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 GEOTAGGING FOTOGRAFIÍ

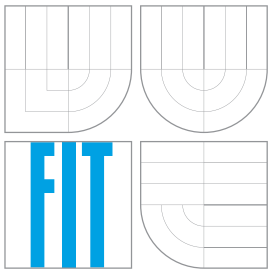
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DANIEL KODEŠ

BRNO 2011

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 GEOTAGGING FOTOGRAFIÍ

PHOTOGRAPHY GEOTAGGING APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DANIEL KODEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KUBÍČEK

BRNO 2011

Abstrakt

Tato diplomová práce popisuje vývoj aplikace pro geotagging fotografií od seznámení s problematikou, přes analýzu a návrh až po výslednou implementaci. Čtenář by po přečtení měl být schopen porozumět problematice časových údajů při geotaggingu využívajícím záznamy GPS tras, bude také rozumět, kde a jak jsou metadata ve fotografiích uložena. Velká část práce se věnuje implementaci aplikace, která čtenáře provede budováním základů architektury aplikace, reprezentací fotografií a načítání jejich metadat programem ExifTool. Postupně budou vysvětleny konvertory souřadnic a parsery GPS tras, které jsou využívány při synchronizaci. Výsledná aplikace umožní uživateli synchronizovat fotografie s GPS trasou a jejich zobrazení na mapě.

Abstract

This master thesis describes development of Photography Geotagging Application from an introduction to the issues, through the analysis and design to the resulting implementation. The reader should be able understand to issues of time in geotagging using the GPS tracks records. He will also understand where and how metadata are stored. The big part of thesis is dedicated to the implementation of application which leads the reader through building the application architecture, the photography representation and loading their metadata with program ExifTool. Gradually will be explained the convertors of coordinates and GPS track parsers, which are used to synchronization. The final application allows the user synchronize photography with GPS track and display them on the map.

Klíčová slova

geotagging, EXIF, GPS, KML, GPX, XML, .NET, C#, MVVM, WPF, ExifTool

Keywords

geotagging, EXIF, GPS, KML, GPX, XML, .NET, C#, MVVM, WPF, ExifTool

Citace

Daniel Kodeš: Aplikace pro geotagging fotografií, diplomová práce, Brno, FIT VUT v Brně, 2011

Aplikace pro geotagging fotografií

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Kubíčka

.....
Daniel Kodeš
23. května 2011

Poděkování

Rád bych poděkoval vedoucímu této práce panu Ing. Radku Kubíčkovi, který mi poskytl odbornou pomoc při psaní práce a pomohl udávat směr, jakým by se aplikace měla vyvíjet.

Obsah

Kapitola 1

Úvod

S rostoucím technologickým pokrokem a minimalizací vznikají nové možnosti, které dříve nebyly možné nebo alespoň pro většinu obyvatel běžné. Typickým příkladem je využití systému GPS. Dnes už jsou běžně dostupné automobilové a turistické navigace a v poslední době se GPS objevuje také v mnoha mobilních telefonech. Díky tomuto masovému rozšíření vznikají pro uživatele nové služby využívající GPS. Jedním z mnoha využití GPS je právě geotagging, typicky zastoupený geotaggingem fotografií. Většina fotografů nefotí fotografie pouze pro sebe, ale chtějí je ukázat přátelům, rodině, či je veřejně vystavit na internetu, aby snadno prezentovali své povedené snímky širokému publiku. Díky geotaggingu autor fotografie nemusí složitě popisovat, kde fotografii pořídil, ale díky zeměpisným souřadnicím uložených v metadatech fotografie jednoduše zobrazí polohu na mapě. Cílem vytvářené aplikace je právě poskytnout uživatelům možnost přiřadit či upravovat polohu pořízení fotografie. To bude možné buď synchronizací vybraných fotografií se záznamem GPS trasy, nebo manuálně, tj. zadáním souřadnic, nebo zvolením polohy na mapě. Žádaným výsledkem bude možnost prohlížení pozice pořízení fotografií na mapě.

Tato práce nejprve v kapitole 2 seznámí čtenáře se systémem GPS, kde získá základní přehled, jak systém funguje. Na GPS logicky navazuje s ním související geotagging včetně popisu způsobu ukládání metadat ve fotografii a formátů GPS tras. Následující kapitola 3 popisuje techniky a postupy použité k analýze, návrhu a implementaci aplikace. Součástí kapitoly je také popis všech nástrojů použitých během celého vývoje aplikace. Poté se v kapitole 4 práce zaměřuje na analýzu a návrh aplikace, jež jsou nezbytné pro úspěšnou implementaci. Z analýzy a návrhu už vychází samotná implementace v kapitole 5, kde jsou popsány některé důležité kroky a klíčové oblasti ve vývoji. Práce je zakončena kapitolou 7 obsahující závěr a zhodnocení práce a možná budoucí rozšíření.

Kapitola 2

Seznámení s GPS a geotaggingem

Cílem této kapitoly je čtenáře seznámit se systémem GPS, pojmem geotagging a s ním souvisejícími informacemi. Čtenář získá přehled o tom, co je a k čemu slouží GPS a bude uveden do problematiky měření a určování času, její znalost napomáhá pochopení problematiky synchronizace, která je zmíněna dále. V druhé části kapitoly bude vysvětlen pojem geotagging a jeho využití v různých aplikacích. Důležitý je také popis formátu pro ukládání metadat ve fotografiích, formátů GPS tras a vysvětlení pojmu interpolace při geotaggingu.

2.1 Global Position System

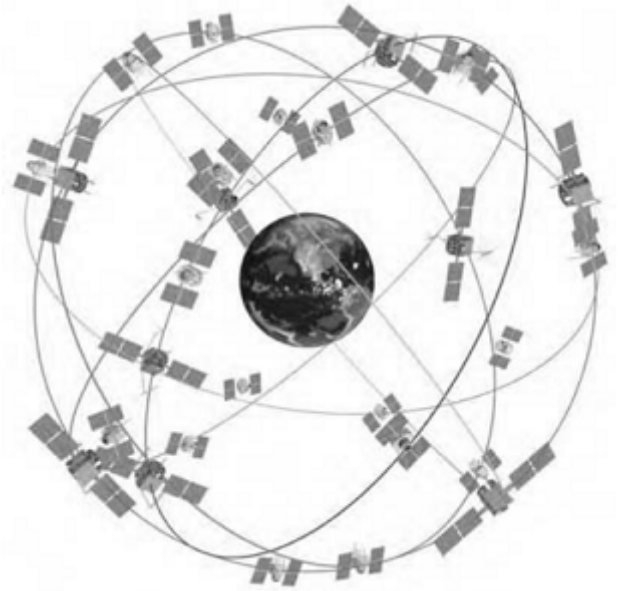
Global Position System, dále už jen GPS, je globální polohový systém vyvinutý a nadále provozovaný Ministerstvem obrany Spojených států amerických[1]. Systém poskytuje přesné, nepřetržité, třírozměrné informace o poloze a rychlosti kdekoliv na Zemi všem uživatelům vybaveným patřičným přijímacím zařízením.

2.1.1 Představení GPS

K určení polohy pomocí GPS se využívá signálu přijímaného ze satelitů umístěných na oběžných drahách Země. Počet satelitů by měl být celkem 24 na 6 oběžných drahách, kde na každé dráze jsou 4 satelity. Takto byl systém původně koncipován, avšak v současnosti je na orbitu mezní počet satelitů, a to 32. Konstelaci satelitů lze spatřit na obrázku 2.1. GPS je systémem pasivním, což znamená, že data ze satelitů jsou pouze vysílána směrem k uživatelským zařízením. Toto však platí pouze z uživatelského pohledu, protože satelity mohou přijímat řídící a kontrolní informace. Pasivnost systému má vyhodu v tom, že může být využíván nekonečným množstvím uživatelských zařízení [2].

Systém využívá konceptu rozmezí času doručení do uživatelského zařízení od času vyslání ze satelitu. Satelity vysílají kódy družic a navigační data všesměrově na několika frekvencích použitím techniky CDMA[2]. Tyto frekvence byly záměrně zvoleny tak, aby pracovaly za jakéhokoliv počasí. Každý satelit generuje krátký kód označovaný jako *hrubý/přírůstkový* neboli C/A kód a dlouhý kód označovaný jako *přesný* nebo P(Y) kód. Tyto kódy slouží uživatelskému přijímači k určení času přenosu signálu a následnému určení vzdálenosti mezi přijímačem a satelitem, zatímco navigační data obsahují informace o poloze satelitu v době vysílání informace. Aby mohl být proveden výpočet polohy, tak je zapotřebí, aby uživatelský přijímač obsahoval hodiny. K určení správné třírozměrné informace

[1] United States Department of Defense (DOD).
[2] Code Division Multiple Access.



Obrázek 2.1: Konstelace satelitů na orbitu [2].

o poloze je nutný příjem signálu alespoň ze 4 satelitů, pomocí kterých se určí zeměpisná šířka a výška, nadmořská výška a odchylka hodin přijímače od času GPS. Pokud je některý z těchto parametrů již znám, tak není zapotřebí všech 4 satelitů.

GPS je využíván dvěma různými způsoby. To znamená, že poskytuje oddělené služby pro civilní a vojenské účely. Služba pro civilní účely se nazývá *Standard Positioning Service* (SPS) a je dostupná zdarma kdekoliv a komukoliv na světě. V současnosti je tato služba využívána miliony uživatelů v různých zařízeních od mobilního telefonu po autonavigační systémy. Služba pro vojenské účely se nazývá *Precise Positioning Service* (PPS). U PPS, stejně jako u SPS, je dle specifikace uváděna nejhorší možná přesnost 15 metrů. Ve skutečnosti je přenos mnohem vyšší. Jak už název služby vypovídá, služba pro vojenské účely je velmi přesná a na rozdíl od služby pro civilní účely není limitována použitím pouze do určité nadmořské výšky. Toto omezení bylo zavedeno za účelem zabránění zneužití GPS v různých balistických střelách. Přístup do PPS je kryptován a pokud někdo z civilního sektoru vyžaduje využití GPS ve vyšších nadmořských výškách musí získat speciální povolení od Ministerstva obrany Spojených států amerických. Kryptování se provádí pomocí technik antispoofing (AS) a selective availibility (SA). Antispoofing je ochrana proti zneužití vojenského signálu. Útočník by mohl zneužít vojenský signál tak, že by zkopíroval družicové kódy, navigační data a nosnou frekvenci Dopplerova efektu za účelem oklamání přijímače oběti. Mechanismus SA sloužil ke snížení přesnosti určení polohy uživatelů SPS. Tento mechanismus byl zrušen 1.5.2000, což právě pomohlo rozmachu GPS v civilním sektoru [3].

2.1.2 Časový systém

Čas hraje velmi důležitou roli v určování polohy pomocí GPS. Signál GPS je ovládán přesnými atomovými hodinami, ty zajišťují dosažení přesné polohy při výpočtu vzdálenosti uživatelského zařízení od satelitu. GPS lze tedy také využít pro synchronizaci času. Celosvětově se používá velké množství časových systémů, jedním z nich je koordinovaný světový

čas UTC[3] využívaný systémem GPS. Zkratka UTC neodpovídá přesně názvu systému, byla zvolena jako určitý kompromis mezi anglickou zkratkou CUT a francouzskou zkratkou TUC[4]. Význam slova světový v názvu je v tom, že UTC není založen na rotaci Země, takže je ve všech časových pásmech stejný. UTC je založen na Mezinárodním atomovém času TAI[5]. Ten je řízen atomovými hodinami, které jsou rozmístěné v několika časových laboratořích po celém světě a při určování přesného času odečte Mezinárodní úřad pro míry a váhy[6] z těchto atomových hodin přesnou hodnotu času. Při navigaci je však požadován časový systém relativní k rotaci Země (UT1) a ne atomický čas. Jelikož je rotace Země nepravidelná, dochází k odchylkám od atomového času, který je pochopitelně velmi přesný. Aby se dosáhlo synchronizace s tímto systémem, tak je třeba jednou za čas upravit UTC čas o jednu, tzv. přestupnou sekundu. Čas v UTC se udržuje v rozmezí 0,9 sekundy od UT1, po jehož překročení je čas UTC upraven [2].

Přestupná sekunda se zavádí vždy 31. prosince nebo 30. června o půlnoci UTC času. Vzniká tedy situace, kdy jedna minuta má 61 nebo 59 sekund. Dosud však nikdy nedošlo k ubrání přestupné sekundy, protože rotace Země se stále mírně zpomaluje. Přestupná sekunda zatím byla aplikována 24-krát, první byla zavedena 30. června 1972 a naposledy se tak stalo 31. prosince 2008. Čas TAI není takto upravován, takže čas UTC se nastavuje podle času TAI + celkový počet přestupných sekund [17].

2.2 Geotagging

Geotagging je proces přidání geografických informací do metadat mediálního obsahu. Typickým příkladem takového mediálního obsahu jsou fotografie. Mezi další se řadí videa, SMS, webové stránky a další. Geografické informace obsahují především informaci o zeměpisné šířce a výšce, mohou však také obsahovat dodatečné informace jako nadmořskou výšku, azimut (orientovaný úhel určující směr natočení při zaznamenání polohy), přesnost polohy nebo zeměpisné názvy.

Jeho využití závisí na mediálním obsahu, avšak hlavním využitím je vždy zprostředkování nějaké informace k zeměpisným souřadnicím. U fotografií uživatel například vidí, kde byly pořízeny, nebo opačným přístupem je vyhledat si místo na mapě a zobrazit si fotografie, které byly v okolí pořízeny. U videí je využití víceméně totožné, u webových stránek chytře využívá geotagging např. Wikipedia, kde informace, u kterých to má smysl, mají přiřazené zeměpisné souřadnice a uživatel si lehce zobrazí jejich polohu na mapě.

2.2.1 Geotagging fotografií

Geotagging fotografií může probíhat dvěma způsoby, a to pomocí zabudovaného GPS přijímače ve fotoaparátu nebo spárováním GPS trasy zachycené externím GPS přijímačem s fotografiemi. Pokud se využívá fotoaparát se zabudovaným přijímačem, jsou fotografie doplněny o zeměpisné souřadnice okamžitě v době pořízení. V případě, že se jedná o spárování s externím přijímačem, jsou fotografie označeny manuálně uživatelem. Zde nastává problém se synchronizací času. Je zapotřebí mít čas u obou zařízení shodný, nebo provést korekturu času při zpracování. Druhým problémem s časem je použití rozdílných časových

[3] Coordinal Universal Time.
[4] Temps Universel Coordonné.
[5] International Atomic Time. Zkratka pochází z francouzštiny.
[6] Bureau International des Poids et Mesures.

systémů. Časový systém použitý ve fotoaparátu je založen na rotaci Země, takže se v závislosti na časovém pásmu, kde byla fotografie pořízena, mění. Důležité je také nastavení správného časového pásma ve fotoaparátu. Jak již bylo zmíněno v sekci 2.1.2, časový systém je v GPS nezávislý na rotaci Země a je na všech místech stejný. Toto je nutné brát v potaz při spárování a učinit kroky ke správnému nastavení času.

Veškeré informace o fotografiích včetně geografických informací se ukládají do metadat fotografie. Tyto informace nejsou při pouhém prohlížení obrázku viditelné, ale mohou být speciálními programy čteny či modifikovány. Metadata se ukládají v jedné ze dvou specifikací – EXIF[7] a XMP[8], kde známější je EXIF a bude podrobněji popsán v sekci 2.2.2. Specifikace formátu XMP je standardem společnosti Adobe, jeho rozšířenost však není taková jako u formátu EXIF [15].

Také zeměpisné souřadnice obsahující zeměpisnou výšku a šířku lze reprezentovat několika způsoby, které jsou popsány v tabulce 2.1. Důležitou informací je, že hodnotu souřadnice ovlivňuje, zda jde o jižní výšku, či západní šířku. Hodnoty souřadnic uloženy jako reálné číslo jsou pro jižní výšku a západní šířku záporné. V případě stupňů se poloha řídí podle anglické zkratky zeměpisných stran - N, S, E a W, tedy *North* (Sever), *South* (Jih), *East* (Východ) a *West* (Západ).

Tabulka 2.1: Formáty zápisu GPS souřadnic.

Vzor	**Hodnota**
[-]d.d, [-]d.d	40.446195, -79.948862
d m.m' {N|S}, d m.m' {E|W}	40 26.7717 N, 79 56.93172 W
{N|S} d m.m' {E|W} d m.m'	N 40 26.7717, W 79 56.93172
d°m' s"{N|S}, d°m' s"{E|W}	40°26'47"N, 79°58'36"W
{N|S} d°m' s", {E|W} d°m' s"	N 40°26'47", W 79°58'36"

2.2.2 Exchangeable Image File Format

EXIF je nejpoužívanější specifikací pro formát metadat obsahující informace o fotografii. Specifikace využívá existujících formátů souborů fotografií jako jsou JPEG a TIFF 6. revize, ke kterým přidává metadata ve formě tagů. Tag představuje štítek, který je v dané fotografii jedinečný a je mu přiřazena patřičná hodnota. Tagů existuje velké množství, ale dále budou popsány pouze některé, které se přímo týkají času a geolokačních informací. Metada umísťují do fotografií digitální fotoaparáty již v době pořízení. Specifikace umí přiřazovat metadata i k audio souborům, konkrétně k souborům formátu RIFF WAV, ale její popis je již mimo rozsah této práce a lze ji najít v [16].

Metadata jsou v souboru fotografie uloženy v přesně dané struktuře, kde výchozím bodem je TIFF hlavička. Za hlavičkou je první IFD (*Image File Directory*), IFD-0, v němž jsou uloženy informace o fotografii, jako jsou např. rozlišení a orientace fotografie, výrobce a model fotoaparátu, autor, copyright a další. Mimo tagů jsou zde také uloženy ukazatele na další IFD, kde hlavními jsou Exif IFD a GPS IFD. Exif IFD obsahuje mimo jiné tagy s nastavením fotoaparátu v době pořízení, kde jsou např. údaje o citlivosti ISO, blesku, expozici, přiblížení, atd. Jsou tu také tagy s časem a datem pořízení fotografie. Zde se však

[7] Exchangeable Image File Format.
[8] Extensible Metadata Platform.

naráží na chybu specifikace, která u těchto údajů neudržuje informaci o časovém pásmu. Tím je určení času pořízení fotografie nejednoznačné. Tento problém se dá vyřešit rozdílem mezi časem pořízení a časem z GPS, který je v UTC, ale jen za předpokladu, že tyto informace jsou dostupné. Jak už název napovídá, GPS IFD obsahuje informace získané z GPS. Je zde možné vyčíst informace o zeměpisné výšce a šířce, nadmořské výšce, času v systému UTC a další informace jako azimut, rychlost, počet satelitů atd. Popis základních geografických metadat lze nalézt v tabulce 2.2. Další informace o EXIF a popis všech tagů lze nalézt v literatuře [16].

Tabulka 2.2: Důležité tagy z GPS IFD.

Tag	**Popis**
GPSLatitude	Zeměpisná šířka ve stupních, minutách a sekundách.
GPSLatitudeRef	Severní nebo jižní zeměpisná šířka.
GPSLongitude	Zeměpisná výška ve stupních, minutách a sekundách.
GPSLongitudeRef	Východní nebo západní zeměpisná výška.
GPSTimeStamp	Čas pořízení v UTC.

2.2.3 Formáty GPX a KML

Formáty GPX (*GPS Exchange Format*) a KML (*Keyhole Markup Language*) jsou ve zkratce formáty pro ukládání geografických informací, jako jsou jednotlivé pozice nebo trasy skládající se z většího množství pozic. U jednotlivých pozic formáty udržují informace o zeměpisné šířce a výšce, nadmořské výšce a času. Oba formáty jsou aplikací metajazyka XML, díky tomu je velikost uložených dat malá a umožňuje jednoduché parsování a vytváření.

GPX

Kořenovým prvkem ve struktuře dokumentu formátu GPX je prvek `gpx`. Ten může obsahovat jeden prvek s metadaty popisující dokument. V metadatech se uchovávají informace o autorovi, datum, popis a jiné. Důležitější však je, že kořenový prvek může obsahovat tři základní typy prvků udržující geografické informace. Každého typu může být 0 až n prvků.

Základním typem je prvek `waypoint` reprezentující bod na mapě, který byl uložen manuálně uživatelem pro zaznamenání aktuální pozice. Hodnoty zeměpisných souřadnic jsou atributy prvku uloženy jako reálná čísla ve stupních. Vnitřní strukturu prvku tvoří prvky popisující daný bod, kam patří např. název, komentář a hlavně prvek s informací o datu a čase zaznamenání. Formát zápisu data a času je definovaný mezinárodním standardem ISO 8601 a jednotlivé variace zápisu jsou popsány v tabulce 2.3. Je důležité zmínit, že čas může být uložen jako lokální nebo jako UTC. Pokud je čas uložen v systému UTC, tak je na konci označen velkým písmenem Z, což značí *zero*, neboli nulovou odchylku od UTC času. V případě lokálního času je uvedení časového posuvu určující časové pásmo nepovinné, avšak jeho uvedení je silně doporučeno, protože bez něj se tato informace nenávratně ztrácí a snižuje se tak vypovídající hodnota času.

Dalšími dvěma typy prvků jsou `rte` a `trk`, které tvoří trasu. Rozdíl mezi nimi je v tom, že prvek `trk` je již zaznamená trasa v minulosti, tedy záznam o tom, kde se GPS přijímač skutečně pohyboval. Jednotlivé body na trase by tak měly mít přiřazený čas, kdy se daná pozice zaznamenala. Oproti tomu, prvek `rte` je určen pro plánovanou trasu, vytvořenou

Tabulka 2.3: Formát zápisu data a času dle ISO 8601 [8].

Typ informace	**Formát zápisu**
Datum	YYYY-MM-DD YYYYMMDD
Základní čas	hh:mm:ss hhmmss
Čas	<Základní čas>Z <Základní čas> ±hh <Základní čas> ±hh:mm
Datum a čas	<Datum>T<Čas>

např. v počítači, kde jednotlivé body by neměly mít přiřazený čas. Typ trasy `rte` může sloužit např. k nahrání do GPS přijímače, který uživatele vede po jím definované trase. Tyto prvky mohou obsahovat další informace jako je název trasy, komentář, atd. Body trasy představují až prvky `rtept` nebo `trkseg`, které obsahují několik prvků typu `waypoint`. Formát GPX dále dovoluje do struktury dokumentu přidávat vlastní rozšíření přidávající některé vlastnosti chybějící ve specifikaci formátu. Ukázku trasy obsahující jediný bod lze vidět ve výpisu 2.1 [4].

```
<trk>
  <name>Ukázková trasa</name>
  <trkseg>
    <trkpt lat="48.93517000" lon="14.48747000">
      <ele>393.00000</ele>
      <time>2011-03-15T23:38:14Z</time>
    </trkpt>
  </trkseg>
</trk>
```

Výpis 2.1: Ukázka trasy s jediným bodem ve formátu GPX.

KML

Formát KML byl vyvinut společností Keyhole, Inc., která byla později koupena společností Google včetně jejich virtuálního glóbusu Earth Viewer. Ten pracoval právě s daty ve formátu KML a společností Google byl přejmenován na Google Earth. Využití v této aplikaci se projevilo na komplexnosti formátu. KML ve verzi 2.2 se v roce 2008 stal standardem Open Geospatial Consortium [20]. Kromě klasických prvků pro popis bodu a trasy jako ve formátu GPX umožňuje KML vytváření polygonů, modelů a různé druhy překryvu, jak povrchu, tak i obrazovky. Je možné také nastavení různých stylů, úhlů pohledu, kamer a přeletů mezi jednotlivými body. V rámci této práce si však vystačíme pouze s popisem uložení bodů a tras.

Výchozím prvkem pro ukládání geometrických útvarů, jako je bod nebo trasa, je prvek `Placemark`. Pro bod máme prvek `Point` a pro trasu `LineString`. Pro účely této práce je problém v tom, že tyto prvky neuchovávají záznam o čase, ten je pouze v `Placemark`. V případě jediného bodu to není překážkou, ale u trasy ano. Pro zápis trasy tedy nemůžeme

využít prvek `LineString`, ale několik prvků `Placemark` vždy s jediným bodem uvnitř. Geografické informace jsou v `Point` uzavřeny v povinném prvku `coordinates`. Hodnoty jsou uloženy stejně jako ve formátu GPX jako reálné číslo a jsou odděleny čárkou. Nadmořská výška je v metrech a je volitelná. Význam její hodnoty je definován v prvku `altitudeMode`. Význam hodnot tohoto prvku je popsán v tabulce 2.4. Ukázku s příkladem jednoho bodu lze vidět ve výpisu 2.2 [10].

Tabulka 2.4: Význam hodnot prvku altitudeMode.

Hodnota altitudeMode	**Význam**
clampToGround	Výchozí hodnota, nadmořská výška bude ignorována.
relativeToGround	Nadmořská výška je relativní k výšce v dané poloze, např. pro hodnotu 9 a výšku 20 metrů v dané poloze je nadmořská výška 29 metrů.
absolute	Hodnota je absolutní bez ohledu na nadmořskou výšku v dané poloze.

```
<Placemark>
  <TimeStamp>
    <when>2010-08-24T08:51:00Z</when>
  </TimeStamp>
  <Point>
    <coordinates>-16.831612,28.211588,80.0</coordinates>
  </Point>
</Placemark>
```

Výpis 2.2: Ukázka bodu ve formátu KML.

Kapitola 3

Použité nástroje a techniky

V této kapitole se práce zaměřuje na popis jednotlivých technik a nástrojů použitých k návrhu aplikace. Je zde popsán jazyk UML, který je velkým pomocníkem při analýze a návrhu aplikace. Poté následuje popis objektově orientovaného programování, jehož zásady a principy napomáhají ke kvalitnímu návrhu a výsledné implementaci. Kapitola dále pokračuje popisem použitých nástrojů, které byly použity ve fázi návrhu, kam se řadí Visual Paradigm pro návrh systému pomocí UML, a Microsoft Expression Blend pro návrh uživatelského rozhraní. Mezi implementační nástroje patří Microsoft .NET Framework a jeho podmnožina pro vytváření uživatelského prostředí WPF. Kapitola je uzavřena popisem návrhového vzoru Model-View-ViewModel pro vývoj ve WPF.

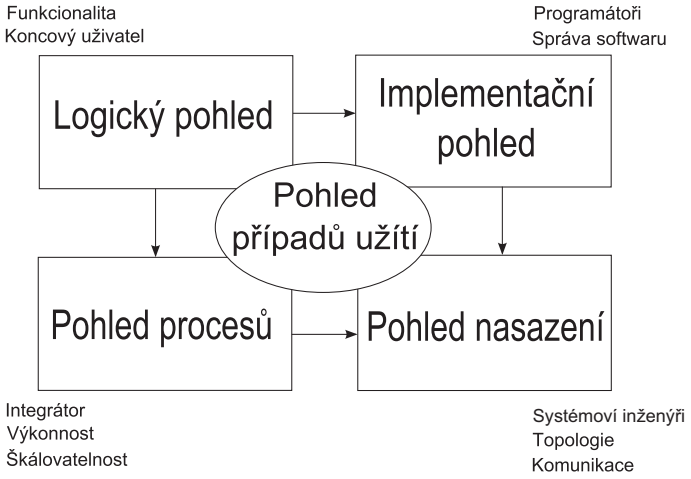
3.1 Unified Modeling Language

Unified Modeling Language (UML) je grafický jazyk, který slouží pro modelování, vizualizaci, specifikaci, stavbu a dokumentaci objektově orientovaných softwarových systémů. Dle specifikace UML obsahuje celkem 15 diagramů logicky rozdělených do 2 kategorií: strukturní diagramy a diagramy chování.

Umožnuje modelovat aplikace pomocí pevně dané syntaxe, proto není problém vytvořené modely sdílet s ostatními návrháři. UML je navrženo tak, aby modelům porozuměl i zadavatel, který nemusí být úplně technologicky zdatný. Tímto se zajistí, že případná nedorozumění nebo nepochopení požadavků se odhalí již ve fázi analýzy a návrhu.

Architekturu systému lze z pohledu UML rozdělit na několik částí tak, aby byly na vysoké úrovni zachyceny všechny důležité aspekty struktury systému. Tato koncepce se jmenuje *4+1 pohled* a byla navržena Philippem Kruchtenem [7]. Těchto několik pohledů dovoluje různým účastníkům projektu (koncový uživatel, vývojář, projektový manažer, ...) různé pohledy na systém tak, aby byly zřejmé věci pro něj a nebyl zatěžován věcmi, které s jeho zaměřením nesouvisejí. Vazby mezi pohledy znázorňuje obrázek 3.1. Jak z názvu vyplývá, koncepce obsahuje 4 základní pohledy a jeden speciální popisující jejich interakci [22]:

- **Logický pohled**
  Logický pohled představuje podporu pro funkcionální požadavky na systém. Systém je dekomponován na jednotlivé objekty, třídy a jejich vazby v určité úrovni abstrakce, aby byly pokryty klíčové části systému. Do logického pohledu typicky patří tyto diagramy: diagram tříd, složené struktury, objektů, balíčků a stavového automatu.



Obrázek 3.1: Model 4+1 pohledu [7].

- **Pohled procesů**
  Pohled procesů se zaobírá některými nefunkcionálními požadavky jako výkonnost a dostupnost systému. Zaměřuje se také na konkurečnost, distribuovatelnost, integritu systému, toleranci chyb a také, jak abstrakce z logického pohledu zapadá do architektury procesů. Nejjednodušší definicí je, že je to ve skutečnosti logický pohled, avšak orientován na procesy v systému. Do pohledu procesů patří následující diagramy: diagram tříd, složené struktury a objektů.

- **Implementační pohled**
  Tento pohled poskytuje náhled na organizaci softwarových modulů ve vývojovém prostředí. Software je brán jako balíček skládající se z menších částí (programové knihovny a subsystémy), které jsou vyvíjeny vývojáři. Subsystémy jsou organizovány do hierarchie vrstev, kde každá vrstva poskytuje dobře definované rozhraní vrstvě, která je nad ní. Základními požadavky na software, které se zde musejí uplatňovat, jsou znovupoužitelnost, přenositelnost a bezpečnost. Do implementačního pohledu patří diagramy komponent a balíčků.

- **Pohled nasazení**
  Hlavním úkolem v tomto pohledu je nasazení softwaru na množinu výpočetních uzlů (hardware). Primárně se zaměřuje na nefunkcionální požadavky systému jako dostupnost, spolehlivost (tolerance chyb), výkonnost (propustnost) a škálovatelnost. Software může být nasazen na několika fyzických konfiguracích, např. ve vývoji, testování nebo u zákazníka. Měl by tedy být dostatečně flexibilní a mít minimální dopad na změnu zdrojových kódů při nasazování. V tomto pohledu je jediný diagram, a to diagram nasazení.

- **Pohled případů užití**
  V pohledu případů užití jsou jednotlivé elementy předchozích pohledů, které spolu bezproblémově spolupracují, zachyceny jako množina případů užití. Pokud se na to podíváme z druhé strany, tak všechny předchozí modely vycházejí z těchto případů užití. Do tohoto pohledu patří diagramy případů užití a interakce.

Informace, které byly získány o modelu 4+1 pohledu, vycházejí z literatury [7].

3.1.1 Diagram případů užití (Use Case diagram)

Diagram případů užití modeluje interakci uživatelů se systémem, aby se lépe pochopily požadavky zadavatelů a přesně vymezil rozsah vytvářeného systému. To znamená, že systém bude obsahovat pouze to, co je uvedeno v případů užití.

Uživatelé komunikující se systémem se nazývají aktéři. Aktér definuje roli, ve které uživatel vystupuje v rámci komunikace se systémem. Jeden fyzický uživatel může vystupovat v několika rolích, aktérem muže být i externí systém nebo čas.

3.1.2 Diagram tříd (Class diagram)

Diagramy tříd zobrazují statickou stránku systému, především vztahy mezi třídami. UML rozlišuje několik druhů tříd a množství vztahů, které jednotlivé třídy propojují (asociace, agregace, kompozice, ...). Diagram tříd by měl být ve finále ekvivalentní zdrojovému kódu.

3.1.3 Diagram balíčků (Package diagram)

Diagram balíčků znázorňuje balíčky obsahující jednotlivé diagramy tříd. Pomocí šipek jsou mezi balíčky znázorněny závislosti mezi diagramy tříd, balíčky jsou znázorněny jako složky uchovávající třídy z diagramu tříd. Diagram balíčků se používá pro znázornění vrstev aplikační architektury.

3.1.4 Diagram nasazení (Deployment diagram)

Diagram nasazení zobrazuje rozložení hardwarových zdrojů, na kterých běží nasazovaná aplikace. V jednotlivých zdrojích mohou být definovány softwarové komponenty, které vyjadřují závislosti, jež musí být splněny ke správnému běhu aplikace. Hardwarovými zdroji mohou být různé webové a databázové servery či klientské stanice.

Pro popis jednotlivých diagramů byla čerpáno z literatury [6].

3.2 Microsoft .NET Framework

Microsoft .NET Framework je softwarový framework pro některé operační systémy Windows. Díky open source projektu Mono je možné aplikace spouštět i na jiných platformách, jako je Linux nebo Mac. Obsahuje velkou knihovnu připravených řešení obecných programových problémů. Tato knihovna se nazývá Base Class Library (BCL). Obsahuje podporu pro programování uživatelského prostředí, přístup k datům, kryptografii, vytváření webových stránek a další.

Programy je možné psát ve vybraném programovacím jazyku, které .NET Framework podporuje. Mezi podporované jazyky patří např. VB.NET, C#, C++, JScript a F#. Tato nezávislost je zajištěna pomocí CLI[1]. Programy napsané ve vybraném jazyce se kompilují do přechodného jazyka CIL[2], kód v tomto přechodném jazyku je vykonáván virtuálním strojem CLR[3] [18].

[1] Common Language Infrastructure.
[2] Common Intermediate Language.
[3] Common Language Runtime.

3.2.1 Microsoft Visual Studio

Microsoft Visual Studio je integrované vývojové prostředí (IDE). Podporuje několik programovacích jazyků a je primárně určeno k vývoji aplikací pomocí .NET Frameworku. Je možné v něm vytvářet aplikace ve Windows Forms, Windows Presentation Foundation, webové aplikace, webové služby i například aplikace pro mobilní zařízení.

Obsahuje v sobě mimo jiné editor kódu podporující IntelliSense (automatické doplňování) a zvýrazňování syntaxe, dále má v sobě integrovaný debugger (nástroj pro ladění aplikací) a taktéž obsahuje vizuální návrhový nástroj, který značně urychluje implementaci.

3.2.2 Windows Presentation Foundation

Windows Presentation Foundation (WPF) je poměrně nový prezentační systém k vytváření aplikací pro operační systém Windows. WPF umožňuje vytvářet uživatelsky přívětivé aplikace, které se starším systémem Windows Forms nebyly možné. Narozdíl od staršího systému nevyužívá pro vykreslování procesor, ale výkon grafické karty díky DirectX, čímž se dosahuje dobrého výkonu aplikací i na relativně starších počítačích. Velkou předností je, že aplikace jsou psány deklarativně značkovacím jazykem XAML[4]. Ten umožňuje oddělit práci programátora, který může psát nezávisle kód v pozadí, a designéra navrhujícího uživatelské prostředí [21]. Pro návrh uživatelského prostředí je vhodným nástrojem Microsoft Expression Blend popsaný v části 3.2.3. S tímto přístupem také přichází lepší možnosti pro testování aplikace, protože prezentační vrstva není tak pevně svázána s aplikační logikou. Tento návrhový vzor se jmenuje Model-View-ViewModel a bude více popsán v části 3.3.

WPF poskytuje velké možnosti pro vytváření bohatého uživatelské prostředí. Kromě klasických komponent známých z Windows Forms umožňuje v aplikaci 2D a 3D grafiku, animace, typografii, práci se šablonami pro stylizování uživatelských prvků, práci s médii a mnohé další. Nespornou výhodou WPF je také obousměrné vázání dat. To zahrnuje jak zobrazení dat v uživatelských prvcích z tříd aplikační logiky, tak jejich zpětné zapsání do objektů při změně hodnoty v uživatelském prvku. Tato funkčnost velmi usnadňuje práci s daty [18].

3.2.3 Microsoft Expression Blend

Microsoft Expression Blend je nástroj pro snadnou tvorbu uživatelského rozhraní. S tímto nástrojem lze navrhovat grafické rozhraní v technologiích využívajích pro zápis XAML. Kromě výše zmíněného WPF lze navrhovat aplikace i pro Silverlight a mobilní operační systém Windows Phone 7. Oproti Microsoft Visual Studiu jsou některé programátorské prvky potlačeny, ale zase se rozšiřují možnosti pro návrh uživatelského rozhraní.

Mezi klíčové vlastnosti patří například podpora importu grafických prvků vytvořených programy Adobe Photoshop a Adobe Illustrator. Další užitečnou funkcí je naplnění datových prvků testovacími daty pro lepší názornost v době návrhu. Díky způsobu využití datových zdrojů se při dalším vývoji nemusí měnit uživatelské rohraní, pouze se data budou načítat z jiného zdroje. To umožňuje dobře oddělit práci programátora a designéra aplikace [21].

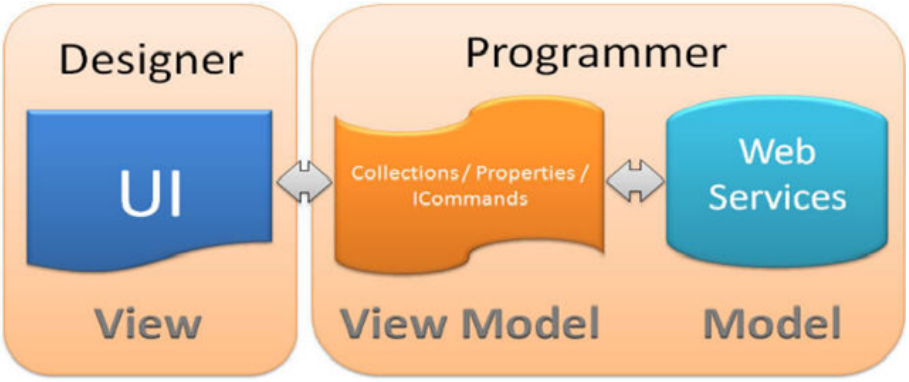
[4] eXtensible Application Markup Language.

3.3 Model-View-ViewModel

Model-View-ViewModel, zkráceně MVVM, je návrhový vzor speciálně vyvinutý pro vývoj aplikací pomocí XAML, typicky WPF, Silverlight a nyní také pro platformu Windows Phone. Snaží se o co nejmenší vazbu mezi definicí uživatelského rozhraní a jejím kódem v pozadí. Klasický způsob implementace s využitím kódu v pozadí ztěžuje přenositelnost na jinou platformu a znemožňuje testování. MVVM je vhodný pro velké a náročné aplikace. Jeho využití na malých projektech nepřináší tolik výhod, protože úsilí vynaložené v počátku vývoje se v takto malém projektu nevrátí. Samotný Microsoft tento vzor využil například při vývoji výše popsaného nástroje Microsoft Expression Blend.

Název vzoru vznikl ze tří logických částí. Model představuje pouze objekt reálného světa uchovávající data, typicky je umístěn v aplikační logice aplikace. Zbylé dvě části již představují prezentační vrstvu. Objekt View je samotné uživatelské rozhraní zapsané v XAML. Objekty Model a View nemají mezi sebou přímou vazbu a komunikují přes prostředníka ViewModel. Komunikace probíhá tak, že po interakci uživatele s objektem View je příkaz obsloužen objektem ViewModel, který provede požadovanou operaci s objektem Model. Jednotlivé části a kdo se na jejich vývoji podílí znázorňuje obrázek 3.2.

Pokud by mělo být uvedeno několik výhod tohoto návrhového vzoru, tak k nim bezesporu patří již zmíněné oddělení kódu v pozadí od definice uživatelského rozhraní. Dále sem patří obousměrné provázání dat, kdy změna ve View nebo ViewModel se projeví i na opačné straně. Velmi užitečné je také využití datových šablon ve View. V XAML lze lehce zapsat strukturu zobrazování dat, kterou poté mohou sdílet více View. Celkově MVVM přináší vývojáři velkou podporu pro vytvoření znovu použitelné, testovatelné a lépe spravovatelné aplikace [14].



Obrázek 3.2: Reprezentace vzoru MVVM [19].

Kapitola 4

Analýza a návrh aplikace

Tato kapitola obsahuje přesné zadání aplikace, z kterého se vychází při samotné analýze a návrhu. Návrh je prováděn pomocí jazyka UML tak, aby se minimalizovaly nejasnosti. Ke konci kapitoly je také navržen základ uživatelského rozhraní za pomoci nástroje Microsoft Expression Blend.

4.1 Zadání

Cílem této aplikace je umožnit uživateli přiřadit GPS souřadnice k jím vybraným fotografiím. Uživatel by měl mít možnost přiřadit polohu k jedné nebo více fotografiím manuálně zadáním přesné polohy, výběrem na mapě, a nebo pomocí synchronizace s GPS trasou. Aplikace by měla podporovat minimálně formáty tras GPX a KML. Samozřejmostí je, že aplikace bude schopna zobrazit pozici pořízení fotografie na mapě. K jednotlivým fotografiím by mělo být umožněno prohlížení metadat ve formátu EXIF. Dalším požadavkem na aplikaci je umožnění vytváření alb tak, aby uživatel měl možnost znovu otevřít sérii fotografií, které dříve pořídil např. na výletě a sesynchronizoval s GPS trasou.

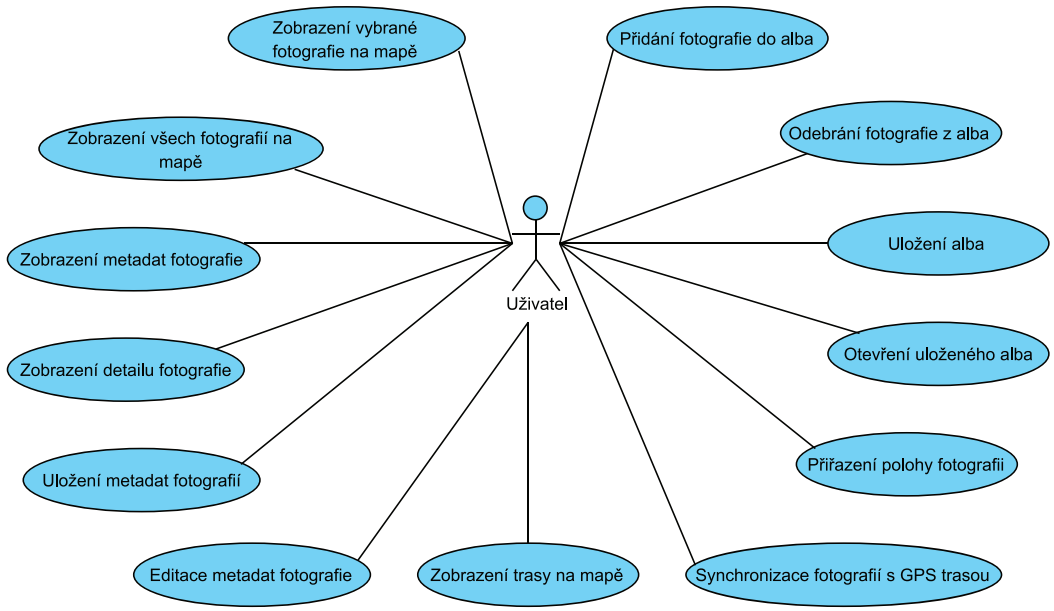
Stěžejní oblastí je synchronizace série fotografií s trasou obsahující GPS souřadnice. Problematika již byla popsána výše, zde je však potřeba uvést, že korekce času by měla být uživatelsky přívětivá, tedy aby fotografie dokázal sesynchronizovat i méně zkušený uživatel.

4.2 Návrh

Tato sekce je zaměřena na návrh aplikace tak, aby byly předem stanoveny všechny její důležité části. Návrh bude vycházet ze zadání a bude popsán pomocí UML zmíněného dříve. Výchozím diagramem je diagram případů užití znázorňující všechny operace, které může uživatel s aplikací provádět. V rámci diagramu případů užití je zapotřebí detailně popsat specifikace jednotlivých případů užití. Aby byly specifikace případů užití lépe pochopitelné, bude vhodné je graficky reprezentovat pomocí diagramu aktivit.

4.2.1 Návrh realizace aplikace

Jak již bylo zmíněno výše, pro návrh aplikace bude výchozím diagramem diagram případů užití. Aplikace je primárně určena pro jediného uživatele, takže diagram bude obsahovat pouze jediného aktéra, který může vykonávat veškeré operace. Výsledným diagram je na obrázku 4.1.



Obrázek 4.1: Diagram případů užití.

Aktéři:

- **Uživatel**
  Aktér Uživatel je jediným uživatelem aplikace. Jako takovému mu přísluší právo vykonávat všechny případy užití uvedené v diagramu. Aktér tedy po spuštění aplikace nemusí absolvovat přihlášení a může okamžitě provádět požadované operace.

Případy užití:

- **Zobrazení vybrané fotografie na mapě**
  Pokud daná fotografie obsahuje v metadatech informace o zeměpisných souřadnicích určující polohu pořízení snímku, měla by se po označení fotografie uživatelem zobrazit tato pozice pomocí ukazatele na mapě. Ukazatel pozice pořízení se však ukáže pouze za předpokladu, že jsou na mapě povoleny ukazatele vybrané fotografie. Mapa by se také měla dostatečně přiblížit, aby byla zřejmá přesná poloha.

- **Zobrazení všech fotografií na mapě**
  Pozice všech fotografií v albu obsahující informace o zeměpisné poloze se zobrazí na mapě. Fotografie se zobrazují na mapě ihned po přidání do alba, avšak na mapě musí být povoleno zobrazení ukazatelů určující pozici pořízení fotografie. Tento a předchozí příklad užití předpokládá připojení uživatelského počítače k internetu, jelikož mapové podklady nebudou uloženy lokálně, ale budou se stahovat v reálném čase.

- **Zobrazení metadat fotografie**
  Uživatel bude moci u vybrané fotografie zobrazit její metadata, která mu zprostředkují detailní informace vázané mimo jiné k jejímu pořízení. Uživatel si bude moci zvolit mezi zobrazením všech metadat nebo několika hlavními, které definují charakter fotografie. Metadata bude možné také třídit podle formátu metadat nebo podle skupin daného formátu.

- **Zobrazení detailu fotografie**
  Ze seznamu fotografií v albu bude umožněno uživateli zobrazit její detail, tj. její zobrazení v samostatném okně. V tomto zobrazení stále bude možné přecházet mezi jednotlivými fotografiemi v albu. Přechod na jinou fotografii vyvolá případ užití „Zobrazení vybrané fotografie na mapě" tak, aby uživatel mohl na mapě vidět polohu pořízení fotografie.

- **Přiřazení polohy fotografii**
  Tento případ užití dovoluje uživateli manuálně nastavit polohu pořízení fotografie. Polohu lze nastavit třemi způsoby. Prvním je zadáním zeměpisných souřadnic do metadat fotografie. Druhým způsobem je přiřazením polohy z vybraného ukazatele na mapě. Posledním způsobem je přiřazení vybraného bodu z načtené GPS trasy.

- **Zobrazení trasy na mapě**
  V aplikaci bude uživatel moci načíst GPS trasy v podporovaném formátu. Pokud uživatel trasu označí a na mapě bude povoleno zobrazení tras, tak se na mapě vykreslí trasa s dostatečným přiblížením, aby žádný bod trasy nebyl na mapě skryt.

- **Editace metadat fotografie**
  U vybrané fotografie bude možné editovat metadata, která jsou podstatná vzhledem k charakteru této aplikace. Myšleno je datum pořízení a metadata určující geodata snímku.

- **Uložení metadat fotografií**
  Uložení provede zapsání podstatných metadat do zdrojového souboru fotografie. Případ užití se provede na všech fotografiích v albu, které mají příznak, že byly upraveny.

- **Synchronizace fotografií s GPS trasou**
  Uživatel u fotografií v albu bude moci provést synchronizaci s GPS trasou, kterou poskytne aplikaci. Před provedením synchronizace uživatel bude muset vyplnit některé parametry potřebné ke správné synchronizaci. Jak již bylo zmíněno, tento případ užití je stěžejním v celé aplikaci a bude dále přesně specifikován.

- **Přidání fotografie do alba**
  Do aktuálního alba uživatel může přidat libovolný počet fotografií. Po přidání fotografií následuje případ užití "Zobrazení všech fotografií na mapě".

- **Odebrání fotografie z alba**
  Z otevřeného alba lze odebrat jednu či více fotografií. Tyto fotografie se fyzicky neodstraňují z pevného disku, pouze z alba aplikace. Před smazáním bude uživatel dotázán, zda si je prováděnou operací jistý, aby nedošlo k nechtěnému odstranění.

- **Uložení alba**
  Nové a dosud neuložené nebo otevřené album lze uložit ve stávajícím stavu, aby bylo možné jeho pozdější otevření. Spolu s tímto případem užití se provede také "Uložení metadat fotografií".

- **Otevření uloženého alba**
  Aktér Uživatel může otevřít z pevného disku dříve uložené album. Po otevření by aplikace měla načíst veškeré fotografie, kterou jsou v albu obsaženy, a vyznačit je na mapě. V případě, že umístění jedné či více fotografií bylo změněno nebo byly smazány,

tak samozřejmě nebudou zobrazeny a uživatel bude informován o těchto fotografiích s možností jejich odstranění z alba.

Případ užití: Synchronizace fotografií s GPS trasou

Tabulka 4.1 detailně popisuje případ užití Synchronizace fotografií s GPS trasou. Případ užití je jednoznačně identifikován svým ID a jsou definováni účastníci tohoto případě užití. Stěžejní částí je Tok událostí specifikující tok operací, které se v tomto případu provádějí.

Tabulka 4.1: Případ užití: Synchronizace fotografií s GPS trasou.

Případ užití: Synchronizace fotografií s GPS trasou
ID: UC02
Stručný popis: Aplikace za pomocí časových údajů synchronizuje GPS souřadnice trasy s fotografiemi.
Primární aktéři: Uživatel
Sekundární aktéři: Žádný
Vstupní podmínky: Označení fotografie, jejíž paramtery lze využít k nastavení synchronizace s GPS trasou.
Výstupní podmínky: Fotografie obsahující v metadatech odpovídající informace o poloze.
Tok událostí: 1. Pro spuštění případu užití, uživatel vybere volbu "Synchronizovat fotografie s GPS trasou". 2. V případě, že v aplikaci už jsou načteny nějaké trasy, tak budou předvyplněny. Poté bude možné zadání jednoho či více souborů s GPS trasou. 3. Uživatel provede nastavení parametrů pro nejlepší výsledek synchronizace. 4. Volbou náhled se zobrazí očekávané výsledky synchronizace, aniž by se provedlo jejich uložení do fotografií. 5. Uživatel potvrdí dialog. 5.1 Jestliže u některé fotografie nelze určit polohu, zobrazí se náhled s výsledky synchronizace a možností pokračovat. 5.1.1 POKUD si uživatel nepřeje pokračovat A. Návrat k události číslo 3. 6. Fotografie jsou synchronizovány.
Alternativní toky: Storno. Neplatný soubor GPS trasy.

4.2.2 Parsování GPS tras

Je důležité správně navrhnout parsování GPS tras formátů, které byly popsány v oddíle 2.2.3, a reprezentaci jejich dat v aplikaci. Z popisu vyplývá, že bez ohledu na formát jsou jednotlivé pozice určeny geografickými informacemi a časovým razítkem. Třída reprezentující

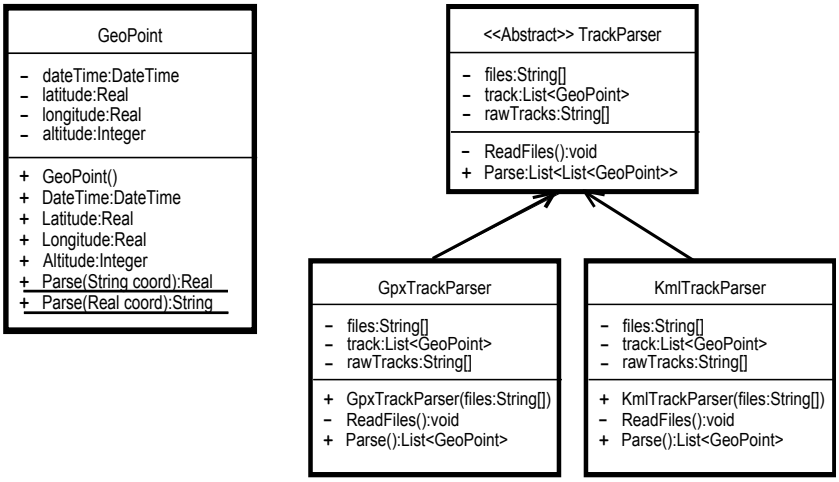
pozici se bude jmenovat `GeoPoint` a její vazby s dalšími třídami souvisejícími s parsováním GPS tras jsou znázorněny v diagramu tříd na obrázku 4.2. Trasu bude představovat seznam skládající se z objektů třídy `GeoPoint`.

Problematickou částí je znázornění zeměpisných souřadnic. Jak již bylo popsáno v tabulce 2.1, existuje více možných formátů zápisu. Pro práci programátora je z výše uvedených nejvýhodnější pracovat s reálnými čísly. Aby program nebyl omezen pouze na zobrazení a práci s tímto formátem, tak třída `GeoPoint` bude obsahovat statické metody pro převod mezi jednotlivými formáty. Tyto metody napomůžou jak zobrazení souřadnic v uživatelsky přívětivějším formátu, tak například při práci se souřadnicemi zadanými uživatelem.

Parsery

Protože výsledek parsování obou formátů by měl být totožný, tj. seznam objektů třídy `GeoPoint`, je vhodné navrhnout abstraktní třídu pokrývající společnou funkcionalitu. Ta bude například obsahovat metodu pro otevření a načtení souboru s GPS trasou do paměti, která bude společná pro oba formáty. Zároveň bude definovat rozhraní abstraktních metod, jejichž implementace bude ve třídě příslušné danému formátu. Název abstraktní třídy bude `TrackParser` a je znázorněna v diagramu tříd na obrázku 4.2.

Při návrhu musí být také brána v potaz situace, kdy bude parsováno více souborů s GPS trasami nebo soubor obsahující více segmentů. V ideální situaci uživatel zaznamenává jedinou trasu během celého výletu. Může však nastat situace, kdy se například v GPS přijímači vybijí baterie a trasa bude rozdělena do více segmentů či souborů. Další možnou situací je třeba přestávka na oběd a dočasné vypnutí přijímače z důvodu úspory baterií nebo náhlý výpadek signálu. Tyto dvě různé GPS trasy pak bude zapotřebí spojit do jedné, seřazené podle časového razítka, pokud to bude možné.



Obrázek 4.2: Diagram tříd parsování GPS tras.

4.2.3 Diagram tříd

Diagram tříd představuje logický pohled na aplikační logiku. Diagram byl vygenerován z vývojového prostředí Microsoft Visual Studio, čímž je dosaženo, že diagram přesně odpovídá vytvořené aplikaci. Z diagramu lze také vyčíst hierarchickou strukturu dědění a získat tak bližší pohled na architekturu vyvíjené aplikace. Kompletní diagram obsahuje velké množ-

ství tříd a při umístění do práce by jeho obsah nebyl čitelný, proto byl umístěn zvlášť jako příloha.

4.2.4 Synchronizace

Synchronizace je velmi důležitou částí aplikace, a proto je jí třeba věnovat dostatečnou pozornost již při analýze a návrhu. Vstupním bodem synchronizace bude načtení záznamů GPS tras ze zadaných souborů. Zde již nebude rozlišováno, o jaký formát trasy se jedná, ale bude se pracovat pouze s kolekcí objektů třídy `GeoPoint`. Úkolem tedy je synchronizovat tyto zaznamenané body s fotografiemi podle data a času pořízení a zadaných parametrů. To umožní uživateli nalezení optimálního bodu v trase pro jednotlivé fotografie, kterým jsou následně přiřazeny geografické informace z nalezeného bodu.

Hlavní úloha z pohledu uživatele spočívá v nastavení správných parametrů, které povedou k přidání geografických informací k co nejvíce fotografiím. Prvním takovýmto parametrem je nastavení odchylky data pořízení fotografie od data zaznamenání bodu. Při zaznamenávání trasy GPS přijímačem jsou body ukládány v určitých intervalech, které jsou závislé na použitém přijímači nebo uživatelském nastavení. Ideálním případem by bylo zaznamenávání bodů každou vteřinu, ale to by se výrazně podepsalo na spotřebě přijímače a rychlém vybití baterií. Nastavený interval může být např. 10 sekund a v tu chvíli je potřeba nastavit vhodnou odchylku, aby mohl být bod k fotografii přiřazen. Při nenastavení odchylky je totiž velmi malá šance, že dojde k přiřazení bodu, protože čas by se musel shodovat na sekundu přesně. Dalším parametrem je výběr data fotografie, ke kterému se bude hledat bod trasy. Logickou volbou je datum pořízení fotografie. Uživateli by však mělo být umožněno synchronizovat i podle jiného data, například datum vytvoření souboru fotografie nebo datum zaznamenání geografických souřadnic.

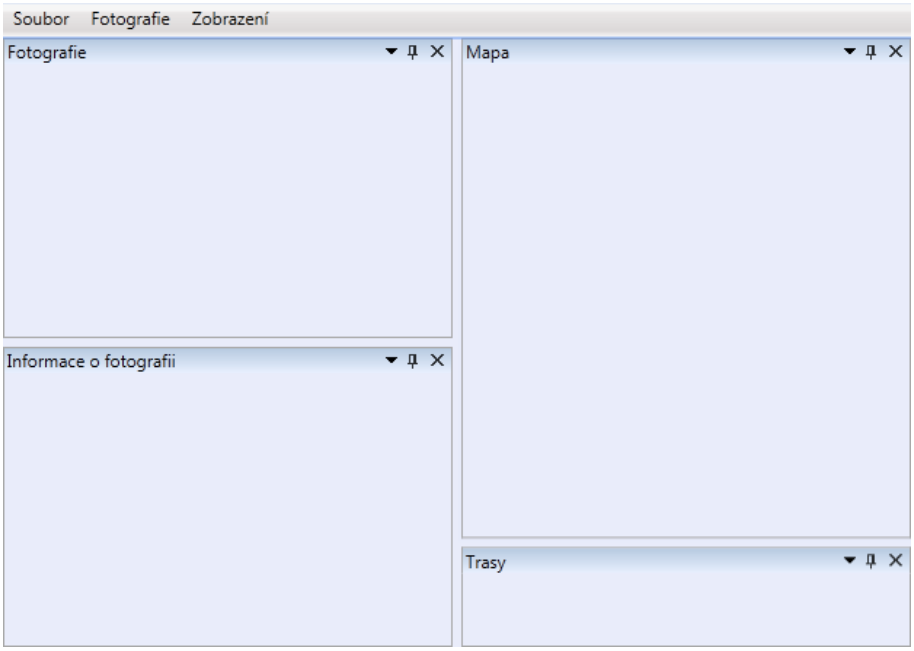
Běžnou věcí je, že datum z fotografie a bodu trasy se liší. Situace, kdy by tato data byla shodná a stačilo by pouze upravit odchylku, je výjimečná. Důvodů, proč se data liší, může být několik. Tím nejběžnějším bude pořízení fotografie v jiné časové zóně. Dalším důvodem může být nesprávně nastavený datum a čas ve fotoaparátu. Tento rozdíl je potřeba vyrovnat, aby mohlo dojít k synchronizaci. Uživateli by mělo být umožněno nastavit posun času a časovou zónu, kterými by došlo ke korekci času a data fotografie, aby byla synchronizace úspěšná. Při hledání bodu z trasy se nemůže do fotografie uložit první nalezený bod, který spadá do rozsahu odchylky od data z fotografie. Je zapotřebí hledat dále a do fotografie uložit až bod, který je datu z fotografie nejblíže.

4.2.5 Uživatelské rozhraní

Částí návrhu, která by neměla být opomenuta, je návrh uživatelského rozhraní. Při návrhu jsou hlavními cíli poskytnout uživateli aplikaci, která se bude lehce ovládat a bude přehledná. Návrhu uživatelského rozhraní se věnuje celý obor zvaný User Experience nebo pouze zkratkou UX. Při práci s fotografiemi uživatel potřebuje vidět velké množství informací, kterými jsou např. seznam načtených fotografií, informace o fotografii nebo třeba mapa s označenými body, kde byly fotografie pořízeny. Vhodnou technikou pro zobrazení těchto informací je tzv. MDI[1]. Díky ní jsme do okna aplikace schopni vložit další okna, které mají každé svůj účel. K vytváření uživatelského rozhraní je ideálním nástrojem již popsaný Microsoft Expression Blend. Bohužel framework WPF neposkytuje přímou podporu pro vytváření těchto MDI oken. Z toho důvodu byl využit ovládací prvek AvalonDock

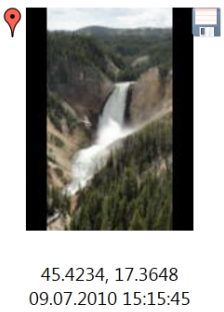
[1] Multiple Document Interface.

[11], který kromě MDI oken umožňuje i další operace s okny, jako je dokování oken či jejich náhledy z minimalizovaného stavu. Za použití tohoto ovládacího prvku bylo vytvoření rozhraní, jak je znázorněno na obrázku 4.3, otázkou několika řádků kódu v jazyce XAML. Díky použitému ovládacímu prvku si uživatel bude moci přizpůsobit aplikaci přesně svým potřebám.



Obrázek 4.3: Návrh uživatelského rozhraní aplikace.

Další částí návrhu, která si zaslouží bližší pozornost, je návrh zobrazení fotografie v albu. Zde si uživatel může procházet své fotografie a po jejím výběru si je nechat zobrazit na mapě nebo vypsat více informací z metadat. Proto by mu měl výpis poskytnout rychlý přehled o fotografii a jejích podstatných datech. Kompletní návrh lze provést pouze zápisem v jazyce XAML. Výsledkem je datová šablona pro fotografii, kterou lze naplnit ukázkovými daty. Takto není potřeba napsat jediný řádek kódu logiky a je možné zaměřit se pouze na grafickou část aplikace. Stěžejním bodem šablony z obrázku 4.4 je miniatura fotografie pro rychlé rozpoznání. Navrchu šablony jsou dvě ikony znázorňující, zda fotografie obsahuje zeměpisné souřadnice a zda byla fotografie po načtení upravena. Hlavním úkolem aplikace je práce se souřadnicemi a časem, proto jsou tyto důležité informace zobrazeny i pod miniaturou fotografie.



Obrázek 4.4: Návrh uživatelského rozhraní náhledu fotografie.

Kapitola 5

Implementace

Tato kapitola popisuje průběh implementace aplikace od navržení architektury, vytvoření základů pro návrhový vzor MVVM, až k implementaci jednotlivých částí aplikace. Konkrétně je popsán způsob využití programu ExifTool sloužící pro extrahování a ukládání metadat z fotografií. Následuje popis zobrazení metadat fotografií, dále třeba implementace parserů GPS tras, které byly využity při implementaci synchronizace.

5.1 Architektura aplikace

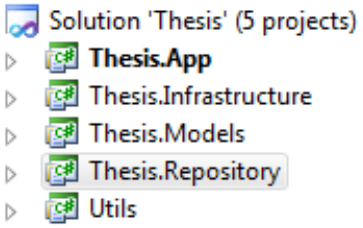
Rozvržení architektury aplikace je důležitou oblastí v počátku vývoje, která nám rozděluje aplikaci do jednotlivých vrstev obsahujících mezi sebou vazby určující závislosti. Ideálním řešením je mít minimum závislostí, to však v praxi není možné, ale díky správnému rozvržení je možné úroveň závislosti snížit. Výhoda v nízkém počtu vazeb se projeví například při výměně vrstvy, přehlednosti a udržovatelnosti kódu a s tím související refaktorizaci.

V případě této aplikace architektura vychází z již popsaného návrhového vzoru MVVM 3.3. Ten sám o sobě definuje prezentační vrstvu a vrstvu modelů. Prezentační vrstva obsahuje z návrhového vzoru části View a View-Model. V rámci umožnění přenositelnosti aplikace na jiné platformy byla přidána vrstva obsahující funkčnost a logiku aplikace a vrstva pro přístup k datům. Poslední vrstvou je vrstva poskytující obecnou funkcionalitu, která není závislá na konkrétní aplikaci a může být použita i v jiné aplikaci. Jednotlivé vrstvy jsou detailněji popsány níže.

Implementace architektury ve vývojovém prostředí Visual Studia spočívá ve vytvoření tzv. řešení[1] představujícího celou aplikaci. Řešení obsahuje jednotlivé projekty reprezentující vrstvy aplikace, mezi kterými jsou nastaveny vazby. Užitečným prvkem pro zvýšení přehlednosti je použití jmenných prostorů. Celá aplikace je uzavřena ve jmenném prostoru `Thesis`, který je dále dělen dle názvu vrstvy, jak ukazuje obrázek 5.1. Výjimkou je vrstva `Utils`, která nespadá do jmenného prostoru `Thesis`. Jmenný prostor vrstvy může být dále dělen do menších logických celků. Dělicím symbolem pro rozdělení jmenného prostoru je tečka.

Vrstva `App` je ve Visual Studiu aplikací WPF a může obsahovat různé speciální prvky jako formuláře, uživatelské a další prvky týkající se uživatelského rozhraní. Všechny ostatní vrstvy jsou typu `Class Library`, tudíž obsahují pouze jednoduché objekty jako jsou třídy, rozhraní, výčtové typy a podobné.

[1] Solution.



Obrázek 5.1: Rozdělení do vrstev ve vývojovém prostředí Visual Studio.

5.1.1 Utils

První popisovanou vrstvou je `Utils`, která není závislá na žádné jiné vrstvě a poskytuje svoji funkcionalitu ostatním vrstvám. Vrstva obsahuje převážně pomocné třídy pro návrhový vzor MVVM. Některé úkony nelze totiž provést přímo, ale je zapotřebí použití pomocných tříd či tříd poskytujících společnou funkcionalitu. Mezi takovéto pomocné třídy patří například různé konvertory, jejichž použití je velmi časté, ale ve standardní knihovně chybějí. Vyjmutí těchto tříd z prezentační vrstvy umožní jejich znovu použití v jiné aplikaci.

5.1.2 Models

Další vrstvou je `Models`, která již přímo vychází z návrhového vzoru MVVM. Vrstva obsahuje prosté deklarace aplikačních objektů a výčtových typů bez jakékoliv logiky. Mohla by být součástí vrstvy aplikační logiky `Infrastructure`, ale tyto objekty jsou zapotřebí ve vrstvě přístupu k datům `Repository` pro načítání a ukládání dat. Vazba mezi těmito vrstvami není možná z důvodu vzniku cyklické vazby, jenž ve Visual Studiu nelze vytvořit.

5.1.3 Repository

`Repository` je vrstva pro přístup k datům. Tato vrstva pracuje přímo s daným datovým úložištěm, kde plní aplikační objekty daty nebo data z aplikačních objektů ukládá na datové úložiště. V případě této aplikace je jediným datovým úložištěm pevný disk a tak se může zdát její použití zbytečné. Avšak je zde z důvodu logického oddělení části přístupu k datům a umožnění rozšiřitelnosti aplikace, kdyby bylo potřeba načítat data z jiného datového úložiště, kterým může být např. webová služba. V tom případě by stačilo pouze pozměnit tuto vrstvu se zachovaným rozhraním, na kterém jsou závislé vyšší vrstvy.

Hlavním úkolem této vrstvy je načítat metadata fotografií a také jejich uložení do fotografie. Tuto práci vykonává externí program ExifTool, jehož použití je popsáno v oddíle 5.3. Dalším jejím úkolem je načítání souborů GPS tras, které budou zpracovány ve vyšší vrstvě.

5.1.4 Infrastructure

Vrstva `Infrastructure` obsahuje aplikační logiku pracující s aplikačními objekty z vrstvy `Model` a poskytuje metody vrstvy `Repository` vyšším vrstvám, aby došlo k odstínění vrstvy přístupu k datům od prezentační vrstvy. Snahou této vrstvy je poskytnout co nejvíce aplikační logiky, která by ušetřila úsilí při přenosu aplikace na jinou platformu.

Největšími oblastmi, které aplikační logika obsahuje, jsou synchronizace fotografií s GPS trasou a třídy, které se starají o práci s GPS trasou. Sem patří konvertory GPS souřadnic

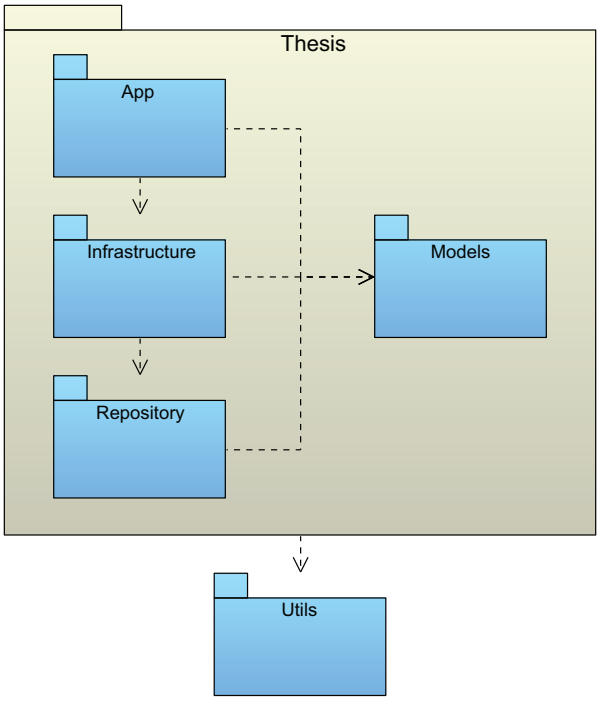
mezi jednotlivými formáty a parsery, které načítají trasy ze souborů v daném formátu do objektů představující GPS trasu. Tyto objekty jsou později využity právě pro synchronizaci.

5.1.5 App

`App` je prezentační vrstva této aplikace využívající framework WPF a jak již bylo řečeno, vychází z návrhového vzoru MVVM. Z tohoto vzoru jsou zde využívány oblasti View a View-Model. Popis těchto oblastí je důležitý pro pochopení implementace a proto je detailněji rozebrán v oddíle 5.2.

Kromě implementace různých View pro zobrazení fotografií, jejich metadat a synchronizace s GPS trasou je velkou oblastí také implementace mapy. Ta umožňuje zobrazení nejen fotografií, ale také GPS tras pomocí ukazatelů na mapě. Bližší pohled na implementaci mapy přináší oddíl 5.7.

Vazby mezi jednotlivými vrstvami popisuje obrázek 5.2.



Obrázek 5.2: Znázornění závislosti vrstev v aplikaci pomocí diagramu balíčků.

5.2 Implementace MVVM

Cílem této sekce je podrobně popsat způsob implementace aplikace s návrhovým vzorem MVVM, jež nám vytvoří potřebné základy pro snadnější a rychlejší vývoj. Konkrétně bude rozebráno, jak pracuje část View-Model, které rozhraní musí implementovat, a jak funguje propagace změny hodnot do View. U této části bude znázorněno, jak navázat proměnné a příkazy do View-Modelu. A nesmí být ani opomenut princip komunikace mezi View-Modely, jemuž je přikládán velký význam kvůli konceptu nezávislosti.

5.2.1 Implementace View-Modelu

Jak již bylo řečeno dříve, View-Model představuje kód v pozadí pro View, který na rozdíl od klasického kódu v pozadí není na View vůbec závislý. View-Model by měl být proto implementovaný tak, aby nejen poskytl vazbu mezi View a Modelem, ale také aby umožnil testovatelnost tohoto kódu. Striktní dodržování zásad tohoto vzoru může implementaci ztížit nebo v konkrétních situacích úplně znemožnit.

Úkolem View-Modelu je také zaobalit konkrétní Model a pokud to bude potřeba, tak poskytnout dodatečnou funkcionalitu, která je však úzce spjata s danou prezentační vrstvou. V opačném případě tato funkcionalita patří do aplikační logiky, kde je možné její opětovné použití. Jako názorný příklad může posloužit `TrackViewModel`, který přidává funkcionalitu pro získání cesty k souboru s trasou.

Každý View-Model musí implementovat rozhraní `INotifyPropertyChanged`. Výsledkem implementace rozhraní je deklarace delegáta `PropertyChangedEventHandler` na událost, která odešle informaci o změně dané vlastnosti do View. Dalším úkolem je vytvoření metody `OnPropertyChanged`, která s parametrem názvu vlastnosti vytvoří událost a vyvolá ji. Implementaci této metody lze spatřit ve výpisu 5.1. Do View můžeme informovat o změně buď jediné vlastnosti, kdy metodu zavoláme s názvem vlastnosti, nebo všech vlastností, pokud zavoláme metodu s parametrem `null`. Tato metoda musí být v View-Modelu ručně volána, kdykoliv si přejeme aktualizovat hodnotu ve View.

```
public event PropertyChangedEventHandler PropertyChanged;

public virtual void OnPropertyChanged(string propertyName)
{
    PropertyChangedEventHandler handler = this.PropertyChanged;
    if (handler != null)
    {
        var e = new PropertyChangedEventArgs(propertyName);
        handler(this, e);
    }
}
```

Výpis 5.1: Výchozí kód pro implementaci View-Modelu.

Taková implementace pro každý View-Model by byla zdlouhavá a v každém View-Modelu by se vyskytoval totožný kód. Proto je vhodné umístit tento společný kód do abstraktní třídy, z které budou všechny View-Modely dědit. Ještě je však třeba upozornit, že tento společný kód bude využit v některých třídách vrstvy `Model`, což umožní aktualizaci hodnot vybraných vlastností. S tím vyvstává otázka, zda tímto nevytváříme v této vrstvě vazbu na konkrétní prezentační vrstvu. Lze dospět k názoru, že nikoliv. Přímá vazba na prezentační vrstvu vytvořena není, pouze přibyla reference na součást .NET Frameworku, který musí být na počítači v každém případě. Nebude ani problém při přechodu na jinou platformu. Pokud by byl kód přenesen bez úprav, tak by se sice vykonávaly některé metody zbytečně, ale prováděné operace jsou nenáročné a výslednou funkčnost by to neovlivnilo. Abstraktní třída obsahující výše zmíněný kód se jmenuje `NotifyObject`. Z této třídy, pokud je to potřeba, dědí jednotlivé modely. Kdybychom však chtěli přidat společnou funkcionalitu specifickou pro View-Modely, tak toto by nebylo nejvhodnější místo. Z toho důvodu vznikla další abstraktní třída `ViewModelBase`, která dědí právě z `NotifyObject`. Tyto třídy nejsou závislé na konkrétní aplikaci, proto byly umístěny do vrstvy `Utils`, kde mohou být opětovně využity v jiné aplikaci.

5.2.2 Binding a Commands

Binding, neboli česky vázání, je způsob odkazování z View na vlastnosti View-Modelu. Jak již bylo zmíněno dříve, WPF umí obousměrný binding, takže hodnota zadaná ve View se zapíše do vlastnosti View-Modelu a obráceně za použití výše popsané metody na propagaci informace o změně hodnoty. Název vlastnosti z View-Modelu není ve View kompilátorem kontrolován, což ztěžuje implementaci, protože případná chyba je objevena až za běhu aplikace.

Popsaný postup, jak z View-Modelu vyvolat událost se změnou hodnoty, je vhodný pro jednoduché objekty. Pokud bychom chtěli použít kolekci hodnot, tak ideálním řešení je využít generickou kolekci `ObservableCollection`, která už sama implementuje rozhraní `INotifyPropertyChanged`. Díky tomu veškeré operace nad kolekcí jako přidávání či třeba mazání prvků jsou automaticky promítnuty do View.

Binding lze využít také k navázání příkazů, typicky tlačítek. Z pohledu View je binding totožný jako při vázání jakýchkoliv jiných hodnot, jen se váže na atribut `Command`. Hlavní změna se odehrává ve View-Modelu, kde vlastnost musí být rozhraní `ICommand` nebo třída, která ho implementuje. Z tohoto rozhraní jsou nejzajímavější metody `Execute` a `CanExecute`, kde první je volána po stisku tlačítka a měla by vykonat odpovídající operaci. Druhá metoda obsahuje podmínku, která pokud není splněna, tak tlačítko není možné potvrdit. Vhodné použití je pro validace. Pro zjednodušení práce a vyhnutí se opakování kódu byla vytvořena třída `RelayCommand`, která právě implementuje toto rozhraní. Konstruktor této třídy přijímá delegáta na funkci z View-Modelu, která provede operaci po stisku tlačítka. Pokud je to potřeba, tak druhým nepovinným parametrem je predikát, určující, za jakých podmínek má být tlačítko aktivní. Třída `RelayCommand` opět poskytuje obecnou funkčnost bez závislosti na konkrétní aplikaci a proto byla umístěna do vrstvy `Utils` [14].

5.2.3 Komunikace mezi View-Modely

Důležitou oblastí při implementaci návrhového vzoru MVVM je komunikace mezi View-Modely. Ty jsou izolovanými objekty a o ostatních View-Modelech neví. V některých případech je potřeba předat informaci z jednoho View-Modelu do druhého, což není přímo možné. Pro vyřešení tohoto problému byl využit návrhový vzor `Mediator`, který je implementovaný jako `Singleton`, takže k jeho instanci mají přístup všechny View-Modely. Pokud v nějakém View-Modelu chceme odeslat zprávu, zavoláme metodu `NotifyColleagues` s klíčem identifikující typ zprávy. Druhým parametrem může předat do přijímajícího View-Modelu hodnotu. Jako klíč jsou použity hodnoty z výčtového typu `ViewModelMessages`, který je definován ve vrstvě `App`. View-Model, který má přijímat zprávu, se musí pro odpovídající klíč zaregistrovat metodou `Register`, které předá ukazatel na funkci, jež se má vykonat. Třída `Mediator` byla opět umístěna do vrstvy `Utils` [1].

5.3 Fotografie a ExifTool

V této sekci se práce zaměřuje na implementaci tříd reprezentující fotografie v aplikaci včetně jejích metadat. Bude také představen program ExifTool, který byl využit k načítání metadat z fotografie a také k jejich zpětnému uložení. Po představení následuje řešení problémů, se kterými se bylo třeba vypořádat. Mezi ně patří hlavně optimalizace načítání, nastavení správného kódování znaků a reprezentace metadat.

5.3.1 Reprezentace fotografií

Výchozím prvkem při implementaci reprezentace fotografií se stalo rozhraní `IPhoto`. To definuje minimální požadované vlastnosti, které byly potřeba ve zbytku aplikace. Zde patří hlavně cesta k souboru s fotografií, název souboru, objekt reprezentující metadata fotografie a vlastnost určující zeměpisnou polohu pořízení fotografie. Poloha je definována třídou `GeoPoint`, jejíž návrh byl představen v oddíle 4.2.2. V aplikaci se pracuje pouze s tímto rozhraním, jež nám určuje minimální požadavky na vlastnosti fotografie. Konkrétní implementací rozhraní je třída `Photo`, která je zároveň potomkem třídy `NotifyObject`. Jedním z důvodů proč třída implementuje rozhraní namísto dědění z abstraktní třídy je, že v jazyce C# je možné dědit pouze z jedné třídy, ale implementovat lze více rozhraní. Tím, že jsme tedy implementovali rozhraní, jsme si ponechali prostor na případné dědění funkcionality z jiné třídy. Rozšíření ve třídě `Photo` spočívají v nastavení oprávnění k přístupu k jednotlivým vlastnostem, uchování původních dat fotografie, jejichž význam bude vysvětlen v sekci synchronizace, a využití funkcionality ze třídy `NotifyObject`. Díky ní se nám při změně některé hodnoty aktualizuje grafické rozhraní aplikace. To je potřeba např. v případě změny polohy pořízení fotografie, kdy se nám v aplikaci aktualizují souřadnice a změní pozice fotografie na mapě.

Veškerá metadata o fotografii jsou uložena ve třídě metadat, kterou v sobě obsahuje třída fotografie. Třída metadat `Metadata` obsahuje jak obecné informace, tak objekty tříd reprezentující metadata formátů EXIF a XMP. Formátu XMP nebyla v úvodu věnována velká pozornost, proto i zde je jeho popis velmi stručný. Pro účely této aplikace uchovává formát XMP důležitou vlastnost, a to informaci o časové zóně v době pořízení. Tato informace je jedinou vlastností třídy `MetadataXMP`, která reprezentuje tento formát. Umístění této vlastnosti do třídy nese výhodu v možnosti přidání dalších informacích z formátu XMP. Naproti tomu třída `MetadataEXIF`, reprezentující metadata formátu EXIF, obsahuje veškerá metadata, která určuje specifikace tohoto formátu.

5.3.2 Načítání fotografií

Načítání fotografií se provádí ve dvou krocích. Nejdříve se vytvoří instance fotografií pouze s cestou k souboru a až následně se začínají načítat metadata. Toto rozdělení má svůj důvod, pokud by uživatel načítal velké množství fotografií, tak by aplikace po dobu načítání přestala reagovat. Proto bylo každé načítání umístěno do vlastního vlákna. Pro načítání fotografií byla vytvořena třída `PhotosLoadingThread`, kde se vytvářejí instance fotografií a ve vlastním vlákně se následně volá načítání metadat. Po vytvoření instancí a spuštění vlákna načtení metadat je asynchronně volána metoda ve ViewModel, která zařídí zobrazení fotografií na mapě. Zde nastává problém, kdy aplikace při zobrazování přestane odpovídat. Je to způsobeno tím, že se vykreslují veškeré náhledy fotografií, i když nejsou zrovna zobrazeny. Při standardním použití komponenty pro vypsání fotografií je použita virtualizace, kdy jsou vykresleny pouze viditelné objekty. Avšak tato komponenta byla upravena, aby bylo možné zobrazovat fotografie v matici, čímž dochází k vypnutí virtualizace. V současné době řešení této situace neexistuje. Oficiální komponenta od firmy Microsoft chybí a komponenty vytvořené komunitou nejsou použitelné, protože mají problém s mazáním a s výkonností během postupného vykreslování dalších fotografií. O načítání metadat ve vlastním vlákně se stará třída `MetadataLoadingThread`. Po načtení je opět asynchronně volána metoda z ViewModelu, která zajistí vykreslení načtených metadat, zobrazení fotografií na mapě a ukončení signalizace o načítání.

5.3.3 Seznámení s ExifTool

ExifTool je knihovna pro čtení, zápis a editaci meta informací z velkého množství různých typů souborů. Kromě obrázků umožňuje pracovat s video a audio soubory, dokumenty a mnoho dalšími typy souborů. Knihovna dokáže pracovat s velkým množstvím formátů metadat, z nichž nejzajímavější jsou pro nás EXIF a XMP. Výstupní i vstupní metadata lze libovolně formátovat k dosažení co nejpřijatelnější podoby dat. Formátovat lze i celková struktura výpisu metadat. K dispozici je třeba formátování do CSV, XML, HTML, JSON a další. Další výhodou je, že knihovna je vícejazyčná a podporuje i český jazyk. Překlad se týká nejen hodnot metadat jako např. informace o blesku v době pořízení, ale také názvů tagů. Názvy tagů nejsou překládány vždy, záleží na konkrétním formátu výstupu. V době psaní této práce je aktuální verze 8.56, kterou aplikace také využívá.

Knihovna je napsána v programovacím jazyce Perl, díky čemuž je přenositelná mezi různými platformami. Pro použití na platformě Windows bez potřeby prostředí Perlu je k dispozici spustitelný soubor. S ním lze pracovat v prostředí příkazové řádky a ovládání se provádí pomocí parametrů při spuštění. Výchozím kódováním výstupu je UTF-8. Změna na jiné kódování se provádí přepínačem `charset`, viz dokumentace [5]. Při spuštění bez parametrů se vypíše kompletní dokumentace k programu.

Pro potřeby spouštění aplikací v prostředí příkazové řádky z prostředí .NET slouží třída `Process`. Nejdůležitější vlastností je název spustitelného souboru, který se má vykonat. Mezi další vlastnosti, které budeme potřebovat, patří parametry při spuštění, schování okna spouštěné aplikace, kódování a přesměrování vstupu a výstupu. Přesměrování vstupu je zapotřebí z důvodu omezení počtu znaků příkazu v příkazové řádce. Toto omezení záleží na verzi operačního systému a na globálních proměnných. Ve Windows XP a novějších je limit 8191 znaků a ve Windows 2000 nebo NT 4.0 je 2047 [9]. Tento limit může být překročen například při snaze načíst nebo uložit velké množství souborů. Výstup není nikterak omezen, ale pokud by nebyl přesměrován, tak bychom ho nemohli načíst v aplikaci a vypsal by se standardní cestou v příkazové řádce.

5.3.4 Optimalizace načítání metadat

V případě, kdy uživatel bude potřebovat načíst do aplikace velké množství fotografií, se naskýtá otázka, jak rychle lze metadata z fotografií přečíst. Nejpřímější cestou by bylo volání programu ExifTool pro každou fotografii. Tento způsob je však velmi neefektivní, protože program by se musel volat pro každou fotografii. Jelikož je knihovna napsána v Perlu, musí se při každém spuštění volat interpret, který vykonání operace také zpomaluje. ExifTool však umožňuje i dávkové zpracování, díky kterému je možné načíst velké množství souborů pouze jedním příkazem. Rychlost čtení lze dále zvýšit přepínači `fast` a `fast2`. Použití těchto přepínačů je však na úkor jiné funkcionality. Po nahlédnutí do dokumentace je patrné, že použití těchto přepínačů neovlivňuje data potřebná v této aplikaci, proto byly použity. Dalšího možného zrychlení lze dosáhnout zvolením správného výstupního formátu, který lze efektivně parsovat. Pro svou nenáročnost, pro kterou se třeba velmi často využívá ve webových aplikacích, byl vybrán formát JSON. Parsování výstupu je prováděno pomocí výkonné knihovny Json.NET [13]. Její použití je demonstrováno v oddíle 5.3.6.

5.3.5 Nastavení kódování

Během implementace byl objeven problém s kódováním, kdy nebylo možné načítat fotografie, které v cestě k souboru obsahují diakritiku. Problém spočívá v nemožnosti na-

stavení libovolného kódování standardního vstupu. Výchozí kódování je definováno regionálním nastavením v operačním systému. V případě České republiky je používána znaková sada pro reprezentaci středoevropských jazyků `Windows-1250`. Parametry předávané programu ExifTool musí být tedy ve výchozím kódování, jinak by speciální znaky nebyly správně interpretovány. U názvu souborů by se tak poškodila cesta k souboru a ExifTool by je nemohl načíst. V prostředí .NET získáme výchozí kódování následujícím způsobem `Encoding.Default`.

S tím souvisí problém při kódování výstupních dat. Kódování dat, která jsou předána na vstup, není během zpracování měněno. To znamená, že na výstupu jsou např. názvy souborů ve výchozím kódováním. Bohužel všechny ostatní údaje zůstávají v kódování `UTF-8`. To znamená, že data nelze načíst pouze jedním kódováním, protože text se v každém případě interpretuje jinak. Dle dokumentace ExifTool umožňuje měnit výstupní kódování, ale to zůstává pořád stejné. Byly provedeny různé pokusy ve spojení s nastavením kódování výstupu v prostředí příkazové řádky, ale výsledek zůstával pořád stejný. Nakonec bylo zvoleno řešení, kdy se data fotografií z výstupu nezískají přímo jako řetězec, ale čtou se ze standardního výstupu jako posloupnost bytů. Ty se následně přečtou jak ve výchozím kódování, tak i v kódování `UTF-8`. V datech ve výchozím kódování je nalezena pouze cesta k souboru, zbývající data budou získána z textu v kódování `UTF-8`.

5.3.6 Načtení metadat

Nyní bude popsán kompletní postup k získání metadat včetně řešení problémů, které byly popsány výše. Výchozím bodem je spuštění programu ExifTool a předání vhodných parametrů pro získání očekávaného výstupu. Výpis 5.2 ukazuje konkrétní příklad spuštění aplikace ExifTool.

```
Process oP = new Process();
oP.EnableRaisingEvents = false;
oP.StartInfo.CreateNoWindow = true;
oP.StartInfo.LoadUserProfile = false;
oP.StartInfo.RedirectStandardError = false;
oP.StartInfo.RedirectStandardOutput = true;
oP.StartInfo.RedirectStandardInput = true;
oP.StartInfo.StandardErrorEncoding = null;
oP.StartInfo.StandardOutputEncoding = Encoding.UTF8;
oP.StartInfo.UseShellExecute = false;
oP.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
oP.StartInfo.FileName = @"exiftool.exe";
oP.StartInfo.Arguments = "-fast2 -j -lang cs -common " +
    "-EXIF:GPSAltitudeRef# -EXIF:GPSLatitudeRef# " +
    "-EXIF:GPSLongitudeRef# -EXIF:all " +
    "-xmp:DateTimeOriginal -G -c \"%.6f\" -@ -";
oP.Start();
```

Výpis 5.2: Spuštění aplikace ExifTool.

Pro lepší pochopení je vhodné si některé významné parametry popsat. Zde patří vlastnosti `CreateNoWindow` a `WindowStyle` zajišťující, aby okno, ve kterém se spouští aplikace, zůstalo skryté. Je to vhodné pro zvýšení uživatelského komfortu, kdy uživatel nebude rušen blikajícím oknem. Další podstatné vlastnosti jsou `RedirectStandardOutput`

a `RedirectStandardInput` sloužící pro přesměrování standardního vstupu a výstupu a jejichž význam již byl popsán výše. Důležité je povšimnout si nastavení výstupního kódování na `UTF-8` tak, aby se shodovalo s výchozím kódováním výstupu z programu ExifTool. Posledními dvěma parametry jsou název spouštěného souboru a parametry pro spuštění. Jak je patrné, neudává se absolutní cesta k souboru, ale pouze jeho název. To předpokládá, že soubor s aplikací ExifTool je umístěn ve stejném adresáři jako hlavní aplikace. Význam jednotlivých parametrů, které se předávají aplikaci, popisuje tabulka 5.1.

Tabulka 5.1: Parametry pro načtení metadat.

Parametr	**Popis**
-fast2	Optimalizace způsobená vynecháním poznámek výrobců.
-j	Nastavení výstupního formátu JSON.
-lang cs	Nastavení jazyka pro překlad hodnot metadat.
-common	Vypiš obecná metadata jako cesta k souboru, velikost, ...
-EXIF:GPSAltitudeRef#	Vypiš referenční hodnotu nadmořské výšky jako hodnotu namísto slov.
-EXIF:GPSLatitudeRef#	Vypiš referenční hodnotu zeměpisné šířky jako hodnotu namísto slov.
-EXIF:GPSLongitudeRef#	Vypiš referenční hodnotu zeměpisné délky jako hodnotu namísto slov.
-EXIF:all	Vypiš všechna metadata formátu EXIF, která fotografie obsahuje.
-xmp:DateTimeOriginal	Vypiš datum pořízení z formátu XMP, které obsahuje časovou zónu.
-G	Přidá před název metadat název skupiny/formát metadat.
-c "%.6f"	Nastavení formátování zeměpisných souřadnic jako reálné číslo zaokrouhlené na 6 desetinných míst.
-@ -	Příkaz, aby program očekával na standardním vstupu další parametry. V tomto případě cesty k fotografiím.

Metadata, která si přejeme vypsat, můžeme definovat jednotlivě nebo můžeme nechat vypsat celou skupinu metadat. V našem případě vypisujeme tři metadata jednotlivě, jež mají navíc znak # definující vypnutí konverze při výpisu. Zde se např. u `GPSAltitudeRef#` pro kladnou nadmořskou výšku vypíše 0. Pokud by byla konverze vypnuta, vytisklo by se *Nadmořská výška (kladná hodnota)*. Obdobné je to i u ostatních referenčních hodnot. Práce pouze s jednoduchými hodnotami je poté v aplikaci jednodušší než s dlouhým řetězcem. ExifTool umožňuje také nastavit formátování výstupního tvaru data, čehož bylo v počátku využíváno. Ve výchozím tvaru jsou totiž jednotlivé číslice odděleny dvojtečkou. Problém nastal u data pořízení ve formátu XMP, které obsahuje informace o časové zóně. Pokud bychom zvolili vlastní formátovaní, tak není možné tuto informaci vypsat. Proto se nakonec v aplikaci očekává výchozí formátování.

Nyní je zapotřebí běžící aplikaci předat názvy souborů, jejichž metadata požadujeme. Z důvodu výše popsaného limitu délky příkazu, zapíšeme názvy souboru přímo na standardní vstup, jak je znázorněno ve výpisu 5.3. Za zmínku stojí poukázání na použití výchozího kódování, jehož význam byl vysvětlen v oddíle 5.3.5. ExifTool očekává každý název souboru na vlastním řádku.

```
byte[] data = Encoding.Default.
            GetBytes(String.Join(Environment.NewLine, fileNames));
oP.StandardInput.BaseStream.Write(data, 0, data.Length);
oP.StandardInput.BaseStream.Close();
```

Výpis 5.3: Přesměrování na standardní vstup.

Následující výpis 5.4 ukazuje způsob čtení posloupnosti bytů ze standardního výstupu. Po načtení všech dat je důležité počkat na ukončení aplikace, aby nezůstávaly otevřené. Nakonec se přečtená posloupnost bytů převede na dva různé řetězce dle použitého kódování. Význam tohoto převodu byl popsán v oddíle 5.3.5. Z těchto řetězců už se provádí parsování jednotlivých fotografií a jejich jednotlivých metadat.

```
byte[] buffer = new byte[32768];
MemoryStream ms = new MemoryStream();
while (true)
{
    int read = oP.StandardOutput.BaseStream.
                Read(buffer, 0, buffer.Length);
    if (read <= 0)
        break;
    ms.Write(buffer, 0, read);
}

oP.WaitForExit();

byte[] data = ms.ToArray();
```

Výpis 5.4: Čtení ze standardního výstupu.

Parsování je prováděno pomocí již zmíněné knihovny Json.NET. Získání formátovaných dat, ze kterých je prováděno parsování, v obou kódováních znázorňuje výpis 5.5. Po získání názvu souboru z řetězce ve výchozím kódováním je v metodě `GetFileName` kontrolováno, zda soubor na disku opravdu existuje. Pokud by totiž došlo k poškození názvu souboru, tak bychom měli načíst pouze fotografie, se kterými je možné pracovat. V metodě `ParsePhoto` už se čtou jednotlivé metadata. Z nich je vyparsována jejich skupina, název a hodnota, které se poté ukládají do příslušné proměnné. Při parsování metadat se ještě provádí kontrola shody názvů metadat. Toto je z důvodu, že ExifTool pojmenovává některá metadata jinak, než uvádí specifikace. Typickým příkladem je název `ImageWidth` ze specifikace EXIF, který má však ve výstupu z ExifTool název `ExifImageWidth`. Při parsování se prohledává seznam těchto rozdílných názvů, aby došlo ke správnému přiřazení hodnoty. Seznam nyní obsahuje pouze dva názvy metadat, ale v případě budoucího rozšíření je připraven na přidání dalších.

Během vývoje aplikace byly používány různé verze ExifToolu, ale až do verze 8.56 obsahovaly chyby, které znesnadňovaly práci. První používanou verzí byla 8.41, jež však obsahovala chybu, kdy nebylo možné načíst hodnotu referenční nadmořské výšky. Později byla aktualizována na novější verzi 8.55, kde již tato chyba byla opravena. Bohužel autor programu přidal vlastnost, kdy při použití znaku # pro konverzi se tento znak přidal i do názvu tagu. Tento problém se bohužel vyskytoval jen ve formátu JSON. Bylo sice možné upravit aplikaci, aby správně načítala data, ale bylo usouzeno, že problém je na straně ExifToolu, který pro různé formátování vypisuje nekonzistentní data. Proto padlo rozhodnutí

```
IPhoto photo;

JArray utf8Photos = JArray.Parse(Encoding.UTF8.GetString(data));
JArray defPhotos = JArray.Parse(Encoding.Default.GetString(data));

for (int i = 0; i < utf8Photos.Count; i++)
{
    JToken defaultPhoto = defPhotos[i];

    photo = GetPhoto(photos, GetFileName(defaultPhoto));

    if(photo != null)
        ParseMetadata(utf8Photos[i], photo);
}
```

Výpis 5.5: Parsování metadat fotografií.

řešit situaci na oficiálním fóru aplikace, kde proběhla komunikace přímo s autorem a ten chybu během pár dní opravil. Oprava je součástí verze 8.56 a její popis je také uveden v historii: *Reverted JSON output to pre-8.51 behaviour by removing '#' symbol from tag names when print conversion is disabled on a per-tag basis.*

5.3.7 Uložení metadat

Postup při ukládání metadat je velmi podobný jako při načítání. Opět se vytváří proces, který spustí aplikaci ExifTool. Jediným parametrem při spuštění je `-@ -`, po kterém bude aplikace očekávat zapsání příkazu na standardní vstup ve výchozím kódování. Příkaz se zapisuje přímo na vstupu ze stejného důvodu jako při načítání, a to kvůli limitu počtu znaků v příkazu. I při ukládání je možné uložit metadata k více fotografiím jedním příkazem. Slouží k tomu přepínač `-execute`, který umožňuje přidat další příkaz. Takto můžeme za sebe skládat příkazy pro všechny fotografie, jak je znázorněno ve výpisu 5.6.

```
String args = String.Empty;

foreach (IPhoto photo in photos)
{
    if(!String.IsNullOrEmpty(args))
        args += Environment.NewLine + "-execute" +
                Environment.NewLine;

    args += GetSaveArgument(photo, saveOriginal) + photo.Source;
    photo.IsModified = false;
}

ExecuteSave(args);
```

Výpis 5.6: Uložení metadat fotografií.

Důležitou věcí, kterou je třeba zmínit, je, že ve výchozím nastavení ExifTool při ukládání zálohuje původní fotografie ve formátu `(původní název souboru)_original`. Toto zálohování je z důvodu ochrany před ztrátou dat. Může se totiž při ukládání stát, že

se zápisem metadat poškodí struktura dat v souboru a fotografie již nepůjde zobrazit. Zálohování lze vypnout přepínačem `-overwrite_original`, ale je doporučeno zálohování ponechat a tyto soubory smazat až po ověření, že je vše v pořádku. Kompletní seznam parametrů, které se pro každou fotografii ukládají, obsahuje tabulka 5.2.

Tabulka 5.2: Parametry pro uložení metadat fotografie.

Parametr	**Popis**
-EXIF:GPSLatitude	Zeměpisná šířka.
-EXIF:GPSLatitudeRef#	Referenční hodnota zeměpisné šířky.
-EXIF:GPSLongitude	Zeměpisná délka.
-EXIF:GPSLongitudeRef#	Referenční hodnota zeměpisné délky.
-EXIF:GPSAltitude	Nadmořská výška.
-EXIF:GPSAltitudeRef#	Referenční hodnota nadmořské výšky.
-EXIF:GPSTimeStamp	Čas zaznamenání GPS souřadnic.
-EXIF:GPSDateStamp	Datum zaznamenání GPS souřadnic.
-EXIF:DateTimeOriginal	Datum a čas pořízení z formátu metadat EXIF.
-XMP:DateTimeOriginal	Datum a čas pořízení včetně informace o časové zóně z formátu metadat XMP.
-overwrite_original	Nastavení, zda přepsat původní soubor s fotografií.

Při zápisu metadat do fotografie se vždy ukládají datum a informace o poloze pořízení. Zvlášť se ukládá datum a čas zaznamenání GPS souřadnic. Tyto informace totiž nemusí být vyplněny vždy, kdy fotografie obsahuje zeměpisné souřadnice. Příkladem může být ruční přiřazení souřadnic z mapy. Použití posledního parametru o přepsání původní fotografie závisí na uživatelském nastavení v aplikaci.

5.4 Zobrazení metadat fotografie

Aplikace poskytuje uživateli nejen prosté načtení geolokačních informací a dat, ale také umožňuje zobrazit různé údaje o fotografii v době pořízení. Tyto informace jsou převážně z formátu metadat EXIF. Ke každé fotografii je tak zobrazeno, zda byl při pořízení použit blesk, nastavení clony, expozice a další. Formát EXIF obsahuje velké množství těchto informací, ve kterém se obtížněji hledají konkrétní informace. Proto byly vybrány základní metadata, která se k fotografii zobrazují ve výchozím stavu. Tato metadata popisuje tabulka 5.3. V tomto zobrazení uživatel získá rychle přehled o vlastnostech dané fotografie. Kromě základního zobrazení je možné také zobrazit veškerá metadata, kdyby uživatel rád viděl i jiné informace, které nejsou obsaženy v základním zobrazení.

Metadata je možné dále třídit do vybraných skupin. Třídění je výhodné pro lepší orientaci při zobrazení všech metadat. Třídit je možné například podle formátu metadat, což je v našem případě formát EXIF a obecné informace přímo popisující fotografii. Druhou skupinou pro třídění je podle skupiny vlastností. Například ve formátu EXIF jsou to jednotlivá IFD, která byla již popsána v části 2.2.2. Takto si uživatel může rozdělit jednotlivá metadata do skupin, čímž si ušetří práci hledáním konkrétní informace.

Informaci, která metadata patří do základního zobrazení nebo do jaké skupiny vlastností, je zapotřebí někde definovat. Zde bylo využito atributů, které lze přidávat k deklaracím vlastností, metod, tříd, aj. Ve výpisu 5.7 lze vidět použití atributů u deklarace vlastnosti

Tabulka 5.3: Metadata v základním zobrazení.

Tag	Popis
ImageWidth	Šířka fotografie v pixelech.
ImageHeight	Výška fotografie v pixelech.
Make	Výrobce fotoaparátu, kterým byla fotografie pořízena.
Model	Konkrétní model fotoaparátu.
DateTimeOriginal	Datum a čas pořízení fotografie.
ExposureTime	Expozice.
Flash	Informace o použití blesku.
WhiteBalance	Vyvážení bílé barvy.
GPSLatitudeRef	Referenční hodnota určující, zda jde o severní nebo jižní zeměpisnou délku.
GPSLatitude	Hodnota souřadnice zeměpisné délky.
GPSLongitudeRef	Referenční hodnota určující, zda jde o západní nebo východní zeměpisnou šířku.
GPSLongitude	Hodnota souřadnice zeměpisné šířky.
GPSAltitudeRef	Referenční hodnota určující, zda jde o hodnotu nadmořské výšky pod nebo nad hladinou moře.
GPSAltitude	Hodnota nadmořské výšky.
GPSTimeStamp	Čas zaznamenání zeměpisných souřadnic.
GPSDateStamp	Datum zaznamenání zeměpisných souřadnic.

metadat, konkrétně hodnoty zeměpisné délky ve třídě `MetadataEXIF` pro formát metadat EXIF. Atribut `Category` slouží pro přiřazení do dané skupiny vlastností. Druhý atribut `CommonTagAttribute` slouží pro definici, že tato informace má být zobrazena v základním zobrazení.

```
[Category("GPS")]
[CommonTagAttribute()]
public string GPSLatitude { get; set; }
```

Výpis 5.7: Použití atributů u deklarace metadat.

Při zobrazování a případném následném třídění se používá reflekce nad vlastnostmi metadat. Výchozím bodem je objekt `Metadata` dané fotografie, jehož vlastnosti se procházejí a podle parametrů se přidávají do seznamu metadat určených k zobrazení. Pro procházení objektů s metadaty formátů EXIF a XMP se využívá rekurze.

5.5 Konvertory souřadnic a parsery GPS tras

Tato sekce vychází z návrhu popsaného v oddíle 4.2.2. Bude zde popsán způsob konverze zeměpisných souřadnic a jejich využití v aplikaci. Dále bude také ukázána implementace parserů GPS tras. Tato část je podstatná pro další průběh implementace, protože načtené trasy jsou zapotřebí při synchronizaci.

5.5.1 Konvertory souřadnic

Převod souřadnic je možný dvěma způsoby. Prvním a jednodušším je převod hodnoty souřadnice z reálného čísla do řetězce. Druhým způsobem je převod souřadnic v textovém tvaru do reálného čísla. Zde je hlavní komplikací různý způsob zápisu souřadnic. Tyto konvertory jsou v aplikaci využívány na několika místech. Ať už je to při zobrazení souřadnic u fotografie nebo u jednotlivých bodů trasy, kde se daný formát souřadnic zobrazuje podle uživatelského nastavení. Další oblastí použití je také jejich využití při validaci vstupních dat při manuální editaci souřadnic. Při editaci je uživateli povoleno zadávat oba formáty souřadnic. Pro kontrolu správnosti vstupních dat se použijí konvertory k převodu souřadnic do opačného formátu. Pokud je tento převod úspěšný, jsou data platná.

Při převodu souřadnice z reálného čísla je zapotřebí z čísla získat stupně, minuty a sekundy, ze kterých se skládá výstupní řetězec. Také je zapotřebí získat referenční hodnotu souřadnice, aby se mohlo v řetězci definovat, zda jde o severní nebo jižní zeměpisnou délku či o západní nebo východní zeměpisnou šířku. Protože se jednotlivé souřadnice převádějí zvlášť, musí být při konverzi definováno, zda se jedná o zeměpisnou délku nebo šířku. Pro tuto definici byl vytvořen výčtový typ `GeoCoordinateType` obsahující hodnoty `Latitude` pro zeměpisnou délku a `Longitude` pro zeměpisnou šířku, který je vstupním parametrem při konverzi. Získání konkrétní referenční hodnoty se provádí na základě znaménka hodnoty souřadnice, jak ukazuje tabulka 5.4.

Tabulka 5.4: Definice získání referenční hodnoty.

Souřadnice	**Hodnota souřadnice**	**Referenční hodnota**
Zeměpisná délka	Kladná	Severní (N).
	Záporná	Jižní (S).
Zeměpisná šířka	Kladná	Východní (E).
	Záporná	Západní (W).

Výpočet dílčích hodnot souřadnic znázorňuje výpis 5.8. Vstupní proměnnou je `coord`, která představuje souřadnice jako reálné číslo. Další proměnné jsou zřejmé už svým názvem, tedy `deg` jsou stupně, `min` minuty a `sec` sekundy. Tyto vypočtené dílčí hodnoty se následně spolu s vypočtenou referenční hodnotou vloží do formátovaného řetězce, který je na posledním řádku v ukázce kódu. Aplikace umožňuje používat více formátů souřadnic a zde se použije pouze jiný formátovaný řetězec. Při vkládání hodnot do formátovaného řetězce se hodnoty převádějí na řetězec. Zde je problém se sekundami, protože při převodu se používá regionální nastavení a jako oddělovač desetinné části by se vložila čárka. Formát souřadnic je však mezinárodní a v tom případě se používá jako oddělovač tečka. Proto se převod na řetězec musí provést zvlášť a s parametrem pro použití univerzálního prostředí `CultureInfo.InvariantCulture`.

```
tmp = (int)Math.Round(coord * 3600);
deg = tmp / 3600;

min = Math.Abs(tmp % 3600) / 60;

sec = ((coord * 3600) % 3600) % 60;
```

Výpis 5.8: Výpočet dílčích hodnot souřadnic.

Jak již bylo řečeno, opačný postup převodu je komplikovanější. Je zde třeba brát v potaz různé formáty souřadnic, které jsou popsány v tabulce 2.1. Jedná se o všechny formáty kromě toho na 1. řádku, pro který slouží převod souřadnic jako reálné číslo, jež je popsán výše. Jelikož by manuální parsování těchto různých formátů bylo obtížné, byly k parsování využity regulární výrazy. Pomocí nich lze velmi jednoduše zkontrolovat, zda zadaný vstup odpovídá některému z formátů a zároveň lze získat vyparsované hodnoty. Nejobtížnější částí je tedy definice regulárních výrazů tak, aby odpovídaly vybraným formátům a zároveň byly flexibilní při různém počtu bílých znaků. Celkem bylo zapotřebí definovat 4 regulární výrazy, jejichž definice obsahuje výpis 5.9.

```
// např. N 20 nebo E20 30.12
"^\\s*(?<ref>N|S|E|W)\\s*(?<degree>\\d+)($||\\s*)" +
"(?<minutes>(\\d)+(\\.\\d+)?)?\\s*$"

// např. 20N nebo 20 30.127 E
"^\\s*(?<degree>\\d+)\\s*(?<minutes>(\\d)+(\\.\\d+)?)?\\s*" +
"(?<ref>N|S|E|W)\\s*$"

// např. N20° 10′ 12.123″
"^\\s*(?<ref>N|S|E|W)\\s*(?<degree>\\d+)\\s*°\\s*" +
"(?<minutes>\\d+)\\s*′\\s*(?<seconds>\\d+(\\.\\d+)?)\\s*\"\\s*$"

// např. 20° 10′ 12.123″ N
"^\\s*(?<degree>\\d+)\\s*°\\s*(?<minutes>\\d+)\\s*′\\s*" +
"(?<seconds>\\d+(\\.\\d+)?)\\s*\"\\s*(?<ref>N|S|E|W)\\s*$"
```

Výpis 5.9: Regulární výrazy pro různé formáty souřadnic.

V regulárních výrazech si lze povšimnout vzoru `(?<název>výraz)`, kde `název` je pojmenování regulárního výrazu v části `výraz`. Pomocí tohoto vzoru lze lehce získat řetězec, který splňuje daný regulární výraz. Tento vzor se používá k získání referenční hodnoty souřadnice a stupňů, které jsou v každém formátu povinné, ale také k získání minut a sekund. Při převodu se musí nejdříve zjistit, jestli a jakému regulárnímu výrazu dané souřadnice vyhovují. Proto se postupně procházejí všechny regulární výrazy a testuje se, zda souřadnice odpovídá výrazu. Tento kód znázorňuje výpis 5.10.

```
Regex Valid(String coord)
{
    Regex regex = null;

    foreach (String r in Regexs)
    {
        regex = new Regex(r, RegexOptions.Compiled);

        if (regex.IsMatch(coord))
            return regex;
    }
    return null;
}
```

Výpis 5.10: Kontrola, kterému regulárnímu výrazu odpovídá souřadnice.

V případě, že souřadnice neodpovídá žádnému regulárnímu výrazu, konvertor navrací hodnotu `Double.NaN` značící, že se nejedná o číslo. Za zmínku také stojí, že regulární výrazy jsou kompilované. Toto je nutné nastavit speciálním parametrem při vytváření objektu regulárního výrazu. Výhoda kompilovaných výrazů se projeví, pokud se konverze provádí velmi často. To může být např. případ u zobrazení trasy, která obsahuje velké množství bodů a nastavení zobrazení souřadnic ve stupních. Proměnná `Regexs` obsahuje regulární výrazy popsané výše.

Po nalezení odpovídajícího regulárního výrazu se díky použitým vzorům získají hodnoty ze souřadnic. Při zjišťování znaménka z referenční hodnoty se provádí opačný postup získávání na základě údajů v tabulce 5.4. Z vyparsovaných jednotlivých hodnot vypočteme již výslednou hodnotu souřadnice, kterou zaokrouhlíme na 4 desetinná místa. Výpočet ukazuje výpis 5.11. Po zaokrouhlení na 4 desetinná místa je souřadnice stále dostatečně přesná a zároveň je pro uživatele lépe čitelná a zapamatovatelná.

```
sign * Math.Round(degree + (minutes / 60) + (seconds / 3600), 4);
```

Výpis 5.11: Výpočet výsledné hodnoty souřadnice z dílčích hodnot.

5.5.2 Parsery tras

Jak již bylo řečeno v úvodu, implementace parserů vychází z návrhu popsaném v oddíle 4.2.2. Při implementaci se ovšem vyskytly některé problémy, se kterými nebylo v návrhu počítáno. Návrh však poskytuje pouze obecný pohled a jednotlivé implementační detaily se řeší až během implementace. Třída `TrackParser` stále poskytuje společnou funkcionalitu a deklaruje vlastnosti a metody pro třídy implementující parsery jednotlivých formátů. Důležitou vlastností je deklarace jmenného prostoru. Oba implementované formáty jsou aplikací jazyka XML, který právě použití tohoto prostoru umožňuje. Název jmenného prostoru je uložen v atributu `xmlns` v kořenovém prvku daného formátu. Název tohoto jmenného prostoru je zapotřebí k přístupu k jednotlivým prvkům. V názvu jmenného prostoru je patrný formát trasy a také verze specifikace. Body získané tímto parserem z tras jsou ukládány do objektů třídy `GeoPoint` a výsledkem parsování je kolekce těchto objektů.

Další podstatnou částí je parsování data. Tabulka 2.3 ukazuje možné formáty data, které se mohou v souborech s trasami vyskytnout. Jediným rozdílem je nepoužití informace o časové zóně, protože při zaznamenávání GPS souřadnic je čas ukládán v univerzálním časovém systému. Při parsování data se nejdříve zkouší vyparsovat datum za použití standardních formátů. Bohužel tyto standardní formáty nepokrývají všechny možné formáty, které se mohou v trasách objevit. Proto bylo zapotřebí definovat několik vlastních formátů, pomocí kterých jsme schopni datum a čas v jayzce C# vyparsovat. Tyto formáty ukazuje tabulka 5.5.

Jednotlivé implementace parserů formátů GPX a KML se liší hlavně v tom, jak parsují jednotlivé body z trasy. Společnou vlastností je použití `Linq2Xml`, pomocí něhož lehce nalezneme v dokumentu prvky, ze kterých už se parsují konkrétní hodnoty. U formátu KML se parsují prvky bodu `Placemark`, k němuž při parsování musí být přidán prefix se jmenným prostorem. Jednotlivé hodnoty v bodu se parsují z prvků, to je rozdíl oproti parsování formátu GPX, kde se hodnoty parsují z atributů prvku. Další odlišností u formátu GPX je, že jsou parsovány dva prvky obsahující bod. Jsou to prvky `wpt` pro manuálně uložený bod a `trkpt` pro bod ze zaznamenané trasy. Z obou formátu se parsují pouze body obsahující časové razítko. Pokud bod časový údaj neobsahuje, je ignorován, protože tyto

Tabulka 5.5: Definované formáty dat.

Formát data
yyyyMMddTHHmmss
yyyyMMddTHHmmssZ
yyyy-MM-ddTHHmmss
yyyy-MM-ddTHHmmssZ
yyyyMMddTHH:mm:ss
yyyyMMddTHH:mm:ssZ
yyyy-MM-ddTHH:mm:ss
yyyy-MM-ddTHH:mm:ssZ

body pro synchronizaci nemají význam. Body z obou formátů se uloží do stejné kolekce a poté jsou ve výsledné kolekci ze všech parsovaných souborů seřazeny vzestupně podle časového razítka.

5.6 Synchronizace

Při implementaci synchronizace budeme vycházet z analýzy popsané v oddíle 4.2.4. V této sekci budou postupně probrány čtyři části týkající se synchronizace. První a hlavní částí je popis samotného algoritmu synchronizace, za kterým následuje popis implementace parametrů pro synchronizaci. Dalšími částmi jsou vysvětlení interpolace polohy a ukázka seznámení s náhledem výsledků synchronizace.

5.6.1 Implementace synchronizace

Nejpodstatnější třídou pro synchronizaci je třída `PhotoSynchronization`. Vstupními parametry této třídy jsou kolekce fotografií, které budou synchronizovány, trasa s body pro synchronizaci a parametry uložené v objektu třídy `SyncSettings`, které budou popsány v oddíle 5.6.2. Výchozím bodem synchronizace je metoda `Synchronize`. Zde se postupně procházejí všechny fotografie určené k synchronizaci. Z každé fotografie se podle zadaných parametrů získá datum, podle kterého bude synchronizace prováděna, a toto datum se upraví podle nastaveného posunu času. Automatický výpočet posunu data nastává pouze v případě vypočtení posunu mezi bodem a fotografií. V případě ručního nastavení posunu je zapotřebí brát ještě v úvahu možnost nastavení časové zóny. Pro tuto manuální synchronizaci byla vytvořena třída `PhotoSynchronizationManual`, která je potomkem hlavní třídy pro synchronizaci. Jedinou funkcionalitou navíc je přidání hodnoty z časové zóny při ukládání posunu k datu.

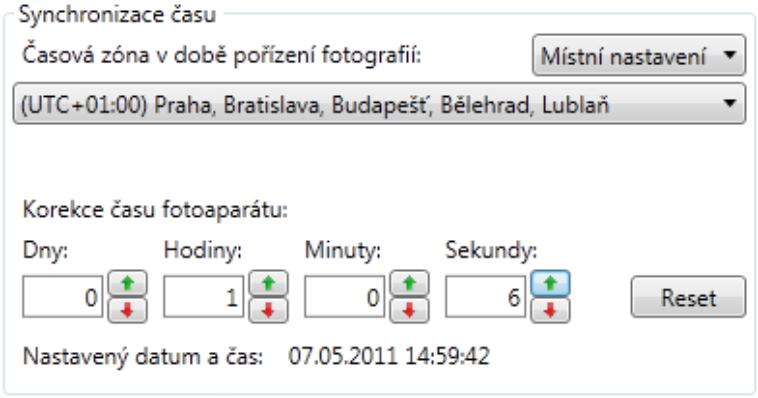
Při vybírání data z fotografie, podle kterého se bude synchronizovat, záleží také na parametru, zda se má datum brát jako relativní nebo absolutní. Při použití absolutního nastavení se načte vybraná hodnota data ve stavu v jakém byla načtena z fotografie. Při načítání fotografií se totiž všechny čtyři data zálohují do speciálního objektu právě pro tyto účely. Při relativním nastavení se použije aktuální hodnota data, která již mohla být od načtení upravena. Typickým případem tohoto využití může být například, pokud uživatel data některých fotografií editoval a při synchronizaci si přeje synchronizovat podle výchozích hodnot. Takto uživatel neztratí svou rozdělanou práci a může pohodlně synchronizovat.

Po získání hodnoty data nastává čas pro nalezení vhodného bodu z trasy. Při hledání se procházejí všechny body trasy a testuje se, zda byl bod zaznamenán v čase spadajícího do rozmezí odchylky od vypočteného data fotografie. Pokud bod nespadá do rozmezí, pokračuje se dalším bodem. V opačném případě se dále testuje zda je bod nejbližším k datu fotografie. Pokud se jedná o první nalezený bod je nastaven jako nejbližší a pokračuje se v hledání ještě bližšího. Jestliže je nalezen další bod, který spadá do rozmezí odchylky, porovná se z posledním nejbližším nalezeným bodem. Kontrola se provádí jako porovnání rozdílů absolutních hodnot. Takto lze porovnat body před datem i po datu z fotografie. Tímto hledáním nejbližšího bodu zajistíme, že do fotografie budou uloženy co nejpřesnější geografické souřadnice. Pokud žádný bod nebyl nalezen, stávající informace se z fotografie nemažou. Během synchronizace se ukládá seznam fotografií, které byly synchronizovány a tyto fotografie se poté předávají k výpočtu interpolace, pokud je její použití povoleno.

Při ukládání informací z nalezeného bodu do fotografie se využije extenzivní metody `SetPosition` ve třídě `PhotoExtensions`. Extenzivní metody slouží k rozšíření funkčnosti již vytvořených tříd. Pomocí této metody přidáme funkcionalitu třídě `Photo`, která samotná neobsahuje žádnou funkcionalitu. Informace z nalezeného bodu jsou ukládány do odpovídajících metadat, a to ve stejném formátu, ve kterém jsou načítána. To nám umožní jednodušší zpětné uložení metadat do fotografie.

5.6.2 Typy synchronizace

Uživatel si může zvolit mezi dvěma typy synchronizace, a to mezi manuální a z trasového bodu. Při manuální synchronizaci, jak ukazuje obrázek 5.3, si musí uživatel sám nastavit posun času. U nastavení časové zóny může uživatel využít přednastavení hodnoty buď podle místního nastavení, nebo z fotografie, pokud je ve fotografii uložena. Dále lze nastavit korekci času fotoaparátu pro případ, že byl během pořizování fotografií špatně nastaven. Pro okamžitou kontrolu je zobrazen aktuální čas včetně korekce.

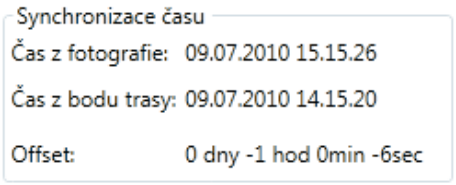


Obrázek 5.3: Nastavení manuální synchronizace.

Druhým způsobem je vypočtení posunu času z rozdílu mezi datem z fotografie a označeným bodem trasy v okně tras. Na obrázku 5.4 je vidět příklad takového výpočtu.

5.6.3 Interpolace polohy

V nastavení synchronizace lze zvolit použití interpolace polohy. Výsledkem interpolace je dopočítání polohy pořízení u těch fotografií, ke kterým nebyla nalezena poloha přímo v trase.



Obrázek 5.4: Nastavení synchronizace z bodu trasy.

Pro výpočet byla použita nejjednodušší lineární interpolace. K dosažení správného výpočtu je zapotřebí seznam fotografií, které byly synchronizovány, i ty, které synchronizovány nebyly, seřadit podle data pořízení. Následně se pro každou nesesynchronizovanou fotografii hledají pozice zaznamenané před a za touto fotografií. Pokud jsou tyto body nalezeny, vypočte se nová pozice. Fotografie s vypočteným bodem je poté doplněna do seznamu synchronizovaných fotografií a je využita k výpočtu pozice další fotografie. Při použití interpolace je třeba brát v úvahu, že přiřazená poloha je pouze orientační a může v některých situacích vést k nepřesným výsledkům.

5.6.4 Náhled výsledků

Náhled výsledků synchronizace na obrázku 5.5 slouží uživateli k nahlédnutí, jak synchronizace dopadne, ještě předtím než, se skutečně provede. Uživatel takto může zamezit některým nechtěným chybám či si jenom zkontrolovat správné nastavení parametrů.

Fotografie	Původní datum a čas	Datum a čas po korekci	Datum nejbližšího bodu	Odchylka
DSC01748.JPG	09.07.2010 15:12:42	09.07.2010 14:12:42	09.07.2010 14:15:00	00 dnů 00:02:18
DSC01752.JPG	09.07.2010 15:15:26	09.07.2010 14:15:26	09.07.2010 14:15:20	00 dnů 00:00:06
DSC02698.JPG	18.09.2010 08:11:38	18.09.2010 07:11:38	18.09.2010 07:11:38	00 dnů 00:00:00
DSC03006.JPG	20.09.2010 12:50:31	20.09.2010 11:50:31	19.09.2010 07:13:38	01 dnů 04:36:53
DSC03099.JPG	21.09.2010 18:36:43	21.09.2010 17:36:43	19.09.2010 07:13:38	02 dnů 10:23:05

Obrázek 5.5: Náhled výsledků synchronizace.

Řádky zvýrazněné oranžovou barvou znamenají, že tyto fotografie nebudou synchronizovány, protože k ním nebyl nalezen žádný bod, který by splňoval nastavené parametry. Ve sloupci odchylka je vypočten rozdíl od nejbližšího nalezeného bodu. Uživatel se tak může u zvýrazněných řádků rozhodnout, zda postačí zvýšit velikost odchylky nebo musí hledat problém na jiném místě.

Pro výpočet náhledu výsledků synchronizace byl využit stejný algoritmus jako u samotné synchronizace. Rozdílem však je, že se nejdříve neprovádí kontrola, zda je bod v rozsahu odchylky, ale ihned se hledá nejbližší bod. Takto jsme schopni určit odchylku i u fotografií, které by jinak nebyly synchronizovány. Pro zobrazení jednotlivých výsledků synchronizace byla vytvořena speciální třída `SyncResult`, kam se ke každé fotografii uloží očekávaný výsledek její synchronizace.

Tento náhled si může uživatel vyvolat sám nebo je automaticky spuštěn v případě potvrzení synchronizace, kdy nebudou všechny fotografie úspěšně synchronizovány. Zde bude pro kontrolu uživatel dodatečně dotázán, zda si přeje fotografie takto synchronizovat.

5.7 Práce s mapou GMap.NET

V této sekci bude čtenář seznámen s komponentou GMap.NET, která byla použita pro zobrazení fotografií na mapě. Dále budou vysvětleny jednotlivé ukazatele, které se mohou na mapě objevit a jakou mají funkčnost. Posledním bodem bude popis zobrazení tras na mapě.

5.7.1 GMap.NET

GMap.NET je open source knihovna pro Windows Forms a Windows Presentation Foundation. Díky dalšímu open source projektu Mono je dokonce přenositelná mezi více platformami. Mezi její hlavní výhody patří podpora velkého množství mapových podkladů. Programátor se při práci s knihovnou vůbec nemusí starat o mapové podklady, jejich překreslování řeší tato knihovna za něj. Mezi podporované mapové podklady patří jak známé Google Maps, Bing Maps, OpenStreetMap, tak i české Mapy.cz. Další užitečnou vlastností je ukládání mapových podkladů do mezipaměti pro snížení zátěže internetového připojení nebo také podpora vykreslování tras [12].

Bohužel, jako všechny open source projekty, má i tento některé své nevýhody. V době psaní této práce autor projektu musel řešit stížnost ze strany firmy Google. Této firmě se nelíbí, že tato knihovna přistupuje k mapovým dlaždicím přímo generováním odkazu a nevyužívá oficiální API. Výsledkem je provedení některých úprav a odstranění loga Googlu ze stránky knihovny ke splnění jejich podmínek. V současné době mapové podklady stále fungují, ale do budoucna je možné, že Google může přístup z této knihovny k jejich mapám zakázat.

5.7.2 Ukazatele na mapě

Všechny objekty, které jsou zobrazovány na mapě, musí být typu `GMapMarker` nebo jeho potomkem. Tyto objekty přijímá vlastnost mapy v kolekci `Markers`, ze které vykresluje jednotlivé objekty na mapu. Pozice objektu na mapě je určena vlastností `Position`, která je strukturou `PointLatLang`. Tato struktura představuje bod se zeměpisnou délkou a zeměpisnou šířkou. Všechny tyto struktury a třídy jsou definovány knihovnou a při implementaci je nutné je používat. Proto nemohl být použit již vytvořený objekt třídy `GeoPoint`. Důležitou vlastností je také `ZIndex`, který představuje z-ovou osu při vykreslování. Je zapotřebí ho u jednotlivých objektů vhodně zvolit, protože některé objekty by měly být vykresleny nad jinými.

Aby mapa věděla, jaký tvar ukazatele má vykreslit, je zapotřebí ho definovat a nastavit do ukazatele. Tvar ukazatele je očekáván ve vlastnosti `Shape` třídy `GMapMarker`. Objektem může být jakýkoliv prvek uživatelského rozhraní. Jednotlivé ukazatele na mapě mohou mít různé chování, ať už je to reakce po kliknutí nebo možnost přetáhnutí na jinou pozici. Abychom toto chování nemuseli implementovat pro každý ukazatel zvlášť, byla vytvořena základní třída `BaseMarker`, která obsahuje tuto společnou funkcionalitu. Tato třída kromě reakce na události myši poskytuje vlastnost `Icon`, do které je nastavován obrázek reprezentující ukazatel. Aby třída mohla poskytovat toto obecné chování, musí dědit z nějakého uživatelského prvku, v tomto případě ze třídy `UserControl`.

Ve třídě `BaseMarker` se provádějí pouze operace s uživatelským rozhraním, což neodporuje návrhovému vzoru MVVM. Abychom však mohli implementovat logiku pro ukazatele, je zapotřebí vytvořit nový ViewModel, a to `MapMarkerViewModel`. Tento ViewModel byl vytvořen, aby opět poskytoval společnou funkcionalitu pro všechny ukazatele. Již

v úvodu bylo řečeno, že několik View může mít jeden společný ViewModel pro vyhnutí se psaní opakujícího se kódu. ViewModel obsahuje například funkcionalitu pro přiřazení aktuální polohy do označené fotografie nebo vystavuje aktuální fotografii do View, kde může sloužit pro výpis informací o fotografii.

Na základech implementovaných ve třídách `BaseMarker` a `MapMarkerViewModel` byly vytvořeny celkem tři ukazatele. Tím nejzákladnějším je ukazatel pozice pořízení fotografie z alba, jehož ikona má červenou barvu. Pro zobrazení fotografie na odpovídající pozici na mapě je zapotřebí převést souřadnice z fotografie do struktury `PointLatLang`. Pro snazší převod se již při načítání nebo manuálním nastavení souřadnic ukládají souřadnice do vlastnosti `Position` ve fotografii. V té už jsou souřadnice vyparsované z hodnot načtených programem ExifTool do reálného čísla a převod do této struktury je již následně snadný. Dále se při zastavení kurzoru myši nad ukazatelem vypíší důležité informace o fotografii, jako jsou název souboru, datum pořízení a souřadnice. Pro získání více informací o fotografii lze na ukazatel kliknout, čímž se fotografie označí v albu a vypíší se její metadata. Dalším ukazatelem je ukazatel označené fotografie v albu, jehož ikona má modrou barvu. Ten poskytuje stejnou funkcionalitu jako předchozí, ale navíc dovoluje změnit polohu fotografie pouhým přetáhnutím ukazatele po mapě. Před uložením nové polohy je uživatel vyzván k potvrzení, aby nedošlo k nechtěným změnám polohy. Posledním ukazatelem je kurzorový ukazatel, jehož ikona má zelenou barvu. Tento ukazatel lze libovolně umístit na mapu pouhým kliknutím. Po umístění ukazatele na libovolnou pozici na mapě lze přes kontextovou nabídku nastavit vybranou polohu jedné či více fotografiím. Zobrazení všech těchto ukazatelů lze na mapě vypnout a nebo zapnout, pokud by to uživatel potřeboval ke zvýšení přehlednosti při práci s mapou. Ikony představující jednotlivé ukazatele lze vidět na obrázku 5.6.

Obrázek 5.6: Možnosti zobrazení ukazatelů na mapě.

5.7.3 Trasy na mapě

Kromě ukazatelů reprezentujících jediný bod na mapě lze na mapě zobrazit také trasu. Načtení a zobrazení trasy je pro uživatele přínosné v tom, že si před synchronizací může prohlédnout a zkontrolovat, zda se trasa zaznamenala správně. Pro vykreslení trasy se opět využívá třída `GMapMarker`, ale namísto vlastnosti `Shape` se použije vlastnost `Route`. Ta je kolekcí struktury `PointLatLang`, takže při načtení trasy je nutné opět provést konverze bodů do této struktury. Na rozdíl od synchronizace se z načtených souborů nevytvoří jediná trasa, ale každá trasa bude načtena zvlášť. Tímto uživatel získá větší přehled o zaznamenaných trasách, než kdyby byly všechny načtené do jediné.

Pro rychlý přechod na načtenou trasu umožňuje aplikace její přiblížení. Zde bylo nutné se vypořádat s tím, aby po přiblížení byla na obrazovce celá trasa a žádný kus nechyběl. To bylo možné pouze po nalezení minimální a maximální hodnoty pro zeměpisnou délku a zeměpisnou šířku, což lze provést velmi snadno za využití standardních metod pro nalezení minima a maxima generické kolekce `IEnumerable<T>` a lambda výrazů jako parametrů těchto metod. Z těchto hodnot lze vytvořit objekt struktury `RectLatLang`, která představuje pravoúhelník. Podle něj už knihovna umí nastavit správnou pozici na mapě a odpovídající přiblížení. Získání pravoúhelníku pokrývajícího trasu znázorňuje výpis 5.12.

```
latMin = track.Min(p => p.Point.Latitude);
latMax = track.Max(p => p.Point.Latitude);
lngMin = track.Min(p => p.Point.Longitude);
lngMax = track.Max(p => p.Point.Longitude);

MapControl.SetZoomToFitRect(
    RectLatLng.FromLTRB(lngMin, latMax, lngMax, latMin));
```

Výpis 5.12: Přiblížení na celou trasu.

5.8 Alba

Alba umožňují uživateli znovuotevření fotografií. To je výhodné například, pokud uživatel provedl synchronizaci a později by chtěl přátelům ukázat na mapě, kde byly fotografie pořízeny. Album v aplikaci reprezentuje třída `Album`, která obsahuje fotografie patřící do alba a název souboru, ve kterém je album uloženo. Nové album se vytváří automaticky při spuštění aplikace a uživatel do něj může okamžitě nahrávat fotografie. Při prvním vložení fotografie je poznamenáno, že došlo ke změně alba, a při opouštění aplikace je uživatel dotázán, zda si přeje album uložit, aby nedošlo ke ztrátě dat.

Album se ukládá do strukturovaného dokumentu XML, kde kořenovým prvkem je `album`. Poté již následují prvky `photo`, které reprezentují jednotlivé fotografie. Obsahem těchto prvků je cesta k fotografii. V současnosti se počítá pouze s fotografiemi uloženými na pevném disku. Pokud by byla aplikace do budoucna rozšířena o možnost načítání fotografií z webových služeb, mohl by tento zdroj být identifikován jako atribut prvku `photo`. Ukázka obsahující album v souboru XML je ve výpisu 5.13.

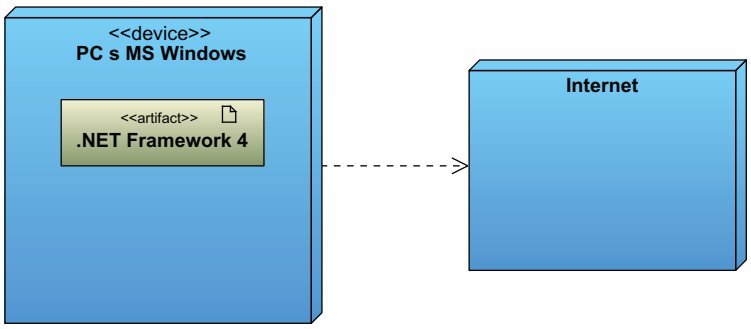
```
<?xml version="1.0"?>
<album>
  <photo>D:/fotky/DSC01457.JPG</photo>
  <photo>D:/fotky/DSC01458.JPG</photo>
</album>
```

Výpis 5.13: Album uložené v XML souboru.

5.9 Nasazení aplikace

Poslední částí v kapitole implementace je popis nasazení aplikace. Ta byla celá vyvíjena v .NET Frameworku 4, který je nutné mít nainstalovaný na počítači, kde bude aplikace spouštěna. .NET Framework verze 4 se ještě dělí na tzv Full Profile a Client Profile. Full Profile je kompletní balík, který je určen vývojářům a je nutný ke zkompilování aplikace. Client Profile je naopak pouze pro uživatele, kteří budou s aplikací pracovat. Instalační soubor je na rozdíl od Full Profilu velmi malý. Aby mohla být aplikace spuštěna pouze s verzí Client Profile, musí být kompilována v konfiguraci `Release`. Pokud by totiž byla kompilována s výchozí konfigurací `Debug`, tak by aplikace obsahovala ladící informace, se kterými umí pracovat pouze plná verze frameworku. Instalace .NET Frameworku 4 je možná pouze na operační systémy Microsoft Windows XP SP3 a novější. Pro práci s mapou je zapotřebí připojení k internetu pro stahování mapových podkladů. Připojení však není

nutné a jeho nedostupnost nijak nebrání uživateli k provedení synchronizace. Pro lepší představu je možné si prohlédnout obrázek 5.7.



Obrázek 5.7: Diagram nasazení.

Kapitola 6

Testování

Tato kapitola popisuje průběh testování synchronizace. Budou ukázány testovací fotografie a trasa, která byla během pořizování fotografií zaznamenána.

6.1 Získání testovacích dat

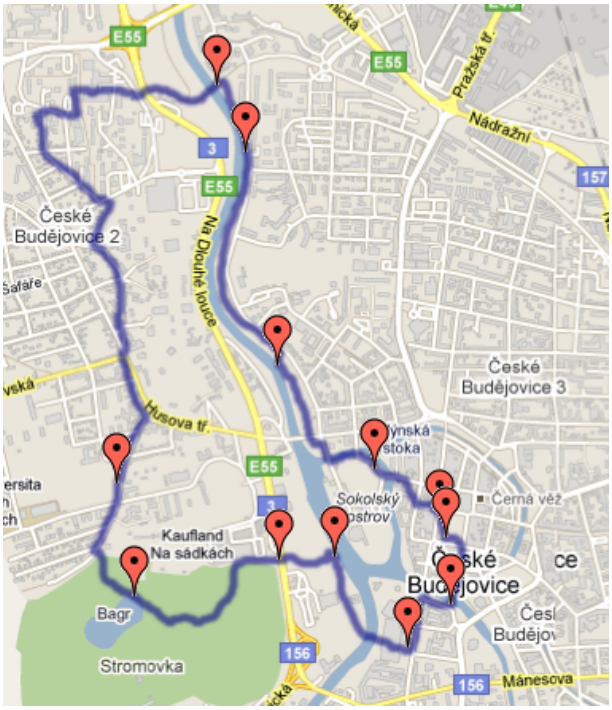
Aby bylo možné ověřit funkčnost aplikace, bylo nejdříve zapotřebí získat data, nad kterými bude testování prováděno. Rozhodl jsem se projet na kole mé rodné město České Budějovice a během cesty udělat několik fotografií. Pro záznam trasy jsem využil svůj mobilní telefon s integrovaným GPS přijímačem a potřebnou aplikaci pro snímaní polohy. Celkem bylo pořízeno 16 fotografií na různých místech, aby byl vidět výsledek synchronizace. Před samotným fotografováním proběhlo ověření, že je ve fotoaparátu nastavené správné časové pásmo. Tato testovací data jsou uložena na přiloženém CD v adresáři `testovaci data`. Přímo v tomto adresáři jsou uloženy zaznamenané trasy ve formátu GPX i KML s názvem `trasa`. Dále jsou v adresáři ještě podadresáře obsahující zdrojové fotografie a fotografie, nad kterými již byla provedena synchronizace.

6.2 Testování synchronizace

Před samotnou synchronizací byla nejdříve do aplikace načtena zaznamenaná trasa pro kontrolu, že byla uložena správně. Následně byly do aplikace načteny fotografie. Nyní máme jako uživatel dvě možnosti synchronizace. Tou jednodušší je vybrání jedné fotografie a následné vybrání bodu z trasy, u kterého víme, že zde byla fotografie pořízena. V případě těchto dat je to pro první fotografii bod v čase 11:40:24. Po vybraní fotografie a bodu můžeme spustit formulář synchronizace, kde je automaticky nastaven typ synchronizace `Z trasového bodu` a načtená trasa, která je zobrazená na mapě. Při tomto typu synchronizace se automaticky vypočte rozdíl času mezi těmito body. Kromě očekávaného posunu o jednu hodinu daným časovou zónou byl čas navíc posunut o dvě minuty a pět sekund. Tento posun navíc byl způsoben špatným nastavením času ve fotoaparátu. Před potvrzením synchronizace se můžeme ještě podívat na náhled výsledků k ověření, že ke všem fotografiím byl nalezen bod trasy.

Druhým možným typem synchronizace je manuální. U tohoto typu je nutné znát posun času, který se musí nastavit, aby byla synchronizace úspěšná. První volbou je nastavení časové zóny, která je přednastavena podle nastavení v počítači. Časovou zónu lze také načíst z vybrané fotografie. Zde však bylo zjištěno, že použitý fotoaparát informace o časové

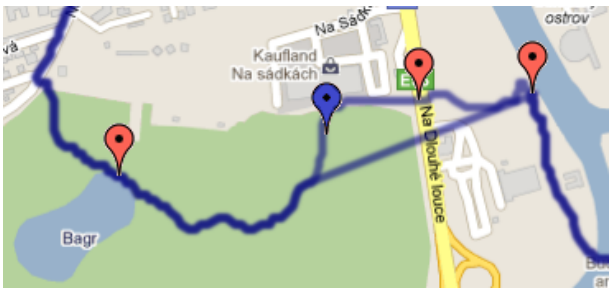
zóně do fotografie neukládá, bylo tedy ponecháno výchozí nastavení. Pro dosažení stejných výsledků byl nastaven posun času -2 minuty a -5 sekund. Dalšího nastavení nebylo třeba a výsledek synchronizace je vidět na obrázku 6.1.



Obrázek 6.1: Výsledek testování synchronizace.

6.3 Testování interpolace

Pro otestování interpolace byly z trasy odstraněny cca dvě minuty záznamu, během kterých byla zrovna pořízena fotografie. Tato trasa byla uložena do souboru `trasa_intepolace`. Opět byla provedena synchronizace, ale nyní se zatrženou volbu "Použít interpolaci polohy". Fotografie, jejíž poloha byla vypočtena, je na obrázku 6.2 označena modře. Na obrázku je také patrná chybějící část trasy. Pro znázornění, o kolik se vypočtená poloha liší, byla do alba načtena fotografie s přesně přiřazenou polohou. Jak je vidět, výpočet není přesný. To však ani není očekáváno, interpolace slouží pouze pro přibližné přiřazení. Přesnou polohu lze poté upravit ručně posunem na mapě nebo editací souřadnic.



Obrázek 6.2: Výsledek testování synchronizace s interpolací.

Kapitola 7

Závěr

Úkolem této diplomové práce bylo vytvořit aplikaci pro synchronizaci GPS tras s fotografiemi a umožnění jejich zobrazení na mapě. Před samotnou implementací bylo zapotřebí se nejdříve blíže seznámit s tématem práce. Bez nastudování informací o systému GPS a jím používaném časovém systému, způsobu uložení metadat ve fotografiích či formátu souřadnic a GPS tras, by úspěšnost implementace byla velmi mizivá. Všechny tyto informace jsou podstatné pro pochopení problematiky a následnou analýzu a návrh. V této části vývoje bylo důležité se podívat na aplikaci jako budoucí uživatel tak, aby aplikace umožňovala řešit jeho problémy a potřeby. Části návrhu bylo třeba věnovat patřičnou pozornost, protože čas ušetřený při návrhu se může vrátit během implementace, kdy po čase zjistíme, že řešení není možné a naše práce byla zbytečná. V počátku implementace bylo třeba věnovat velké úsilí správnému rozvržení architektury a implementaci bázových tříd, které později ušetřily spoustu času.

Během vývoje jsem se velmi často věnoval refaktorizaci ve snaze o snížení redundance kódu a jeho zpřehlednění. Bohužel to v některých případech vyústilo v nefunkčnost některých částí aplikace, které bylo následně nutné opravit. Tomuto by se zabránilo použitím jednotkových testů, ale toto téma je velmi rozsáhlé a pokrytí celé aplikace by bylo velmi náročné. Do budoucna mě to však poučilo, že čas vymezený ke psaní jednotkových testů se vyplatí. Velkou výzvou se během implementace také ukázala práce s programem ExifTool. Bylo zapotřebí řešit hlavně problémy s rychlostí načítání fotografií a jejich metadat. Dalším problémem, se kterým se bylo zapotřebí vypořádat, byl problém s diakritikou v cestě k souboru s fotografií.

Samostatnou zmínku si zaslouží také zkušenost s frameworkem WPF a návrhovým vzorem MVVM. Toto byla moje první zkušenost s těmito technologiemi a řešení některých problémů mi nepřišlo příliš vhodné. Například některé ovládací prvky ve WPF byly převzaty přímo z WinForms, což s sebou přineslo problémy, které musely být řešeny nestandardními cestami. U návrhového vzoru MVVM mi například přišlo volání metody pro propagaci změny vlastnosti zbytečné. V některých situacích to bylo při snaze o udržení zapouzdřenosti těžko řešitelné. Při hlubším pronikání do těchto technologií jsem postupně nacházel různé frameworky vytvořené komunitou, které plno problémů řeší a navíc přidávají funkce pro snazší vývoj. Tyto technologie mě však stále zajímají a rád bych se jim v budoucnu více věnoval. Budoucnost zde také vidím při pohledu na velkou aktivitu firmy Microsoft na nových verzích frameworku Silverlight, mobilní platformě Windows Phone, či herní platformě Xbox.

7.1 Možná rozšíření aplikace

Možných rozšíření aplikace může být několik. Asi nejméně náročným rozšířením by bylo přidání podpory i pro jiné formáty GPS tras. Výrobců GPS přijímačů existuje velké množství a ne všichni využívají právě tyto formáty, kterým byla v aplikaci věnována pozornost. Kromě vlastních formátů výrobců může být použit například i standardní CSV soubor. Jako další rozšíření vidím možnost načítat a synchronizovat fotografie z webové služby. V současnosti asi největšími službami pro ukládání fotografií jsou Google Picasa a Flickr. Obě tyto služby poskytují veřejné API, pomocí kterého je možné se službou komunikovat. Rozšíření by se také mohla dočkat práce s metadaty. Editace by se mohla týkat více metadat a zároveň by se mohlo využívat šablon pro hromadnou editaci. V současné době je aplikace pouze v českém jazyce, tak by bylo možné přeložit ji minimálně do anglického jazyka a využít tak možnosti překladu metadat z programu ExifTool.

Literatura

[1] BARBER, Sacha: *MVVM Mediator Pattern*. 2009, [online, cit. 25.4.2011].
URL http://www.codeproject.com/Articles/35277/MVVM-Mediator-Pattern.aspx

[2] D. KAPLAN, Elliott and HEGARTY, Christopher J.: *Understanding GPS, Principles and Applications, Second Edition*. Artech House, Inc., druhé vydání, 2006, ISBN 1-58053-894-0, 726 s.

[3] EL-RABBANY, Ahmed: *Introduction to GPS, The Global Positioning System*. Artech House, Inc., 2002, ISBN 1-58053-183-0, 196 s.

[4] FOSTER, Dan: *GPX: the GPS Exchange Format*. 2010, [online, cit. 13.12.2010].
URL http://www.topografix.com/gpx.asp

[5] HARVEY, Phil: *ExifTool by Phil Harvey*. 2011, [online, cit. 17.11.2010].
URL http://www.sno.phy.queensu.ca/~phil/exiftool/

[6] KANISOVÁ, Hana and MÜLLER, Miroslav: *UML srozumitelně*. Computer Press, 2006, ISBN 80-251-1083-4, 176 s.

[7] KRUCHTEN, Philippe: *The 4+1 View Model of Software Architecture*. 1995, [online, cit. 13.11.2010].

[8] KUHN, Markus: *A summary of the international standard date and time notation*. 2004, [online, cit. 17.1.2011].
URL http://www.cl.cam.ac.uk/~mgk25/iso-time.html

[9] Microsoft Corporation: *Command prompt (Cmd. exe) command-line string limitation*. 2007, [online, cit. 26.4.2011].
URL http://support.microsoft.com/kb/830473/en-us

[10] Open Geospatial Consortium Inc.: *OGC KML*. 2008, [online, cit. 13.12.2010].
URL http://www.opengeospatial.org/standards/kml

[11] Open Source: *AvalonDock*. 2011, [online, cit. 9.5.2011].
URL http://avalondock.codeplex.com/

[12] Open Source: *GMap.NET - Great Maps for Windows Forms & Presentation*. 2011, [online, cit. 9.5.2011].
URL http://greatmaps.codeplex.com/

[13] Open Source: *Json.NET*. 2011, [online, cit. 20.3.2011].
URL http://json.codeplex.com/

[14] SMITH, Josh: *WPF Apps With The Model-View-ViewModel Design Pattern*. 2009, [online, cit. 20.12.2010].
URL http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[15] Stock Artists Alliance: *Photo metadata*. 2009, [online, cit. 10.12.2010].
URL http://www.photometadata.org/

[16] Technical Standardization Comittee on AV & IT Storage Systems and Equipment: *Exchangeable image file format for digital still cameras: Exif Version 2.2*. 2002, [online, cit. 8.12.2010].
URL http://www.exif.org/Exif2-2.PDF

[17] Time Service Department: *LEAP SECONDS*. 2008, [online, cit. 10.11.2010].
URL http://tycho.usno.navy.mil/leapsec.html

[18] TROELSEN, Andrew: *Pro C# 2008 and the .NET 3.5 Platform*. Apress, 2007, ISBN 1-59059-884-9, 1370 s.

[19] WASHINGTON, Michael: *Silverlight MVVM - The Revolution Has Begun*. 2009, [online, cit. 7.1.2011].
URL http://openlightgroup.net/Blog/tabid/58/EntryId/84/Silverlight-MVVM-The-Revolution-Has-Begun.aspx

[20] WEISS-MALIK, Michael: *KML: A new standard for sharing maps*. 2008, [online, cit. 13.12.2010].
URL http://google-latlong.blogspot.com/2008/04/kml-new-standard-for-sharing-maps.html

[21] WILLIAMS, Brennon: *Introduction to Microsoft Expression Blend*. 2008, [online, cit. 20.5.2011].
URL http://www.itworld.com/opinion/53508/introduction-microsoft-expression-blend

[22] ZENDULKA, Jaroslav and BARTÍK, Vladimír and KVĚTOŇOVÁ, Šárka: *Analýza a návrh informačních systémů AIS*. Fakulta Informačních Technologií VUT Brno, 2006, 176 s.

Příloha A

Seznam příloh

Příloha 1. Diagram tříd – na přiloženém CD v `prilohy/classDiagram.jpg`.

Příloha B

Obsah CD

`xkodes00.pdf`	Elektronická verze této práce.
`dokumentace/`	Adresář s programátorskou dokumentací.
`latex/`	Adresář se zdrojovými soubory této práce.
`prilohy/`	Adresář obsahující přílohy.
`src/`	Adresář se zdrojovými kódy aplikace.
`src/App`	Obsahuje prezentační vrstvu.
`src/libs`	Obsahuje potřebné knihovny třetích stran.
`src/Logic`	Obsahuje vrstvu aplikační logiky.
`src/Models`	Obsahuje vrstvu modelů.
`src/Repository`	Obsahuje vrstvu přístupu k datům.
`src/Utils`	Obsahuje pomocné třídy.
`testovaci data/`	Adresář obsahující testovací data.
`testovaci data/fotografie`	Adresář obsahující neupravené fotografie k testování.
`testovaci data/fotografie s polohou`	Adresář obsahující fotografie po synchronizaci.