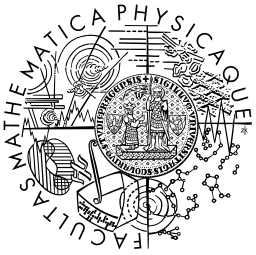


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matej Kukurugya

Trasy v mapách pro orientační běh

Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: RNDr. Ondřej Pangrác, Ph.D.

Studijní program: Informatika (B0613A140006)

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování.

Název práce: Trasy v mapách pro orientační běh

Autor: Matej Kukurugya

ústav: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: RNDr. Ondřej Pangrác, Ph.D., Informatický ústav Univerzity Karlovy

Abstrakt: Původním záměrem této bakalářské práce bylo vytvořit aplikaci, která bude na základe mapových dat hledat vhodnou trasu v prostředí orientačního běhu. Tento záměr však nebyl plně naplněn, neboť až příliš času zabralo vytvoření návrhu a naprogramování samotné aplikace. Tím pádem nezbyl čas na vytvoření vyhledávacích alogirtmů ani mechanizmů na spracováni map na kterých by jse cesty hledaly. Z tohto důvodu vznikla jenom 'skořápka' aplikace, která toho v týhle chvíli sama o sobě nic nedelá. Objektový návrh je za to velmi detailně promyšlen a propracován. Tato bakalářská práce tedy v konečném důsledku pojednává právě o objektovém návrhu vytovřené aplikace a myšlenkách aplikovaných v průběhu jeho tvorby.

Klíčová slova: objektový návrh, MVVM, hledáni tras v otevřeném terénu

Title: Navigation in orienteering maps

Author: Matej Kukurugya

institute: Computer science institute of Charles University

Supervisor: RNDr. Ondřej Pangrác, Ph.D., Computer science institute of Charles University

Abstract: The prior intention of this bachelor thesis was creating and application which based on map data will look for suitable route in orienteering environment. This intention was not completly fulfilled though. Creating and programming of application itsel took to much time so there was left none for creating of usable searching algorithms nor mechanisms for processing of maps on which routes would be looked for. For this reason the 'eggshell' of application was created which currently does not do nothing usefull. On the other side design of application is thought out and elaborated in detail. This bachelor thesis talks in the end specifically about design of created application and ideas applied during its creation process.

Keywords: object desing, MVVM, path finding in open terrain

Obsah

Úvod

Následuje několik ukázkových kapitol, které doporučují, jak by se měla závěrecná práce sázet. Primárně popisují použití TEXové šablony, ale obecné rady poslouží dobře i uživatelům jiných systémů.

1 Nápověda k sazbě

1.1 Úprava práce

Vlastní text práce je uspořádaný hierarchicky do kapitol a podkapitol, každá kapitola začíná na nové straně. Text je zarovnán do bloku. Nový odstavec se obvykle odděluje malou vertikální mezerou a odsazením prvního řádku. Grafická úprava má být v celém textu jednotná.

Práce se tiskne na bílý papír formátu A4. Okraje musí ponechat dost místa na vazbu: doporučen je horní, dolní a pravý okraj 25 mm, levý okraj 40 mm. Číslují se všechny strany kromě obálky a informačních stran na začátku práce; první číslovaná strana bývá obvykle ta s obsahem.

Písmo se doporučuje dvanáctibodové (12 pt) se standardní vzdáleností mezi řádky (pokud píšete ve Wordu nebo podobném programu, odpovídá tomu řádkování 1,5; v TEXu není potřeba nic přepínat).

Primárně je doporučován jednostranný tisk (příliš tenkou práci lze obtížně svázat). Delší práce je lepší tisknout oboustranně a přizpůsobit tomu velikosti okrajů: 40 mm má vždy *vnitřní* okraj. Rub titulního listu zůstává nepotištěný.

Zkratky použité v textu musí být vysvětleny vždy u prvního výskytu zkratky (v závorce nebo v poznámce pod čarou, jde-li o složitější vysvětlení pojmu či zkratky). Pokud je zkratek více, připojuje se seznam použitých zkratek, včetně jejich vysvětlení a/nebo odkazů na definici.

Delší převzatý text jiného autora je nutné vymezit uvozovkami nebo jinak vyznačit a řádně citovat.

1.2 Jednoduché příklady

K různým účelům se hodí různé typy písma. Pro běžný text používáme vzpřímené patkové písmo. Chceme-li nějaký pojem zvýraznit (třeba v okamžiku definice), používáme obvykle *kurzívu* nebo **tučné písmo.** Text matematických vět se obvykle tiskne pro zdůraznění *skloněným (slanted)* písmem; není-li k dispozici, může být zastoupeno *kurzívou.* Text, který je chápan doslova (například ukázky programů) píšeme `psacím strojem`. Důležité je být ve volbě písma konzistentní napříč celou prací.

Čísla v českém textu obvykle sázíme v matematickém režimu s desetinnou čárkou: $\pi \doteq 3{,}141\,592\,653\,589$. V matematických textech je často lepší používat desetinnou tečku (pro lepší odlišení od čárky v roli oddělovače). Nestřídejte však obojí. Numerické výsledky se uvádějí s přiměřeným počtem desetinných míst.

Mezi číslo a jednotku patří úzká mezera: šířka stránky A4 činí 210 mm, což si pamatuje pouze 5 % autorů. Pokud ale údaj slouží jako přívlastek, mezeru vynecháváme: 25mm okraj, 95% interval spolehlivosti.

Rozlišujeme různé druhy pomlček: červeno-černý (krátká pomlčka), strana 16–22 (střední), $45 - 44$ (matematické minus), a toto je — jak se asi dalo čekat — vložená věta ohraničená dlouhými pomlčkami.

V českém textu se používají „české“ uvozovky, nikoliv “anglické”.

Na některých místech je potřeba zabránit lámání řádku (v~TEXu značíme vlnovkou): u~předložek (neslabičnych, nebo obecně jednopísmenných), vrchol~v,

před k~kroky, a~proto, … obecně kdekoliv, kde by při rozlomení čtenář „škobrtnul“.

1.3 Matematické vzorce a výrazy

Proměnné sázíme kurzívou (to TEX v matematickém módu dělá sám, ale nezapomínejte na to v okolním textu a také si matematický mód zapněte). Názvy funkcí sázíme vzpřímeně. Tedy například: $\operatorname{var}(X) = \mathsf{E}\, X^2 - \left(\mathsf{E}\, X\right)^2$.

Zlomky uvnitř odstavce (třeba $\frac{5}{7}$ nebo $\frac{x+y}{2}$) mohou být příliš stísněné, takže je lepší sázet jednoduché zlomky s lomítkem: $5/7$, $(x+y)/2$.

Není předepsáno, jakým písmem označovat jednotlivé druhy matematických objektů (matice, vektory, atd.), ale značení pro tentýž druh objektu musí být v celé práci používáno stejně. Podobně používáte-li více různých typů závorek, je třeba dělat to v celé práci konzistentně.

Nechť

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{pmatrix}.$$

Povšimněme si tečky za maticí. Byť je matematický text vysázen ve specifickém prostředí, stále je gramaticky součástí věty a tudíž je zapotřebí neopomenout patřičná interpunkční znaménka. Obecně nechceme sázet vzorce jeden za druhým a raději je propojíme textem.

Výrazy, na které chceme později odkazovat, je vhodné očíslovat:

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_n^\top \end{pmatrix}. \tag{1.1}$$

Výraz (1.1) definuje matici $\mathbf{X}$. Pro lepší čitelnost a přehlednost textu je vhodné číslovat pouze ty výrazy, na které se autor někde v další části textu odkazuje. To jest, nečíslujte automaticky všechny výrazy vysázené některým z matematických prostředí.

Zarovnání vzorců do několika sloupečků:

$$\begin{aligned} S(t) &= \mathsf{P}(T > t), \qquad t > 0 \qquad \text{(zprava spojitá)}, \\ F(t) &= \mathsf{P}(T \leq t), \qquad t > 0 \qquad \text{(zprava spojitá)}. \end{aligned}$$

Dva vzorce se spojovníkem:

$$\left.\begin{aligned} S(t) &= \mathsf{P}(T > t) \\ F(t) &= \mathsf{P}(T \leq t) \end{aligned}\right\} \quad t > 0 \qquad \text{(zprava spojité)}. \tag{1.2}$$

Dva centrované nečíslované vzorce:

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

$$\mathbf{X} = \begin{pmatrix} 1 & \boldsymbol{x}_1^\top \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\top \end{pmatrix}.$$

Dva centrované číslované vzorce:

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{1.3}$$

$$\mathbf{X} = \begin{pmatrix} 1 & \boldsymbol{x}_1^\top \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_n^\top \end{pmatrix}. \tag{1.4}$$

Definice rozdělená na dva případy:

$$P_{r-j} = \begin{cases} 0, & \text{je-li } r-j \text{ liché,} \\ r!\,(-1)^{(r-j)/2}, & \text{je-li } r-j \text{ sudé.} \end{cases}$$

Všimněte si použití interpunkce v této konstrukci. Čárky a tečky se dávají na místa, kam podle jazykových pravidel patří.

$$\begin{aligned} x &= y_1 - y_2 + y_3 - y_5 + y_8 - \cdots = && \text{z (1.3)} \\ &= y' \circ y^* = && \text{podle (1.4)} \\ &= y(0)y' && \text{z Axiomu 1.} \end{aligned} \tag{1.5}$$

Dva zarovnané vzorce nečíslované (povšimněte si větších závorek, aby se do nich vešel vyšší vzorec):

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{n} f_i(y_i;\, \boldsymbol{\theta}), \\ \ell(\boldsymbol{\theta}) &= \log\Big\{L(\boldsymbol{\theta})\Big\} = \sum_{i=1}^{n} \log\Big\{f_i(y_i;\, \boldsymbol{\theta})\Big\}. \end{aligned}$$

Dva zarovnané vzorce, první číslovaný:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \prod_{i=1}^{n} f_i(y_i;\, \boldsymbol{\theta}), \\ \ell(\boldsymbol{\theta}) &= \log\Big\{L(\boldsymbol{\theta})\Big\} = \sum_{i=1}^{n} \log\Big\{f_i(y_i;\, \boldsymbol{\theta})\Big\}. \end{aligned} \tag{1.6}$$

Vzorec na dva řádky, první řádek zarovnaný vlevo, druhý vpravo, nečíslovaný:

$$\begin{aligned} \ell(\mu, \sigma^2) = \log\Big\{L(\mu, \sigma^2)\Big\} &= \sum_{i=1}^{n} \log\Big\{f_i(y_i;\, \mu, \sigma^2)\Big\} = \\ &= -\frac{n}{2}\,\log(2\pi\sigma^2) \,-\, \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2. \end{aligned}$$

Vzorec na dva řádky, zarovnaný na =, číslovaný uprostřed:

$$\begin{aligned} \ell(\mu, \sigma^2) = \log\Big\{L(\mu, \sigma^2)\Big\} &= \sum_{i=1}^{n} \log\Big\{f(y_i;\, \mu, \sigma^2)\Big\} = \\ &= -\frac{n}{2}\,\log(2\pi\sigma^2) \,-\, \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2. \end{aligned} \tag{1.7}$$

1.4 Definice, věty, důkazy, ...

Konstrukce typu definice, věta, důkaz, příklad, ... je vhodné odlišit od okolního textu a případně též číslovat s možností použití křížových odkazů. Pro každý typ těchto konstrukcí je vhodné mít v souboru s makry (`makra.tex`) nadefinované jedno prostředí, které zajistí jak vizuální odlišení od okolního textu, tak automatické číslování s možností křížově odkazovat.

Definice 1. *Nechť náhodné veličiny* $X_1, \ldots, X_n$ *jsou definovány na témž pravděpodobnostním prostoru* $(\Omega, \mathcal{A}, P)$. *Pak vektor* $\boldsymbol{X} = (X_1, \ldots, X_n)^\top$ *nazveme* náhodným vektorem.

Definice 2 (náhodný vektor)**.** *Nechť náhodné veličiny* $X_1, \ldots, X_n$ *jsou definovány na témž pravděpodobnostním prostoru* $(\Omega, \mathcal{A}, P)$. *Pak vektor* $\boldsymbol{X} = (X_1, \ldots, X_n)^\top$ *nazveme* náhodným vektorem.

Definice 1 ukazuje použití prostředí pro sazbu definice bez titulku, definice 2 ukazuje použití prostředí pro sazbu definice s titulkem.

Věta 1. *Náhodný vektor* $\boldsymbol{X}$ *je měřitelné zobrazení prostoru* $(\Omega, \mathcal{A}, P)$ *do* $(\mathbb{R}_n, \mathcal{B}_n)$.

Lemma 2 (Anděl [1], str. 29)**.** *Náhodný vektor* $\boldsymbol{X}$ *je měřitelné zobrazení prostoru* $(\Omega, \mathcal{A}, P)$ *do* $(\mathbb{R}_n, \mathcal{B}_n)$.

Důkaz. Jednotlivé kroky důkazu jsou podrobně popsány v práci Anděl [1, str. 29]. □

Věta 1 ukazuje použití prostředí pro sazbu matematické věty bez titulku, lemma 2 ukazuje použití prostředí pro sazbu matematické věty s titulkem. Lemmata byla zavedena v hlavním souboru tak, že sdílejí číslování s větami.

2 Odkazy na literaturu

Při zpracování bibliografie (přehledu použitých zdrojů) se řídíme normou ISO 690 a zvyklostmi oboru. V LaTeXu nám pomohou balíčky biblatex, biblatex-iso690. Zdroje definujeme v souboru `literatura.bib` a pak se na ně v textu práce odkazujeme pomocí makra `\cite`. Tím vznikne odkaz v textu a odpovídající položka v seznamu literatury.

V matematickém textu obvykle odkazy sázíme ve tvaru „Jméno autora/autorů [číslo odkazu]“, případně „Jméno autora/autorů (rok vydání)“. V českém/slovenském textu je potřeba se navíc vypořádat s nutností skloňovat jméno autora, respektive přechylovat jméno autorky. K doplňování jmen se hodí příkazy `\citet`, `\citep` z balíčku natbib, ale je třeba mít na paměti, že produkují referenci se jménem autora/autorů v prvním pádě a jména autorek jsou nepřechýlena.

Jména časopisů lze uvádět zkráceně, ale pouze v kodifikované podobě.

Při citování je třeba se vyhnout neověřitelným, nedohledatelným a nestálým zdrojům. Doporučuje se pokud možno necitovat osobní sdělení, náhodně nalezené webové stránky, poznámky k přednáškám apod. Citování spolehlivých elektronických zdrojů (maji ISSN nebo DOI) a webových stránek oficiálních institucí je zcela v pořádku. Citujeme-li elektronické zdroje, je třeba uvést URL, na němž se zdroj nachází, a datum přístupu ke zdroji.

2.1 Několik ukázek

Aktuální verzi této šablony najdete v gitovém repozitáři [2]. Také se může hodit prohlédnout si další návody udržované Martinem Marešem [3].

Mezi nejvíce citované statistické články patří práce Kaplana a Meiera a Coxe [4, 5]. Student [6] napsal článek o t-testu.

Prof. Anděl je autorem učebnice matematické statistiky [7]. Teorii odhadu se věnuje práce Lehmann a Casella [8]. V případě odkazů na specifickou informaci (definice, důkaz, ...) uvedenou v knize bývá užitečné uvést specificky číslo kapitoly, číslo věty atp. obsahující požadovanou informaci, např. viz Anděl [1, Věta 4.22].

Mnoho článků je výsledkem spolupráce celé řady osob. Při odkazování v textu na článek se třemi autory obvykle při prvním výskytu uvedeme plný seznam: Dempster, Laird a Rubin [9] představili koncept EM algoritmu. Respektive: Koncept EM algoritmu byl představen v práci Dempstera, Lairdové a Rubina [9]. Při každém dalším výskytu již používáme zkrácenou verzi: Dempster et al. [9] nabízejí též několik příkladů použití EM algoritmu. Respektive: Několik příkladů použití EM algoritmu lze nalézt též v práci Dempstera a kol. [9].

U článku s více než třemi autory odkazujeme vždy zkrácenou formou: První výsledky projektu ACCEPT jsou uvedeny v práci Genbergové a kol. [10]. V textu *nenapíšeme:* První výsledky projektu ACCEPT jsou uvedeny v práci Genberg, Kulich, Kawichai, Modiba, Chingono, Kilonzo, Richter, Pettifor, Sweat a Celentano [10].

3 Tabulky, obrázky, programy

Používání tabulek a grafů v odborném textu má některá společná pravidla a některá specifická. Tabulky a grafy neuvádíme přímo do textu, ale umístíme je buď na samostatné stránky nebo na vyhrazené místo v horní nebo dolní části běžných stránek. LaTeX se o umístění plovoucích grafů a tabulek postará automaticky.

Každý graf a tabulku očíslujeme a umístíme pod ně legendu. Legenda má popisovat obsah grafu či tabulky tak podrobně, aby jim čtenář rozuměl bez důkladného studování textu práce.

Na každou tabulku a graf musí být v textu odkaz pomocí jejich čísla. Na příslušném místě textu pak shrneme ty nejdůležitější závěry, které lze z tabulky či grafu učinit. Text by měl být čitelný a srozumitelný i bez prohlížení tabulek a grafů a tabulky a grafy by měly být srozumitelné i bez podrobné četby textu.

Na tabulky a grafy odkazujeme pokud možno nepřímo v průběhu běžného toku textu; místo *„Tabulka 3.1 ukazuje, že muži jsou v průměru o* 9,9 kg *těžší než ženy“* raději napíšeme *„Muži jsou o* 9,9 kg *těžší než ženy (viz Tabulka 3.1)“*.

3.1 Tabulky

U **tabulek** se doporučuje dodržovat následující pravidla:

- Vyhýbat se svislým linkám. Silnějšími vodorovnými linkami oddělit tabulku od okolního textu včetně legendy, slabšími vodorovnými linkami oddělovat záhlaví sloupců od těla tabulky a jednotlivé části tabulky mezi sebou. V LaTeXu tuto podobu tabulek implementuje balík `booktabs`. Chceme-li výrazněji oddělit některé sloupce od jiných, vložíme mezi ně větší mezeru.

- Neměnit typ, formát a význam obsahu políček v tomtéž sloupci (není dobré do téhož sloupce zapisovat tu průměr, onde procenta).

- Neopakovat tentýž obsah políček mnohokrát za sebou. Máme-li sloupec *Rozptyl*, který v prvních deseti řádcích obsahuje hodnotu 0,5 a v druhých deseti řádcích hodnotu 1,5, pak tento sloupec raději zrušíme a vyřešíme to jinak. Například můžeme tabulku rozdělit na dvě nebo do ní vložit popisné řádky, které informují o nějaké proměnné hodnotě opakující se v následujícím oddíle tabulky (např. *„Rozptyl* = 0,5 “ a níže *„Rozptyl* = 1,5 “).

Efekt	**Odhad**	**Směrod. chyba**[a]	**P-hodnota**
Abs. člen	−10,01	1,01	—
Pohlaví (muž)	9,89	5,98	0,098
Výška (cm)	0,78	0,12	$< 0{,}001$

Pozn: [a] Směrodatná chyba odhadu metodou Monte Carlo.

Tabulka 3.1 Maximálně věrohodné odhady v modelu M.

- Čísla v tabulce zarovnávat na desetinnou čárku.
- V tabulce je někdy potřebné používat zkratky, které se jinde nevyskytují. Tyto zkratky můžeme vysvětlit v legendě nebo v poznámkách pod tabulkou. Poznámky pod tabulkou můžeme využít i k podrobnějšímu vysvětlení významu některých sloupců nebo hodnot.

3.2 Obrázky

Dodejme ještě několik rad týkajících se obrázků a grafů.

- Graf by měl být vytvořen ve velikosti, v níž bude použit v práci. Zmenšení příliš velkého grafu vede ke špatné čitelnosti popisků.
- Osy grafu musí být řádně popsány ve stejném jazyce, v jakém je psána práce (absenci diakritiky lze tolerovat). Kreslíme-li graf hmotnosti proti výšce, nenecháme na nich popisky `ht` a `wt`, ale osy popíšeme *Výška [cm]* a *Hmotnost [kg]*. Kreslíme-li graf funkce $h(x)$, popíšeme osy x a $h(x)$. Každá osa musí mít jasně určenou škálu.
- Chceme-li na dvourozměrném grafu vyznačit velké množství bodů, dáme pozor, aby se neslily do jednolité černé tmy. Je-li bodů mnoho, zmenšíme velikost symbolu, kterým je vykreslujeme, anebo vybereme jen malou část bodů, kterou do grafu zaneseme. Grafy, které obsahují tisíce bodů, dělají problémy hlavně v elektronických dokumentech, protože výrazně zvětšují velikost souborů.
- Budeme-li práci tisknout černobíle, vyhneme se používání barev. Čáry rozlišujeme typem (plná, tečkovaná, čerchovaná,...), plochy dostatečně rozdílnými intensitami šedé nebo šrafováním. Význam jednotlivých typů čar a ploch vysvětlíme buď v textové legendě ke grafu anebo v grafické legendě, která je přímo součástí obrázku.
- Vyhýbejte se bitmapovým obrázkům o nízkém rozlišení a zejména JPEGům (zuby a kompresní artefakty nevypadají na papíře pěkně). Lepší je vytvářet obrázky vektorově a vložit do textu jako PDF.

3.3 Programy

Algoritmy, výpisy programů a popis interakce s programy je vhodné odlišit od ostatního textu. Pro programy se hodí prostředí `lstlisting` z LaTeXového balíčku `listings`, které umí i syntakticky zvýrazňovat běžné programovací jazyky. Většinou ho chceme obalit prostředím `listing`, čímž z něj uděláme plovoucí objekt s popiskem (viz Program 1).

Pro algoritmy zapsané v pseudokódu můžeme použít prostředí `algorithmic` z balíčku `algpseudocode`. Plovoucí objekt z něj uděláme obalením prostředím `algorithm`. Příklad najdete v Algoritmu 1.

Program 1 Můj první program

```
#include <stdio.h>

int main(void)
{
  printf("Hello, world!\n");
  return 0;
}
```

Algoritmus 1 Primitivní pravděpodobnostní test prvočíselnosti. Pokud odpoví NE, číslo x určitě není prvočíslem. Pokud odpoví ANO, nejspíš se mýlí.

```
1: function IsPrime(x)
2:     r ← rovnoměrně náhodné celé číslo mezi 2 a x − 1
3:     z ← x mod r
4:     if z > 0 then
5:         Vrátíme NE                          ▷ Našli jsme dělitele
6:     else
7:         Vrátíme ANO                         ▷ Možná se mýlíme
8:     end if
9: end function
```

Další možností je použití LaTeXového balíčku `fancyvrb` (fancy verbatim), pomocí něhož je v souboru `makra.tex` nadefinováno prostředí `code`. Pomocí něho lze vytvořit např. následující ukázky.

V základním nastavení dostaneme:

```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Můžeme si říci o menší písmo:

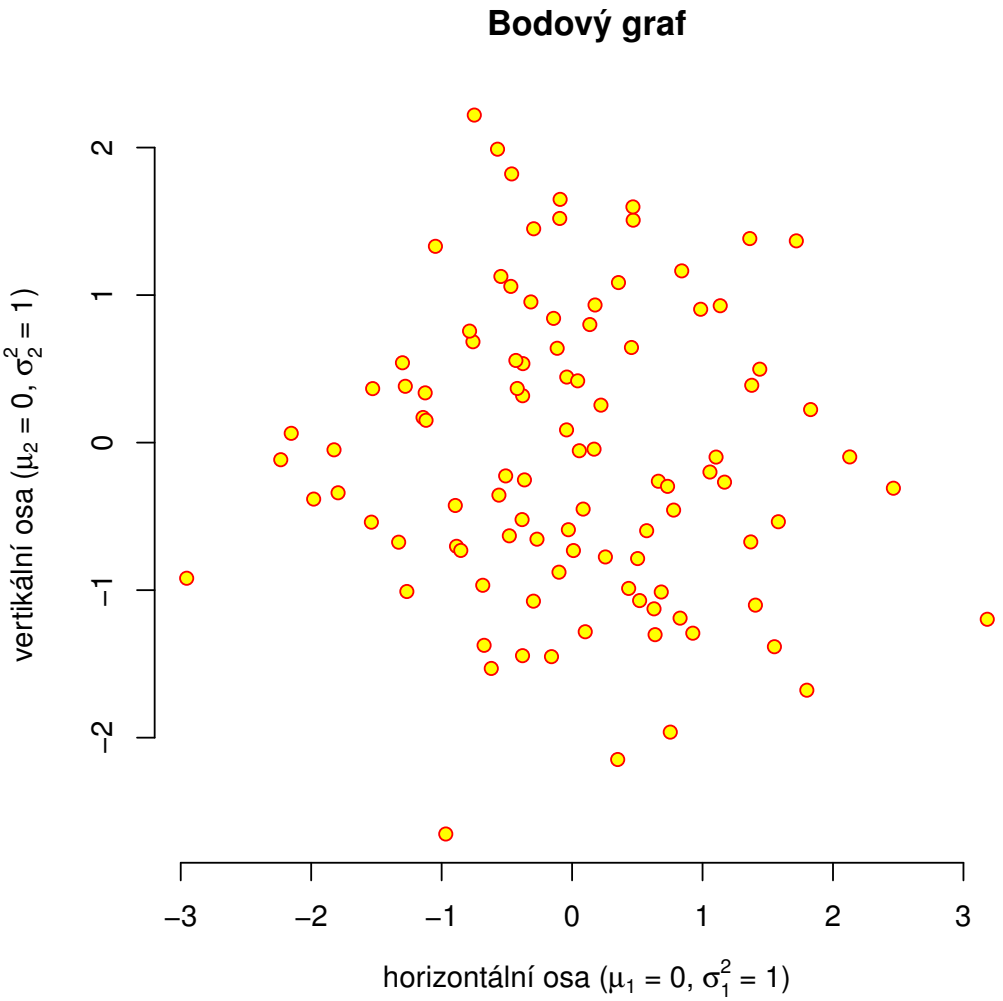
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Nebo vypnout rámeček:

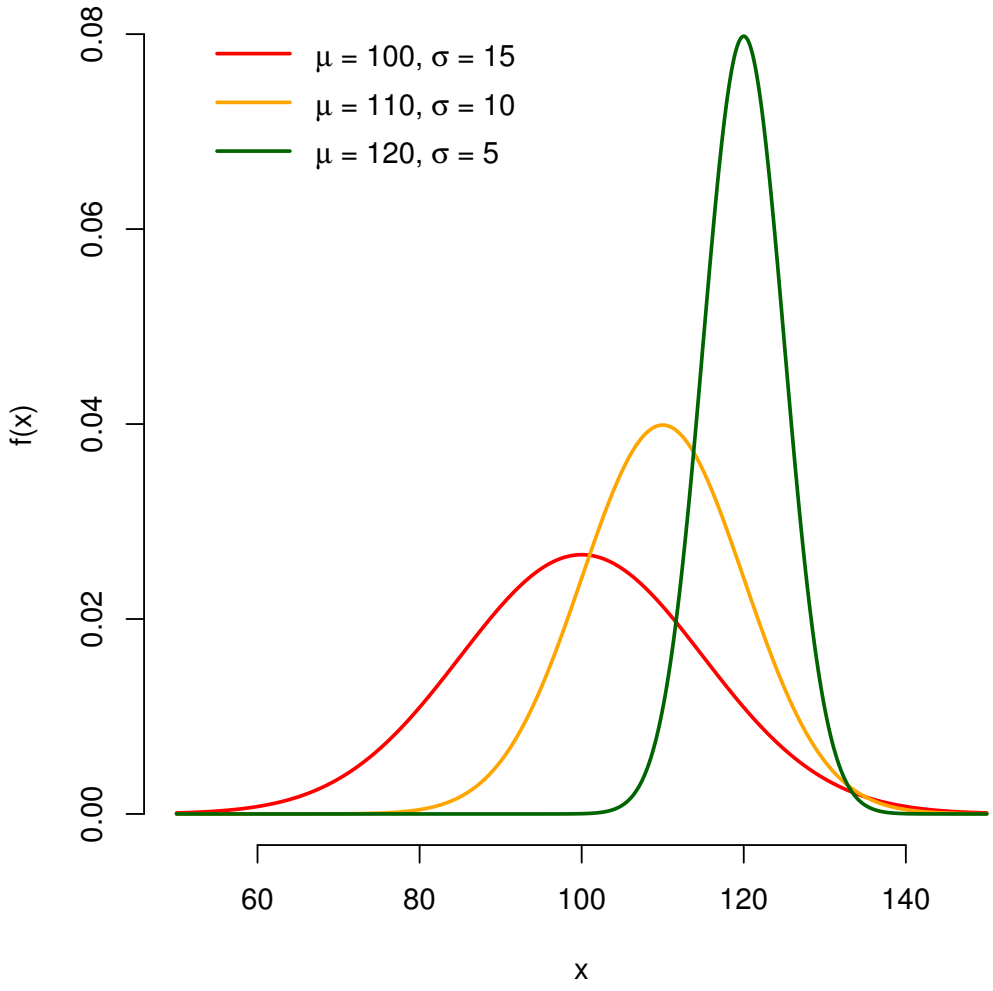
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```

Případně si říci o užší rámeček:

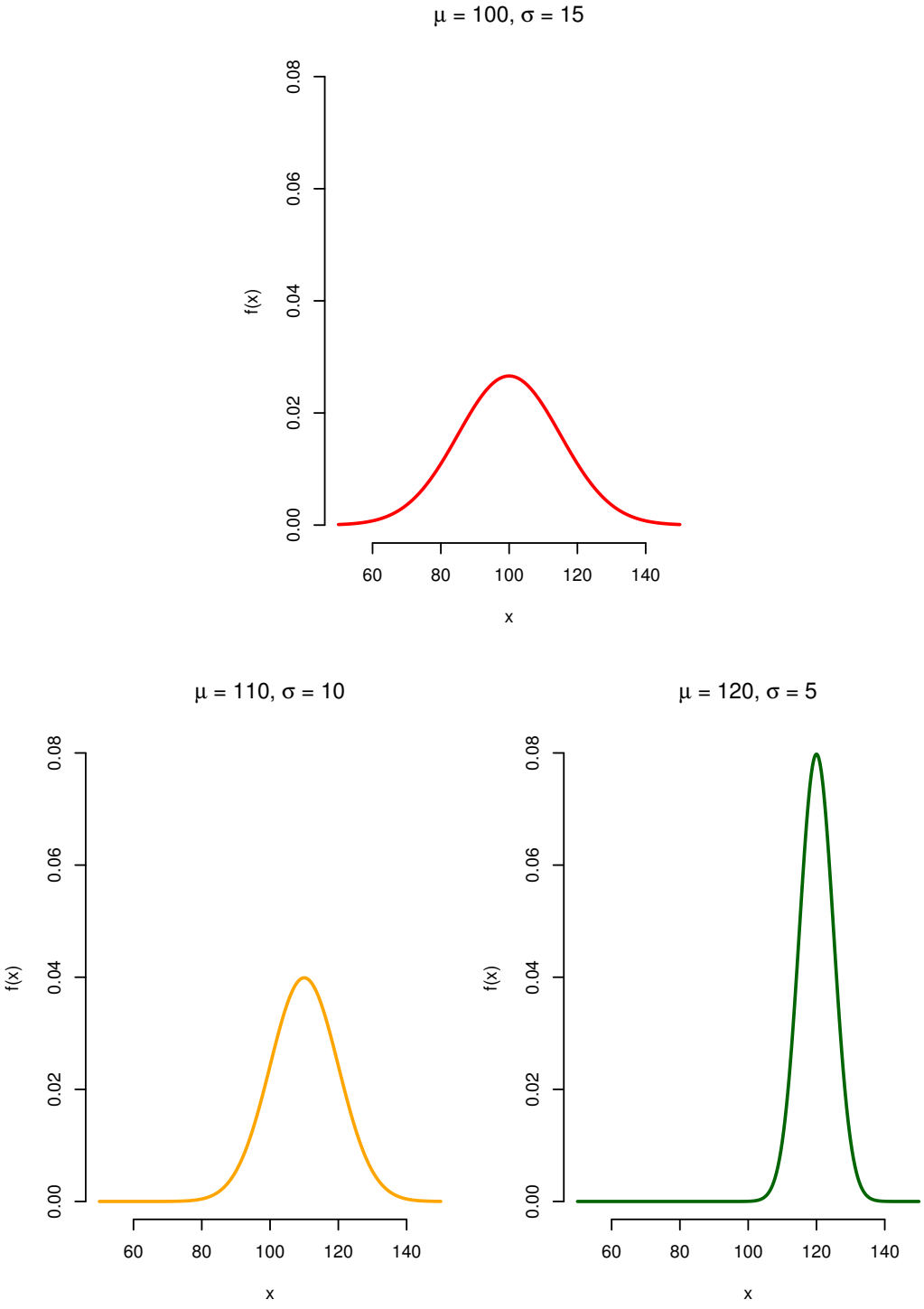
```
> mean(x)
[1] 158.90
> objekt$prumer
[1] 158.90
```



Obrázek 3.1 Náhodný výběr z rozdělení $\mathcal{N}_2(\mathbf{0}, I)$.



Obrázek 3.2 Hustoty několika normálních rozdělení.



Obrázek 3.3 Hustoty několika normálních rozdělení.

4 Formát PDF/A

Opatření rektora č. 13/2017 určuje, že elektronická podoba závěrečných prací musí být odevzdávána ve formátu PDF/A úrovně 1a nebo 2u. To jsou profily formátu PDF určující, jaké vlastnosti PDF je povoleno používat, aby byly dokumenty vhodné k dlouhodobé archivaci a dalšímu automatickému zpracování. Dále se budeme zabývat úrovní 2u, kterou sázíme TEXem.

Mezi nejdůležitější požadavky PDF/A-2u patří:

- Všechny fonty musí být zabudovány uvnitř dokumentu. Nejsou přípustné odkazy na externí fonty (ani na „systémové“, jako je Helvetica nebo Times).
- Fonty musí obsahovat tabulku ToUnicode, která definuje převod z kódování znaků použitého uvnitř fontu to Unicode. Díky tomu je možné z dokumentu spolehlivě extrahovat text.
- Dokument musí obsahovat metadata ve formátu XMP a je-li barevný, pak také formální specifikaci barevného prostoru.

Tato šablona používá balíček `pdfx`, který umí LaTeX nastavit tak, aby požadavky PDF/A splňoval. Metadata v XMP se generují automaticky podle informací v souboru `prace.xmpdata` (na vygenerovaný soubor se můžete podívat v `pdfa.xmpi`).

Validitu PDF/A můžete zkontrolovat pomocí nástroje VeraPDF, který je k dispozici na `http://verapdf.org/`.

Pokud soubor nebude validní, mezi obvyklé příčiny patří používání méně obvyklých fontů (které se vkládají pouze v bitmapové podobě a/nebo bez unicodových tabulek) a vkládání obrázků v PDF, které samy o sobě standard PDF/A nesplňují.

Další postřehy o práci s PDF/A najdete na `http://mj.ucw.cz/vyuka/bc/pdfaq.html`.

5 Obecné informácie o aplikácii

5.1 Motivácia

Veľmi často v živote narážame na situácie, kedy sa potrebujeme dostať z bodu A do bodu B čo najrýchlejším spôsobom. V mnohých prípadoch mám k dispozícii sieť cestných komunikácii či chodníkov na základe ktorých sa môžeme rozhodovať, ktorá trasa je pre nás vyhovujúca. Pre takéto prípady nám veľmi dobre poslúžia už existujúce navigačné aplikácie ktoré majú podrobne zmapovanú cestnú sieť a vedia nám na základe cestných parametrov určiť najrýchlejšiu alebo najúspornejšiu trasu.

Avšak môžu nastať v živote aj situácie, kedy cestná sieť je priveľmi riedka, až takmer neprítomná. Väčšinou takéto situácie nastávajú vo voľnej prírode, v odľahlých častiach od civilizácie. V takých prípadoch nám konvenčné navigácie veľmi dobre neposlúžia. Ak máme kvalitnú, detailnú mapu, môžeme sa pokúsiť v nej nájsť vyhovujúcu trasu vlastnými silami ale často existuje príliš mnoho možností ktorými sa môžeme vydať. Preto by sa niekedy hodilo mať software, ktorý na základe zadaných parametrov nájde na predloženej mape najrýchlejšiu trasu po ktorej sa človek môže vydať k vytýčenému cieľu.

Príklady využitia takéhoto software-u by sme mohli nájsť v špecifických profesiách ako sú napríklad lesníctvo, záchranné služby či ozbrojené sily. Vo všetkých je potrebné sa z času na čas dostať na miesto ‚uprostred ničoho' aby mohli byť vykonané ich zámery.

Takýto software by však našiel uplatnenie taktiež v rekreačných aktivitách. Či už turizmus alebo športové aktivity sa často odohrávajú vo voľnej prírode. Z rôznych dôvodov sa potom môže zísť schopnosť nájdenia najrýchlejšej trasy späť do civilizácie, či už po alebo mimo cesty.

Rád by som vyzdvihol špecificky jedno športové odvetvie, ktoré veľmi úzko súvisí s hľadaním ciest v otvorenom teréne a to *orientačný beh*. „Orientačný beh (skratka OB) je športové odvetvie vytrvalostného charakteru, pri ktorom je úlohou prejsť alebo prebehnúť podľa mapy a buzoly trať vyznačenú na mape za čo najkratší čas. V teréne nie je trať vyznačená, sú tam umiestnené iba kontrolné stanovišťa (Kontroly)"[11]. Tento šport bol hlavnou motiváciou pre vytvorenie aplikácie, v ktorej by si užívateľ mohol nechať vykresliť na mape najrýchlejší postup pre ním zadanú trať.

5.2 Aspekty hľadania najrýchlejšej trasy (v OB)

Mapy pre orientačný beh bývajú veľmi detailné a bežec má preto dostatok informácie na to aby si mohol vybrať ideálnu trasu. Za predpokladu že bežec nerobí chyby a beží presne podľa svojho zámeru, sú pre výber najrýchlejšieho postupu dôležité dva druhy objektov: líniové (cesty, potoky, prieseky, ...) a polygonálne (lúky, kroviská, vodné objekty, močiare, ...). Tie určujú typ terénu nachádzajúci sa v danej časti mapy a tým pádom aj veľkosť odporu ktorý je kladený rýchlosti behu pretekára. Táto veličina sa dá použiť ako hlavný parameter ktorý bude určovať preferencie výberu postupu.

Proces hľadania cesty sa skladá z dvoch hlavných častí:

- **vytvorenie mapovej reprezentácie** - Na začiatku je potrené na základe mapového súboru vygenerovať mapovú reprezentáciu na ktorej bude možné vyhľadávať. Mapovou reprezentáciou by mal byť konštrukt, ktorý dobre vystihne topografiu mapy a umožní hľadanie najideálnejšej trasy. V našej aplikácii budú týmito konštruktami *ohodnotené grafy*.

- **aplikácia vyhľadávacieho algoritmu** - mapová reprezentácia je predaná algoritmu a on za použitia jej interface-u v nej vyhľadá najkratšiu cestu. V našom prípade najkratšia znamená najrýchlejšia.

V následujúcich podsekciách spomeniem koncepty, ktoré budú v procese hľadania najrýchlejších ciest vystupovať. Medzi jednotlivými konceptami sú tvorené rozne závislosti. Grafické zhrnutie týchto závislostí je možné nahliadnuť na konci sekcie v Obrázku 5.1.

5.2.1 Mapy

Na začiatok je potrebné spomenúť koncept Mapy. V procese hľadania ciest bude na viacerých miestach potrebné agregovať dáta z užívateľom vybraného mapového súboru. Aby sme nemuseli neustále čítať priamo zo súboru, budeme si udržovať jeho obsah v pamäti v prívetivejšej forme.

Touto formou bude práve koncept *Mapy*. Mapa si bude udržovať všetky objekty definované v mapovom súbore a keď bude potrebné agregovať z daného súboru nejakú informáciu (mapovú reprezentáciu, mapovú grafiku, ...) použije sa namiesto súboru odpovedajúci mapový objekt.

Je zahodno zmieniť že koncept mapy je odlišný od konceptu *mapovej reprezentácie*. Mapa narozdiel od jej reprezentácie neobsahuje žiadne zložité prepojenia medzi objektami ktoré v sebe drží. Jej vytvorenie by malo byť rýchle, s lineárnou časovou zložitosťou v závislosti na velkosti mapového súboru.

5.2.2 Mapové reprezentácie

Mapové reprezentácie sú jednou z dôležitých zložiek procesu hľadania ciest. Sú to jednotky, na ktorých sa samotné vyhľadávanie uskutočňuje. Mapové objekty su oproti *mapám* zložitejšie konštrukty ktoré už v sebe zahrňujú plno závislostí a prepojení. Ich generovanie môže zabrať oveľa viac času ako generovanie mapových objektov.

Ako už bolo spomenuté v aplikácii budú mapovými reprezentáciami *ohodnotené grafy* (naďalej iba grafy). Napriek tomu, že mapová reprezentácia a graf budú v aplikácii reprezentovať rovnaký objekt, ich významy sú odlišné.

- **Mapová reprezentácia** hovorí o tom, ako daný objekt funguje vnútorne. Popisuje jeho vlastnosti, mechanizmy a spôsoby akými generuje výsledný *graf*, v ktorom sa následne hľadá cesta. Objekt ktorý je abstrahovaný z mapy je v prvom rade mapová reprezentácia a až v druhom graf.

  Príkladom myšlienky ktorú môže mapová reprezentácia vyjadrovať je „samozahusťujúci“ sa graf. Takáto mapová reprezentácia počas behu algoritmu

zahusťuje predom pripravený graf o ďalšie vrcholy na miestach, v ktorých sa prehľadávanie aktuálne uskutočňuje. Redukuje sa tým veľkosť vygenerovaného grafu.

- **Graf** na druhej strane hovorí o vlastnostiach objektu ktoré sú viditeľné navonok. Bude informovať o „grafových“ službách objektu ktoré dokáže poskytnúť. Tieto služby môžu byť obmedzené práve vnútornou štruktúrou ktorá je definovaná *mapovou reprezentáciou*. Graf sa berie ako druhotný produkt mapového spracovania.

  Príkladom vlastnosti, ktorú môže graf prezentovať vonkajšiemu svetu je možnosť poskytnutia kompletne vygenerovaného grafu pri ktorom sa vonkajší užívateľ nemusí obávať, že by sa počas práce s ním graf nejakým spôsobom modifikoval. Túto vlastnosť napríklad nevie zaručiť graf ktorý koresponduje s vyššie zmienenou mapovou reprezentáciou ktorá stav grafu počas práce neustále modifikuje, zahusťuje ho.

Pojmy *mapová reprezentácia* a *graf* budú naďalej v práci brané ako zameniteľné a ich použitie bude závisieť od okolitého kontextu a ich špecifického významu.

5.2.3 Užívateľské modely

Je potrebné si uvedomiť, že rôzni bežci majú rôzne schopnosti a preto aj ich preferencie na výber trasy nemusia byť rovnaké. Niekto sa dokáže rýchlejšie predierať cez husté pasáže, inému ide rýchlejšie beh cez močiar a ďalšiemu vyhovujú dlhšie postupy po cestách.

Preto by bolo vhodné, aby užívateľ aplikácie mal možnosť aplikovať svoje preferencie do procesu vyhľadávania. K tomuto účelu boli vytvorené tzv. „užívateľské modely“. Užívateľ si pomocou nich môže vytvoriť vlastný „profil“ na základe ktorého bude vyhľadávanie prispôsobené.

Užívateľské modely vďaka svojej informovanosti o užívateľských preferenciách nadobúdajú zodpovednosť za výpočty hodnôt ktoré sú závislé práve na daných preferenciách. Stávajú sa teda dôležitou zložkou procesu hľadania ciest, kde sa vyhľadávacie algoritmy stávajú závislé na nimi vykonávaných výpočtoch.

5.2.4 Výškové dáta

Jedným z dôležitých a neopomenuteľných faktorov voľby najrýchlejšieho postupu je prevýšenie, ktoré je potrebné pri jeho prevedení zdolať. Mnoho typov máp obsahuje popisuje reliéf krajiny pomocou takzvaných *vrstevníc*. Vrstevnica je krivka na mape, ktorá spája body rovnakej nadmorskej výšky. Pre človeka je vrstevnicová abstrakcia výšky terénu veľmi ľahko pochopiteľná a spracovateľná.

Pre strojové spracovanie mapy však vrstevnice predstavujú veľmi neprirodzený spôsob pre reprezentáciu nadmorskej výšky. Vypracovanie reliéfneho obrazu za pomoci vrstevníc samotných je veľmi ťažká úloha. V niektorých prípadoch (mapy pre OB napríklad) dokonca jednotlivé vrstevnice ani nie sú v mapovom súbore reprezentované jedným líniovým objektom. Kvôli dobrej čitateľnosti máp sú často vrstevnice prerušované.

Z vyššie uvedených dôvodov je v aplikácii zahrnutý systém ktorý sprostredkováva užívateľom možnosť stiahnutia a spravovania digitálnych výškových dát

ktoré následne môžu byť použité ako pomocný zdroj v procese tvorby mapových reprezentácií. Pre konštrukciu mapových reprezentácií pre niektoré mapové formáty bude nutná prítomnosť zodpovedajúcich výškových dát a pri ich absencii jednoducho nebude možné mapové reprezentácie vytvoriť.

5.2.5 Atribútové template-y

Ďalšia vec, nad ktorou je potrebné sa zamyslieť je, akým spôsobom sa bude v grafoch mapových reprezentácií uchovávať informácia o mapových vlastnostiach a atribútoch, ktoré konkrétne vrcholy a hrany grafu reprezentujú. Zároveň je potrebné aby dotyčné vlastnosti dokázal príslušný užívateľský model spracovať a dopočítať z nich hodnoty potrebné pre beh vyhľadávacích algoritmov. Používané atribúty, ktoré agregujeme z máp do mapových reprezentácií preto musia byť jednotné v celom procese hľadania cesty, od vytvárania mapovej reprezentácie po samotné spustenie vyhľadávacieho algoritmu.

V aplikácii nám definíciu a jednotnosť atribútov budú zabezpečovať tzv. *template*-y. Každý template bude definovať jedinečnú sadu vrcholových a hranových atribútov. Príkladom pre takúto kolekciu pre potreby OB môže byť napríklad:

- vrcholové atribúty - pozícia, výška, indikátory reprezentovaných terénnych objektov (či sa daný vrchol nachádza na ceste, v húštine, na lúke, v dobre priebežnom lese,...)
- hranové atribúty - či daná hrana reprezentuje úsek nejakej cesty, hranu nejakého objektu(lúky, húštiny, močiaru,...), sklon terénu

Na template-och ako takých bude teda závisieť:

- **výber užívateľských modelov** - model musí vedieť spracovať atribúty definované daným template-om a vrátiť od neho požadované výsledky.
- **formát vybraného mapového súboru** - musí existovať konvertor mapy špecifického formátu na odpovedajúcu mapovú reprezentáciu, v ktorej vrcholoch a hranách sú obsiahnuté atribúty definované daným template-om. Túto závislosť môžeme brať aj opačným smerom. Pre vybraný mapový formát môžeme zvoliť len použiteľný template.

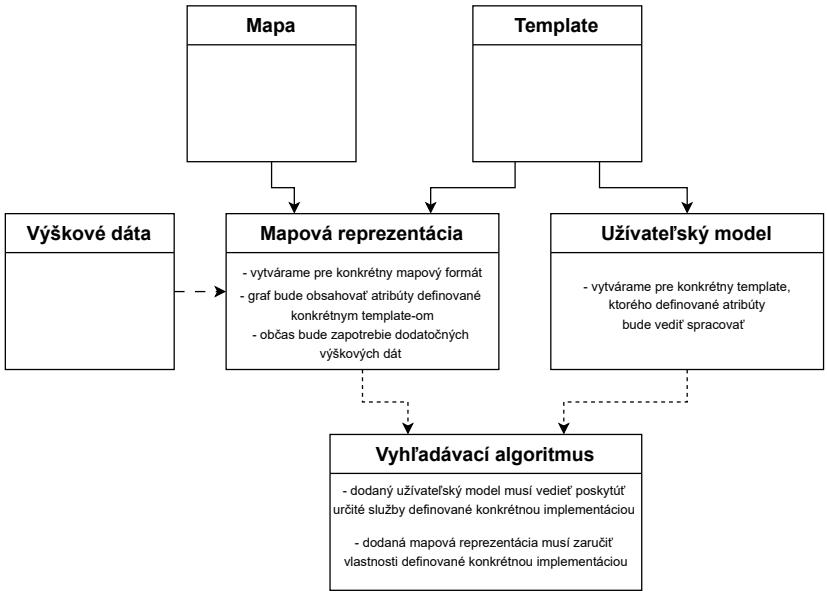
5.2.6 Vyhľadávacie algoritmy

Nakoniec nemôžeme opomenúť koncept samotných vyhľadávacích algoritmov, poslednú neodmysliteľnú súčasť procesu hľadania ciest. Vyhľadávací algoritmus, podobne ako *mapová reprezentácia*, reprezentuje koncept vnútorného mechanizmu ktorým je najkratšia cesta v grafe hľadaná. Príkladom takéhoto algoritmu môže byť napríklad algoritmus *A**.

Vstupom do každého algoritmu sú:

- **graf**, na ktorom je hľadanie najkratšej cesty vykonané a
- **užívateľský model**, ktorý algoritmus nutne potrebuje ku výpočtom váh grafových hrán a iných hodnôt potrebných k jeho správnemu chodu.

Každý algoritmus následne ponúka množinu jeho implementácií. Každá implementácia definuje množinu vlastností, ktoré vložený graf a užívateľský model musia spĺňať. Napríklad implementácie A* algoritmu budú požadovať, aby užívateľský model bol schopný, popri výpočte váh pre hrany grafu, dodať ešte aj výpočet heuristiky využívanej v tomto algoritme.



Obrázek 5.1 Diagram závislostí jednotlivých konceptov

5.3 Zvolené prostriedky

5.3.1 Použitý programovací jazyk

Pre implementáciu aplikácie bol vybraný jazyk C#. Jazyk bol vybraný pre jeho jednoduchosť a bezpečnosť použitia. Alternatívnou volbou by mohol byť jazyk Java, ktorý má blízko práve ku C#. Jeho nedostatočná znalosť v dobe začiatku práce ho však vyradila z použiteľných možností.

Ďalšími možnosťami by mohol byť buď typicky vysoko-úrovňový jazyk Python alebo na druhú stranu nízko úrovňový jazyk C++. Nakoľko v aplikácii bude prebiehať mnoho výpočtov, python by nebol vhodnou voľbou pre nedostatočnú rýchlosť z neho vytvoreného strojového kódu. Na druhú stranu C++ by bol veľmi dobrým kandidátom z hladiska výpočtovej sily. V tomto prípade však narážame opäť na nie veľmi dobrú znalosť tohto jazyka a na nie úplne pohodlnú prácu s ním. Pri takomto väčšom projekte sme považovali za potrebné istotu, že nás použitý jazyk podrží (ak s ním nie sme zžitý na 100%).

Vhodnou úpravou by možno bolo implementovať výpočtovo náročné procesy v jazyku typu C++ a následne túto implementáciu volať externe z C# jadra.

5.3.2 Užívateľské rozhranie

Pre implementáciu užívateľského rozhrania som sa rozhodol pre *Avalonia UI* framework.

V C# existuje viacero možných framework-ov z ktorých bolo možné si vybrať:

- GUI knižnice, ktoré sú súčasťou samotného .NET framework-u:
  - **Windows Forms** je klasická GUI knižnica. Je jednoducho použiteľná a vďaka jej dlhoročnej podpore aj robustná a spoľahlivá. Jej vek je však aj jej nevýhodou, nakoľko vzhľad aplikácií vytvorených za pomoci Windows Forms príde človeku pomerne zastaralý. Ďalšou nevýhodou je rastrová povaha jeho renderovcieho engine-u. Táto vlastnosť sa nehodí pre aplikáciu, ktorej jedným z hlavných účelov je vykreslovanie mapových, vektorových objektov.
  - **Windows Presentation Foundation (WPF)** je modernejší nástupca Windows Forms. Aplikácie tvorené touto knižnicou majú modernejší vzhľad, sú generované renderovacím enginom založeným na vektorovej grafike. Ku programovaniu sa tu využíva popri C# aj XAML v ktorom sa definuje layout užívateľského rozhrania. Avšak spoločne s Windows Forms je ich spoločnou nevýhodou platformová závislosť na operačnom systéme Windows.
  - **.NET MAUI** je nástupcom *Xamarin.Forms*. Je to open-source-ový cross-platformný framework s množinou UI nástrojových balíčkov pre jednotlivé platformy. Podobne ako vo WPF sa pre vytváranie UI využíva kombinácia C# a XAML.
- Alternatívou ku vstavaným .NET GUI knižniciam je práve nezávislá, open-source knižnica **Avalonia UI**. Čo do vlastností je veľmi dobre porovnateľná s *.NET MAUI*. Hlavný rozdiel medzi týmito dvomi knižnicami je v spôsobe,

akým vykresľujú užívateľské rozhranie. Avalonia zapojuje kresliaci engine poháňaný knižnicou pre 2D grafiku *Skia*. Na druhú stranu MAUI využíva natívne nástrojové balíčky pre každú platformu. Ďalším rozdielom je, že Avalonia, na rozdiel od MAUI, podporuje aj niektoré distribúcie Linuxových systémov.

Rozhodnutie nakoniec padlo na využitie framework-u Avalonia UI. Nakoľko je veľmi podobný natívnemu .NET MAUI, rozhodla podpora pre Linuxové systémy a aj kvalitne spravená dokumentácia, z ktorej sa ľahko dalo pochopiť, ako sa s knižnicou má pracovať.

Informované rozhodnutie pre výber GUI knižnice bolo učinené na základe zdrojov [12, 13, 14]. Zároveň väčšinu informácií, ktoré som o Avalonia UI počas tvorby programu čerpal pochádzali z jej dokumentácie [15].

5.3.3 Architektúra Model-View-ViewModel (MVVM)

Jeden z ďalších dôležitých aspektov ktorý hral dôležitú úlohu vo výbere knižnice pre užívateľské rozhranie bola podpora MVVM návrhového vzoru. Dokumentácia Avalonia UI popisuje architektúru MVVM následovne: „The Model-View-View Model (MVVM) pattern is a common way of structuring a UI application. It uses a data binding system that helps move data between its view and view model parts. This means it achieves separation of application logic (view model) from the display of the UI (view). Separation between the application logic and the business services (model) is commonly achieved by a Dependency Injection (DI) system.“[16].

MVVM architektúra je vhodná pre našu aplikáciu, nakoľko pre jej rozsah by klasická *event-driven code-behind* architektúra nemusela postačovať. Týmto spôsobom zaručíme lepšiu separáciu a dostačujúcu vnútornú nezávislosť kódu.

V našej aplikácii bude tento návrhový vzor uplatnený s jemnou obmenou. Vrstva view model, ktorá v pôvodnom návrhovom vzore zastávala úlohu logiky aplikácie bude rozdelená na dve časti: view model a novo vzniknutý *model view*. Viac informácií o tejto úprave je možné nájsť v sekcii 6.1.

5.3.4 Reaktívne programovanie

Ďalším dôležitým aspektom v aplikácii je Avaloniou a architektúrou MVVM iniciované využitie *reaktívneho programovania.* Avalonia pre aplikáciu tohto paradigma využíva framework **Reactive UI**.

Tá samotná definuje reaktívne programovanie ako „Reactive programming is programming with asynchronous data streams.“ a dodáva „Event buses or your typical click events are really an asynchronous event stream, on which you can observe and do some side effects. Reactive programming is that idea on steroids. You are able to create data streams of anything, not just from click and hover events. Streams are cheap and ubiquitous and anything can be a stream: variables, user inputs, properties, caches, data structures, etc.“[17].

V aplikácii sa bude tento framework využívať prevažne vo vrstvách view a view model. Vrstvy model view a model sa väčšinou zaobídu bez jeho využitia, nakoľko reaktívna komunikácia prebieha medzi vrstvami view a view model.

6 Architektúra ako celok

O architektúre aplikácie sa dá premýšlať ako o *mriežke*. Je rozdelená na horizontálne (MVVM návrhový vzor) a vertikálne (*session*y) vrstvy. V nasledujúcich sekciách popíšeme, ako jednotlivé vrstvy vyzerajú, aké su ich úlohy a ako medzi sebou komunikujú.

6.1 MVVM(MV) návrhový vzor

Ako už bolo spomenuté v podsekcii 5.3.3, v aplikácii je využívaný návrhový vzor MVVM s drobnou obmenou. Táto obmena sa týka rozdelenia originálnej vrstvy view model na dve časti: view model a model view. Toto rozdelenie zaručí ešte o niečo lepšiu separáciu kódu a odľahčí tým úlohy view model vrstvy.

MVVM(MV) architektúra teda rozdeluje aplikáciu na 4 vrstvy: View, ViewModel, ModelView a Model. Popis jednotlivých vrstiev je k nahliadnutiu v nasledujúcich podsekciách. Na konci sekcie je v na obrázku 6.1

6.1.1 View

View je vrstva, ktorá popisuje a implementuje grafickú stránku aplikácie a určuje akým spôsobom sa dáta dodané vrstvou View model zobrazia užívateľovi. Viewy sú viazané na zodpovedajúce view modely za pomoci reaktívneho programovania. Spracovávajú akcie užívateľa a iniciujú reakcie zvyšných vrstiev architektúry prostredníctvom naviazaného view modelu. Následne zabezpečujú grafické znázornenie ním dodaných výsledkov reakcie.

View vrstva je kompletne implementovaná za pomoci Avalonia UI framework-u. V náväznosti na tento fakt sú v aplikácii použité tri hlavné typy view-ov:

- **Window** - reprezentuje špecifické okno aplikácie, top-level kontajner, ktorý drží v sebe nejaký obsah. Samo o sebe veľmi nedefinuje vzhľad aplikácie. Slúži predovšetkým ako rám, v ktorom sa striedajú jednotlivé View-y. Každé okno má naviazaný svoj vlastný view model, ktorý drží informáciu o tom, aký View je v danej chvíli obsahom okna. Naviazaný view model taktiež obsahuje vlastnosti ktoré priamo súvisia s vlastnosťami daného okna.

- **View** - sú to hlavné zložky, ktoré nesú grafiku toho, čo sa aktuálne zobrazuje v konkrétnom okne. Reprezentujú jeden obraz ktorý je užívateľovi vykresľovaný v konkrétnom okne. Každý view je naviazaný na špecifický view model. Viaže sa na jeho vlastnosti a vykresluje dáta ktoré tieto vlastnosti obsahujú.

  Býva zvykom že daný View sa zobrazuje práve v jednom konkrétnom okne. V takom prípade si na jeho view model drží referenciu view model okna. Za pomoci tejto referencie potom dokáže okenný view model oznámiť oknu, že sa v ňom má daný view zobraziť.

- **DataTemplate** - definuje grafickú reprezentáciu dát dodávaných view modelom naviazanému view-u. Pre každý dátový typ, ktorý chce byť správne vykreslený pre užívateľa, musí existovať špecifický dátový template, pomocou ktorého sa daný údaj vykreslí. Dátové template-y sú špecifické tým, že sú raz

definované pre celú aplikáciu, aby sa zachovala konzistencia vykreslovania jednotlivých dátových typov.

Je potrebné podotknúť, že na to aby údaj vygenerovaný vo vrstve Modelov bolo možné vykresliť, musí preňho najprv existovať tzv. *data view model* do ktorého sú jeho informácie zabalené a až v takejto podobe predávané nejakému view modelu, ktorý ich následne pomocou svojich vlastností odovzdá view-u na vykreslenie. Pre každý dátový view model sa následne hľadá príslušný DataTemplate, pomocu ktorého v ňom držané informácie zobrazia užívateľovi.

View vrstva je implementovaná pomocou dvoch jazykov a to C# a XAML. Pomocu XAML definujeme všetky polohy a tvary grafických objektov a bindujeme vlastnosti týchto objektov na vlastnosti z vrstvy View model. Pre každý view máme obecne jeden špecifický XAML súbor ktorým ho implementujeme. Ku väčšine XAML súborov je priradený C# zdrojový súbor v ktorom sa implementuje takzvaný *code-behind*. V tomto zdrojovom súbore môžeme doplniť všetku funkcionalitu View-u, ktorú nebolo možné vyjadriť jazykom XAML.

6.1.2 View model

Druhou v poradí je vrstva View model. Táto vrstva je zodpovedná za logiku spracovania akcií užívateľa oznámených reaktívnym spôsobom View vrstvou a disponuje vedomím toho, aké akcie, v akom poradí sa majú vykonať pre zabezpečenie patričnej reakcie na daný impulz. K tomuto účelu využíva služby nižších vrstiev prostredníctvom volania na vrstve Model view. Tá na základe svojej vnútornej logiky vráti odpoveď s požadovanými údajmi. View model následne spracuje dodané dáta a predostrie ich view-u aby ich mohla ukázať užívateľovi.

View model zároveň, na základe aplikačnej logiky, koriguje a obmedzuje akcie užívateľa a tým zabraňuje vzniku nekonzistentných stavov aplikácie. Taktiež v niektorých prípadoch iniciuje interakcie s inými view modelmi za účelom dodania ích doplňujúcich služieb a aplikačnej logiky do jeho vlastného procesu.

Podobne ako vo View vrstve sa view modely delia natri základné typy:

- **Session view model** + *Main window view model* - odpovedajú jednotlivým *session*-om, ktoré sú základným kameňom vertikálnej štruktúry aplikácie. Viac informácií o session-och je možné nájsť v sekcii 6.2. Výnimkou je práve *Main window view model*, ktorý je naviazaný na hlavné okno aplikácie a zabezpečuje preňho aplikačnú logiku. Klasické session view modely sú taktiež naviazané zvyčajne na jedno okno z view vrstvy pre ktoré zabezpečujú aplikačnú logiku.

  Každý session view model obsahuje kolekciu príslušných view model-ov ktoré spoločne implementujú mechanizmus daného session-u. Je zvykom, že v jednej chvíli je aktívny iba jeden view model. O aktívnom view modelu informuje session view model naviazané okno, ktoré potom v sebe zobrazuje odpovedajúci view aktívneho view modelu. Informovanie o aktívnom view modelu je aj hlavnou pracovnou náplňou session view modelu.

  K ďalším jeho povinnostiam patrí spracovávanie užívateľom vyvolaných akcií, ktoré sa týkajú samotného naviazaného okna. Príkladom takej akcie môže byť napríklad požiadavka o zatvorenie dotyčného okna.

- **View model** - reprezentujú zložky, ktorých funkcionalita sa najväčšmi ponáša na obecnú, vyššie popísanú funkcionalitu vrstvy View model. Každý view model vo väčšine prípadov spadá pod réžiu konkrétneho session-u (alebo hlavného okna), pre ktorý implementuje určitú časť jeho mechanizmu. Klasicky sú view modely naviazané na príslušné view-y z View vrstvy. S tými následne reaktívne komunikujú a reagujú na ích podnety. View modely sú navzájom nezávislé. To znamená, že medzi nimi neprebieha takmer žiadna komunikácia ani presun dát. Túto funkciu na seba berie Model view vrstva.

  Väčšina view modelov by mala byť zahrnutá v zodpovedajúcom session view modele (poprípade Main window view modele). Ten potom zabezpečuje správu toho, ktorý view model je v danej chvíli aktívny. Výnimkou sú view modely, ktoré sú výhradne používane pre interakcie z iných view modelov, ktorým týmto spôsobom doručujú svoje služby. Tieto view modely sú väčšinou vytvorené na mieste interakcie a po jej dokončení zanikajú.

- **Data view model** - slúžia ako kontajnery pre informácie abstrahované z dát vygenerovaných v Model vrstve. Dáta sú klasicky konvertované do ich zodpovedajúcich view modelov v Model view vrstve a už takto zabalené informácie sú predávané do View model vrstvy kde sú spracované a pomocou vlastností odovzdané vrstve View na vykreslenie.

  Z toho vyplýva, že na to aby nejaký údaj vygenerovaný v Model vrstve mohol byť prezentovaný užívateľovi, musí preňho existovať zodpovedajúci dátový view model. Zároveň na to, aby informácie obsiahnuté v dátovom view modele mohli byť vykreslené pre užívateľa, musí vo View vrstve existovať zodpovedajúci *dátový template*, ktorý sa postará o ích správne grafické znázornenie.

  Niektoré dátové view modely nielen že obsahujú informácie príslušných dát ale obsahujú aj dáta samotné. Takéto dátové view modely označujeme pomocou slova *wrapping*. (tvoria akýsi obal okolo dátových inštancií). Táto funkcionalita je dôležitá hlavne v prípadoch, kedy dátový view model slúži taktiež pre spätnú komunikáciu s Model view vrstvou. V takých prípadoch musí byť možné identifikovať, ktorú dátovú inštanciu „obaluje“. Wrapping dátové view modely sú stotožnené s ich dátovým objektom a tiež sa pomocou neho identifikujú.

View model je prvá z vrstiev, ktorá je kompletne písaná v jazyku C#. Komunikácia medzi View model a View vrstvami funguje čisto na báze reaktívneho programovania za pomoci konštruktov z *Reactive UI* framework-u.

6.1.3 Model view

Treťou v poradí je, do klasickej MVVM architektúry pridaná, Model view vrstva. Táto vrstva je zodpovedná za „vnútornú“ logiku aplikácie. Priamo komunikuje s Model vrstvou a využíva jej zdroje pre zabezpečenie svojich služieb pre View model vrstvu. Dá sa povedať, že nedisponuje vlastným „vedomím“. Medzi jej hlavné úlohy patria:

- prijímanie a spracovávanie požiadaviek od View modelu a dodávanie očakávaných výsledkov.

- zabezpečovať vnútro-session-ovú komunikáciu. Model view-y v rámci jedného session-u si na seba držia referencie a predávajú si medzi sebou držané dáta. Táto komunikácia by mala byť vyšším vrstvam skrytá a na povrch by mal byť vidieť iba interface, pomocou ktorého prebieha komunikácia s View modelom.

- konverzia dát, získaných od Model vrstvy, do odpovedajúcich Data view modelov pri ích posielaní do vyšších vrstiev.

Na druhú stranu medzi jej povinnosti nepatrí kontrola konzistentnosti jej vlastného stavu. O konzistenciu stavu aplikácie sa má starať View model.

V aplikácii používame model view-y dvoch typov:

- **Session model view** - odpovedajú jednotlivým session-om. Ich hlavnou a v podstate jedinou úlohou je vytvoriť a distribuovať model view-y odpovedajúce danému session-u. Pri ich tvorbe vytvorí medzi nimi väzby, ktoré sú následne počas behu aplikácie využívané na spomínanú vnútro-session-ovú komunikáciu.

  Každému session model view-u odpovedá jeden session view model. Ten pri svojej konštrukcii predá model view-y, dodané v session model view-e, zodpovedajúcim view modelom.

- **Model view** - typ, ktorý nesie vyššie popísanú funkcionalitu Model view vrstvy. Zvyčajne pre každý view model existuje dedikovaný model view ktorý sa preň stará o zabezpečenie ním požadovaných služieb. Nie je to však pravidlo, view model môže mať odkazy na viacero model view-ov ktorých služby následne využíva alebo sú predané, kvôli interakciám vytvoreným, view modelom.

  Väčšina model view-ov spadá pod nejaký konkrétny session. V takom prípade je daný model view vytváraný v odpovedajúcom session model view-e. Výnimkou sú model view-y, ktoré zodpovedajú view modelom pracujúcim v hlavnom okne. Model view-y dotyčných view modelov v základe nezabezpečujú vzájomnú komunikáciu. Pri aktuálnom návrhu aplikácie to nie je potrebné. Ak takáto potreba vznikne, môže sa návrh týchto model view-ov prerobiť.

6.1.4 Model

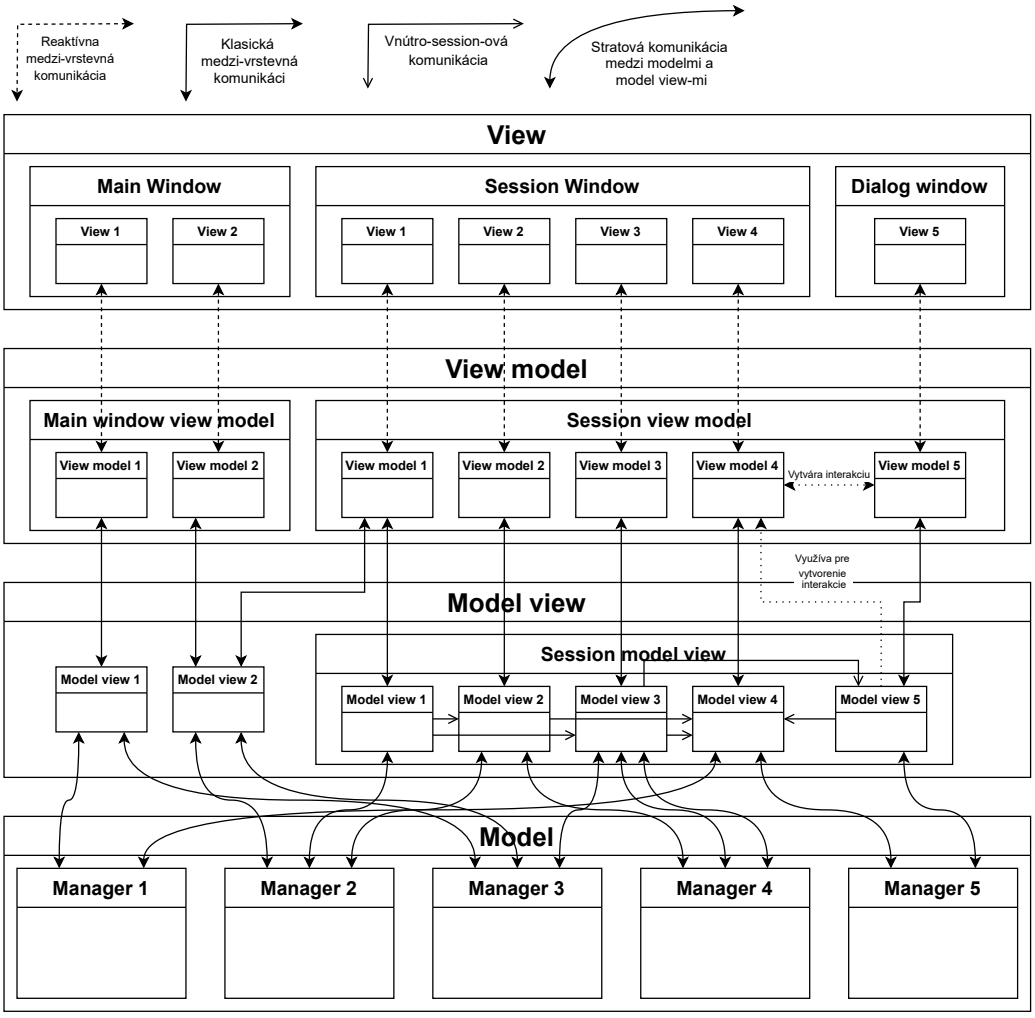
Poslednou „horizontálnou“ vrstvou je Model. Model sa svojou štruktúrou diametrálne odlišuje od predchádzajúcich vrstiev. Je tvorený jednotlivými oblasťami, ktoré spravujú dedikovaní *manažéri*. Manažéri sú pristupovaný z jednotlivých model view-ov a doručujú im svoje služby, či už informačné alebo výpočtové. Predstavujú interface-y ponúkajúce prívetivejší spôsob práce s vnútornými mechanizmami modelov.

Špecifickým znakom komunikácie medzi Model a Model view vrstvami je, že pri nej dochádza ku strate typovej informácie dodávaných dát. Táto vlastnosť je motivovaná jednoduchým faktom, ktorým je udržanie vrstvy Model view jednoduchou. V Model vrstve sa totiž vo veľkom využívajú generiká pre jednoduché prenášanie typovej informácie v ích mechanizmoch.

Využívanie generík v Model view vrstve by však prinieslo značné komplikácie v jej implementácii a tomu odpovedajúce zneprehladnenie kódu. Už len Model samotný trpí jemnou, generikami spôsobenou neprehľadnosťou. Z tohto dôvodu bolo rozhodnuté zabezpečiť jednoduchosť vrstvy Model view za cenu straty typovej informácie dát tečúcich z modelov do model view-ov.

Pri opačnej komunikácii, z model view-ov do modelov, je manažérmi typová správnosť dodaných parametrov opäť testovaná/získaná (väčšinou za pomoci tzv. *generic visitor pattern*. Viac informácii o tejto modifikácii klasického *visitor pattern* návrhového vzoru nájdete v

V modelu existujú popri manažéroch ešte aj tzv. *sub-manažéri*. Tieto entity sú ale určené pre využitie priamo z modelov. Podporujú generickú komunikáciu, na ktorej báze modely fungujú, bez straty typovej informácie a teda sú príjemnejšie pre modelovú komunikáciu než klasickí manažéri.



Obrázek 6.1 Príklad možnej horizontálnej štruktúry aplikácie

6.2 Session-y + *hlavné okno*

Aplikácia, či už z vizuálneho, logického či implementačného hladiska je rozdelená do tzv. *session*-ov. Session-y sú najväčšie stavebné jednotky z ktorých každá predstavuje jedinečný mechanizmus dodávaný aplikáciou. Session-ov (aj rovnakého druhu) môže byť v aplikácii pustených viacero naraz.

Session-y sú vytvárané v *hlavnom okne*.

Závěr

Literatura

1. Anděl, J. *Základy matematické statistiky*. Praha: Matfyzpress, 2007. Druhé opravené vydání. ISBN 80-7378-001-1.
2. Mareš, Martin (ed.). *Šablona závěrečné práce na MFF UK v LaTeXu* [online]. [cit. 2024-03-02]. Dostupné z: `https://gitlab.mff.cuni.cz/teaching/thesis-templates/thesis-cs`.
3. Mareš, Martin. *Jak psát bakalářskou (či jinou) práci* [online]. [cit. 2024-03-02]. Dostupné z: `https://mj.ucw.cz/vyuka/bc/`.
4. Kaplan, E. L.; Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*. 1958, roč. 53, č. 282, s. 457–481.
5. Cox, D. R. Regression models and life-tables (with Discussion). *Journal of the Royal Statistical Society, Series B*. 1972, roč. 34, č. 2, s. 187–220.
6. Student. On the probable error of the mean. *Biometrika*. 1908, roč. 6, s. 1–25.
7. Anděl, J. *Statistické metody*. Praha: Matfyzpress, 1998. Druhé přepracované vydání. ISBN 80-85863-27-8.
8. Lehmann, E. L.; Casella, G. *Theory of Point Estimation*. New York: Springer-Verlag, 1998. Second Edition. ISBN 0-387-98502-6.
9. Dempster, A. P.; Laird, N. M.; Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*. 1977, roč. 39, č. 1, s. 1–38.
10. Genberg, B. L.; Kulich, M.; Kawichai, S.; Modiba, P.; Chingono, A.; Kilonzo, G. P.; Richter, L.; Pettifor, A.; Sweat, M.; Celentano, D. D. HIV risk behaviors in Sub-Saharan Africa and Northern Thailand: Baseline behavioral data from project Accept. *Journal of Acquired Immune Deficiency Syndrome*. 2008, roč. 49, s. 309–319.
11. SZOS. *Čo je to orientačný beh?* [online]. [cit. 2024-06-29]. Dostupné z: `https://www.orienteering.sk/page/co-je-to-orientacny-beh`.
12. adegeo; ihsansfd; alexbuckgit; v-trisshores; DCtheGeek. Desktop Guide (WPF .NET). 2023. Dostupné také z: `https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0`.
13. davidbritch; mhrastegari; jonpryor; Banovvv; jconrey. What is .NET MAUI? 2024. Dostupné také z: `https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0`.
14. JAMES, M. Avalonia UI and MAUI - Something for everyone. 2024. Dostupné také z: `https://avaloniaui.net/blog/avalonia-ui-and-maui-something-for-everyone`.
15. *Welcome* [online]. [cit. 2024-06-30]. Dostupné z: `https://docs.avaloniaui.net/docs/welcome`.
16. *The MVVM pattern* [online]. [cit. 2024-06-30]. Dostupné z: `https://docs.avaloniaui.net/docs/concepts/the-mvvm-pattern/`.

17. *Reactive Programming* [online]. [cit. 2024-06-30]. Dostupné z: `https://www.reactiveui.net/docs/reactive-programming/index.html`.

Seznam obrázků

Seznam použitých zkratek

- OB - orientačný beh
- GUI - grafické užívateľské rozhranie (graphical user interface)
- XAML - Extension application markup language
- MVVM - Model-View-ViewModel

A Přílohy

A.1 První příloha