
- centrálna línia
- poloha ťažiska koruny
- rozmery koruny (výška a šírka)
- výška nasadenia koruny
- poloha prsnej výšky

Po vymedzení požadovaných výstupov nasledovala fáza, ktorej cieľom bolo vytvoriť súbor vstupov, ktoré musí užívateľ do aplikácie vložiť. Medzi ne patrí každá informácia, ktorá je k výpočtu potrebná a zároveň sa nedá odvodiť z iných vstupov. Snaha zminimalizovať tieto údaje viedla k nasledovnému zoznamu užívateľských akcií. Ďalšie údaje nie sú pre požadované výstupy potrebné.

- vloženie fotografie nadzemnej časti stromu
- zadanie mena majiteľa a GPS súradnice lokácie
- označenie najnižšieho bodu kmeňa a smeru rastu kmeňa na fotografii
- vyznačenie koruny stromu na fotografii
- vloženie údajov o výške stromu a rozmeroch kmeňa v prsnej výške
- selekcia druhu stromu z pripravenej databázy, ktorá obsahuje hodnoty koeficientu aerodynamického odporu a pevnosti dreva
- vloženie údajov o nadmorskej výške lokácie, teplote okolitého vzduchu a referenčnej rýchlosti vetra
- výber typu okolitej krajiny – kategórie terénu

Ďalšou požiadavkou na systém bola jeho spustiteľnosť na operačnom systéme Windows vo verzii XP a vyššej. Preto bol ako implementačný jazyk doporučený jazyk C# na platforme .NET Framework 4.0. Vzhľadom k širšiemu okruhu užívateľov výsledného produktu bolo na mieste zvážiť použitie *Windows Presentation Foundation* ako nástroja pre tvorbu užívateľsky prívetivých grafických rozhraní.

3.2 Windows Presentation Foundation

Windows Presentation Foundation [8] (ďalej len WPF) je grafický podsystém pre renderovanie užívateľských rozhraní v aplikáciách bežiacich nad operačným systémom Windows. Je súčasťou .NET Framework-u od verzie 3.0. WPF je UI[9] framework, pomocou ktorého je možné vytvárať škálovateľné a vizuálne príťažlivé grafické rozhrania. Vznikol pod snahou zjednotiť rednerovanie 2D a 3D objektov, typografie, vektorovej grafiky a animácií. Namiesto staršieho vykreslovacieho jadra GDI (Graphics Device Interface) využíva *DirectX*.

DirectX je súbor aplikačných rozhraní pre správu úloh súvisiacich s multimédiami. Poskytuje rozhrania pre prácu so zvukom, videom i grafikou. Pristupuje priamo k ovládačom hardware-u a aplikácie využívajúce DirectX tak dosahujú vysoký výkon. Distribuuje sa v binárnej forme ako sada knižníc spolu s API dokumentáciou pre ľahšie použitie. WPF využíva DirectX najmä pre hardwarovú akceleráciu dynamického vykreslovania 2D a 3D objektov.

XAML

WPF oddeľuje užívateľské rozhranie od aplikačnej logiky. Dosahuje to pomocou súborov, napísaných v jazyku XAML [9]. V nich vývojár definuje celé užívateľské rozhranie, jeho vzhľad a rozloženie jednotlivých prvkov. Zjednodušuje tak tvorbu užívateľských rozhraní pre aplikácie postavené na .NET Frameworku.

Princíp je inšpirovaný úspechom značkovacích jazykov ako je XML v použití pre webové technológie. XAML je skratka pre eXtensible Application Markup Language a je to jazyk svojou štruktúrou podobný jazyku XML – je značkovací a deklaratívny. WPF disponuje implementáciou virtuálneho procesoru, ktorý poskytuje podporu tohoto jazyka a spracováva XAML súbory. Všetky triedy, ktoré WPF poskytuje pre tvorbu grafických užívateľských rozhraní sa tak dajú použiť nielen pomocou procedurálneho programovania ale aj v ich XAML reprezentácii. Obecne je v XAML súbore deklarovaný vzhľad prvkov a v triede ležiacej hierarchicky vedľa súboru (takáto trieda sa nazýva *code-behind*) je definované chovanie týchto prvkov. Výhodou je možnosť tvorby grafického rozhrania celkom separátne od zvyšku aplikácie, pretože nezávisí na použitých metódach v code-behind a dizajnér nepotrebuje poznať jazyk, v ktorom je zvyšok aplikácie vytvorený. Napomáha to oddeleniu *View* od *ViewModelu* a *Modelu* (viac o architektúre Model View ViewModel v kapitole 3.3). Samozrejme, nie vždy sa dá celé UI vytvoriť deklaratívne. V takom prípade je tu možnosť tvorby aj priamo v code-behind za pomoci klasického prístupu.

[9] UI – skratka z anglického výrazu „user interface“, v preklade užívateľské rozhranie

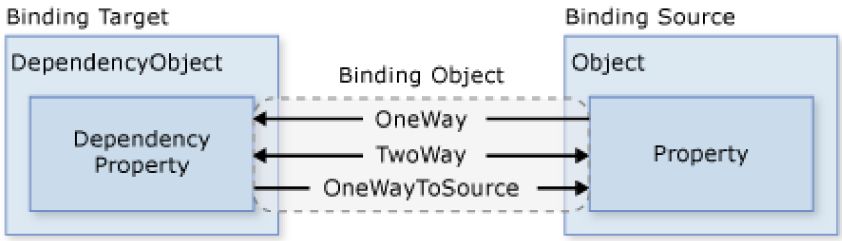
Data-binding

WPF predstavuje mnoho nových konceptov. Jedným z najdôležitejších je *data-binding* [10]. Poskytuje jednoduchý spôsob ako prezentovať dáta. UI elementy sa dajú naviazať na dáta z rôznych zdrojov, pričom tie zostávajú skryté pred prezentačnou časťou. Tento koncept umožňuje v XAML súbore deklarovať vlastnosť UI prvku (farba textu, rozmery prvku, poloha atď.) a jeho hodnotu naviazať na atribút zvolenej triedy. Vždy keď sa zmení tento atribút, užívateľské rozhranie automaticky reflektuje túto zmenu (či už je to zmena popisku tlačítka alebo zmena farby určitého prvku). Môže to platiť aj opačne. Ak užívateľ stlačí tlačítko alebo vpíše text do textového poľa, aktualizuje sa hodnota príslušného atribútu. Data-binding ešte viac podporuje separáciu aplikačnej logiky od UI.

Typické použitie je vytvorenie triedy, ktorá reprezentuje model určitých dát a následne s použitím data-bindigu spojiť tieto dáta s formulárom vytvoreným v užívateľskom rozhraní. Je tak možné implementovať napríklad triedu *Osoba*, ktorá má atribút *Adresa*, a jeho obsah naviazať na textové pole v UI formulári.

V data-bindingu existujú 4 rôzne smery toku dát. Použitie závisí na konkrétnej situácii.

1) **OneWay** binding aktualizuje vlastnosť cieľového UI prvku po zmene naviazaného atribútu priamo v kóde zdrojovej triedy. Tento typ bindingu je vhodný predovšetkým tam, kde vlastnosť prvku v UI je určená iba na čítanie a užívateľ nemá možnosť túto hodnotu meniť prostredníctvom rozhrania. Napríklad farba pozadia, text popisku atď. Ak teda nie je potrebné monitorovať zmeny tejto vlastnosti, OneWay binding je správna voľba.
2) **TwoWay** binding spôsobuje navyše aj aktualizáciu atribútu po zmene naviazaného elementu. Je vhodný všade tam, kde je vlastnosť UI prvku editovateľná a zároveň môže nastať situácia, kedy je zmena vlastnosti spôsobená programom samotným.
3) **OneWayToSource** binding je presným opakom OneWay bindingu. Aktualizuje atribút zdrojovej triedy prostredníctvom UI bez možnosti zmeniť vlastnosť UI prvku zo strany aplikácie.
4) **OneTime** binding je najmenej používaný typ, ktorý dovoľuje zdrojovému atribútu zmeniť vlastnosť v UI pri inicializácii ale ďalšie zmeny sa už neprejavia. Používa sa vtedy ak sú reprezentované dáta statické a nemenia sa v čase.



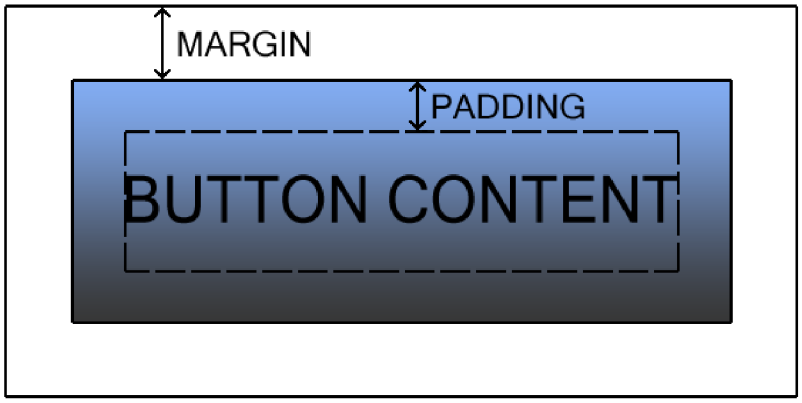
obrázok 3.1 – grafické znázornenie toku dát v data-bindingu (prevzaté z [11])

UI elementy a ich usporiadanie

Vo WPF je k dispozícii mnoho pripravených UI prvkov ako sú napríklad tlačítko, textové pole, popisok, zoznam, tabuľka, zaškrtávacie políčko a iné. Všetky sa dajú modifikovať a programátor tak môže do veľkej miery upraviť ich vzhľad a chovanie. Dajú sa neobmedzene kombinovať do väčších častí, ktoré možno skompilovať a používať opakovane v rôznych častiach aplikácie, prípadne v inej aplikácii. Takto vytvorený prvok sa nazýva *User Control* alebo *Custom Control*. Ide o mocný nástroj, ktorý dáva slobodu vo vytváraní komplikovanejších elementov so svojim vlastným a špecifickým správaním.

Každý UI prvok (či už základný, alebo User Control) musí mať v rozhraní svoje miesto [12], kde bude vykreslený. Technika, ktorou je rozhranie vytvárané (XAML) podmieňuje hierarchické usporiadanie prvkov. Základné vizuálne elementy sú teda vždy vložené v špeciálnych elementoch. Sú nimi napríklad *Grid*, ktorý umiestňuje prvky do tabuľky, pričom prvok si sám určí stĺpec aj riadok, kde bude umiestnený. Alebo *Canvas*, ktorý prvky vykresľuje do súradnicového systému s počiatkom v jeho ľavom hornom rohu. Prvok má opäť možnosť nastaviť, kde bude v Canvas-e vykreslený, udaním svojej polohy relatívne k počiatku súradnicového systému. Takýmto spôsobom sa elementy dajú skladať do väčších celkov a na pohľad pôsobia uceleným dojmom.

Hierarchické usporiadanie tried vo WPF umožňuje uplatniť princíp dedičnosti a tak je možné každému prvku nastaviť základné vizuálne vlastnosti ako sú šírka a výška. Navyše k lepšiemu usporiadaniu napomáhajú aj vlastnosti *Padding* a *Margin*. Padding udáva, koľko pixelov od každého okraja ostane voľných predtým ako sa vykreslí obsah prvku, Margin naopak určuje počet pixelov, ktoré musia zostať voľné zvonka prvku. Podobný koncept je uplatnený napríklad v jazyku CSS. Situáciu lepšie ilustruje obrázok 3.2.



obrázok 3.2 – grafické znázornenie vlastností Padding a Margin

Routed Events

Grafické rozhranie typickej WPF aplikácie pozostáva z viacerých UI elementov, ktoré sú usporiadané v stromovej štruktúre. *Routed Event* [13] je typ eventu[10], ktorý je obsluhovaný handlermi na viacerých prvkoch v strome UI elementov a nielen na zdrojovom elemente, ktorý event vyvolal. Takýto event „cestuje“ cez všetky prvky nadradené zdrojovému prvku. Ak teda existuje panel, ktorý obsahuje tlačítko, tak potom event kliknutia myši obslúži najprv tlačítko a následne aj panel. Tento spôsob sa nazýva *Bubbling* a existuje aj druhý typ routed eventu, kde je smer obsluhy eventu opačný. Nazýva sa *Tunneling* a v jeho prípade by sa najskôr obslúžil handler v paneli a až potom v tlačítku samotnom.

3.3 Model View ViewModel

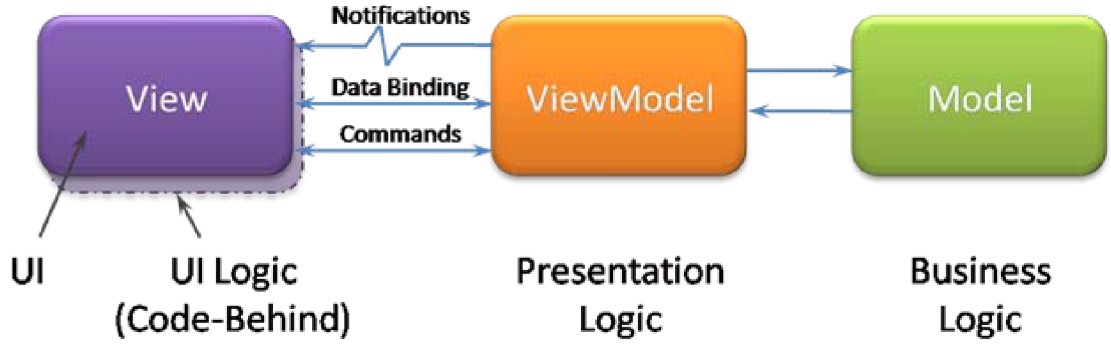
Implementácia software-u na profesionálnej úrovni nie je ľahká úloha. Častokrát sa počas vývoja stane vnútorná štruktúra programu neprehľadnou. Programátor tak ľahko môže narušiť integritu kódu. Ide najmä o vytváranie objektov, ktoré nesú príliš veľa zodpovednosti, starajú o viac úloh naraz alebo obsluhujú úlohy, ktoré z pohľadu logického usporiadania patria do inej časti kódu.

Tento problém sa snažia riešiť architektonické návrhové vzory. Najznámejšími sú *Model View Controller* (ďalej len MVC) a *Model View Presenter* (ďalej len MVP), ktoré sa úspešne používajú pri vývoji mnohých systémov. Spoločnosť Microsoft vyvinula vzor na podobnom princípe ako MVC a MVP a nazvala ho *Model View ViewModel* [14] (ďalej len MVVM).

[10] event – špeciálny dátový typ, ktorú reprezentuje udalosť ako napríklad stlačenie tlačítka alebo kliknutie myši; môže však vystupovať ako akákoľvek udalosť, ktorá v systéme nastala a je potrebné ju nejakým spôsoboom riešiť

Tento návrhový vzor sa skladá z 3 vzájomne prepojených súčastí, ktoré definujú základné skupiny pre usporiadanie tried v kóde aplikácie.

- **View** Súbor tried, ktoré obsahujú prvky grafického rozhrania a zobrazujú ich. Je to jediná časť, s ktorou príde užívateľ do priameho styku.
- **Model** Doménový model aplikácie. Popisuje dáta, s ktorými aplikácia pracuje. Často sem patria triedy, ktoré majú jeden účel a nespolupracujú s triedami, ktoré do domény nepatria. Väčšina tried má jasne definovaný vstup a výstup, pričom nemenia stav iných objektov. Príkladom môže byť trieda reprezentujúca geometrický útvar.
- **ViewModel** Reprezentuje strednú vrstvu, model prezentovaných dát. Je abstrakciou View a vystupuje ako prostredník medzi vrstvami View a Model. Mení informácie z Model-u tak, aby boli prezentovatelné vo View a naopak. Navyše deleguje príkazy, ktoré prichádzajú z View a rozhoduje sa, ktoré časti Model-u budú pre daný príkaz použité a akou formou.



obrázok 4.1 – schéma prepojenia jednotlivých častí v MVVM (prevzaté z [15])

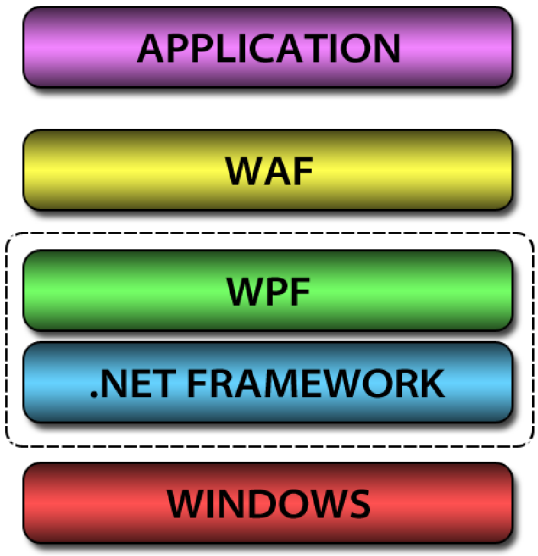
Návrhový vzor Model View ViewModel prináša riešenie [16], ako pristupovať k architektúre aplikácie. Je zameraný na vývoj aplikácií, ktoré sú založené na úzkej interakcii s užívateľom. S použitím WPF sa takýto typ aplikácií vytvára a preto je použitie vzoru MVVM vhodné.

Najdôležitejším aspektom WPF, ktorý podporuje použitie MVVM je data-binding (popísaný v kapitole 3.2). Naviazaním vlastností prvkov z View do ViewModel sa dosahuje voľnejšieho spojenia medzi komponentami. ViewModel nemusí „ručne“ aktualizovať View, všetko zariadi systém data-bindingu. Hlavným cieľom použitia MVVM je snaha o čo najčistejší code-behind. Všetky akcie spojené so spracovaním dát z View by mali byť smerované priamo na ViewModel. Ďalšie pravidlo, ktoré by sa malo dodržiavať, je úsilie presunúť čo najviac kódu

do Model-u. Táto časť je najjednoduchšie testovatelná a zaručuje dodržiavanie princípov voľného spojenia komponent. Zároveň ViewModel by mal byť najtenšia vrstva architektúry, pretože jeho hlavnou úlohou je delegovať požiadavky prichádzajúce z View a nemal by neobsahovať žiaden kód, ktorý by sa dal presunúť do Model-u.

3.4 WPF Application Framework

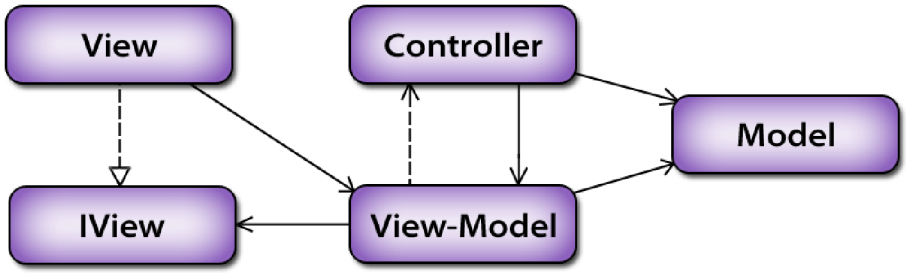
Pri tvorbe WPF aplikácie s pomocou návrhového vzoru MVVM je vhodné využiť niektorý z frameworkov, dostupných pre uľahčenie implementácie. Jedným z nich je WPF Application Framework [17] (ďalej len WAF). Je to jednoduchý open-source framework, súbor základných tried a nástrojov, ktorý pomáha dobre štruktúrovať kód aplikácie. Jeho umiestnenie v rámci celého vývojového prostredia je medzi aplikáciou a .NET Framework-om.



obrázok 4.2 – vrstevná achritektúra prostredia aplikácie

Medzi hlavné výhody WAF patrí podpora využívania návrhového vzoru MVVM. Obsahuje abstraktné triedy a rozhrania, určené pre využitie dedičnosti. Pomocou nich možno diferencovať triedy podľa ich príslušnosti k súboru View, Model alebo ViewModel. WAF navyše zavádza základné typy *IView* a *Controller*. Každá trieda, ktorá dedí z View musí implementovať interface, ktorý dedí z IView. Následne sa k View pristupuje iba pomocou tohto interface-u. Pri väčších systémoch je možné vytvoriť aj triedy, ktoré nie sú ViewModel pre žiaden View, ale plnia funkciu Controller-u. Typu triedy, ktorá úzko interaguje s ViewModel-om a Model-om. Spravuje udalosti a riadi tok dát.

Ďalšou užitočnou zložkou je trieda *DelegateCommand*, ktorá umožňuje jednoducho vytvárať handler-y[11] pre príkazy, prichádzajúce z View. Takto sa dajú viazať na ViewModel nielen vlastnosti ale aj udalosti.



obrázok 4.3 – zákadný vzťahový diagram pre aplikácie využívajúce WAF

3.5 Zhrnutie analýzy

Po diskusii so zadávateľom práce som vytvoril zoznam požiadaviek na software z technického aj obsahového hľadiska. Aplikácia musí byť spustiteľná pod operačným systémom Windows, verzie XP a vyššej. Bude vytvorená v implementačnom jazyku C# v prostredí .NET Framework-u verzie 4.0 s použitím Windows Presentation Foundation ako nástroja pre tvorbu vizuálne príťažlivých a užívateľsky prívetivých rozhraní. Bol vytvorený zoznam požadovaných užívateľských vstupov aj programových výstupov systému. Na jeho základe sa bude odvíjať návrh grafického rozhrania a logického usporiadania jednotlivých častí aplikácie.

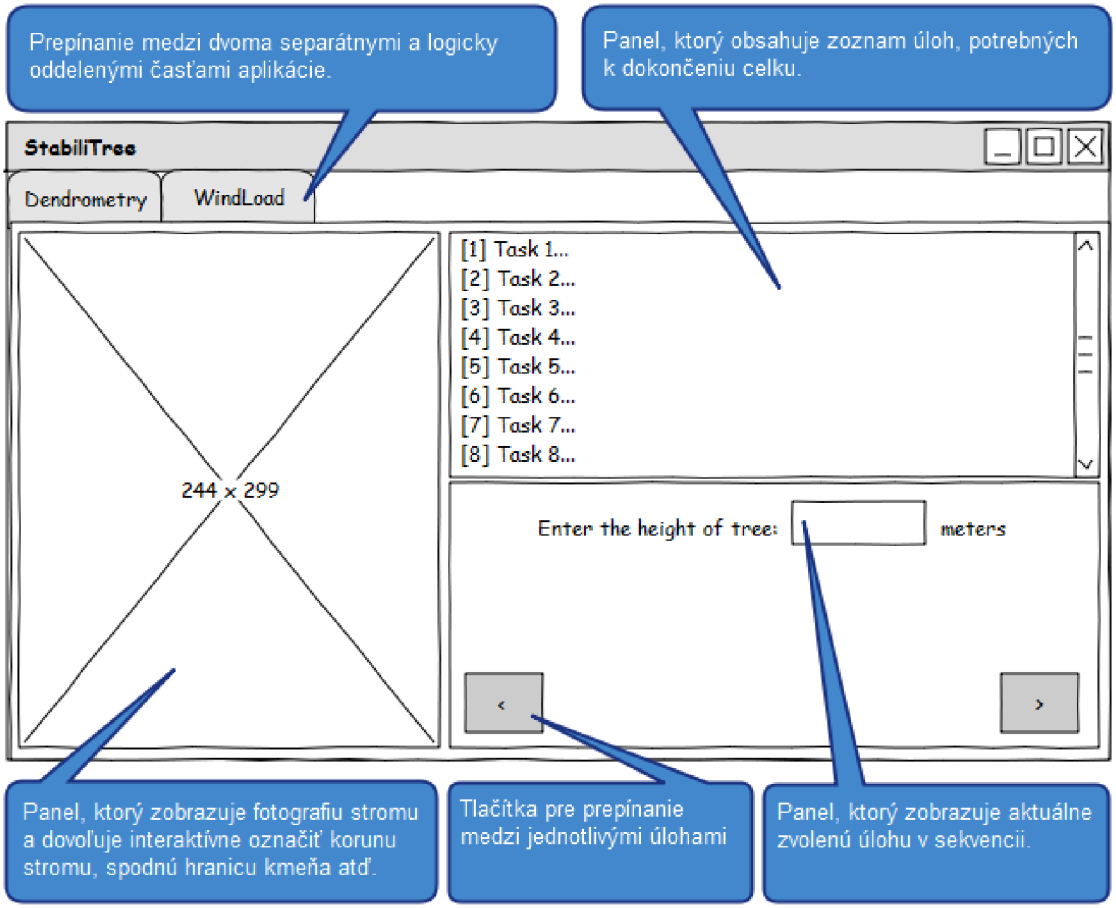
[11] handler – metóda, ktorá obsluhuje určitú udalosť

4 Návrh aplikácie

Kapitola popisuje akým spôsobom bude aplikácia štruktúrovaná a aké postupy sa použijú pri implementácii. Prezentuje návrh základných dátových štruktúr a ich vzájomných väzieb tak, aby architektúra aplikácie bola prehľadná, flexibilná, ľahko rozšíriteľná a udržiavatelná.

Predchádzajúca analýza definovala požiadavky kladené na software z pohľadu informačného prínosu pre užívateľa. Fáza architektonického návrhu vytvára vnútornú štruktúru kódu a pomáha lepšie zorganizovať jednotlivé časti do uceleného celku. Prvým krokom však bola tvorba prototypu grafického rozhrania.

4.1 Grafické rozhranie



obrázok 4.4 – návrh GUI

Grafické rozhranie by malo reflektovať účel danej časti aplikácie. Jednotlivé prvky by mali byť zoskupené podľa ich príslušnosti k celku so spoločným informačným prínosom. So znalosťou požiadavok užívateľa na vstupy a výstupy aplikácie som podľa týchto zásad navrhol grafické rozhranie pre aplikáciu (obrázok 4.4).

Úlohy, ktoré musí užívateľ splniť aby sa dopracoval k výsledku sa dali jednoducho formovať podľa analýzy požadovaných vstupov. Kostru aplikácie som potom navrhol tak, aby práca užívateľa s programom bola intuitívna a krok za krokom ho viedla k výsledku. Jednotlivé úlohy, nutné k jeho dosiahnutiu som rozčlenil do dvoch logických celkov. Prvý sa týka geometrie stromu – dendrometrie a cieľom celku je podať všetky dostupné informácie, týkajúce sa tejto oblasti. Druhý obsahuje údaje o prostredí stromu a jeho cieľom je výpočet koeficientu stability, k čomu využíva ďalšie užívateľské vstupy ale aj dáta vypočítané v prvej časti.

4.2 Prezentačná vrstva

Návrh prezentačnej vrstvy vyžadoval hlbokú analýzu možností neskoršej implementácie v oblasti hierarchie tried. Do značnej miery bol využitý princíp dedičnosti a polymorfizmu.

Hlavné okno a záložky

Základným elementom rozhrania je hlavné okno. Obsahuje 2 záložky, každá reprezentuje informačne jednotnú časť programu. Prvá záložka – *DendrometryTab* – spravuje údaje o geometrii stromu, druhá – *WindLoadAnalysisTab* – údaje o okolitom prostredí a zaoberá sa záťažou kladenou na strom. Pôvodný návrh grafického rozhrania bol koncipovaný tak, že podmienil celú vnútornú štruktúru a vzťahy medzi triedami vrstvy. Obe záložky majú zdieľať 3 panely, ktoré tvoria hlavný prístupový bod pre užívateľa.

- ***ImagePanel*** – Zobrazuje importovanú fotografiu stromu a poskytuje možnosť vyznačiť korunu stromu, najnižší bod kmeňa a jeho sklon. Rovnako zobrazuje grafickú reprezentáciu výsledkov.
- ***TaskSequencePanel*** – Obsahuje zoznam úloh pre aktívnu časť. Zoznam na záložke Dendrometry bude iný ako na záložke Wind Load Analysis a panel musí svoj obsah meniť v závislosti na aktívnej záložke.
- ***TaskDetailPanel*** – Zobrazuje detailný popis úlohy a navigačné tlačítka pre posun medzi úlohami. Jeho obsah sa teda mení spolu so zmenou aktuálnej úlohy.

Vo WPF nie je povolené aby viacero GUI prvkov malo spoločných potomkov. Ak teda 2 záložky majú zdieľať jednu instanciu každého panelu, musí byť vytvorený mechanizmus, ktorý odstráni panely z deaktivovanej záložky a vloží ich do práve aktivovanej. K tomuto účelu bude vytvorený interface *ITabView*. Hlavné okno podrží referencie na 2 instancie typu ITabView. Obe záložky potom implementujú tento interface, ktorý obsahuje 2 základné metódy.

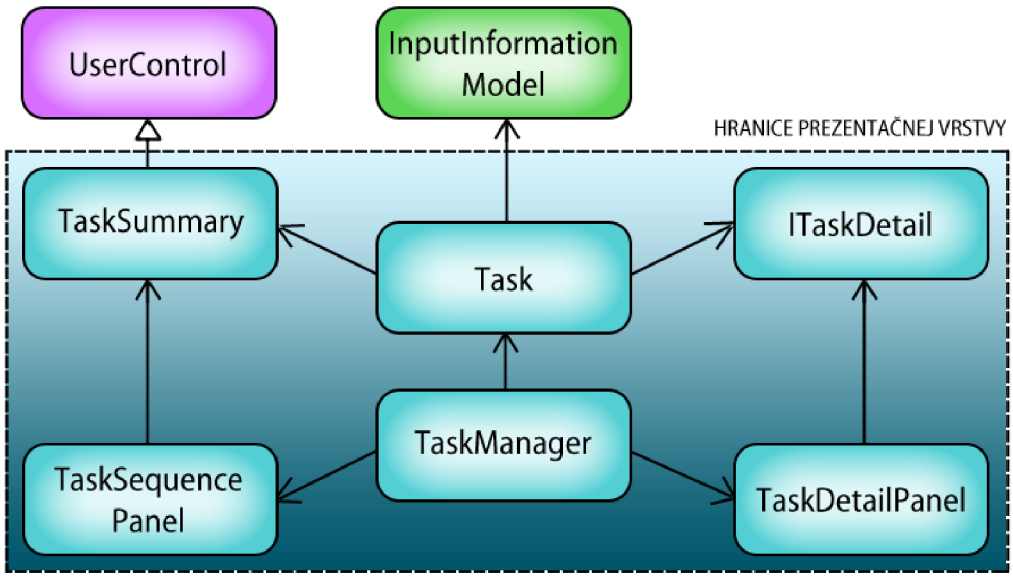
- *ActivateContent* – záložka pridá panely do zoznamu svojich potomkov
- *DeactivateContent* – záložka odstráni panely zo zoznamu potomkov

Vždy, keď hlavné okno zachytí žiadosť o prepnutie záložky, deaktivuje obsah na pôvodnej záložke a aktivuje ho na požadovanej bez toho, aby vedelo o akú záložku ide. Panely sa tak presunú z jednej záložky do druhej a nezmenia svoj obsah.

Správa úloh

Najdôležitejším bodom návrhu bola interná správa úloh, ktoré musí užívateľ splniť. Ich zoznam sa udržiava v TaskSequencePanel-i. Keď užívateľ klikne na položku, reprezentujúcu určitú úlohu, jej zadanie sa musí zobraziť v TaskDetailPanel-i.

Navrhol som triedu *Task*, ktorá reprezentuje jednu úlohu pre užívateľa. Okrem iného obsahuje referencie na objekt *TaskSummary* a objekt implementujúci rozhranie *ITaskDetail*. TaskSummary je *UserControl*, ktorý sa zobrazuje v TaskSequencePanel-i a obsahuje stručný popis úlohy. ITaskDetail je interface, ktorý musia implementovať všetky elementy, ktoré predstavujú zadanie úlohy. Toto zadanie je potom zobraziteľné v TaskDetailPanel. Panely tak nemusia poznať obsah týchto prvkov, iba ich zobrazujú. Ich životný cyklus bude spravovať objekt nazvaný *TaskManager*. Je zodpovedný za vytvorenie úloh a ich smerovanie do panelov.



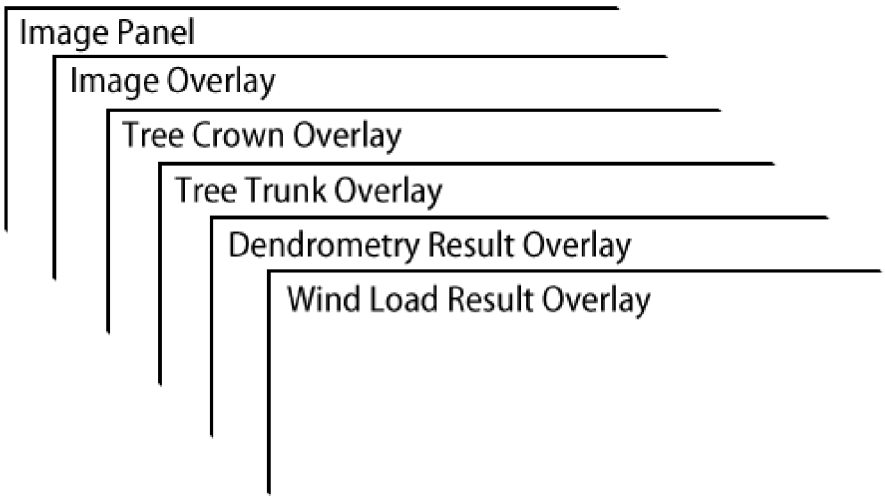
obrázok 4.7 – vzťahový diagram tried pre správu úloh

Image Panel

Hlavný účel triedy ImagePanel som popísal vyššie. Dôležitou úlohou bolo vymyslieť spôsob, akým v ňom môže byť usporiadaných viacero UI elementov tak, aby sa ich viditeľnosť menila podľa aktuálneho stavu programu. Niektoré úlohy totiž vyžadujú zobrazenie ďalších prvkov. Základným elementom, ktorý je zobrazený počas celého behu aplikácie je fotografia stromu. Počas plnenia úloh sa vyskytnú 4 situácie, kedy je nutné v ImagePanel zobraziť aj ďalšie prvky.

- nástroj, pomocou ktorého užívateľ vyznačí stred kmeňa a jeho sklon
- nástroj, ktorým užívateľ označí celú korunu stromu
- UI prvky, ktoré reprezentujú výsledky dendrometrie
- UI prvky, ktoré reprezentujú výsledky záťažovej analýzy

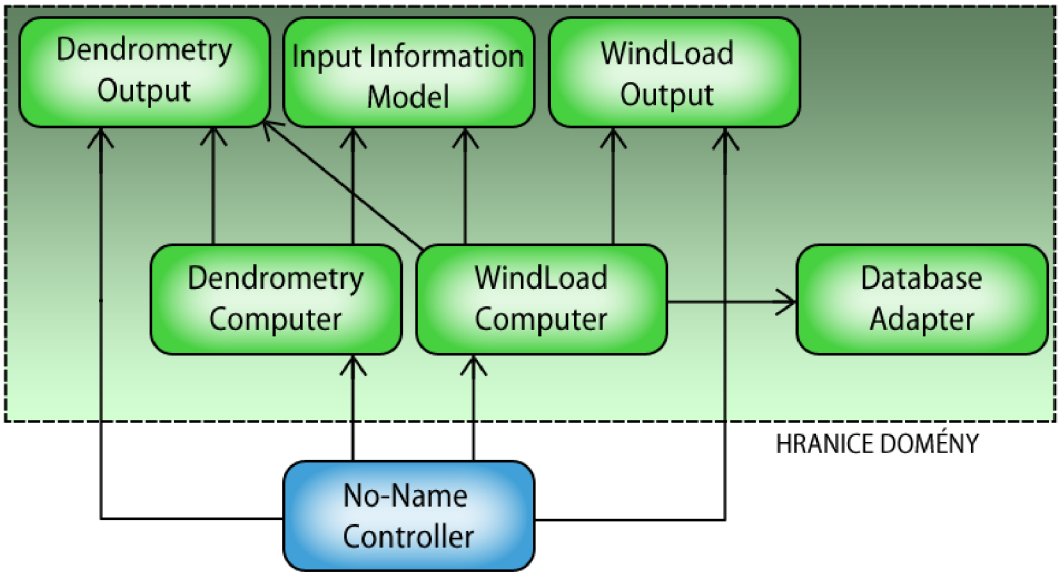
K tomuto účelu som navrhol systém vrstiev. Bude existovať trieda *Overlay*, ktorá bude zložená z transparentného plátna, na ktorý budú v odvodených triedach vykreslované UI elementy. Z tejto triedy budú špecializované vrstvy dediť. Obsahuje jednu metódu *OnParentSizeChanged*, ktorá sa volá vždy, keď ImagePanel zmení svoju veľkosť. Samotný ImagePanel teda nemusí poznať konkrétny typ vrstvy, pretože každá túto metódu zdedila od *Overlay*. Vrstva tak môže pružne reagovať a zmeniť polohu či veľkosť UI prvkov, ktoré obsahuje. Tieto 4 vrstvy sa potom vložia do panelu nad seba. Vďaka mechanizmu RoutedEvents (popísaný v kapitole 3.2) sa udalosti, ako napr. kliknutie myšou, zachytia vo všetkých vrstvách. Reagovať však na udalosť bude iba aktívna vrstva.



obrázok 4.6 – systém vrstiev

4.3 Doménová vrstva

Doménová vrstva predstavuje základný dátový a výpočtový model aplikácie. Vzhľadom k nadobudnutým znalostiam o cieľovom produkte a povahe aplikácie som vytvoril návrh jadra doménovej vrstvy.



obrázok 4.5 – diagram vzťahov medzi triedami doménovej vrstvy

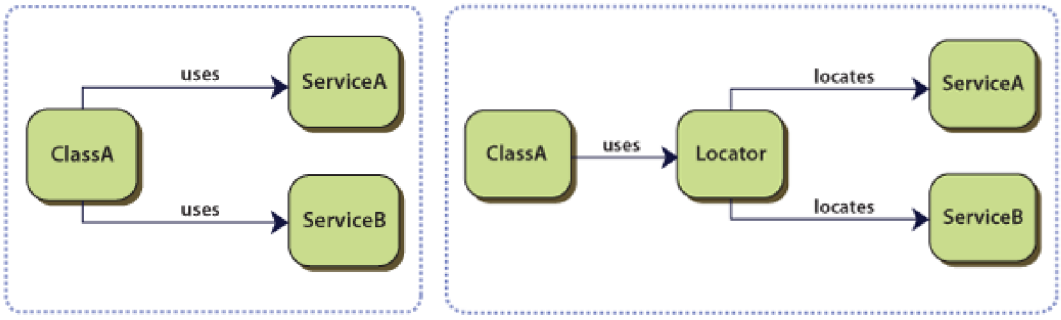
Trieda *InputInformationModel* udržiava infromácie zadané užívateľom v jednotlivých úlohách. *DendrometryComputer* a *WindLoadComputer* disponujú metódami, ktoré počítajú požadované výsledky a následne ich ukladajú do tried *DendrometryOutput* a *WindLoadOutput*, ku ktorým potom možno ľubovoľne pristupovať a čítať dáta. Trieda *DatabaseAdapter* reprezentuje triedu servisnej vrstvy. Abstrahuje súborový systém počítača a navonok poskytuje prístup k databázi stromov.

4.4 Inversion of Control

Klasický prístup k tvorbe objektov v OOP[12] vytvára väzby medzi objektami priamo. Objekt triedy **A** volá konštruktor triedy **B** a ten vytvorí nový objekt triedy **B**. Objekt **A** naň potom drží referenciu.

Inversion of Control [18] [19] je programovacia technika, pri ktorej objekty vytvára tretia strana a referencie vkladá tam, kde je to potrebné. Navyše tieto referencie sú často abstrahované použitím rozhraní (rozhranie tak ako ho chápe OOP). Takýto postup popisuje princíp nazvaný *Dependency Injection* [20]. V praxi to vyzerá tak, že objekt **A** potrebuje referenciu na objekt implementujúci rozhranie **I**. Tretia strana vytvorí objekty **A** a **B**, pričom **B** implementuje **I**. Následne vloží referenciu na **B** do objektu **A**. Výhodou takéhoto prístupu je najmä uvoľnenie väzieb medzi objektami.

Existuje niekoľko rôznych variácií techniky Inversion of Control. Niekdy sa využívajú rozhrania a kód je ľahšie testovateľný. Inokedy stačí vložiť referenciu do verejného atribútu objektu pomocou tretej strany. Ja som vzhľadom na zložitosť aplikácie zvolil jednoduchý prístup, ktorý najlepšie popisuje návrhový vzor *Service Locator*. Funguje ako „run-time“ linker, ktorý vkladá referencie do objektov až za behu programu, nie v čase kompilácie. Variácia konceptu Inversion of Control s použitím vzoru ServiceLocator je jednoduchá a pre daný účel efektívna. Konkrétny príklad použitia v aplikácií je popísaný v kapitole 5.2.



obrázok 4.8 – použitie vzoru Service Locator (prevzaté z [21])

[12] OOP – skratka pre objektovo orientované programovanie

5 Implementácia

Nasledujúca kapitola prezentuje čitateľovi proces implementácie programu. Venuje sa dôležitým alebo zaujímavým častiam a popisuje základnú štruktúru kódu. Koniec kapitoly sumarizuje túto časť vývojového procesu a zameriava sa na testovanie a profiling.

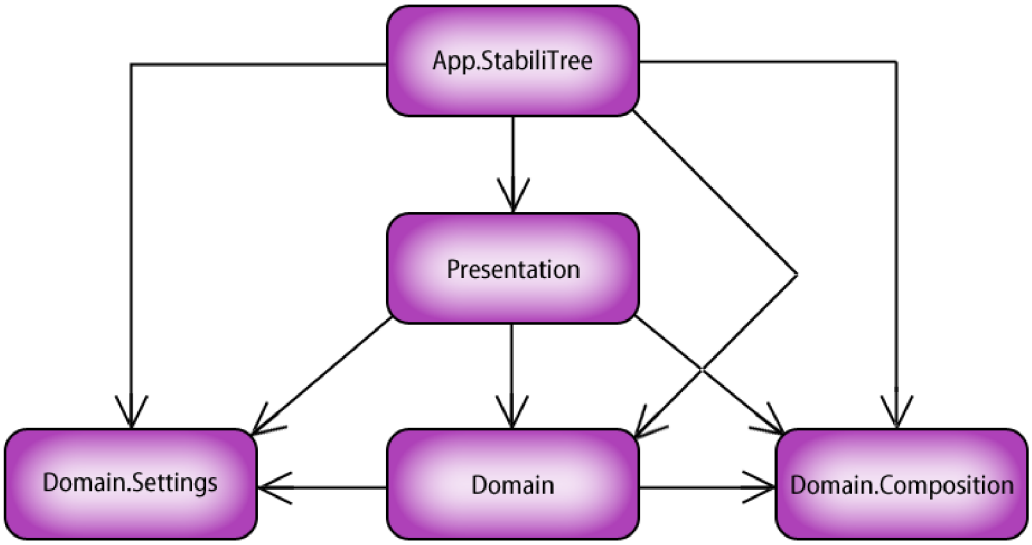
Vzhľadom na určený implementačný jazyk som ako vývojové prostredie zvolil Microsoft Visual Studio 2010. Disponuje množstvom prostriedkov a funkcií pre pohodlnú prácu v prostredí .NET. Napriek tomu, že je k dispozícii nástroj *Microsoft Blend for Visual Studio* pre tvorbu GUI, celé grafické rozhranie som vytváral ručne, deklarovaním prvkov v XAML súboroch. Okrem prostredia .NET Framework 4.0 som použil knižnice tretích strán:

- WPF Application Framework 2.5 – popísaný v kapitole 3.4
- WPF Property Grid 2.0 – Umožňuje zobraziť nejaký objekt ako zoznam jeho atribútov vo forme intuitívnej pre užívateľa. Ten potom môže atribúty editovať a hodnoty sa uložia do objektu.
- WPF Themes 1.0 – Sada grafických tém pre WPF aplikácie.

5.1 Štruktúra zdrojového kódu

Zdrojové kódy aplikácie sú usporiadané do piatich modulov, skompilovaných do dynamicky linkovaných knižníc (.dll súbory). Ako koreňový namespace som zvolil heslo Zephyr, ktoré reprezentuje pôvodné označenie projektu.

- **Zephyr.Domain.Settings** – obsahuje nastavenia aplikácie, externé zdroje (obrázky, ikony apod.) a reťazcové literály.
- **Zephyr.Domain.Composition** - obsahuje *CompositionCatalog* – triedu implementujúcu IoC koncept a jej popis možno nájsť v kapitole 5.2.
- **Zephyr.Domain** – súbor tried, ktoré logicky patria do domény aplikácie podľa vzoru MVVM (Model)
- **Zephyr.Presentation** – súbor tried, ktoré logicky patria do kontrolnej a prezentačnej vrstvy podľa vzoru MVVM (View a ViewModel)
- **Zephyr.App.StabiliTree** – obsahuje triedu *App*, ktorá reprezentuje vstupný bod aplikácie a triedu *ApplicationController*, ktorej funkcia je popísaná v kapitole 5.2.



obrázok 5.1 – diagram závislostí modulov v aplikácii

5.2 Composition Catalog

Pri tvorbe aplikácií je častým javom potreba vytvoriť iba jednu instanciu určitej triedy. Takúto potrebu spoľahlivo uspokojuje návrhový vzor *Singleton*. Ak sa však v aplikácii vyskytuje väčší počet tried, ktoré takýto princíp využijú, je jeho implementácia zdĺhavá a menej flexibilná. Navyše jeho uplatnenie znižuje znovupoužitelnosť triedy a pružnosť návrhu.

Vo svojej aplikácii som potreboval mechanizmus, ktorým bude možné jednoducho zdieľať jedinú instanciu triedy medzi viacerými objektami. Preto som navrhol triedu *ApplicationController*, ktorá zodpovedá za tvorbu hlavných objektov aplikácie. ApplicationController vytvorí instanciu z každého takého objektu ihneď po spustení programu. Následne ich vloží do špeciálneho kontajneru, ktorý sa nazýva *CompositionCatalog*. Je to objekt obsahujúci katalóg instancií. Pre každú triedu však môže byť vložená iba jedna instancia a tak sa zaistí dodržanie princípu uvedeného vyššie.

Základné metódy zaisťujúce kľúčovú funkcionalitu sú *ImportObject* a *ExportObject*. Funkcia ImportObject ako parameter obdrží objekt, ktorý je potrebné do katalógu vložiť. Detekuje triedu, ktorej objekt prináleží a zaznamená ju do interného registra. Akýkoľvek pokus o import instancie rovnakej triedy skončí výnimkou. Funkcia ExportObject je následne volaná ľubovolným objektom, ktorý stanoví triedu požadovanej instancie a funkcia vráti referenciu na vopred vložený objekt. CompositionCatalog sa nachádza v separátnej knižnici a sám o sebe je implementovaný ako Singleton. Obdržať naň referenciu potom môže prakticky ktokoľvek. Takýmto spôsobom sú odstránené príliš tesné väzby medzi objektami, tvoriacimi nosnú konštrukciu aplikácie.

5.3 User Controls

Definícia toho, čo je to User Control je popísaná v kapitole 3.2. V snahe vylepšiť intuitívnosť práce som vytvoril niekoľko vlastných User Control prvkov. Poskytujú užívateľovi väčší komfort pri práci s aplikáciou.

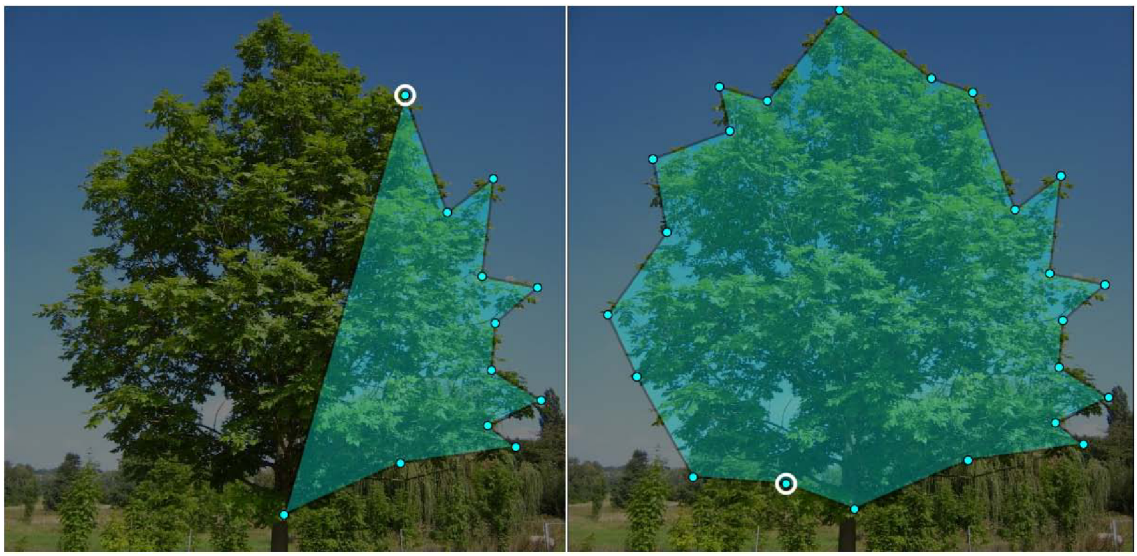
Protractor

Dve z úloh, ktoré musí užívateľ vykonať v priebehu práce sú označenie spodnej hranice kmeňa a vyznačenie sklonu kmeňa vzhľadom na zem. Tieto úlohy som zjednotil a pre ich splnenie vytvoril User Control, ktorý umožňuje oba úkony. Pozostáva z ukazovateľa v tvare kríža, ktorý možno presúvať po ploche. Slúži k tomu, aby ho užívateľ presunul na miesto, kde začína kmeň stromu. Následne je možné uchopiť vrchný ukazovateľ v tvare trojuholníka/šípky a jeho posunutím určiť sklon kmeňa. Vertikálna pozícia hrotu zároveň musí reflektovať spodnú hranicu koruny stromu. Takto sa určí tzv. výška nasadenia. Pomocné prerušované linky slúžia k lepšej vizuálnej orientácii.

obrázok 5.2 – Protractor

Flexible Polygon

Ďalšou dôležitou úlohou pre užívateľa je označiť korunu stromu. K tomuto účelu slúži Flexible Polygon. Kliknutím na plochu fotky sa vytvorí uzol – vrchol polygónu. Takýmto spôsobom užívateľ postupne vyznačí vonkajší obvod koruny. Jeho tvar by mal odrážať tvar koruny stromu. Jednotlivé vrcholy sa dajú presúvať, či zmazať, rovnako ako celý polygón. Posledný pridaný vrchol je vizuálne odlíšený.



obrázok 5.3 – Flexible Polygon

Môže však nastať situácia, kedy sa v korune vyskytuje diera, prípadne viac dier. V takomto prípade program umožňuje pridať ďalšie polygóny, ktoré sa od hlavného odčítajú a vznikne tak výsledný polygón, ktorý lepšie popisuje tvar koruny. Pre zaistenie takejto funkcionality bolo nutné implementovať booleovské operácie nad obecným polygónom. Použil som mierne upravený *Weiler-Atherton* algoritmus pre orezávanie polygónov aj s vnútornými otvormi. Funguje na princípe vytvorenia zoznamov vrcholov a priesečníkov hrán dvoch polygónov. Rôznymi priechodmi cez zoznamy možno získať usporiadaný zoznam vrcholov výsledného polygónu. Vytvoril som tak triedu *PolygonGeometry*, ktorá reprezentuje jeden obecný polygón a disponuje metódami pre booleovské operácie s iným objektom triedy PolygonGeometry. Zároveň je možné využiť funkcie, ktoré spočítajú jeho obsah, ťažisko alebo bounding-box[13].

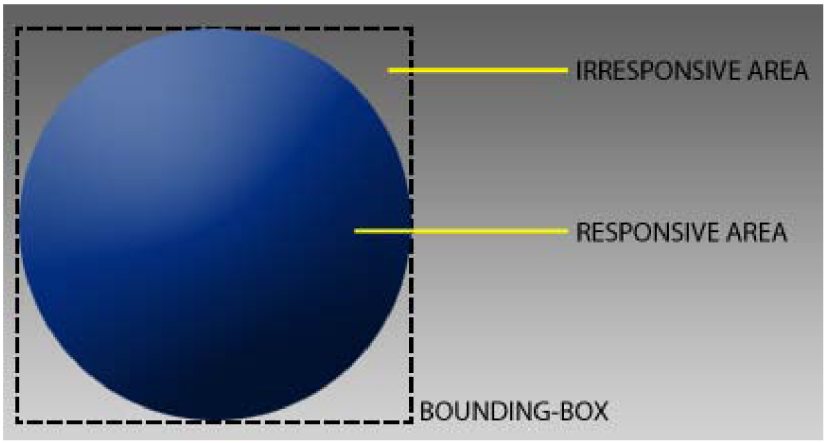
Transparent Image

Často je žiaduce v programe vytvoriť tlačítko, ktoré nemá klasický popisok ale nahrádza ho obrázok. Vo WPF sa dá tento problém vyriešiť jednoducho – priradením objektu reprezentujúceho obrázok do atribútu *Content* triedy *Button*. Avšak takéto tlačítko ostáva responzívne po celej ploche, určenej bounding-boxom obrázku. Preto som vytvoril triedu *TransparentImage*, dediacej z triedy *Image*, ktorá je súčasťou WPF. Metóda *HitTestCore* triedy *Image* má za úlohu zistiť, či sa súradnice ukazovateľa myši nachádzajú nad daným elementom alebo nie. Jej preťažením možno ovplyvniť výsledok.

Trieda *TransparentImage* v nej testuje či na mieste, kde sa súradnice nachádzajú, nie je alfa kanál obrázku nastavený na hodnotu 0 (úplná priehľadnosť). Ak áno, vráti výsledok

[13] bounding-box – najmenší obdĺžnik, ktorý ohraničuje 2D útvar

symbolizujúci, že súradnice sa nachádzajú mimo obrázku. V takom prípade je výsledné tlačítko neresponzívne. Tento algoritmus umožňuje vytvárať tlačítka ľubovolného tvaru, záleží len na nastavení alfa kanálu v obrázku.



obrázok 5.4 – Transparent Image

5.4 Testovanie

Počas implementácie som časti programu viackrát podrobil testovaniu s cieľom nájsť odstrániť zvyšné chyby, prípadne vylepšiť stávajúci systém. Najskôr som program testoval sám, pričom som sa zameriaval na vytvorenie málo pravdepodobných stavov a okrajových podmienok pri užívateľských vstupoch. Snažil som sa postihnúť čo najširšie spektrum situácií, do ktorých sa môže program dostať. Výsledkom bolo ošetrenie viacerých chybových stavov.

Dôležitým krokom pri testovaní bola validácia výsledkov. K tomu bol použitý program TreeStab, dodaný Ing. Luďkom Prausom Ph.D. z Lesníckej a dřevařskej fakulty Mendelovej Univerzity v Brně. Do oboch programov boli zadané rovnaké vstupy a porovnávali sa spočítané výsledky. Vzhľadom na použitie rovnakého matematického modelu boli výsledky totožné.

Ďalej nasledovala fáza užívateľského testovania, pri ktorom aplikáciu používalo viacero ľudí, väčšina z nich s IT vzdelaním. Na základe ich podnetov som do aplikácie implementoval niektoré doplňujúce funkcie, ktoré zvyšujú komfort pri práci. Zároveň im bol predložený dotazník obsahujúci 2 otázky týkajúce sa práce s programom.

obrázok 5.5 – spätná väzba od testovacej skupiny

Na záver som výkon aplikácie preveril profilingom[14]. Microsoft Visual Studio 2010 disponuje nástrojom pre automatický profiling a dynamickú analýzu programu. Nenašiel som však žiadny úsek kódu, ktorý by značne brzdil aplikáciu alebo mal výrazne väčšie nároky na operačnú pamäť. V tomto smere sa pozitívne prejavilo predošlé úsilie a testovanie aplikácie už počas implementácie.

[14] profiling – vyhľadanie miest v programe, ktoré sú vhodné k optimalizácii z hľadiska výkonu

6 Záver

Záverečná kapitola rekapituluje celú prácu a proces vývoja aplikácie. Obsahuje zhodnotenie dosiahnutých výsledkov a ponúka možnosti ďalšieho vývoja.

6.1 Proces vývoja

Pred začiatkom vývoja som musel preniknúť do pozadia problému a pochopiť čo je to stabilita stromov, ako sa meria a prečo je dôležité ju poznať. Preto mojou prvou úlohou bolo štúdium teoretického základu, pri ktorom som mal k dispozícii materiály poskytnuté Lesnickou a dřevařskou fakultou Mendelovy Univerzity v Brně.

S programovaním desktopových aplikácií v jazyku C# v prostredí .NET Framework-u som mal skúsenosti už pred začiatkom vývoja programu StabiliTree. Ďalším cieľom preto bolo štúdium platformy Windows Presentation Foundation a možností jej využitia. Potom som sa musel oboznámiť s návrhovým vzorom Model View ViewModel ako architektonického návrhového vzoru pre tvorbu rozsiahlejších aplikácií. Po zvládnutí tejto úlohy som sa sústredil na výber frameworku, ktorý podporuje vývoj vo WPF s použitím MVVM. Vybral som si WPF Application Framework pre jednoduchosť použitia a open-source licenciu.

Vo fáze návrhu aplikácie som sa zameral na interaktivitu programu a celú aplikáciu koncipoval tak aby bol užívateľ krok za krokom vedený k výsledku. Najťažším bodom bolo navrhnúť grafické rozhranie a následne premyslieť systém správy úloh, ktoré treba splniť na ceste k finálnemu zhodnoteniu stability.

Implementácia prebiehala podľa plánu a z veľkej časti som sa pridržal konceptu vytvorenom vo fáze návrhu. Následne bol program testovaný viacerými užívateľmi a dosiahnuté výsledky validované.

6.2 Zhodnotenie výsledkov

Výsledkom tejto bakalárskej práce je aplikácia, ktorá umožňuje arboristom jednoducho spočítať koeficient stability stromu a pomáha tak pri hodnotení jeho prevádzkovej bezpečnosti. Predstavuje základnú verziu softwaru, ktorý sa zaoberá stabilitou stromov a jej hodnotením. Bude prezentovaná ako voľne dostupná verzia, pričom jej vývoj bude pokračovať, smerujúc

k platenej verzii pre komerčné účely. Aplikácia s takotou funkcionalitou momentálne na trhu neexistuje a arboristi sú nútení hodnotiť stabilitu stromu ručne, prípadne s využitím vlastných nástrojov. Program StabiliTree teda ponúka možnosť ako im prácu uľahčiť a celý proces zjednodušiť.

6.3 Možnosti ďalšieho vývoja

Aplikácia je vytvorená ako súčasť väčšieho projektu. Zatiaľ pozostáva z dvoch častí: geometria stromu a výpočet záťaže kladenej vetrom. Vývoj bude pokračovať s cieľom ešte spoľahlivejšie ohodnotiť stabilitu. V ďalšej časti bude výpočet koeficientu stability spresnený a s využitím počítačovej tomografie sa vyšetrí kmeň stromu. Užívateľ potom vyznačí skutočný tvar kmeňa vrátane dutín a tak je možné presnejšie spočítať prierezový modul a tým aj napätie, ktoré v kmeni pri zaťažení vzniká. Následne je možné do aplikácie zakomponovať časť, kde sa vložia výsledky reálneho experimentu – ťahovej skúšky. Namerané hodnoty sa porovnajú s výsledkami matematického modelu a bude možné spoľahlivejšie zhodnotiť bezpečnosť stromu.

7 Literatúra

[1] Přístrojové hodnocení stability stromů. [online]. [cit. 2013-05-03]. Dostupné z: http://www.zahrada-park-krajina.cz/index.php?option=com_content&view=article&id=263:pistrojove-hodnoceni-stability-strom&catid=61:zakladani-a-udrba-zelen&Itemid=122

[2] Andreas Detter, Erk Brudi, Frank Bischoff; Statics Integrated Methods; 2005

[3] Stanovení zatížení větrem podle ČSN EN 1991-1-4. [online]. [cit. 2013-05-03]. Dostupné z: http://concrete.fsv.cvut.cz/~mosty/Pomucky/Zatizeni/Zat_Vitr_EN-1991-1-4.pdf

[4] Luděk Praus; Mechanická stabilita stromů a metody jejího zjišťování; 2006

[5] Praus L., Horacek P.; Assessment of tree stability - The mechanical behaviour of a tree; 2005

[6] Odpor prostředí. [online]. [cit. 2013-05-03]. Dostupné z: http://cs.wikipedia.org/wiki/Odpor_prost%C5%99ed%C3%AD

[7] ČSN EN 1991-1-4, Eurocode 1: Actions on structures - General actions - Part 1-4: Wind actions, 2004

[8] MSDN - Introduction to WPF. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/aa970268.aspx

[9] MSDN - XAML Overview (WPF). [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms752059.aspx

[10] MSDN - Data Binding Overview. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms752347.aspx

[11] MSDN - Data Binding Overview. [online]. [cit. 2013-05-03]. Dostupné z: http://i.msdn.microsoft.com/dynimg/IC58485.png

[12] MSDN - Layout. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms745058.aspx

[13] MSDN - Routed Events Overview. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms742806.aspx

[14] MSDN – WPF Apps With The Model-View-ViewModel Design Pattern. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[15] MSDN – Model View ViewModel. [online]. [cit. 2013-05-03]. Dostupné z: http://blogs.msdn.com/blogfiles/dphill/WindowsLiveWriter/CollectionsAndViewModels_EE56/ViewModel_2.png

[16] Model-View-ViewModel (MVVM) Explained. [online]. [cit. 2013-05-03]. Dostupné z: http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

[17] WPF Application Framework (WAF). [online]. [cit. 2013-05-03]. Dostupné z: http://waf.codeplex.com/

[18] Inversion of Control. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff921087(v=pandp.20).aspx

[19] Inversion of Control – An Introduction with Examples in .NET. [online]. [cit. 2013-05-03]. Dostupné z: http://joelabrahamsson.com/inversion-of-control-an-introduction-with-examples-in-net/

[20] Design pattern – Inversion of control and Dependency injection. [online]. [cit. 2013-05-03]. Dostupné z: http://www.codeproject.com/Articles/29271/Design-pattern-Inversion-of-control-and-Dependency

[21] The Service Locator Pattern. [online]. [cit. 2013-05-03]. Dostupné z: http://msdn.microsoft.com/en-us/library/ff648968.aspx