Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Získání a uchování metadat z heterogenních dat

Plzeň 2015 Martin Kryl

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 11. května 2015

Martin Kryl

Abstrakt

Práce řeší problém čtení metadat z obrazových souborů a jejich zápis do RDF modelu. Zaměřil jsem se na soubory typu JPEG, PNG a TIFF. Ke čtení metadat je využita knihovna Metadata Extractor. Mapování je definováno v ontologii. Je snaha o využití pojmů z existujících RDF slovníků. Pro případy, kdy není žádný vhodný pojem k dispozici, vytvářím vlastní slovník RDF vlastností. Výsledná ontologie mapuje 107 Exif makernote tagů a 207 tagů ostatních metadatových formátů na vlastnosti existujících ontologií. Implementace má formu pluginu pro program MetaMed.

Klíčová slova: RDF, metadata, obrazové soubory, ontologie, RDF slovník, JPEG, PNG, TIFF, Exif, XMP.

Abstract

This work tackles an issue of reading metadata from image files and adding them to RDF model. Image file types in the scope of the work are JPEG, PNG and TIFF. Metadata Extractor library is used for reading metadata from image files. Mapping definition is written in ontology file. Using existing vocabulary terms is preferred. In case no viable vocabulary term exists, a new property is defined in newly created ontology. Created mapping ontology has 107 defined mappings of Exif Makernote tags on existing terms and 207 mappings of other metadata tags. Extraction procedure is implemented as a plugin for MetaMed application.

Keywords: RDF, metadata, image files, ontology, RDF vocabulary, JPEG, PNG, TIFF, Exif, XMP.

Obsah

Úvod

V současné době existuje velké množství formátů souborů, do kterých lze zapisovat data. Datový soubor může kromě samotných dat obsahovat i popis dat, tzn. metadata. Přečtením a pochopením metadat lze získat informace užitečné pro další zpracování souborů, jejich analýzu nebo vyhledávání.

Struktura metadat je většinou definovaná pouze pro konkrétní formát souboru nebo malou množinu formátů. Jeden formát souboru může navíc obsahovat několik typů metadatových struktur. Metadata nacházející se v různých strukturách se mohou významově překrývat.

Cílem této práce je vytvořit plugin pro program MetaMed, který je vyvíjen na Katedře informatiky a výpočetní techniky v rámci výzkumné skupiny Medicínské informační systémy. Plugin umožní číst metadata z obrazových souborů JPEG, TIFF a PNG. Extrahovaná metadata budou zapsána do RDF modelu.

V souvislosti s tím navrhnu mapování obrazových metadat na RDF vlastnosti existujících ontologií. Pro metadata, která nebude možné vhodně přiřadit k existujícím vlastnostem, vytvořím vlastní RDF slovník.

1 RDF a ontologie

1.1 Resource Description Framework

Resource Description Framework, dále jen RDF, je v úvodu [1] představen jako framework pro popis vlastností určitého zdroje (resource). Zdrojem může být cokoliv – dokumenty, osoby, fyzické objekty případně i abstraktní koncepty. RDF je užitečné v situacích, kdy je potřeba, aby informace uložené na webu namísto člověka zpracoval stroj a porozuměl jim. Framework dále umožňuje výměnu dat mezi aplikacemi.

RDF specifikace spravuje World Wide Web Consortium (W3C). W3C je mezinárodní organizace pro vývoj a správu webových standardů, která byla založena v říjnu 1994. Organizace na svých webových stránkách [2] uvádí jako svoji misi vytváření protokolů a doporučení, které zajistí dlouho trvající růst webu a umožní využít celý jeho potenciál.

1.1.1 RDF trojice

Základní strukturou RDF je RDF statement (v češtině se lze setkat s překlady RDF tvrzení nebo prohlášení). Ten má formu trojice subjekt, predikát, objekt. Subjektem je popisovaný zdroj. Predikátem se rozumí vlastnost, kterou subjekt má. Objekt vyjadřuje hodnotu vlastnosti. K trojici se může přidat název grafu, v němž je obsažena, pak je tedy RDF statement tvořen čtveřicí hodnot (v angličtině RDF quad).

```
"Hamlet" - "je" - "tragédie".
"Hamlet" - "byl napsán" - "William Shakespearem".
"Hamlet" - "byl publikován" - "1603".
"Hamlet" - "je" - "princ".
```

Výpis 1.1: Trojice popisující dílo a postavu Hamlet v přirozeném jazyce.

Výpis 1.1 ukazuje, jak by bylo možné základní informace o díle Hamlet zapsat v přirozeném jazyce. Takový zápis je pro člověka čitelný, nicméně pro stroj je těžko uchopitelný. Hlavním problémem je rozlišení, kdy se Hamletem

rozumí název díla a kdy jde o postavu ve hře. RDF toto řeší použitím Internationalized Resource Identifier (IRI) definovaných v [3], což je určitá nadmnožina Uniform Resource Identifier (URI) používaných k jednoznačnému označení zdrojů na webu. Na subjekt je odkazováno jednoznačně a je strojově rozlišitelné, že se jedná o různé entity, i když jejich popisný název může být stejný. Na hru Hamlet může odkazovat například *http://lit.org/play/hamlet* a na postavu *http://lit.org/characters/hamlet*.

Pojem *je* v predikátu může nabývat více významů, vlastnost popisující žánr díla resp. roli osoby ve společnosti. Pro požadovanou vlastnost v predikátu se proto použije také IRI. V případě objektu je možné odkázat na zdroj pomocí IRI, nebo použít literál. Každý ze způsobů je vhodný pro jinou situaci. V tomto případě rok *1603* již pravděpodobně nebude vystupovat v jiné trojici jako subjekt a je tedy vhodné zapsat ho jako literál. Naopak na objekt *William Shakespeare* je lepší odkázat pomocí IRI jak z důvodu rozlišení mezi osobami stejného jména, tak umožnění dalšího popisu osoby.

Sada RDF trojic tvoří strukturu orientovaného grafu, kdy subjekt a objekt jsou vrcholy grafu a predikát je orientovaná hrana ze subjektu do objektu. Uzly mohou mít formu literálu nebo IRI, jak již bylo zmíněno. Kromě toho je možné jako uzel použít tak zvaný prázdný uzel (blank node) také občas nazývaný anonymní zdroj (anonymous resource), jak uvádí [4] v kapitole 3.1. Jedná se o uzel, který nemá přiřazený žádný globální identifikátor a v závislosti na zvoleném způsobu serializace je mu přiřazen místní identifikátor na úrovni grafu. Lei Chen [5] vymezuje jejich užití do následujících případů:

- Vyjmenování množiny prvků, jako například skupinu spoluautorů knihy.
- Zapsání komplexní datové struktury jako například adresu bydliště, která se skládá z názvu ulice, města a čísla domu. Prázdný uzel zde bude vystupovat jako objekt pro predikát *adresa* a zároveň bude subjektem pro predikáty *ulice*, *město* a *popisné číslo*.
- K zapsání kontextové informace pro RDF tvrzení jako den vytvoření nebo jméno autora.
- Může pomoci k popisu zdroje, jehož identifikátor je shodný, jako některá z jeho vlastností. Například u osoby jejíž IRI odpovídá IRI její emailové adresy.

Zápis stejných dat jako v předešlém příkladu do RDF je uveden ve výpisu 1.2. Při převedení tímto způsobem se ale ztratila část informace o názvu

díla a postavy. Není totiž obecně zaručeno, že IRI subjektu bude obsahovat to, co bychom chápali jako název popisovaného zdroje. Název je tak nutné zadefinovat další trojicí.

```
<http://lit.org/play/hamlet> <http://purl.org/dc/terms/type>
    <http://lit.org/terms/Tragedy> .
<http://lit.org/play/hamlet> <http://purl.org/dc/terms/creator>
    <http://lit.org/people/wiliam-shakespeare> .
<http://lit.org/play/hamlet> <http://purl.org/dc/terms/issued>
    "1603" .
<http://lit.org/character/hamlet> <http://lit.org/terms/role>
    "princ" .
```

Výpis 1.2: RDF trojice popisující dílo a postavu Hamlet.

V ukázce 1.2 byl použit formát N-Triples. Tento formát má sice jednoduchou syntaxi, ale na zapsání informace je potřeba relativně velké množství znaků. Navíc informace takto zapsaná není příliš dobře čitelná pro člověka. Existují i další formáty, které se liší čitelností nebo potřebným místem k zapsání dat. Dle mého názoru je jedním z nejčitelnějších a zároveň úsporných formátů Turtle, který budu používat dále v dokumentu. Formát umožňuje definovat na začátku souboru předpony (prefixy), kterými se budou substituovat části IRI se jmenným prostorem. Zápis ekvivalentní k 1.2 je uveden ve výpisu 1.3.

```
@prefix play:    <http://lit.org/play/> .
@prefix terms:   <http://lit.org/terms/> .
@prefix char:    <http://lit.org/character/> .
@prefix people:  <http://lit.org/people/> .
@prefix dc:   <http://purl.org/dc/terms/> .

play:hamlet dc:type terms:tragedy .
play:hamlet dc:creator people:wiliam-shakespeare.
play:hamlet dc:issued "1603".
char:hamlet terms:role "princ".
```

Výpis 1.3: Trojice popisující dílo a postavu Hamlet v Turtle syntaxi.

1.1.2 Linked data

V souvislosti s RDF modelem se často hovoří o konceptu linked data. Koncept prvně uvedl Tim Berners-Lee [6]. Koncept popisuje přístup, jaký je třeba

zvolit k publikování strukturovaných dat tak, aby byly lépe prozkoumatelné a využitelné. Důsledkem by mělo být, že data budou provázána a na základě známých dat bude možné najít jiná relevantní data. V dokumentu [6] Berners-Lee vymezuje čtyři podmínky k fungování a růstu sémantického webu:

1. Používání URI k pojmenování zdrojů.
2. Používání HTTP URI, aby šlo dohledat dané zdroje.
3. Když někdo vyhledá dané URI, poskytnout užitečné informace o zdroji za použití standardních technologií (RDF, SPARQL).
4. V odpovědi uvést i odkazy na jiné relevantní URI, aby bylo možné objevit další informace.

Vztah k RDF je patrný z prvního a třetího bodu. První bod odpovídá způsobu, jakým jsou identifikovány zdroje v RDF. Vyjímkou jsou prázdné uzly, které je možné převést na IRI jak uvádí specifikace [4] v kapitole 3.5. Třetím bodem je přímo jmenováno RDF jako preferovaný způsob podání informace a SPARQL jako jazyk pro dotazování se nad RDF modelem.

Čtvrtý bod a jeho aplikace je v další části dokumentu [6] blíže rozebrán. V důsledku jde opět o koncept, který jsem popsal v předešlé sekci, kdy v objektu RDF trojice je možné, často i chtěné, odkázat pomocí IRI na jiný zdroj, spíše než použít literál.

1.2 Ontologie a slovníky

Sémantický význam RDF trojice vychází z predikátu, který vyjadřuje vlastnost, jenž propojuje subjekt a objekt. V kapitole 1.1.1 popisuji, že na pozici predikátu musí být IRI. To samo o sobě k zajištění sémantické informace nestačí. Uvažujme vyjmenování hlavních postav díla Romeo a Julie trojicemi ve výpisu 1.4. V ukázce používám anglické termíny *hero* a *heroine* pro hlavní mužskou, resp. ženskou postavu. Bez další definice z těchto trojic nedokáže stroj pochopit žádný hlubší význam. Z jeho pohledu jde o neurčité vlastnosti *hero* a *heroine*, které propojují neurčitý zdroj *romeo-and-juliet* se zdroji *romeo* a *juliet*. Aby stroj pochopil, že obě vlastnosti jsou si blízké a liší se pouze v pohlaví osoby, které je vlastností implikována, je nutné vlastnost blíže definovat pomocí ontologie či slovníku. Ontologie je definována jako sada formálních jednoznačných tvrzení popisující určitou část světa [7].

```
@prefix play:  <http://lit.org/play/> .
@prefix terms: <http://lit.org/terms/> .
@prefix char:  <http://lit.org/character/> .

play:romeo-and-juliet terms:hero char:romeo .
play:romeo-and-juliet terms:heroine char:juliet .
```

Výpis 1.4: Trojice popisující hlavní postavy hry Romeo a Julie.

Další sémantický problém může vyvstat, pokud by jiná organizace chtěla popisovat pomocí RDF divadelní hry a pro své potřeby si nadefinovala svoji vlastnost *http://lit.cz/rdf/hlavniPostava*, která by byla pro postavy obou pohlaví. V rámci konceptu linked data by bylo žádané modely obou organizací propojit. Nicméně při snaze o propojování vlastností *hero*, *heroine* a *hlavniPostava* by došlo ke konfliktu ontologické definice pojmů. Řešením by mohla být ontologie definující vztah těchto pojmů. Pokud by stejný postup měl být prováděn pro každou další podobnou organizaci, stane se takové řešení časově i finančně neúnosné. Z tohoto důvodu je vhodné použít v řešení již existující definované pojmy a vlastnosti, pokud se svojí definicí shodují s požadavky řešení. Pojem RDF slovníky (vocabularies) označuje sady pojmů, které jsou určeny k dalšímu použití v cizích ontologiích nebo aplikacích.

1.2.1 RDF schéma

RDF Schema (RDFS) [8], spravované W3C, je základní slovník k vytváření datových modelů a ontologií. Definované pojmy slovníku jsou umístěné na jmenných prostorech *http://www.w3.org/2000/01/rdf-schema#* standardně substituovaným jako *rdfs:* a *http://www.w3.org/1999/02/22-rdf-syntax-ns#* nahrazovaným prefixem *rdf:*. Pojmy se dělí na třídy a vlastnosti. Významné třídy definované v RDFS jsou:

- **rdfs:Resource** každý zdroj vystupující v RDF tvrzení je instancí této třídy.
- **rdfs:Class** každý zdroj, který je třídou, je instancí této třídy. Třídy *rdfs:Class* a *rdfs:Resource* jsou instancemi rdfs:Class.
- **rdfs:Literal** třída pro literály. Je podtřídou *rdfs:Resource*.
- **rdfs:Datatype** třída pro datové typy literálů. Každá instance třídy je podtřídou *rdfs:Literal*.

- **rdf:Property** třída pro RDF vlastnosti.

RDF vlastnosti jsou dle specifikace [4] relace mezi zdrojem v subjektu a zdrojem v objektu. Všechny vlastnosti jsou instancí *rdf:Property*. Významné definované vlastnosti v RDFS jsou:

- **rdfs:range** slouží k definici třídy nebo množiny tříd, jejíž instancí je zdroj *o* v trojici (s, p, o), když *p* je vlastnost, kterou pomocí *rdfs:range* upravujeme.
- **rdfs:domain** je používána obdobně jako *rdfs:range* a definuje třídu nebo množinu tříd, jejíž instancí je zdroj *s* na pozici subjektu.
- **rdf:type** k vyjádření, že popisovaný zdroj je instancí určité třídy.
- **rdfs:subClassOf** k vyjádření, že daná třída je podtřídou jiné. Všechny instance podtřídy jsou zároveň instancemi nadtřídy.
- **rdfs:subPropertyOf** k vyjádření, že daná vlastnost *p* je podřazená jiné vlastnosti *P*. Zdroje, které jsou v relaci *p*, jsou zároveň v relaci *P*.
- **rdfs:label, rdfs:comment** k vložení člověkem čitelného názvu zdroje, resp. jeho popisu. RDF umožňuje k textovým hodnotám přiřadit jazyk, kterým jsou zapsané. Díky tomu je možné udělat vícejazyčné popisky a dokumentaci.

Kromě těchto základních pojmů jsou definované i další, které mají specifické použití. Definované jsou třídy popisující množinu prvků *rdf:Bag*, *rdf:Seq* a *rdf:Alt*. Jejich zamýšlené užití je pro neseřazené množiny, množiny seřazené dle určité vlastnosti, resp. množiny z nichž má být vybrána a použita jedna alternativa.

Dále jsou definovány vlastnosti *rdf:Statement*, *rdf:subject*, *rdf:predicate* a *rdf:object*. Ty slouží k popisu vlastností RDF trojice. To lze využít, jak bylo již uvedeno v souvislosti s blank node v kapitole 1.1.1, například k zapsání času, kdy byla trojice vytvořena nebo kdo je jejím autorem.

RDF schema definuje také několik vlastností k podpoře konceptu linked data. Vlastnost *rdfs:seeAlso* slouží k odkazu na zdroj, který může poskytnou další související informace. Vlastností *rdfs:isDefinedBy* lze odkázat na definici zdroje, například RDF slovník.

1.2.2 Web Ontology Language

Web Ontology Language (OWL), tedy jazyk pro tvorbu webových ontologií, rozšiřuje možnosti jak zapisovat znalosti o vztazích mezi zdroji. OWL umí popisovat třídy, vlastnosti a jedince (individual), což jsou instance třídy definované na úrovni ontologie. Základní syntaxí pro OWL ontologii je RDF/XML. Ontologii vychází z RDF konceptu a lze zobrazit jako graf.

Jazyk je v současnosti ve verzi 2 (z 11. prosince 2012). Jednotlivé možnosti, které OWL nabízí, jsou popsané ve W3C specifikaci [7]. Pro ukázku uvedu použití ontologie pro řešení problému s harmonizací RDF modelů z kapitoly 1.2.

```
<owl:ObjectProperty rdf:about="&terms;hero">
  <rdfs:range rdf:resource="&terms;Hero"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="&terms;heroine">
  <rdfs:range rdf:resource="&terms;Heroine"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="&lit;hlavníPostava">
  <rdfs:range rdf:resource="&lit;HlavníPostava"/>
</owl:ObjectProperty>
```

Ontologie 1.5: Část ontologie definující obor hodnot vlastností popisující hlavní postavy.

Uvažujme definici tříd jako v ontologii 1.5. Ta říká pomocí vlastnosti *rdfs:range*, že objektem v trojici s predikáty *hero*, *heroine* a *hlavniPostava* jsou instance třídy *Hero*, *Heroine*, resp. *HlavniPostava*. Syntaxe umožňuje zkrátit IRI zdrojů použitím prefixu, který je v RDF/XML uvozeny znakem & a ukončeny středníkem. Nyní deklarujme, že třída *HlavniPostava* je ekvivalentní se sjednocením množin tříd *Hero* a *Heroine*. Zápis této deklarace je v ontologii 1.6.

```
<owl:Class rdf:about="&lit;HlavníPostava">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&terms;Hero"/>
        <owl:Class rdf:about="&terms;Heroine"/>
      </owl:unionOf>
```

```
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Ontologie 1.6: Část ontologie definující ekvivalentní vztah mezi třídou *HlavniPostava* a sloučením tříd *Hero* a *Heroine*.

Důsledkem zapsání takové znalosti do ontologie je, že stroj s patřičnou softwarovou výbavou dokáže pochopit spojitost mezi danými pojmy a na jejím základě je schopen vyvodit znalosti, které nejsou explicitně v RDF modelu uvedené. Ukázkou takové znalosti je fakt, že zdroj *romeo*, který vystupuje jako objekt vlastnosti *hero*, je instancí tříd *Hero* a *HlavniPostava*. Tato informace však v trojicích ve výpisu 1.4 není zmíněna, neboť není potřeba z důvodu užití ontologie. Software, který umožňuje sémantiku informací číst a vyvozovat znalosti, se označuje pojmem sémantický reasoner, nebo pouze reasoner. Jeho užitečnost stoupá s komplexitou popisovaných skutečností.

Kromě hledání implicitních znalostí je možné použít reasoner k ověření konzistence dat. Reasoner je nicméně limitován tím, že OWL funguje na předpokladu otevřeného světa (open world assumption). Ten říká, že tvrzení je nepravdivé, jen pokud je řečeno, že je nepravdivé. Naopak v případě uzavřeného světa je nepravdivé každé tvrzení, které není řečené jako pravdivé [9].

Definujme, že vlastnost *hero* má kardinalitu jedna, tzn. že každý zdroj může být spojen danou vlastností pouze s jedním objektem. Pak na výpisu 1.7 neshledá reasoner nic špatného, protože ji interpretuje následující logikou. Romeo je hlavní postavou hry. Julie je hlavní postava stejné hry. Hra může mít nejvýše jednu hlavní postavu. Julie je tedy stejný objekt jako Romeo, pouze s jiným IRI. Aby reasoner shledal v tomto případě data nekonzistentní musela by v modelu být explicitně formulována nějaká negativní tvrzení, nejjednodušeji například taková, kterými budou objekty *juliet* a *romeo* prohlášeny za odlišné. V realitě je však takový přístup nepoužitelný, protože by bylo potřeba k úplné definici formulovat tvrzení pro každou dvojici různých objektů. Úspornější by bylo definovat negativní vztahy na úrovni ontologické definice tříd a v datech pouze explicitně přiřadit objekt určitému typu. V tomto případě tedy formulovat, že *romeo* je instance třídy *Hero*, *juliet* je instance *Heroine* a v ontologii definovat, že třída *Hero* nemá průnik s třídou *Heroine*.

```
@prefix play:  <http://lit.org/play/> .
@prefix terms: <http://lit.org/terms/> .
@prefix char:  <http://lit.org/character/> .

play:romeo-and-juliet terms:hero char:romeo .
play:romeo-and-juliet terms:hero char:juliet .
```

Výpis 1.7: Trojice popisující hlavní postavy hry Romeo a Julie. Julie je deklarovaná vlastností *hero* namísto *heroine*.

V praxi je užitečné v ontologiích konstatovat vedle pozitivní vlastnosti i negativní, je-li to možné. Pokud je u vlastností ve slovníku definován jejich obor hodnot, může reasoner odhalit nesoulad typů literálů v datech a definice.

2 Obrazové soubory a metadata

Obrazové (grafické) soubory mohou být velmi bohaté na metadata. Soubory, které byly vytvořené v grafickém programu, mají omezenou množinu metadat popisující základní vlastnosti jako šířku a výšku obrazu nebo barevnou hloubku, případně popisující program, kterým byly vytvořeny. Na druhou stranu fotografie pořízené digitálním fotoaparátem obsahují navíc řadu údajů o okolních podmínkách při focení a nastavení fotoaparátu.

Existuje řada různých grafických formátů. Základní dělení je na formáty rastrové a vektorové. Vektorové formáty ukládají matematickým zápisem pozici a tvar jednotlivých čar, které jsou použity k vykreslení komplexních tvarů. Rastrové mají strukturu mřížky pixelů. Pro každý pixel je uložena hodnota barevných složek, z kterých je tvořena barva konkrétního pixelu. Vektorové je možné díky způsobu jejich serializace snadno škálovat do potřebné velikosti bez ztráty kvality. U rastrových obrazů dochází k redukci informace při zmenšení velikosti nebo k přidávání nových dat při zvětšování. V závislosti na použité interpolační metodě, obsahu snímku či poměru změny velikosti dojde k různým kvalitativním změnám. Změněný obraz se může jevit rozmazaně, objekty mohou být nevyhlazené nebo se objeví šum u hran přechodů mezi barvami. [10, 11]

Metoda ukládání barevné konfigurace každého pixelu je velmi náročná na úložný prostor. Uvažujeme-li barevný prostor RGB, tedy tři složky s hodnotami 0 až 255, pak zápis jednoho pixelu vyžaduje 3 byty. Jeden snímek z fotoaparátu s rozlišením 10 Mpx by tak měl velikost přibližně 30 MB. Z tohoto důvodu se přistupuje u většiny formátů k nějakému druhu komprese.

Kompresní metoda může být ztrátová (lossy) nebo bezztrátová (lossless). Bezztrátovou kompresí se nepřichází o žádnou obrazovou informaci. Hledá se nejefektivnější způsob, jak uchovat kvalitu a zmenšit velikost. Toho může být docíleno například identifikováním opakujícího se vzoru v datech a jeho substituování zkratkou o menší velikosti. Naopak ztrátová komprese připouští degradaci kvality dat. Taková komprese může ukládat obrazovou informaci s menším rozlišením a pro vykreslení původního obrazu používat statistické odhady hodnot.[12]

V práci se dále zabývám pouze rastrovými formáty, které obsahují oproti vektorovým větší množství metadat ke zpracování. Konkrétně uvažuji formáty JPEG, PNG a TIFF.

2.1 JPEG

JPEG je původem mezinárodní standard ISO/IEC 10918 [13] komprese dat v statických obrazech. Název vychází ze zkratky skupiny Joint Photographic Experts Group, jenž ho vytvořila. V standardu byly definovány dvě metody komprese – ztrátová, založená na diskrétní kosinové transformaci, a bezztrátová využívající predikční přístup. [14] Skupina vytvořila řadu dalších standardů pro specifické způsoby užití, jejichž dokumentaci je možné najít na webových stránkách skupiny[1]. Dále budu uvažovat pouze výše zmíněný standard.

Zkratka JPEG se v současnosti často používá i jako označení datového formátu, který využívá tyto kompresní metody. Základním formátem je JPEG Interchange Format (JIF) definovaný ve standardu [15] v příloze B. Další formáty vždy JIF rozšiřují a jsou s ním zpětně kompatibilní. Často používané formáty jsou JPEG/JFIF pro obrazová data na webu a JPEG/Exif pro fotografie z digitálních fotoaparátů [16].

2.1.1 JPEG Interchange Format

JIF se skládá z několika segmentů začínajících dvou bytovou kódovou značkou definovanou ve standardu [15], příloze B.1. Segmenty mají definované pevné pořadí. Názvy značek dle specifikace uvádím v závorce.

Struktura souboru je dle přílohy B.2.1 [15] následující: začátek obrazu (SOI), definice Huffmanových tabulek (DHT), definice kvantizačních tabulek (DQT), definice restart intervalu (DRI), hlavička rámce, segmenty s komprimovanými daty, konec obrazu (EOI). V hlavičce rámce je použita jedna z šestnácti značek pro začátek rámce (SOFx). Dle zvolené značky je definována použitá metoda komprese. Jednotlivé segmenty s daty se skládají ze začátku scanu (SOS) a dat následovanými restart kódem (RST).

Specifikace dále vyhrazuje šestnáct značek pro označení segmentů pro data jiných aplikací (APPx). Segmenty se zpravidla vyskytují hned za SOI segmentem a je možné vložit více různých APP segmentů. V těchto segmentech se ukládají metadata jednotlivých formátů rozšiřujících JIF.

[1] http://www.jpeg.org/index.html

2.1.2 JPEG File Interchange Format

Tento formát, používající zkratku JPEG/JFIF nebo pouze JFIF, zvolil v roce 1996 Internet Engineering Task Force ve své publikaci jako kódování JPEG souborů pro internetovou datovou komunikaci. V dokumentu byl dále zaveden pojem Multipurpose Internet Mail Extension (MIME), což je rozšíření původního emailového protokolu k umožnění zasílání jiných typů dat než ASCII text. MIME typ je textový hierarchický identifikátor typu obsahu s dvěma stupni hierarchie. Koncept MIME typů se později rozšířil a pracují s ním i další aplikace. Formátu JFIF byl přiřazen MIME typ *image/jpeg*. [17]

JFIF používá APP0 segment, který musí být hned za segmentem začátku obrazu. Specifikace silně doporučuje používat JPEG baseline jako způsob komprese obrazových dat, kvůli zachování maximální kompatibility se všemi aplikacemi používající JPEG. Na rozdíl od JIF je definován barevný prostor obrazu. Může být ve stupních šedi (Y) nebo s třemi barevnými složkami (Y, Cb, Cr). Formát dále uchovává informaci o rozlišení a umožňuje uložit v segmentu APP0 zmenšeninu původního obrazu jako náhled. [18]

2.1.3 Exchangeable Image File Format

Exif je metadatový formát, prvně publikovaný v roce 1998, momentálně ve verzi 2.3. Je definován pro obrazová data v JPEG a TIFF Baseline formátu. Exif může být použit i k popisu zvukových souborů formátu RIFF WAVE Form. V případě JPEG formátu využívá k zápisu svých dat segmenty APP1 a APP2. Segment APP1 musí být hned za SOI. [19] Tímto požadavkem se Exif stává nekompatibilní s JFIF, protože oba vyžadují umístit svůj blok na začátek. V některých souborech se i přesto nachází oba segmenty a je na aplikaci, zda-li a jak si s danou situací poradí.

Metadata jsou roztříděna do několika adresářů (image file directories, IFD). Jednotlivé vlastnosti v adresářích se označují jako tagy. Exif IFD obsahuje údaje o době pořízení snímku, použitém objektivu, vlastnostech dat jako použitý barevný prostor a gamma. Dále jsou zde informace o nastavení fotoaparátu během pořizování snímku. Je zde také prostor k zapsání Makernote.

Makernote, v češtině tedy něco jako poznámka od výrobce, je tag určený pro výrobce zařízení k zapsání libovolné informace. Obsah je plně na uvá-

žení výrobce. [19] Výrobci většinou používají vlastní proprietární formát ke kterému nezveřejňují technickou dokumentaci. Některé formáty se podařilo částečně popsat pomocí reverzního inženýrství. [20]

Dalším adresářem je GPS IFD, který obsahuje data o místě, kde byl snímek pořízen. Původně bylo zamýšleno data získávat výhradně z GPS přijímače zabudovaném v zařízení, nicméně verze 2.3 specifikace již připouští i získání lokační informace z mobilních telefonů nebo pomocí údajů z blízké bezdrátové lokální sítě (WLAN).[19]

Jelikož je na trhu řada výrobců digitálních fotoaparátů, je snaha zajistit interoperabilitu mezi zařízeními různých výrobců. Soubor pořízený na libovolném zařízení by měl být zobrazitelný i na zařízeních jiných výrobců. Vzniklo několik doporučení, jejichž dodržením by měla být interoperabilita zaručena. Adresář Interoperability IFD slouží k určení, jaké sadě doporučení obrazový soubor vyhovuje.

2.1.4 Extensible Metadata Platform

Extensible metadata platform (XMP) je standardem ISO 16684-1:2012 pro vytváření, zpracování a předávání metadat u digitálních dokumentů. XMP byla vytvořeno společností Adobe Systems Inc. U JPEG formátu jsou data uložena v APP1 segmentu, stejně jako data Exifu, ale segment může být umístěn na libovolném místě před SOF. [21] Segmenty jsou od sebe rozlišitelné podle své signatury, která následuje za kódem APP1. XMP je tedy kompatibilní s Exif i JFIF.

Ve specifikaci [22] je popsán datový model, který se shoduje s RDF modelem. Dokumenty [22, 23] definují RDF slovníky týkající se multimediálních zdrojů. Ve slovníku jsou definovány pouze požadované datové typy. Žádná ontologie nad slovníkem není zveřejněna.

Ze své podstaty může být množina metadat zapisovatelných do XMP neomezeně rozšiřována definováním a použitím dalších RDF slovníků. V praxi je ale uživatel limitován aplikací, jenž je použita ke čtení a zápisu informace, která nemusí externí slovníky podporovat. Pro zachování maximální interoperability je proto vhodné využívat slovník definovaný v XMP specifikaci.

2.1.5 Information Interchange Model

International Press Telecommunications Council (IPTC)[2] je organizace založená v Londýně roku 1965, která sdružuje světové zpravodajské organizace. Byla vytvořena za účelem zjednodušení předávání informací a ustanovení technických standardů zefektivňujícím výměnu informace. V jednom ze standardů se zabývá i metadaty v obrazových souborech. Jde o standard Information Interchange Model (IIM), momentálně ve verzi 4.2 [24].

Ve standardu jsou definována metadata popisující obrazový soubor ve smyslu jeho zapouzdření při posílání, do kterých se zapisuje například datum zaslání souboru, použité kódování, adresát nebo formát souboru. V druhé části jsou definována metadata popisující obsah daného souboru. Zde jsou informace o datu a času pořízení, autorských právech, název, popisek obrazu a další. Je zde i prostor pro zapsání textové informace o státu, městu a objektu, který je na fotografii zobrazen. Položky kategorizující kontext obrazové informace jsou vymezeny kontrolovaným slovníkem. To znamená, že do daného pole se nevypisuje hodnota textově (např. „Politika“), ale kódem z pevně definovaného číselníku hodnot (11000000). Číselníky jsou definované v příloze specifikace.

Metadata jsou v JPEG souboru umístěna do segmentu APP13. Segment může být umístěn na libovolném místě před SOF dle kapitoly 1.1.3 dokumentu [21]. Není známo, že by IIM byl nekompatibilní s jinými metadatovými formáty vkládanými do JPEG souboru.

V důsledku rozšíření XMP byl vydán organizací IPTC nový standard, který popisuje způsob zápisu IIM metadat do XMP bloku. [25] V současné době je standard aktuální pro verzi 4.1 IIM, nikoliv tedy pro nejnovější 4.2. Standard popisuje mapování některých metadatových vlastností na již existující vlastnosti v XMP slovníku. Pro zbylé pojmy je vytvořen nový slovník. Ve slovnících je definován datový typ hodnot vlastností a kardinalita vlastností.

2.1.6 Photoshop Image Resources

Adobe Photoshop, program k úpravě a vytváření grafiky od společnosti Adobe Systems Inc., využívá řadu funkcí, jejichž parametry nelze zapsat do

[2]Domovská stránka IPTC: https://iptc.org

běžných rastrových obrazových souborů jako JPEG, PNG nebo TIFF. Jde například o možnost rozložit obraz do několika vrstev, které se navzájem překrývají, a výsledný obraz je funkcí jejich sjednocení. Při uložení do běžného souboru by se uložil pouze výsledný obraz a s jednotlivými vrstvami by nebylo možné již pracovat. Proto společnost vytvořila vlastní formát Photoshop Document, do kterého je program schopen zapsat vše potřebné.

Metadata v tomto formátu jsou uchovávána jako Photoshop Image Resources (PSIR). PSIR je možné použít i v jiných souborech. V JPEG souboru se nalézá v segmentu APP13 a slouží jako kontejner obsahující i jiná metadata jako například výše zmíněný IIM [26]. Specifikace PSIR [27] uvádí seznam definovaných tagů.

2.1.7 ICC profil

V roce 1993 bylo založeno International Color Consortium (ICC)[3], které mělo vytvořit otevřený a multiplatformní systém pro správu barevné konfigurace nezávislý na výrobci zařízení. Výsledkem byla specifikace ICC profilu [28]. Specifikace se stala mezinárodním standardem ISO 15076-1:2010.

ICC profil se používá k transformaci obrazu pořízeného na zařízení s určitou barevnou konfigurací pro zařízení používající jinou konfiguraci tak, aby pozorované barvy na snímku zůstaly zachovány. Profil bývá vkládaný do obrazových souborů. V případě JPEG se vkládá do segmentu APP2. Specifikace JPEG říká, že každý segment musí mít vyjádřenou bytovou délku pomocí 16 bitového integeru. Do délky se započítává i samotné vyjádření délky. Z toho plyne, že velikost dat segmentu může být nejvýše 65 533 bytů. Jelikož data profilu mohou být větší, byl zaveden způsob jak data rozdělit na několik částí. Každá část bude mít indikaci pořadí a celkového počtu částí a každá bude zapsána do jiného APP2 segmentu. Z definice plyne maximální velikost zapisovaného ICC profilu jako 16 707 345 bytů. [28]

[3]Domovská stránka ICC: http://www.color.org

2.2 PNG

Portable Network Graphics (PNG)[4] je grafický formát používající bezztrátovou kompresi. Formát umožňuje zapsat data obrazů v odstínech šedi, indexových barvách nebo True Color (24 bitová barevná hloubka s červeným, zeleným a modrým kanálem). U každé varianty je možnost nastavení průhlednosti. PNG byl vytvořen jako náhrada za formát GIF, u kterého došlo k patentovému sporu u kompresní metody. Jedná se o mezinárodní standard ISO/IEC 15948:2003. MIME typ souboru je *image/png*. [29]

Dokument specifikace [29] v úvodu vyjmenovává několik bodů jako hlavní cíle při vytváření standardu. Požadavek byl kladen na nezávislost procesu kódování, dekódování a přenosu na platformě nebo stroji. Celý formát měl být jednoduchý, aby vývojáři neměli problém standard implementovat, a dostatečně flexibilní, aby případné změny a rozšíření byly kompatibilní s dekodéry původní verze. Byla preferována rychlost vykonání procesů při dekódování a prezentaci na úkor rychlosti zakódování dat. Důležitou podmínkou bylo používání pouze volně dostupných algoritmů, aby nedošlo k dalšímu patentovému sporu.

Vývojové cíle byly naplněny a libovolný PNG dekodér vyhovující standardu je schopen dekódovat libovolný PNG soubor. Soubor je možné kódovat a dekódovat sériově, tedy bez znalosti kompletních dat, aby obrazová data z datového toku bylo možno průběžně zobrazovat. Prezentace obrazu je progresivní. To znamená, že se při zpracování nejprve zobrazí hrubá aproximace obrazu a s dalšími přicházejícími daty se obraz zkvalitňuje. Při přenosu jsou chyby detekovatelné pomocí kontrolního součtu na konci každé části.

2.2.1 Struktura souboru

Soubor se dle specifikace [29] skládá ze signatury a posloupnosti několika chunků. Chunk je část informace o obrazu, který má definovanou svoji délku, typ, může obsahovat datovou část a její kontrolní součet. Datová část je nepovinná, protože některé chunky mohou svoji funkci plnit bez ni. Například jde o IEND užívaný k označení konce souboru. Velikost datové části může být nejvýše $2^{31} - 1$ bytů.

[4]Domovská stránka PNG: http://www.libpng.org/pub/png/

Název chunku se skládá ze čtyř písmen, jejichž velikost určuje vlastnosti daného chunku. Velikost prvního písmena udává, jestli se jedná o chunk kritický, nebo pomocný (ancillary). Kritické chunky mají velké písmeno a musí je rozeznat a správně interpretovat každý dekodér. Pomocné dekodér nemusí dekódovat. Velikost druhého písmena udává, že se jedná o standardizovaný chunk, pokud je velké, nebo soukromý chunk, pokud je malé. Třetí písmeno je vždy velké. Čtvrté malé písmeno značí, že editor může bezpečně chunk zkopírovat, i když ho nedokázal rozpoznat. [29]

Standard [29] definuje šestnáct typů chunků. Čtyři z nich jsou označené jako kritické. Jedná se o IHDR obsahující hlavičku souboru, PLTE s definicí palety pro indexové barvy, IDAT obsahující data obrazu a IEND. Zbylých čtrnáct chunků je pomocných. Obsahují informaci o průhlednosti, nastavení barevného prostoru, času pořízení, barvě pozadí a rozlišení obrazu. V pomocných chuncích je také několik textových, do kterých lze zapisovat dvojice klíč-hodnota.

2.2.2 Další metadata

Do chunků lze vložit i metadata některých dříve zmíněných formátů. Specifikace XMP [21] připouští zápis do PNG chunku iTXt. Tento chunk se řadí mezi textové a je pro něj definováno kódování UTF-8. Specifikace umožňuje existenci pouze jednoho chunku s XMP informací v souboru a doporučuje ho umisťovat na začátek. Podobně lze zapsat i IIM metadata, pokud jsou převedeny do XMP modelu [25]. IIM jako takové však není ve formátu PNG podporováno a nelze jej jinak než převodem zapsat.

ICC profil lze v PNG zapsat do iCCP chunku. V případě, že enkodér zapisuje profil do souboru, doporučuje se zapsat do chunků gAMA a cHRM data získaná aproximací patřičných dat ICC profilu. Tím se zajistí kompatibilita s aplikacemi, které iCCP pomocný chunk nedokáží interpretovat. Z tohoto plyne doporučení pro vývojáře dekodérů, aby v případě nalezení a zpracování chunku iCCP ignorovali chunky gAMA a cHRM. [29]

V rámci projektu [30] byla snaha konvertovat Exif metadata do PNG. Bylo navrženo mapování některých Exif tagů na existující tagy v PNG a pro zbylé byl navržen nový chunk. Projekt ale zůstal ve stádiu návrhu.

Pro Photoshop Image Resource metadata jsem nenalezl žádný způsob vložení do PNG souboru.

2.3 TIFF

Tagged Image File Format je souborový formát k uchování rastrové grafiky. Ukládat lze nekomprimovaná i komprimovaná data, ztrátově i bezztrátově. Původní verze byla vytvořena v roce 1986 společností Aldus. Současná verze 6.0 vyšla v roce 1992. Společnost se později sloučila s Adobe Systems Inc., která teď formát vlastní. Specifikace stanovuje základní verzi, TIFF Baseline, kterou musí všechny dekodéry umět zpracovat. [31] Tato verze má MIME typ *image/tiff*.

Baseline TIFF umožňuje mít v souboru více obrazů, každý ve vlastním adresáři (IFD). Dekodéry nicméně nejsou povinny zpracovávat jiné obrazy než obraz z prvního adresáře. Obraz je dělený na pruhy (strips). V hlavičce je kromě výšky souboru definován i počet řádek v jednom pruhu. Pruhy jsou komprimovány jednotlivě, pokud je potřeba data komprimovat. Baseline nabízí dvě varianty bezztrátové komprese, případně možnost ukládat nekomprimovaný obraz. Barevný prostor Baseline TIFF může být dvojbarevný, v odstínech šedi, indexových barvách a True Color. [31]

Specifikace [31] definuje dále rozšířenou verzi formátu, kterou schválilo TIFF Advisory Committee. Výrobce dekodérů nemá povinnost brát na rozšíření ohled a není tedy zaručeno, že budou podporována na všech zařízeních. Autorům rozšíření je doporučováno, aby úzce spolupracovali s výrobci dekodérů a zajistili tak interoperabilitu.

Rozšíření mimo jiné zavádí možnost komprimace pomocí JPEG nebo LZW, který byl použit ve formátu GIF a stal se předmětem patentového sporu. Je definována možnost rozdělit obraz na několik stejně velkých částí a každou z nich komprimovat samostatně. Je uvedeno, že komprese bývá účinnější nad částmi s podobnou výškou a délkou spíše než nad dlouhými obdélníky, které vznikají u obrazů s vysokým rozlišením při rozdělování na pruhy. [31] Díky rozšíření je možné ukládat obrazy s CMYK a YCbCr barevnými prostory.

2.3.1 Struktura souboru

Struktura TIFF souboru je popsána v kapitole 2 dokumentace [31]. V hlavičce souboru je ofsetový odkaz na nultý adresář. Na začátku každého adresáře je počet položek adresáře. Za ním následuje daný počet položek, každá

o velikosti 12 bytů. Každá položka se skládá z tagu identifikujícího význam položky, tagu datového typu hodnoty, počtu hodnot a ofsetového odkazu na hodnotu. Pokud se hodnota vejde do 4 bytů, je možné místo offsetu vložit do položky přímo hodnotu.

Každý soubor musí mít alespoň jeden adresář. Na konci každého adresáře musí být odkaz na další adresář. Poslední adresář v řetězci musí mít v odkazu na další adresář nulu.

2.3.2 Další metadata

Jak bylo zmíněno v kapitole 2.1.3, Exif je pro nekomprimované grafické soubory definovaný nad TIFF Baseline. Struktura souboru s Exif metadaty je popsána v kapitole 4.5.2 dokumentu [19]. Metadata jsou umístěna jako první adresář.

TIFF má definováno několik tagů pro odkazování na bloky metadat jiných formátů. Pro XMP je využíván tag 700, pro IIM 33723 a pro PSIR 34377, dle [21], kapitola 2.1.6. Tato metadata mohou být snadno vložena do TIFF.

ICC profil je možné vložit pomocí tagu 34675. Tag by měl být v adresáři, ve kterém je i odkaz na obrazová data, na které má být profil aplikován. Jelikož může být v TIFF více obrazů, je možné definovat i více profilů v souboru. V adresáři ale nesmí být více než jeden odkaz na ICC profil. [28]

3 Existující nástroje

V této kapitole popisuji nástroje, které se svým charakterem týkají mojí práce a které bych mohl využít k dosažení cíle. Jde v prvé řadě o aplikace a knihovny pracující s obrazovými soubory, které umí číst jejich metadata. Dále jde o RDF slovníky, které by mohly poskytovat některé pojmy užitečné k uchování informace z obrazových souborů.

3.1 Nástroje pro extrakci metadat

Účelem této práce je najít metadata v obrazových souborech a extrahovat je. Možnost změny a zápisu metadatové informace v souboru, případně vykreslení obrazu, je mimo rámec práce. Z toho důvodu se v této sekci zaměřím především na možnosti nástroje při extrahování metadat.

3.1.1 Exiv2

Exiv2 [32] je knihovna v jazyce C++ a konzolová aplikace pro čtení a editaci metadat obrazových souborů. Knihovna je licencována jako GNU General Public License 2.0[1]. Autor nabízí možnost po domluvě poskytnout knihovnu s jinou licencí tak, aby ji šla použít v closed-source projektech. Poslední stabilní vydaná verze je Exiv2 v0.24 z 2. prosince 2013. Knihovna je aktivně udržována a v projektu[2] je každý den provedeno několik revizí.

Knihovna podporuje metadata formátů Exif, IIM a XMP. V případě Exif a XMP metadat umí získané hodnoty převést na hodnoty pro člověka srozumitelné. Jde především o převod číselníkových hodnot na jejich textové popisy, nebo přidání informace o měrných jednotkách, jak plynou z definice tagu. Na webových stránkách knihovny lze vidět seznam detekovaných metadatových tagů[3]. Rozpoznané jsou i Exif Makernote 10 výrobců fotoaparátů. Knihovna pracuje s 26 XMP slovníky.

[1] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[2] http://dev.exiv2.org/projects/exiv2/activity
[3] http://www.exiv2.org/metadata.html

Knihovna se nezabývá metadaty JIF, JFIF, PNG a Photoshop Image Resource. Hodnoty ICC profilu lze získat pouze jako posloupnost bytů dále nezpracovanou a nepopsanou.

3.1.2 ExifTool

ExifTool [33] je platformě nezávislá knihovna a konzolová aplikace pro čtení a manipulaci s obrazovými metadaty. Podporuje řadu typů souborů včetně JPEG, PNG a TIFF. Knihovna je licencována jako GNU General Public License[4] nebo Artistic License[5]. Poslední stabilní verze knihovny je 9.90 z 14. března 2015. Knihovna je aktivně udržována a každý měsíc vychází průměrně dvě nové vývojové verze. Stabilní verze vycházejí průměrně čtyři ročně.

Knihovna umí číst kromě JIF všechny v této práci diskutované metadatové formáty. Pro formát XMP rozpozná 55 různých slovníků. Exif Makernote jsou čtena pro 22 výrobců zařízení.

3.1.3 Metadata Extractor

Metadata Extractor [34] je knihovna napsaná v jazyce Java sloužící ke čtení metadat z obrazových souborů. Mezi podporovanými formáty jsou JPEG, PNG i TIFF. Produkt je licencovaný jako Apache License 2.0[6]. Verze 2.8.1 vydaná 20. dubna 2015 je poslední stabilní verzí. Projekt je aktivní.

Knihovna čte všechny metadatové formáty diskutované v kapitole 2. V případě XMP dokáže přečíst hodnoty jen ze 4 slovníků. Dokáže dekódovat Exif Makernote od 15 výrobců zařízení. Hodnoty tagů lze převést do tvaru pochopitelného pro člověka s vyjímkou Makernote, kde pro přibližně 20 % tagů převod chybí.

[4] http://dev.perl.org/licenses/gpl1.html
[5] http://dev.perl.org/licenses/artistic.html
[6] http://www.apache.org/licenses/LICENSE-2.0

3.1.4 Aperture

Aperture [35] je Java framework pro extrahování metadat z různých zdrojů a jejich zapisování do RDF modelu. Jako ukázka možných zdrojů jsou uváděny emailové schránky, webové stránky a souborové systémy. Kromě metadat je možné extrahovat i text obsažený v datech, například obsah emailu nebo textového souboru. Framework je vydáván pod BSD licencí, která umožňuje použít pro aplikace užívající Aperture API libovolnou licenci.

Poslední verze projektu je 1.6.0 z 30. prosince 2011. Na domovských stránkách frameworku[7] se nicméně jako poslední uvádí 1.5.0 z 1. července 2010. Jeden z vývojářů, Antoni Mylka, komentoval stav projektu v tiketu v repositáři [36]. Dle jeho slov je projekt mrtvý, a zlom nastal v roce 2011, kdy se hlavní vývojář rozhodl, že nechce používat RDF k uchováváni dat a přešel na jiný projekt. Není prý žádný aktivní projekt, který by na Aperture navazoval ve smyslu extraktoru metadat a jejich uchování v RDF. Zpětně jako největší problém projektu uvádí fakt, že velká část lidí odmítala RDF, a ti, kteří ho podporovali, se nedokázali shodnout na centrální ontologii a místo toho se snažili prosadit svoji vlastní.

Aperture dokáže získat metadata z JPEG souboru. Formáty PNG a TIFF nejsou podporovány. V JPEG souboru čte pouze Exif metadata. Exif Makernote nedokáže zpracovat. Metadata jsou mapována na vlastnosti NEPOMUK ontologie.

3.1.5 Porovnání nástrojů

V tabulce 3.1 uvádím srovnání jmenovaných nástrojů na základě množství jimi rozpoznatelných metadatových tagů. Pro Exif Makernote uvádím pouze počet různých výrobců, jejichž metadata jsou alespoň částečně čtena. Dle mého názoru má tento ukazatel větší vypovídající hodnotu, protože jednotlivé řešení přistupují odlišně k metadatům různých modelů stejného výrobce. ExifTool pro každý z 26 modelů výrobce Canon používá zvláštní sadu tagů, které se významem překrývají. Metadata Extractor používá pro tyto tagy pouze jednu sadu tagů. ExifTool by tak měl mnohonásobně více definovaných tagů v případě, že by oba nástroje rozpoznávaly stejnou množinu metadat.

[7] http://aperture.sourceforge.net/

Všechny nástroje pracující s XMP metadaty pracují na principu hledání nadefinovaných známých řetězců v serializaci RDF modelu vložené do obrazového souboru. Nejsou tedy schopny zobrazit libovolné RDF vlastnosti a jejich hodnoty. Domnívám se, že toto řešení vychází ze snahy vývojářů získaná metadata převést do člověkem srozumitelného tvaru. V souhrnné tabulce 3.1 uvádím počty slovníků, s kterými nástroj pracuje a čte nějaké vlastnosti v nich definované.

	Metadata Extractor	Exiv2	ExifTool	Aperture
Exif	169	290	282	23
JIF	9	0	0	0
JFIF	4	0	4	0
ICC profil	65	0	104	0
IPTC	77	71	84	0
PSIR	43	0	97	0
PNG	24	0	43	0
Exif Makernote	15 výrobců	10 výrobců	22 výrobců	0
XMP	4 slovníky	26 slovníků	55 slovníků	0

Tabulka 3.1: Počet tagů v metadatových formátech, které jednotlivé nástroje dokáží rozpoznat. U makernote je uveden počet výrobců zařízení, jejichž metadata jsou alespoň částečně čtena. U XMP je uveden počet používaných slovníků.

Počty tagů u Metadata Extraktoru čerpám ze zdrojových kódů knihovny verze 2.7.2. U Exiv2 vycházím z webové dokumentace[8] ke dni 20. 4. 2015. U ExifTool čerpám ze seznamů tagů ke dni 20. 4. 2015, na které je odkazováno z domovské stránky projektu[9]. V případě Aperture jsou počty získány ze zdrojového kódu frameworku verze 1.6.0.

3.2 Běžné slovníky a ontologie

V kapitole 1.2 jsem zmínil důležitost využívání pojmů z již existujících a používaných slovníků při návrhu RDF řešení. Prozkoumal jsem známé slovníky a snažil se najít takové, jejichž vlastnosti by byly použitelné pro uchování obrazových metadat. V praxi neexistuje žádný centrální registr slovníků a

[8] http://www.exiv2.org/metadata.html a příslušné podstránky.
[9] http://www.sno.phy.queensu.ca/~phil/exiftool/

je tedy možné, že existují další relevantní slovníky, které jsou málo známé a propagované.

3.2.1 Dublin Core

Dublin Core slovník obsahuje sadu obecných RDF vlastností, které najdou uplatnění v řadě situací. Původní verze obsahující 15 pojmů, popisující např. název, autora, popis, typ, formát nebo zdroj objektu, byla vydána jako IETF RFC 5013, ANSI/NISO Standard Z39.85-2007 a mezinárodní standard ISO 15836-2009. Tato verze pouze vyjmenovává dané pojmy a neklade žádné sémantické požadavky na jejich doménu nebo obor hodnot.

Současná verze z roku 2012 [37] byla oproti standardu rozšířena na 55 vlastností. Kromě toho definuje několik tříd, které využívá k definici domény a oboru hodnot u všech vlastností. Pro pojmy je definovaný jmenný prostor *http://purl.org/dc/terms/*. K zachování kompatibility se standardem zůstávají na jmenném prostoru *http://purl.org/dc/elements/1.1/* definované původní pojmy bez sémantiky. Je tedy v tomto případě možné si podle potřeby vybrat, zda-li je vhodnější použít vlastnost s definovanou sémantikou či nikoliv.

V kontextu mé práce jsou užitečné především vlastnosti původního slovníku popisující jméno autora (*creator*), datum pořízení snímku (*date*), popis obrazu (*description*) či autorská práva (*rights*). K definování mé ontologie budou užitečné vlastnosti pro název a popis.

3.2.2 Friend of a Friend

Slovník Friend of a Friend (FOAF) vznikl v roce 2000. Jeho cílem bylo umožnit popsání jednotlivých osob a jejich vztahů s jinými osobami a informacemi na webu. Svým způsobem se jednalo o jednu z prvních aplikaci konceptu linked data. Specifikace [38] dělí pojmy do tří kategorií. Do hlavních (core) pojmů patří takové třídy a vlastnosti, které popisují charakteristiky lidí nezávisle na technologii sociálních sítí. Kromě třídy pro člověka je zde i definice tříd pro organizaci, projekt a skupinu. V druhé skupině jsou termíny popisující internetové účty osob nebo stránky projektů. Poslední skupinou jsou pomocné termíny pro linked data, které autoři považovali za užitečné.

Slovníkové pojmy mají k sobě definovanou i sémantiku. Třídy mají hierarchickou strukturu. U některých tříd je vymezeno pomocí *owl:disjointWith*, že nemají průnik s jinými třídami. Například třída *Project* nemá průnik s třídami *Document* a *Person*, tzn. nic identifikovaného jako projekt nemůže být zároveň identifikováno jako osoba. Vlastnosti mají definovanou doménu a obor hodnot. U vlastností navzájem inverzních je inverze definována.

Zajímavým pojmem ve vztahu k mé práci je *thumbnail*, jakožto vlastnost propojující obraz se svým náhledem. Dále jsou zde vlastnosti použitelné pro podrobnější popis autora snímku. Vlastnost *depicts* může propojit obraz a objekt, či osobu na něm zachycenou. Teoreticky by se dalo uvažovat i o vlastnosti *based_near*, která propojuje dva objekty, které jsou si blízké ve smyslu geografické pozice.

3.2.3 Kanzaki Exif

Masahide Kanzaki vydal v roce 2003 ontologii [39], která umožňuje zapsat Exif metadata definovaná ve specifikaci verze 2.2. Konkrétně jde o metadata v Exif, GPS a Interoperability IFD. Makernote nejsou podporována.

Definované třídy mají hierarchickou strukturu. Pro tagy obsahující číselníkový kód jsou definovány zvláštní třídy, jejichž instance odpovídají jednotlivým hodnotám číselníkových kódů. Například pro tag *flash* obsahující kód se stavem blesku při pořízení snímku je v ontologii vytvořena třída *Dataflash*, jejímiž instancemi jsou *Fired*, *NotFired*, *Fired-auto* a další. Tyto objekty mají definované pouze popisné vlastnosti *rdfs:label* obsahující název instance a *rdfs:comment* obsahující textové vysvětlení významu včetně číselníkové kódu. Číselníkový kód jako takový není žádnou vlastností samostatně popsán a musel by v případě potřeby být čtený z komentáře.

RDF vlastnosti jsou v ontologii hierarchicky členěné a mají definovanou doménu a obor hodnot. U literálů je oborem hodnot datový typ z XML schématu. Některé vlastnosti mají anotací nastavenou výchozí hodnotu. Každá vlastnost odpovídající některému z Exif tagů má anotací přiřazené číslo daného tagu.

3.2.4 NEPOMUK Information Element

NEPOMUK, zkratka pro Networked Environment for Personal, Ontology-based Management of Unified Knowledge, je projekt snažící se o vytvoření open-source softwarových nástrojů, které by umožnily pracovat s informacemi v počítači sémanticky. Jde primárně o otázku převedení strukturovaných dat z libovolného datového souboru do sémantického modelu, v případě tohoto projektu do RDF. Za tímto účelem vznikla řada ontologií a slovníků. Jejich podmnožinu tvoří sada ontologií označovaná jako NEPOMUK Information Element (NIE) [40].

Specifikace NIE klade na ontologie několik požadavků. Mimo jiné by neměly užívat jazykové konstrukce, které nejsou v RDF schématu, nebo v NEPOMUK Representation Language, což je nadstavba nad RDFS pro projekt NEPOMUK. Je nutné specifikovat doménu a obor hodnot u vlastností v ontologii. U tříd by měla být vytvořena hierarchie, pokud je to možné.

První ontologií v sadě je NIE [40], dle které se jmenuje celá sada. V ní je definována třída Informačního elementu, což je abstraktní třída pro informace uložené v souboru. V ontologii se nachází vlastnosti, které se hodí k zapsání IPTC metadat. Jsou to *copyright*, *contentCreated* popisující dobu vytvoření obsahu, *keyword* s klíčovými slovy a *title* pro název snímku.

NEPOMUK File Ontology [41] nabízí možnosti jak zapsat informace o datovém souboru na disku. Jsou zde vlastnosti k zapsání jména souboru, hash otisku, dobu vytvoření a modifikace. Pro obrazové soubory jsou zajímavé vlastnosti určující, z kolika prvků je složena barevná paleta, zda-li dochází k prokládání obrazu (interlace) nebo jakým způsobem je komprimován.

Poslední zajímavou ontologií je NEPOMUK EXIF ontologie (NEXIF) [42]. Tato ontologie měla vzniknout jako rozšíření Kanzakiho ontologie, která v té době postrádala definici domény a oboru hodnot vlastností a nevyhovovala tak nastaveným požadavkům projektu na ontologie. K důležité změně došlo u vlastností, pro které Kanzaki definoval číselníkové třídy. V specifikaci NEXIF mají tyto vlastnosti obor hodnot integer nebo string. Ontologie tak ztrácí funkci kontrolovaného slovníku. Na druhou stranu je možné do těchto vlastností přiřadit hodnotu, která ve specifikaci Exif není definovaná, ale je zavedená výrobcem v Makernote.

3.2.5 Geo

Exif metadata obsahují i údaj o geografické poloze, kde byl snímek pořízen. K zápisu této informace lze využít slovník vytvořený W3C [43]. Slovník definuje třídu geografického bodu a její vlastnosti zeměpisná délka, šířka a výška. Obor hodnot vlastností není definovaný, je doporučováno používat textový řetězec.

3.2.6 Adobe slovníky

Společnost Adobe Systems využívá metadatový formát XMP, který je založen na RDF. Aby bylo možné uchovávat data, vytvořila několik slovníků obsahujících pojmy k popisu obrazových dat. Slovníky jsou do určité míry popsány ve specifikacích XMP [21, 22, 23] a dokumentu mapování IIM na XMP[25]. Dle dokumentů se zdá, že slovníky pokrývají velkou množinu obrazových metadat.

V dokumentech jsou však pouze informace o jmenném prostoru, názvu vlastnosti a datovém typu. Není známo, zda-li mezi pojmy existují další ontologické vztahy. Společnost nikde nezveřejnila serializaci definice slovníků do některého z RDF formátů. Slovníky tak bohužel nelze využít v sémantických aplikacích pracujících s ontologiemi.

4 Návrh řešení

4.1 Požadavky na řešení

V zadání je vymezeno, že řešení by mělo vycházet z potřeb projektu Medical Research & Education (MRE) na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni[1]. V tomto projektu je vyvíjen nástroj MetaMed, který je používán k extrakci metadat z příchozích lékařských dat v různých formátech.

Metadata obrazových souborů ale nejsou zatím programem řádně extrahována. Program pouze zapíše jejich obecná metadata jako název souboru, datum vytvoření souboru nebo hash otisk. Informace obsažené v obrazových metadatech by byly užitečné pro další práci se soubory a požadavek na jejich extrakci měl vysokou prioritu. Proto v práci řeším právě obrazové soubory. Po konzultaci s vedoucím diplomové práce byly na základě četnosti užití vybrány grafické soubory MIME typů *image/jpeg*, *image/png* a *image/tiff* jako vhodné k řešení.

Řešení bude mít formu pluginu pro aplikaci MetaMed v jazyce Java. MetaMed je licencovaný pod GNU General Public Licence verze 3[2]. Je nutné brát ohled na kompatibilitu licencí externích knihoven s touto licencí. MetaMed není vázán na konkrétní operační systém či platformu, plugin by tedy měl být také multiplatformní. Aplikace poskytuje rozhraní pro připojení pluginu. Posledním požadavkem je uložení získaných metadat do RDF modelu, s kterým MetaMed pracuje.

4.2 Test existujících extrakčních nástrojů

Proces čtení metadat z obrazových souborů není triviální, důkazem tomu budiž fakt, že existující nástroje se liší v množství získatelných metadat, viz tabulka 3.1. Navíc v případě Exif Makernote není k dispozici dokumentace popisující jejich význam a způsob zápisu. Pro tuto činnost by zřejmě bylo vhodné prozkoumat možnost využití již existujícího a ověřeného řešení jiných

[1] Domovská stránka Medical Research & Education: http://medical.kiv.zcu.cz/
[2] http://www.gnu.org/licenses/gpl-3.0.en.html

autorů. V kapitole 3.1 jsem popsal několik nástrojů pracujících s metadaty, které se svým rámcem blíží náplni této práce.

Pro porovnání nástrojů a přehlednou prezentaci jejich schopnosti extrahovat metadata jsem vybral čtyři testovací obrazové soubory, které svými obsaženými metadaty dostatečně ilustrují průměrný stav z dostupných souborů. Jedná se o jeden soubor PNG formátu, jeden TIFF soubor a dva JPEG soubory. Soubory JPEG a TIFF jsou pořízeny fotoaparáty různých výrobců, konkrétně společnostmi Canon, Olympus a Nikon.

Použité verze nástrojů při testování jsou Metadata Extractor 2.7.2, Exiv2 0.24, Exiftool 9.94, Aperture 1.6.0.

Nalezení vhodných souborů, ve kterých by bylo zastoupeno co nejvíce různých metadatových formátů a tagů, bylo obtížné. U souborů často chyběla informace o tom, zda-li je možné je volně užít v mé práci. Nakonec jsem zvolil soubory, které jsou sbírány autorem Metadata Extraktoru a následně poskytovány volně k libovolnému užití. Archiv[3] v současné době obsahuje 385 grafických souborů 22 různých formátů.

Počty nalezených tagů uvádím v souhrnné tabulce 4.1, detailní tabulka včetně seznamu nalezených metadat je na přiloženém DVD. Z podstaty věci nelze určit celkový počet metadat v souborech a vyjádřit podílem nalezené množství z celku. V tabulce pro přehlednost uvádím hodnotu pouze v případě, že v daném souboru některý z extraktorů nalezl alespoň jeden tag příslušného metadatového formátu. Pokud byla metadata ve více souborech, jsou počty výskytů pro jednotlivé soubory odděleny čárkou. V takovém případě je pořadí hodnot významné a je stejné pro všechny pole dané řádky.

Nástroje se neshodují na názvech metadatových formátů. Uvádím označení zavedené v této práci. V jednom ze souborů byla nalezena metadata formátu Adobe JPEG, který v této práci není popsaný. Nástroje ExifTool a Aperture metadata Exif dále nečlení. Oproti tomu Metadata Extractor a Exiv2 je třídí dle IFD, ve kterém jsou obsaženy. Abych mohl smysluplně porovnávat výsledky, uvádím v řádku Exif součet nalezených metadat ze všech Exif IFD kromě Exif Makernote. Ta jsou uvedena samostatně.

V tabulce 4.1 jsou tučně zvýrazněny maxima pro daný metadatový formát a soubor. Z tohoto testu nevzešel dominantní nástroj, tedy takový, který by v každém souboru nalezl nejvíce metadat ve všech metadatových formátech.

[3]https://github.com/drewnoakes/metadata-extractor-images

	Metadata Extractor	Exiv2	ExifTool	Aperture
Exif	42, 44, 72	**45**, **46**, **73**	36, 38, 57	13, 13, 0
Exif Makernote	132, 49	105, 116	**144**, **160**	0, 0
JIF	**8**, **8**	0, 0	0, 0	0, 0
Adobe JPEG	**4**	0	**4**	0
ICC profil	30, 0	0, 0	**39**, **25**	0, 0
IPTC	**3**	**3**	**3**	0
PSIR	**2**	0	**2**	0
XMP	13	**127**	84	0
PNG	**14**	0	**14**	0

Tabulka 4.1: Počet nalezených metadat u testovaných souborů. V případě výskytu metadatového formátu ve více souborech jsou hodnoty pro jednotlivé soubory odděleny čárkou. Tučně jsou zvýrazněny maxima pro konkrétní soubor a metadatový formát.

Jako nevyhovující se jeví Aperture, který, jak již bylo popsáno v kapitole 3.1.4 a ukázáno v tabulce 3.1, umí pracovat pouze s Exif daty JPEG souboru. Ani zde ale nedosahuje úrovně ostatních nástrojů.

Vhodný není ani Exiv2, který nedokáže pracovat s PNG soubory a metadatovými formáty jinými než Exif, IPTC a XMP. Pro řádné otestování schopnosti práce s IPTC bylo v testované sadě málo metadat tohoto typu. Nástroj nalezl 3 metadata stejně jako Metadata Extractor a ExifTool. V případě Exif metadat identifikoval Exiv2 nejvíce metadat ze všech nástrojů. Po hlubším srovnání nalezených tagů se ukázalo, že na rozdíl od ostatních vypisuje i odkazy na IFD formou bytového offsetu v souboru. Bez těchto odkazů odpovídají množiny nalezených tagů těm, které získal Metadata Extractor. Exif Makernote zpracoval Exiv2 v obou případech. V XMP záznamu dokázal Exiv2 přečíst nejvíce vlastností i přesto, že má definováno méně slovníků než ExifTool, viz tabulka 3.1.

Metadata Extractor dokázal získat ve všech případech více Exif metadat než ExifTool. Jako jediný rozpoznal JIF metadata obsažená v JPEG souborech. U metadatových formátů Adobe JPEG, IPTC, PSIR a PNG rozpoznal stejná metadata jako ExifTool. Špatný výsledek měl Metadata Extractor u XMP, kde dokázal přečíst pouze 13 tagů oproti 84 tagů čtených ExifToolem. Tento výsledek pramení z toho, že Metadata Extractor používá pouze 4 slovníky. Metadata Extractor nedokázal extrahovat ICC profil z testovaného souboru TIFF. Z JPEG souboru profil přečetl, ale získal méně metadat než

ExifTool. U výsledku čtení Makernote tagů je důležité poznamenat, že zde Metadata Extractor vypisuje i tagy, u kterých nezná význam. Jde o 50, resp. 3 tagy. Po jejich odečtení je na tom kvantitativně hůře než Exiv2 a ExifTool.

U ExifTool je problém s nemožností zpracovávat JIF metadata. Exif metadata nepřečetl v takové míře jako Metadata Extractor a Exiv2. XMP metadat zpracoval méně než Exiv2. U ostatních metadatových formátů měl nejlepší výsledek.

4.3 Volba nástroje

Z hlediska množství rozpoznaných tagů v předchozím testu nebyl žádný nástroj perfektní. Každý z trojice Metadata Extractor, Exiv2 a ExifTool měl oblast, ve které vynikal. Variantou by tedy bylo využít v mém pluginu všechny nástroje podle jejich silných stránek. Jelikož je každý psán v jiném jazyce, bylo by v rámci jejich integrace nutné hledat způsob jak efektivně volat procedury jiného jazyka. Problém by bylo možné obejít používáním konzolových aplikací místo knihoven a čtením extrahovaných metadat z výstupu aplikací. Tento přístup však otevírá nové problémy s identifikací tagů a jejich hodnoty. ExifTool uvádí pouze názvy vlastností a metadatový formát. XMP metadata se stejným názvem, ale jiným adresářem, nelze proto rozlišit. Takto předávaná metadata by navíc již byla interpretována, což by vadilo při procesu mapování do RDF na existující slovníky, ve kterých může být požadována číselníková hodnota.

Dalším problémem by byla otázka rozdělení řešeného problému mezi nástroje. ExifTool má více jak dvojnásobný počet slovníků pro XMP než Exiv2, ale extrahoval v testu méně metadat. Který z nich má být v takovém případě určen ke zpracování daného formátu? I kdyby se problém rozděloval až na úrovni slovníků a výrobců Makernote, byla by potřeba vypracovat podrobná analýza rozpoznatelných metadat a míry jejich užitečnosti, aby šlo exaktně rozhodnout o nástroji, který bude slovník či daného výrobce řešit.

Toto řešení se mi zdálo nepraktické a proto jsem se rozhodl zvolit pouze jednu knihovnu, s kterou budu pracovat. V této fázi jsem již uvažoval pouze nástroje Metadata Extractor a ExifTool, které podporují všechny požadované souborové formáty. Metadata Extractor je psán v jazyku Java, ExifTool je v jazyku Perl. Jelikož plugin má být v jazyku Java a já nemám žádné zkuše-

nosti s voláním procedur jazyka Perl Java aplikací a ani s jazykem Perl jako takovým, rozhodl jsem se pro Metadata Extractor.

4.4 Výběr ontologií a slovníků

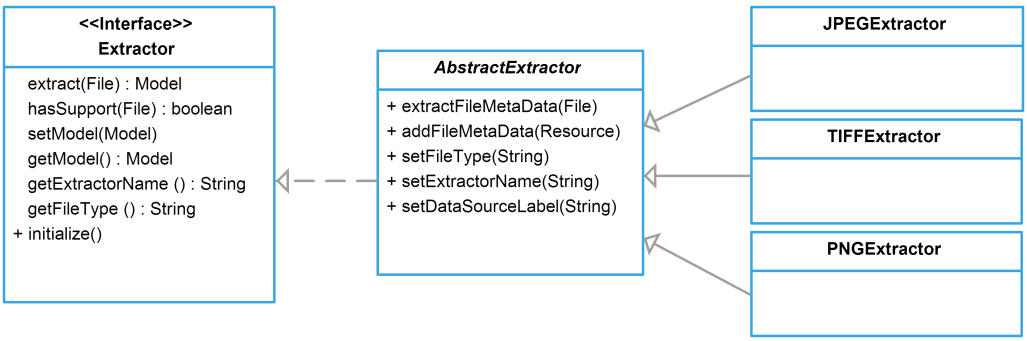
Extrahovaná metadata z obrazových souborů mají být uložena do RDF. V kapitole 3.2 jsem představil slovníky, které jsou používané a definují pojmy týkající se této práce. Neexistuje žádný slovník, který by svým rozsahem pokryl celý řešený problém. V práci tedy kombinuji pojmy z různých slovníků, pokud to jejich sémantika definovaná v ontologii umožňuje.

Kanzakiho Exif slovník a NEXIF, který je součástí NEPOMUK projektu, definují pojmy pro Exif metadata obrazových souborů. Každý ale nabízí jiné ontologické řešení. Kanzaki využívá konceptu kontrolovaných slovníků pro Exif tagy, které mají charakter číselníku. Oproti tomu NEXIF u většiny těchto tagů požaduje textovou hodnotu, které odpovídá číselníkový kód. Výjimkou jsou v současné specifikaci pojmy pro orientaci snímku a zápis tagů GPS IFD Exifu. NEXIF nabízí více pojmů než Kanzakiho slovník (156 oproti 142). Z těchto dvou slovníků jsem se rozhodl pro využití NEXIF na základě toho, že byl vytvořen jako rozšíření Kanzakiho slovníku, je propojený s dalšími slovníky NEPOMUK projektu a obsahuje více pojmů. Tato volba je dále výhodná při vytváření mapování Makernote metadat, které žádný slovník explicitně neřeší. Makernote metadata obsahují některé tagy významem odpovídající tagům specifikovaným v Exif formátu, ale pro číselníkové tagy mohou zapisovat i další kódy. Jelikož NEXIF nemá kontrolovaný slovník pro taková metadata a využívá textové hodnoty, není problém namapovat Makernote tag i přesto, že se neshoduje strukturou číselníku.

Aby bylo možné se zapsanými informacemi do RDF dále pracovat je potřeba mít dostupnou ontologii pro používané slovníky. V tomto ohledu nevyhovují Adobe slovníky, pro které není soubor s ontologií uvolněn. Sémantiku lze částečně dohledat v dokumentech specifikujících XMP, ale takový formát je v tomto případě nepoužitelný. Adobe slovníky, na kterých je vybudován XMP formát, proto nemohu použít.

4.5 Implementované rozhraní

Plugin obsahuje tři třídy extraktorů, jednu pro každý typ obrazových souborů. Tyto třídy implementují rozhraní *Extractor* poskytované knihovnou LibMed aplikace MetaMed. Třídy dědí z abstraktní třídy *AbstractExtractor*, která je také deklarována v knihovně. Tento vztah je zachycen v diagramu tříd na obrázku 4.1. Diagram obsahuje i signatury procedur.



Obrázek 4.1: Diagram tříd zachycující implementované rozhraní a dědičnost extraktorů.

Rozhraní předepisuje sedm metod, které je potřeba implementovat. Metodou *extract(File)* je v parametru předáván odkaz na soubor, z kterého mají být získána metadata. Metadata se uloží do RDF modelu, na který je procedurou vrácen odkaz. MetaMed používá framework Jena[4] k práci s RDF modely a vracený model je instancí třídy implementující rozhraní *Model*[5] z tohoto frameworku.

Metoda *hasSupport(File)* slouží k určení, zda-li extraktor dokáže pracovat s formátem souboru, který je předávaný v parametru. Metodou *setModel(Model)* se extraktoru předá odkaz na RDF model, do kterého má být zapisováno. Metodami *getModel()*, *getExtractorName()* a *getFileType()* se získá z extraktoru odkaz na používaný RDF model, řetězec s názvem extraktoru, respektive řetězec s datovým typem, který je extraktorem zpracováván. Voláním metody *initialize()* by se měla instance extraktoru připravit k použití.

Abstraktní třída *AbstractExtractor* implementuje kromě metod *extract(File)* a *initialize()* všechny předepsané metody.

[4]Domovská stránka frameworku: https://jena.apache.org/
[5]com.hp.hpl.jena.rdf.model.Model

4.6 Struktura Metadata Extractoru

Metadata Extraktor člení známé metadatové tagy do adresářů. Pro každý adresář existují třídy s názvem *kodDirectory.java* a *kodDescriptor.java*, kde *kod* je zkratka názvu adresáře. V prvním ze souborů jsou definované číselné kódy známých metadatových tagů, názvy tagů a procedury pro jejich získání ze souboru. Do procedur bývá zakomponována i kontrola správnosti dat. V druhém souboru jsou definované procedury k převodu extrahované hodnoty do formátu pro člověka srozumitelného.

Extrakce obrazových metadat ze souboru probíhá pomocí funkce *readMetadata(File)*, kterou poskytují třídy *JpegMetadataReader*, *PngMetadataReader* a *TiffMetadataReader*. Funkce vrací objekt třídy *Metadata* se získanými metadaty. Tento objekt v sobě obsahuje několik objektů třídy *Directory*, a každý z nich obsahuje několik objektů třídy *Tag*. Pokud soubor neobsahuje metadata, nebo je nebylo možné přečíst, nebude v *Metadata* instanci žádný *Directory* objekt.

Z objektu *Directory* lze zjistit počet známých tagů pro daný adresář nebo počet nalezených tagů, tedy objektů třídy *Tag* v instanci *Directory*. Objekt uchovává v hashovacích mapách získané hodnoty, získané hodnoty převedené do přijatelnějšího tvaru pro člověka a názvy metadat. Hashovací mapy mají jako klíč číselný kód tagu. Objekt umí vrátit iterátor přes nalezené tagy.

U objektů třídy *Tag* lze získat přeloženou extrahovanou hodnotu, číselný kód tagu decimálně nebo hexadecimálně, či název tagu. Extrahovanou hodnotu nepřeloženou nelze z objektu získat. Je potřeba použít adresářový objekt. Extrahované hodnoty jsou textové řetězce bez ohledu na jejich význam.

5 Implementace

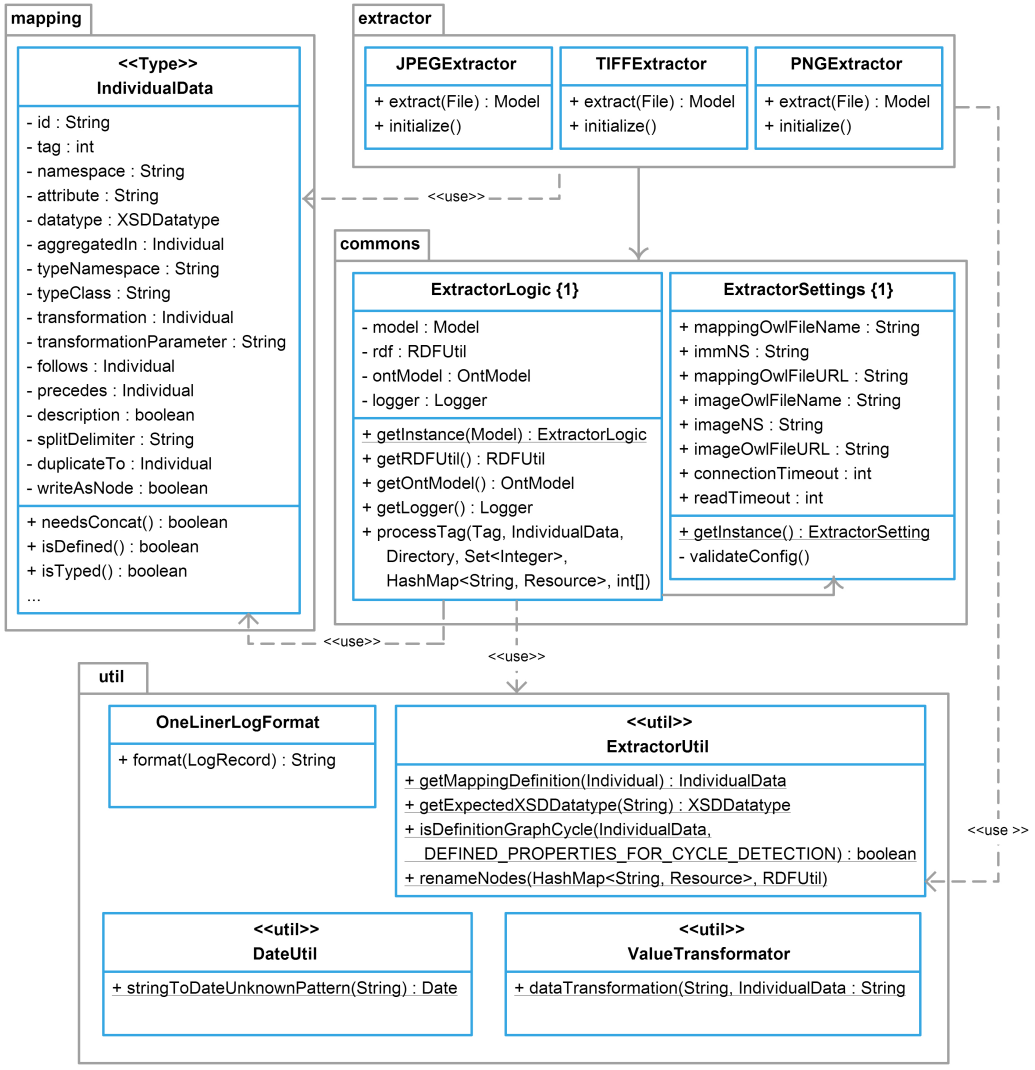
Architektura řešení je zobrazena diagramem tříd na obrázku 5.1. Třídy jsou rozděleny do čtyř balíků. V balíku *extractor* jsou třídy implementující rozhraní *Extractor* MetaMedu. V balíku *mapping* je pouze jedna třída, *IndividualData*, kterou používám pro uchování definice mapování pro určitý tag. Balík *util* obsahuje tři třídy se statickými procedurami užitečnými při zpracování metadat a třídu *OneLinerLogFormat* potřebnou při inicializaci Java loggeru, aby byla zjednodušena syntaxe zapisovaných záznamů do logu. Poslední balík, *common*, obsahuje třídu zpracovávající extrahovaná metadata nezávisle na typu souboru a třídu spravující konfiguraci pluginu.

V diagramu pro přehlednost vynechávám procedury, které slouží k rozčlenění komplexních činností a uvádím pouze hlavní proceduru. Například třída *ValueTransformator* má 15 neuvedených privátních procedur na transformaci hodnot, které se volají v závislosti na vstupním parametru funkce *dataTransformation*. Plné křivky v diagramu značí asociaci, tzn. že třída, z které křivka vychází, má ve svém atributu odkaz na instanci třídy, do které křivka vede. Přerušovaná křivka značí závislost, tedy že třída, z které křivka vychází, v některé ze svých procedur využívá proceduru či atribut třídy, do které je křivka vedena. Pro zmenšení počtu linií v diagramu vykresluji pouze jednu spojnici na okraj balíku v případě, že daná závislost či asociace platí pro každou třídu balíku.

5.1 Balík extractor

Extraktory využívají anotaci *@MetaDataExtractor* definovanou knihovnou LibMed v projektu MetaMed. Díky této anotaci dokáže aplikace MetaMed najít všechny třídy extraktorů aniž by na ně bylo ve zdrojovém kódu přímo odkazováno.

Knihovna Metadata Extractor pracuje s obrazovými soubory JPEG, PNG a TIFF podobně. Při implementaci proto nebylo potřeba navrhovat odlišné algoritmy a kódy v extraktorech jsou si podobné. Každý extraktor si uchovává v privátních atributech odkaz na ontologický model obsahující definici mapování metadat na RDF vlastnosti, konfiguraci, nástroj k zapsání RDF trojice do modelu a instanci *ExtractorLogic*, která obsahuje hlavní logiku ex-



Obrázek 5.1: Diagram tříd v pluginu a jejich vzájemných závislostí.

trakce a zpracování metadat společnou pro všechny extraktory. Kromě toho má každý definován v hashovací mapě třídy metadatových adresářů z Metadata Extractoru, které jsou pro daný typ souboru přípustné, a jejich kódovou zkratku.

Voláním konstruktoru se přiřadí řetězec s MIME typem, který extractor řeší, do zděděného atributu *fileType* a nastaví se název extraktoru dle jména extraktorové třídy. Volání konstruktoru s parametrem používaného RDF modelu přiřadí jeho odkaz do zděděné proměnné *model*.

Pro úplnou implementaci rozhraní musí třídy poskytnout vlastní implementaci dvou procedur. Proceduru *initialize()* volá MetaMed jednorázově na začátku procesu extrakce metadat ze souborů. Procedura získá odkazy na instance singletonů *ExtractorSettings* a *ExtractorLogic* a z nich dále odkazy na instanci utility k zápisu do RDF modelu, *RDFUtil* projektu MetaMed, a ontologického modelu, ve kterém je načtena mapovací ontologie. Druhá procedura, *extract(File)*, je volána pro každý zpracovávaný soubor a obsahuje algoritmus k přečtení, zpracování a zápisu metadatové informace.

5.1.1 Algoritmus extrakce

Algoritmus se skládá z následujících sériově prováděných činností:

1. Vytvoření počitadel uchovávajících statistiky.
2. Čtení obecných souborových metadat, vytvoření *Resource* objektu v modelu pro daný soubor na základě jeho názvu a zapsání čtených metadat.
3. Vytvoření hashovací mapy uchovávající názvy a odkazy na všechny vytvořené objekty typu *Resource* a vložení objektu zpracovávaného souboru.
4. Čtení metadat pomocí knihovny Metadata Extractor.
5. Získání definice z mapovací ontologie pro každý tag z každého adresáře a zapsání čtených metadat do RDF modelu.
6. Změnění IRI všech objektů *Resource* kromě objektu reprezentující zpracovávaný soubor.
7. Vypsání statistik a případného varování při podezření na špatnou konfiguraci extraktoru.

Počitadla sledují čtyři celočíselné údaje. Prvním je počet zapsaných RDF trojic do modelu. Toto číslo nemusí odpovídat počtu nalezených a zapsaných tagů z různých důvodů. Například u IPTC tagu obsahující klíčová slova k popisu souboru je pro každé slovo zapsána jedna trojice. Do zapsaných trojic se nezapočítávají trojice vytvořené ve druhém kroku, protože jsou zapisovány na jiné úrovni než obrazová metadata. Druhý sledovaný údaj je počet tagů, jejichž definice nebyla nalezena v mapovací ontologii. Jde převážně o tagy,

které extrahovací knihovna nezná a extrahovaná hodnota nemusí být korektní. Tyto tagy v mapovací ontologii nejsou obsaženy z důvodu nemožnosti vytvoření jejich konečného výčtu. Další podrobnosti ke způsobu vytváření mapovací ontologie jsou v kapitole 5.5.1. Třetím údajem je počet tagů definovaných v ontologii, ale nemapovaných na žádnou RDF vlastnost. Takové tagy dokáže knihovna pojmenovat, ale hodnoty v nich obsažené nejsou užitečné pro další práci na úrovni MetaMedu. Jde například o hodnoty Makernote obsahující data, jejichž interpretace je neznámá. Posledním sledovaným údajem je počet tagů, jejichž získaná hodnota byla extrahovací knihovnou označena jako neznámá, byla prázdná, či byla během transformace označena jako neplatná. Knihovna jako neznámou označí třeba hodnotu číselníkového tagu, která není definovaná v číselníku.

Obecná metadata souboru jsou čtena zděděnou procedurou abstraktní třídy. Během čtení je mimo jiné získán název souboru, který je použit v parametru funkce *mreResource(String)* utility *RDFUtil* k vytvoření zdroje daného souboru. IRI vytvořeného zdroje má tvaru *http://mre.kiv.zcu.cz/id/HASH*, kde *HASH* je 40 znaků dlouhý řetězec, na který byl převeden název souboru hashovací funkcí SHA-1.

Hashovací mapa vytvořená v bodě 3 slouží k tomu, aby extraktor mohl vytvářet komplexní stromovou strukturu a mohl metadata ukládat do trojic s jiným zdrojem na pozici subjektu než je zdroj generovaný v předchozím kroku. Takto lze například informace o náhledu obrazu obsažené v jednom z Exif IFD zapsat bez toho, aby kolidovala s informacemi o původním obrazu.

Ke čtení obrazových metadat používám výhradně knihovnu Metadata Extractor a její funkce vracející instanci třídy *Metadata*, viz kapitola 4.6.

Při procházení všech tagů adresáře si program uchovává množinu čísel již zpracovaných tagů. Některé tagy mohou být zpracovány ještě před tím, než se k nim iterátor dostane. To nastává v případě, kdy se zapisovaná hodnota skládá z hodnot několika tagů. Například v IPTC jsou datum a čas vytvoření zapsány do dvou tagů, ale pro zápis do RDF potřebuji jejich složenou hodnotu. Pokud se iterátor dostane k již zpracovanému tagu, pokračuje hned následujícím tagem. Toto řešení není nezbytně nutné, protože zpracování skládané hodnoty je definováno tak, aby nezáleželo na pořadí nalezení tagů. Zapisovaná trojice by odpovídala trojici již existující v grafu a nebyla by duplicitně zapsána. Nicméně toto řešení šetří výpočetní čas v případě, kdy se v datech kombinované hodnoty vyskytují.

Pro každý tag nalezený Metadata Extractorem se vyhledá definice mapování z mapovací ontologie. Definicí se rozumí prvek na jmenném prostoru definovaném v konfiguračním souboru s názvem *ADRESÁŘ_KÓD*, kde ADRESÁŘ je zkrácený název metadatového adresáře, ve kterém se tag vyskytuje, a KÓD je hexadecimální číslo tagu, jak ho poskytuje knihovna. Další informace o mapovací ontologii viz kapitola 5.5.1. Definice se zpracuje funkcí *ExtractorUtil.getMappingDefinition(Individual)* a je vrácen odkaz na instanci *IndividualData* umožňující přístup k jednotlivým položkám definice. Samotné zpracování hodnoty na základě definice a následný zápis provádí funkce z třídy *ExtractorLogic*.

Po dokončení iteračního cyklu nad tagy z adresáře mohou být vypsány chyby, na které narazila knihovna při jejich extrakci. Tyto hlášení mají pouze informační charakter a nejsou známkou špatně vykonané extrakce. Vypisuji je proto na standardní výstup. V celé sekci využívající externí knihovnu a pracující se souborem zachytávám výjimky *IOException* a *JpegProcessingException*, resp. ekvivalentní pro jiné typy obrazových souborů. Hlášení o těchto chybách jsou posílána na chybový výstup.

Před dokončením práce s metadaty daného souboru jsou přejmenovány RDF zdroje vytvořené během zpracování. Pojmenování je jednoznačné na základě trojic modelu, ve kterých se zdroj vyskytuje na pozici subjektu. V době vytvoření těchto zdrojů není kompletní model znám a přejmenování musí proběhnout až po zápisu všech trojic. Tímto způsobem je zajištěno, že zdroje budou mít při opětovném spuštění stejný název. Zároveň je zajištěno, že zdroje popisující shodnou skutečnost vytvořené při zpracování různých souborů mají také stejný název. V důsledku je zdroj popisující fotoaparát, z kterého byla série fotografií pořízena, zapsán v RDF modelu pouze jednou a všechny fotografie na něj odkazují.

5.2 Balík commons

Balík commons obsahuje dvě třídy. Třída *ExtractorSetting* slouží k načtení a uchování konfigurace pluginu. *ExtractorLogic* definuje procedury k zapsání extrahovaných metadat do RDF nezávisle na typu obrazového souboru. Obě třídy jsou navržené dle designového vzoru singleton, což znamená, že v každém okamžiku je inicializována nejvýše jedna instance dané třídy.

5.2.1 Třída pro práci s konfigurací

Instance třídy *ExtractorSetting* obsahuje osm atributů popisující konfiguraci pluginu. Jsou jimi atributy *mappingOwlFileName*, *mappingOwlFileURL* a *immNS* týkající se mapovací ontologie. První z nich nese název souboru s ontologií. Druhý je jmenný prostor, který ontologie využívá, a třetí je URL adresa, z které má být soubor případně stažen. Obdobně popisují atributy *imageOwlFileName*, *imageNS* a *imageOwlFileURL* ontologii používanou k zapsání metadat, pro které nejsou definované vhodné vlastnosti v jiných ontologiích. Zbylé atributy *connectionTimeout* a *readTimeout* slouží k nastavení doby, po které se má přerušit pokus o přípojení k serveru při stahování ontologie, resp. doby, po které se přeruší stahování, pokud nepřichází žádná data.

Ke čtení souboru konfigurace dochází při inicializaci. Je použita funkce knihovny LibMed *PropertiesUtil.readConfigs(String)*. Parametrem je relativní cesta k souboru. Konstantou v třídě je stanovena cesta ke konfiguračnímu souboru *image/image.properties*. Funkce jako výchozí složku považuje *.mre/* v domovském adresáři uživatele. Z této složky načítá MetaMed i další konfiguraci a ukládá sem výpis chybového výstupu.

V bloku kódu pracujícího s načtenými daty jsou zachytávány chyby *NullPointerException* a *NumberFormatException*. První z nich je zapříčiněna nenalezením souboru na očekávaném místě. Druhá chyba nastává při pokusu převést textovou hodnotu na celočíselnou. Při výskytu libovolné chyby je celý program ukončen, jelikož proces zpracování metadat a jejich zápisu do RDF závisí na ontologiích, jejichž načtení je podmíněno korektní konfigurací. Chyba je vysvětlena v chybovém hlášení zaslaném na chybový výstup před ukončením běhu programu.

Po zpracování konfiguračního souboru je volána funkce *validateConfig()* k ověření, že všechny atributy byly naplněny a že číselné hodnoty časových limitů k ukončení spojení jsou kladné. Pokud to pro některý atribut neplatí, vrací funkce název problémového atributu. V takovém případě je na chybový výstup zasláno hlášení o chybném prvku konfigurace a program je ukončen.

5.2.2 Třída ke zpracování a zápisu metadat

Inicializace

Třída *ExtractorLogic* v konstruktoru ukládá odkaz na RDF model předávaný parametrem konstruktoru do privátního atributu. Následně vytváří instanci utility *RDFUtil* nad daným modelem a instanci singletonu *ExtractorSettings*.

Na základě hodnot v konfiguraci se načtou obě ontologie do in-memory modelu. Při načítání se nejprve zkouší otevřít soubor s ontologií v pracovním adresáři. Pokud soubor není nalezen, je zaslána zpráva s názvem hledaného souboru na standardní výstup a program se pokusí získat soubor s ontologií z Internetu. Pokud je úspěšný, zapíše soubor na disk na adresu, kde byl původně hledán. V případě, že se ani tímto způsobem nepodařilo ontologii získat, je na chybový výstup zaslána o této situaci zpráva a program se ukončí. Toto pořadí načítání je nutné, aby uživatel mohl provádět změny v definici mapování v lokálním souboru bez nutnosti aplikovat tyto změny i na vzdáleném centrálním úložišti. V opačném případě by samozřejmě řešením mohlo být přepsání URL adresy v konfiguraci, či odpojení stroje od Internetu, ale takové postupy rozhodně nejsou uživatelsky přívětivé. Pokud uživatel chce vynutit stažení aktuální verze ontologie z Internetu, je nutné lokální soubor s ontologií smazat či přejmenovat.

Pokud načtení proběhne v pořádku, vytvoří se logger pro zapisování ladících informací. Logger bude zprávy ukládat na disk do souboru *image-extractor-unknown-tags.log* v podsložce *.mre/image/* domovského adresáře uživatele. Adresa a název souboru jsou pevně definovány konstantami třídy. Jelikož jde o soubor primárně určený k zapisování kódů neznámých tagů, používám v loggeru redukovanou informaci bez časové značky a původce zprávy. Odpovídající formát je definovaný pomocnou třídou *OneLinerLogFormat*. V případě chyby při vytváření souboru program pokračuje v normálním běhu, protože logger není nezbytně nutný ke správné funkci extraktoru. Zpráva o problému je zaslána na standardní výstup.

Nakonec je v konstruktoru čten soubor *.mre/image/prefix.properties* v domovském adresáři, který obsahuje definici prefixů. Prefixy se používají při serializaci RDF modelu k nahrazení části IRI se jmenným prostorem. Jako takové nemají na funkci žádný vliv, slouží především ke zlepšení čitelnosti a redukování velikosti výstupního RDF souboru. Pokud by prefixy nebyly definovány, vygeneruje Jena vlastní. Z tohoto důvodu je chyba při zpraco-

vání souboru pouze zaznamenána na standardní výstup a program pokračuje v běhu.

Instanci třídy *ExtractorLogic* lze získat voláním funkce *getInstance(Model)*. Z instance je voláním funkcí *getRDFUtil()*, *getOntModel()* a *getLogger()* získatelný odkazy na inicializovaný *RDFUtil* pro používaný model, model obsahující ontologie k mapování, resp. logger k výpisu ladících informací.

Zpracování metadat

Třída definuje proceduru *processTag(. . .)* s šesti vstupními parametry, která obsahuje část logiky extraktoru shodnou pro všechny souborové typy. Prvním vstupními parametrem je objekt metadatového tagu nalezený Metadata Extractorem. Druhým je definice mapování tagu získaná z mapovací ontologie. Třetí je adresář nalezených tagů v souboru. Následuje odkaz na seznam čísel tagů, jenž již byly zpracovány, a hashovací mapa názvů vytvořených uzlů na jejich odkazy. Posledním parametrem je počitadlo statistických údajů.

Posloupnost činností vykonávaných v proceduře je následující:

1. Nalezení dalších definic mapování a jejich rekurzivní provedení.
2. Získání zapisované hodnoty čtením z tagu nebo složením z více tagů.
3. Získání odkazu na zdroj, který bude v subjektu zapisované trojice.
4. Zapsání trojice.

Pro každý tag lze v mapovací ontologii definovat více mapování. Na další mapování je odkazováno objektovou vlastností *duplicateTo*. V prvním kroku je zjišťováno zda-li v definici mapování zpracovávaného tagu je tato vlastnost použita. V kladném případě je získána definice následujícího mapování a je rekurzivně volána procedura se stejnými parametry kromě druhého. Jelikož hrozí nebezpečí nekonečného rekurzivního volání při použití nevhodné ontologie, je před voláním ověřeno, že v řetězci vlastností *duplicateTo* není cyklus. Pokud by cyklus byl nalezen, je o tom zaslána zpráva na standardní výstup a provede se mapování pouze podle původní definice.

Dále se testuje, zda-li definice obsahuje název a jmenný prostor RDF vlastnosti, na kterou má být mapováno. Pokud neobsahuje, zapíše se infor-

mace o prázdné definici do logu pro ladící výpisy a běh programu opustí proceduru. Tento test nemůže být vykonán hned na začátku, protože i v tomto ohledu prázdná definice může mít definované další mapování, které nemusí být prázdné. Existence použité RDF vlastnosti není programem ověřována.

Druhým krokem je získání hodnoty, která bude do trojice zapsána na pozici objektu. Metadata Extractor poskytuje hodnotu přeloženou, či původní. Rozdíl je patrný především u číselníkových tagů, kdy původní hodnota je například kód *3* u tagu měrných jednotek a přeložená hodnota je řetězec *cm*. Další variantou získání hodnoty je její složení z několika tagů stejného adresáře.

K složení hodnot slouží privátní funkce *concatValue(IndividualData, Directory, Set<Integer>)*. Výsledná hodnota je získána zřetězením hodnot tagů v pořadí definovaném v ontologii. K určení pořadí jsou použity objektové vlastnosti *concatAfter* a *concatBefore*, které jsou vzájemně inverzní. Funkce *concatValue* nejprve zkontroluje, že v ontologii není těmito vlastnostmi tvořen cyklus. Testuje se načítáním dalších prvků v obou směrech, dokud takové prvky existují, a uchováváním seznamu čtených objektů. Pokud byl objekt načten víckrát, je v definici cyklus, informuje se o tom zprávou na standardní výstup a vrací se prázdný řetězec. V opačném případě začne být tvořen výsledný řetězec. Pro každý tag může být definováno, zda-li má být použita přeložená hodnota a zda-li je potřeba provést transformaci. Před přidáním do řetězce je hodnota očištěna na okrajích od bílých znaků. Hodnoty jsou v budovaném řetězci odděleny mezerou. Ve funkci je řešena speciálně situace, kdy dochází k přidání časového údaje k datu. V takovém případě jsou hodnoty odděleny středníkem a po spojení do řetězce je zavolána transformační metoda k převodu do správného tvaru.

Ve třetím kroku je zjišťováno, zda-li subjektem v zapisované trojici bude instance *Resource* obrazového souboru, nebo nějaký jiný uzel. Pokud se jedná o jiný uzel, je zjišťováno, zda-li není v definici cyklus, podobně jako v prvním kroku. V případě nalezení cyklu nebude trojice zapsána a na standardní výstup bude zasláno sdělení o problému. Pokud je definice v pořádku, volá se metoda *getTypedNode(IndividualData, HashMap<String, Resource>, int[])* s parametry definice mapování zpracovávaného tagu, hashovací mapa s vytvořenými uzly a počitadlo statistik. Volaná funkce najde nebo vytvoří odpovídající uzel a odkaz na něj vrací. Při vytváření nového uzlu je čtena jeho definice z ontologie a je možné volat funkci rekurzivně k vytvoření dalších uzlů, s kterými je vytvářený uzel spojen. Pokud není vazba na jiný uzel de-

finována, je zapsána trojice, kde subjektem je obrazový soubor, predikát je čten z definice a objektem je vytvářený uzel.

Ve čtvrtém kroku proběhne zápis extrahované hodnoty. K zápisu slouží procedura *addValueWithType(Resource, IndividualData, String, int[])*. Parametry jsou zdroj použitý v subjektu nalezený ve 3. kroku, definice mapování tagu, zpracovaná hodnota ze 2. kroku a počitadlo. Procedura na začátku nahradí nepovolené znaky mezerou. Mezi nepovolené patří řídící znaky 0x00 až 0x1f, znak 0x7f a speciální znaky unicode 0xfff0 až 0xffff. Výskyt těchto znaků v extrahované hodnotě může indikovat chybu při extrakci, chybu v datech souboru, či špatně vložená metadata uživatelem. Z testování vyplynulo, že se tyto znaky nachází většinou v některém z komentářových tagů. Jelikož může jít o chybu uživatelskou, rozhodl jsem se hodnoty s nepovolenými znaky nezahazovat, ale pouze odstranit dané znaky.

Zapisovací procedura dále kontroluje, zda-li řetězec ukazuje na hodnotu null, je prázdný nebo obsahuje pouze bílé znaky. V takovém případě není hodnota zapsána, protože nemá smysl. Dále nejsou zapsány hodnoty začínající řetězcem *unknown* a *undefined*. Takové hodnoty poskytuje Metadata Extraktor v případě, že nezná význam číselníkového kódu. Pokud by bylo uživatelem přesto chtěno zapsat hodnotu, může v definici požadovat zápis kódu místo překladu, či může definovat vlastní překlad číselníku.

Po zkontrolování zapisovaných dat procedura zjišťuje, zda-li je definován v ontologii oddělovač, kterým má být získaná hodnota rozdělena do několika řetězců. Každý z řetězců je pak zapsán dle stejné definice. Oddělovačem může být regulární výraz.

Hodnoty jsou zapsané jako literál s definovaným datovým typem, nebo jako objekt kontrolovaného slovníku. Při zapisování literálu datového typu XSDDateTime je volána pomocná funkce z třídy *DateUtil*, která se pokusí vstupní textovou hodnotu převést do hodnoty formátu Date. Pokud se převod nepodaří, není hodnota zapsána. Pojmy kontrolovaných slovníků lze použít pouze takové, které jsou definované v obrazové ontologii načtené při inicializaci objektu třídy *ExtractorLogic*. IRI pojmu musí mít tvar *VLASTNOST_HODNOTA*, kde *VLASTNOST* je IRI vlastnosti definované pro daný tag a *HODNOTA* je extrahovaný řetězec upravený předešlými kroky. Pokud prvek s daným IRI neexistuje, trojice nebude zapsána.

5.3 Datová třída IndividualData

Instance třídy *IndividualData* jsou používány k přístupu k definici mapování tagu. Třída má 16 atributů plynoucích ze struktury mapovací ontologie. Přístup k atributům je pomocí get a set metod.

Atribut *id* uchovává název instance *imageMetadata* v mapovací ontologii. Pokud má být stejný tag řešen i jiným mapováním, je odkaz na takovou instanci v atributu *duplicateTo*.

Integer *tag* je číslo, které má daný tag v adresáři Metadata Extractoru. Toto číslo je potřeba pokud je přistupováno k extrahovaným hodnotám jiných metadatových tagů, například při skládání hodnoty z několika tagů. Pořadí skládání hodnot je dané ve vlastnostech *follows* a *precedes*, které obsahují odkaz na předcházející, resp. následující část.

Atributy *namespace* a *attribute* obsahují jmenný prostor a název RDF vlastnosti, která bude použita jako predikát při zapisování získané hodnoty.

Se zapisovanou hodnotou souvisí atributy *description* a *writeAsNode* obsahující pravdivostní hodnotu, zda-li má být použita extrahovaná hodnota v člověkem čitelném tvaru, resp. zda-li má být v objektu použit pojem z kontrolovaného slovníku. Atributy *transformation* a *transformationParameter*, popisují způsob, jakým bude extrahovaná hodnota transformována a parametry transformace. V atributu *datatype* je datový typ zapisovaného literálu. Pokud má být hodnota rozdělena na několik podřetězců, je v atributu *splitDelimiter* uveden regulární výraz pro nalezení oddělovačů.

Má-li být subjektem trojice jiný zdroj než obrazový soubor, je na definici takového zdroje odkázáno v atributu *aggregatedIn*. Typ zdroje může být určen atributy *typeNamespace* a *typeClass*, které obsahují jmenný prostor a název třídy, jejíž instancí je zdroj.

Třída dále definuje tři funkce, které pracují s často používanými kombinacemi atributů. Funkce *isTyped()* vrací hodnotu true, pokud je v definici uzlu obsažena informace o jeho typu. To znamená, že atributy *typeNamespace* a *typeClass* nejsou null. Funkce *isDefined()* vrací true, je-li známý jmenný prostor a název vlastnosti v predikátu, tedy atributy *namespace* a *attribute* nejsou null. Funkce *needsConcat()* vrací true, pokud alespoň jeden z atributů *follows* a *precedes* odkazuje na objekt a je nutné spojovat extrahované hodnoty.

5.4 Balík util

5.4.1 Formátování logu

Třída *OneLinerLogFormat* je jednoduchá pomocná třída, která dědí z třídy *Formatter*, a slouží k poskytnutí vhodnějšího formátování pro ladící log. Třída překrývá metodu *format(LogRecord)* tak, aby byla do logu zapisována pouze informace o úrovni hlášení a jeho text. Čas vzniku události, původce události ani žádné další informace nejsou zapisovány. Záznam je zapsán do jednoho řádku logu.

Tento formát je zamýšlen k použití pro logy, které nemají sloužit k zapisování chyb při běhu programu, ale spíše k zaznamenávání údajů pro další zpracování. Takový log by měl být co nejstručnější, aby jeho zpracování bylo rychlé, a syntaxí jednoduchý, aby bylo možné získat potřebná data bez nutnosti tvorby složitých regulárních výrazů.

5.4.2 Práce s daty a časem

Třída *DateUtil* poskytuje statickou funkci pro přečtení data a času z textového vstupu, u kterého není znám datový formát. V privátní konstantě je definována sada formátů, které jsou postupně aplikovány na vstupní data. Pořadí, v jakém jsou aplikovány, je významné. Hledá se první formát, při jehož použití v parsovací funkci není vyhozena chyba *ParseException*. Datum a čas získaný použitím takového formátu je považován za správný, pokud je po 31. 12. 1969. Není-li podmínka splněna pokračuje se v cyklu dalším formátem.

Podmínka částečně eliminuje problém, kdy čas ve formátu *HHmmssZ*, například *181920-0000*, byl bez vykázání chyby interpretován jako formát *yyyyMMdd*. Hodnota však byla nesmyslná, v tomto případě *Mon Jul 31 01:00:00 GMT+01:00 1820*. Formát obsahující pouze časovou složku je silný v tom ohledu, že zpracuje bez chyby i vstupy obsahující pouze datum, leč samozřejmě nesprávně. Změna pořadí testování by v tomto případě problém neřešila, naopak by způsobila další problémy.

Původně jsem chtěl podmínku rozšířit na testování, že získaný datum je před současným systémovým datem. Nicméně tato podmínka by činila funkci

nekonzistentní v tom ohledu, že v závislosti na současném datu by vracela různé výstupy. Zároveň by mohl nastat problém u jednoho z IPTC tagů udávající datum platnosti obsahu. Tento datum z podstaty nemusí být před současným datem.

Při práci s daty používám anglický locale, jelikož řetězce, které poskytuje Metadata Extractor, jsou zapsané v anglické notaci. Funkce by samozřejmě v případě potřeby šla upravit, nebo přetížit, aby byl locale variabilní podle prostředí uživatele.

Řešení v této funkci nepracuje korektně s časovými zónami. Exif metadata s datovými a časovými hodnotami neobsahují údaj o časové zóně. Metadata Extractor sice zná tag 0x882A, který by informaci o zóně měl obsahovat, ale tag není čtený u žádného z testovacích souborů. U případů, kdy není zóna uvedena, předpokládá funkce pásmo UTC+0.

Pro třídu je napsán unit test zkoušející, že jsou korektně čteny řetězce všech definovaných formátů.

5.4.3 Transformace extrahovaných hodnot

V třídě *ValueTransformator* je definována statická funkce pro transformaci extrahovaných dat *dataTransformation(String, IndividualData)*. Vstupem jsou řetězec k transformaci a mapovací definice tagu. Funkce vrací změněný řetězec při standardním běhu. Pokud je požadována transformace, která není definována, nebo není použitelná pro daná data, je vrácen vstupní řetězec. Pokud dojde k chybě při zpracování, nebo je vstupní řetězec vadný, vrací se prázdný řetězec. Ke transformačním funkcím je vytvořen unit test.

Způsob transformace je pro tag definován odkazem na instanci transformace v mapovací ontologii. Funkce nejprve iterátorem projde vlastnosti transformace a hledá její decimální kód. Pokud byl kód nalezen volá pomocnou funkci, které předává kód transformace, parametry transformace a vstupní řetězec. Je definováno 16 různých transformací. V mapovací ontologii mají příslušné instance jména *transformation_KÓD*.

Před provedením transformace jsou z řetězce odstraněny bílé znaky na začátku a konci. Transformace s kódem 1 slouží pouze k nahrazení znaků mezera znakem tečka. Transformace je použitá u tagu GPS verze, jehož výstup je formátován jinak, než požaduje použitá ontologie.

Transformace s kódem 2 slouží k převodu velikosti úhlu z formátu úhly, minuty, vteřiny na desetinné číslo. Ve vstupním řetězci jsou očekávány 3 zlomky. Funkce rozdělí vstup na několik podřetězců za použití oddělovacího znaku mezera a lomítko. Z nich jsou pak čteny celočíselné hodnoty a vypočtena výsledná hodnota. Pokud není získáno 6 podřetězců nebo při čtení celých čísel dojde k chybě *NumberFormatException*, je vrácen prázdný řetězec. Jelikož Metadata Extractor vrací řetězce, ve kterých jsou desetinná čísla zapsána s desetinnou tečkou, převádím případné české desetinné čárky na tečky před vrácením hodnoty. Tato transformace se používá u několika Exif GPS tagů.

Kód 3 převede sérii celých čísel oddělených mezerou na textový řetězec tak, že každé číslo bude nahrazeno znakem odpovídající danému kódu ve výchozím kódování. Mezery nejsou zachovány. Ve většině případů umí tuto transformaci vykonat již Metadata Extractor a poskytuje ji jako přeloženou hodnotu tagu. Knihovna uvažuje více kódování a případně speciální charakteristiky řetězce v závislosti na metadatovém adresáři. U GPS tagů ale tuto možnost neposkytuje a používám zde proto tuto funkci.

Transformace číslo 4 převede racionální číslo na desetinné. Ze vstupu jsou odstraněny všechny písmenné znaky. Návrh programové logiky neumožňuje provést více než jednu transformaci extrahované hodnoty a v některých případech bylo potřeba kromě převodu i odstranit text. Naopak odstranění znaků v žádném z případů nevadí. Proto bylo přistoupeno k této variantě. Funkce dokáže převést i více čísel v jednom řetězci. Je ale potřeba, aby při rozdělení na podřetězce jako v tr. 2, byl sudý počet prvků. Při lichém počtu je vracen vstupní řetězec bez písmenných znaků. Při chybě *NumberFormatException* je vracen prázdný řetězec. Parametrem transformace je počet desetinných míst, nejméně je však jedno. Výsledný řetězec používá desetinnou tečku.

Transformace s kódem 5 převede sérii čísel oddělených mezerou na číslo verze takovým způsobem, že mezery budou odstraněny a na pozici oddělující verzi a podverzi bude vložena tečka. Pozice, ve smyslu počtu prvků, po kterých je tečka umístěna, je parametrem transformace. Při chybě čtení celých čísel, stejně jako při neplatném parametru, je vrácen prázdný řetězec. Transformace se uplatňuje u Exif tagu s verzí interoperability.

Transformace 6 je použita k převodu hodnoty počtu bitů na každou složku do tvaru požadovaného v NEXIF ontologii. Knihovna například vrací řetězec *8 8 8* pokud pro každou z barevných složek je vymezeno 8 bitů. NEXIF požaduje pouze jedno číslo platné pro všechny složky. Funkce vrací podřetězec

od začátku vstupu k prvnímu znaku mezera. Tento postup by nebyl korektní pro případ, kdy každá složka bude mít jiný počet bitů. S tím jsem se nicméně nesetkal a tedy to nepředpokládám.

Tag v JFIF adresáři používá jiný číselník pro jednotky rozlišení než ostatní adresáře. Pro převod na číslo odpovídající ostatním slouží transformace 8. Jde pouze o inkrementaci čísla na vstupu, neboť je celý číselník posunutý právě o jeden prvek. Transformace nekontroluje, zda-li je vstup z platného rozsahu. Kontrola tohoto typu je prováděna při zápisu hodnoty pomocí kontrolovaného slovníku. Transformaci je tedy možné použít i k jiným účelům.

Transformace 9 přidá za každé číslo na vstupu řetězec */100000*. Touto formou je řešen problém s PNG tagy barevnosti, které jsou uváděné jako celá čísla po vynásobení číslem 100 000. Je vracen prázdný řetězec při chybě čtení celých čísel. Řešení je velmi specifické a těžko použitelné u jiného tagu.

Transformace 10 převádí číselníkovou hodnotu PNG tagu s typem barev na hodnotu počtu barevných vzorků v pixelu. PNG připouští varianty odstíny šedi (0), RGB (2), paletové barvy (3), odstíny šedi s průhledností (4), a RGB s průhledností (6). Jsou definovány následující převody: 0 na 1, 2 na 3, 3 na 1, 4 na 2, 6 na 4. Ostatní vstupy jsou převedeny na prázdný řetězec.

V transformaci 11 je čtena celočíselná hodnota. Pokud je větší než 0, je vrácen řetězec *true*. V opačném případě je vrácen řetězec *false*. Při chybě čtení čísla je vrácen prázdný řetězec. Transformace se používá u PNG tagů při určení, zda-li je obraz transparentní a zda-li je použito prolínání (interlace). Dále je použita u některých makernote tagů.

Transformace číslo 12 slouží ke spojení dvou tagů obsahujících datum a čas do jedné hodnoty. Datum a čas je rozdělen do různých tagů ve formátu IPTC a v Olympus makernote. Při spojování dojde k vložení znaku středník mezi hodnoty, jak bylo popsáno v kapitole 5.2.2. Tento znak je využit jako oddělovač a každý podřetězec je převeden funkcí z *DateUtil* třídy na datum. Pokud se převod podaří, jsou data sečteny. Součet je převeden na řetězec a vrácen. V případě chyby při zpracování je vrácena původní hodnota. To je z důvodu, že při skládání řetězce ze dvou tagů, je transformace volána po načtení každého z tagů a po dokončení spojení. Při vracení prázdného řetězce jako v předchozích transformacích by se nikdy nevytvořil celý řetězec.

Kódem 13 je volána transformace odstraňující písmenné znaky ze vstupního řetězce. Používá se hlavně u tagů, které je potřeba získat z Metadata

Extractoru přeložené, ale není chtěné, aby řetězec obsahoval měrné jednotky, nebo jiné textové informace. Typické je užití u tagů clonového čísla, kde knihovna správně interpretuje data, ale přidá před číslo písmeno F.

Transformace 14 převede sérii celých čísel oddělených mezerou na jejich hexadecimální reprezentaci. Mezery nejsou zachovány. Přijímá pouze čísla v rozmezí 0 až 255. Čísla jsou vždy zapsána dvěma znaky. Pokud dojde k problému při čtení čísel je vrácen prázdný řetězec. Transformace je použita u Canon tagu obsahujícího unikátní identifikátor fotografie.

Transformací 15 lze převést textovou hodnotu z tagů popisujících použité digitální přiblížení na číslo požadované NEXIF specifikací. Pokud vstup začíná podřetězcem *no*, obsahuje *not used* nebo se rovná řetězci *off*, je vrácena hodnota 0. Pokud vstupní řetězec obsahuje *unknown*, je vrácen prázdný řetězec. V ostatních případech jsou odstraněny písmenné znaky a vrácen zbylý řetězec, protože knihovna poskytuje hodnoty zvětšení jako číslo následované písmenem x.

Transformace číslo 16 slouží k vydělení čísla na vstupu číslem zadaným v parametru transformace. Chyba při dělení, včetně dělení nulou, vrací prázdný řetězec. Počet desetinných míst je určen v závislosti na hodnotě výsledku. Vrácená hodnota obsahuje desetinnou tečku. Transformace je použita u dvou tagů Casio makernote, které používají špatné měrné jednotky.

Poslední transformace, s kódem 17, poskytuje mechanismus k nadefinování vlastních popisných hodnot pro číselníkové kódy. Parametrem funkce je řetězec obsahující textové hodnoty oddělené středníkem. Funkce rozdělí tento řetězec na podřetězce s tím, že první odpovídá indexu 0. Je vrácen popisek na indexu, který je dán vstupní hodnotou. Vrací prázdný řetězec, pokud vstup není celé číslo, nebo index neodpovídá žádnému prvku.

5.4.4 Třída ExtractorUtil

Tato třída obsahuje čtyři statické funkce volané třídami extraktorů nebo třídou *ExtractorLogic*. Pro všechny funkce kromě přejmenování uzlů je sestaven unit test.

Funkce *getMappingDefinition(Individual)* slouží k načtení mapovací definice do instance *IndividualData*, kterou pak vrací. Funkce iteruje nad všemi vlastnostmi uzlu předaného jako vstupní parametr. Jména vlastností porov-

nává s názvy známých vlastností mapovací definice. Při shodě je objekt vlastnosti přečten a zapsán do příslušného atributu mapovací instance. Vlastnost popisující typ literálu je jediný případ, kdy je čtený objekt nutné zpracovat před zapsáním.

K zpracování je použita funkce *getExpectedXSDDatatype(String)*, která převede textovou definici datového typu na příslušnou instanci *XSDDatatype* a tu pak vrací. Pro lepší uživatelskou přívětivost a syntaktickou benevolenci netestuji řetězec na vstupu na rovnost, ale pouze na výskyt specifického podřetězce. V důsledku nezáleží na tom, jestli v mapovací ontologii je definován požadovaný datový typ jako *http://www.w3.org/2001/XMLSchema#string* (IRI), *xsd:string* (prefixová cesta), *XSDstring* (instance Java třídy) nebo *string* (název primitivního datového typu). Funkce je definována pouze pro datové typy *xsd:string*, *xsd:integer*, *xsd:float*, *xsd:dateTime* a *xsd:boolean*. Pokud vstup neodpovídá žádnému z nich, je vrácena hodnota null.

Další funkcí je *isDefinitionGraphCycle(. . .)* hledající cyklus v definici objektových vlastností. Parametrem je načtená definice mapování tagu a enumerační hodnota popisující vlastnost k prozkoumání. Funkce je definována pouze pro vlastnosti *duplicateTo* a *isAggregatedIn*. Testování na výskyt cyklu u definice návaznosti při spojování je prováděna přímo ve funkci *concatValue*. Jelikož jsou obě podporované vlastnosti definované jako funkční, hledání probíhá lineárním průchodem grafu od uzlu ze vstupního parametru přes danou vlastnost. Pokud je navštíven uzel, který byl navštíven již dříve, je definice vlastnosti zacyklená a je vrácena pravdivostní hodnota true. Pokud pro procházený uzel není vlastnost definována, je vrácena hodnota false.

Poslední funkce, *renameNodes(HashMap<String, Resource>, RDFUtil)*, slouží k přejmenování uzlů na základě vlastností, které jsou pro ně definovány. Vstupním parametrem je mapa názvů vytvořených uzlů na jejich instanci a instance RDFUtil pro zapisování do RDF modelu. Funkce pro všechny vytvořené uzly, kromě uzlu obrazového souboru, projde jejich vlastnosti. Řetězec vzniklý sloučením IRI vlastnosti a textové serializace objektu vloží do seznamu. Po přečtení všech vlastností je seznam abecedně seřazen. Uzel je pak přejmenován na název odpovídající výstupu SHA-1 hashovací funkce nad seznamem.

5.5 Vytvořené ontologie

V projektu využívám dvě vlastní ontologie. První z nich obsahuje definici mapování extrahovaných tagů. Druhou ontologií vytvářím vlastnosti, které vyplývají ze zpracovávaných metadat a neexistují v jiném slovníku či ontologii. RDF/XML serializace obou ontologií jsou na přiloženém DVD.

5.5.1 Mapovací ontologie

Definice mapování by jistě mohla být zanesena do zdrojových kódů programu. Takové řešení je ale nepraktické při potřebě provést změnu, protože je nutné celý projekt znovu zkompilovat. Rozhodl jsem se tedy pro vynesení definice do externího souboru i za cenu jejího složitějšího čtení.

Ontologie obsahuje instanci třídy *ImageMetadata* pro všechny metadatové tagy, které jsou definovány v Metadata Extractoru 2.7.2 u souborových typů JPEG, PNG a TIFF. Názvy prvků jsou složeny ze zkratky názvu adresáře, do kterého knihovna tag řadí, a hexadecimálního čísla identifikující tag. Například prvek vztahující se k Exif tagu s označením modelu zařízení má název *exifIFD0_0x0110*. Aby šlo lépe poznat, co prvek popisuje, uvádím u každé vlastnosti anotací *rdfs:label* textovým popisek. Předchozí prvek má popisek *TAG_MODEL*. Popisky jsou získány ze zdrojových kódů knihovny, konkrétně jde o název konstanty uchovávající číslo tagu. Takový popisek mi připadá jako dostatečný pro potřeby ontologie.

Prvky mohou definovat mapování pomocí deseti datových a pěti objektových vlastností. Jejich užití bylo popsáno v kapitole 5.3. Vlastnosti mají definovanou doménu i obor hodnot. Všechny vlastnosti jsou funkční, tzn. prvek může mít každou vlastnost definovanou maximálně jednou. Pokud by například měl být tag zapsán pomocí dvou dalších definic, bude ve vlastnosti prvku uveden odkaz pouze na jednu definici a v té bude odkaz na druhou definici.

Kromě prvků odpovídajících známým metadatovým tagům se v ontologii nachází prvky, jejichž název začíná slovem *duplicate*. Jedná se o prvky obsahující další definici mapování k danému tagu. Ty využívám v případě, kdy vlastnost, na kterou je tag mapován, vyžaduje v objektu číselníkový kód. Pro takový tag použiji druhé mapování na vlastnost v mojí ontologii, která využívá kontrolovaný slovník. Dalšími zvláštními prvky jsou ty, jenž obsahují

v názvu *agg*. Jde o definici uzlů, které jsou v některém z mapování použity na pozici subjektu v RDF trojici. Například *gps_agg1* je definován jako uzel typu *geo:Point*, na který budou dle definic *gps_0x0002*, *gps_0x0004* a *gps_0x0006* připojena extrahovaná metadata o zeměpisné šířce, délce a nadmořské výšce.

V ontologii je zavedena třída *DataTransformation*, která má definováno 16 instancí. V definici transformace je vždy odkazováno na jednu z instancí. Instance mají definovaný pouze číselný kód transformace a několik anotačních vlastností pro popis jejich funkce. Transformace jsou blíže popsány v kapitole 5.4.3 zabývající se jejich programovou implementací.

Při tvorbě mapování obrazových metadat na RDF vlastnosti byly nejprve nalezeny existující slovníky řešící podobný problém, viz kapitola 3.2. Jako základ byla zvolena NEXIF ontologie, která poskytovala nejvíce relevantních pojmů. U většiny pojmů byl v popisu odkaz na Exif tag, kterému odpovídají. Následně byly na NEXIF ontologii namapovány i tagy se stejným významem z metadatových adresářů jiných než Exif. Poté byly v ostatních ontologiích hledány pojmy využitelné pro dosud nenamapované tagy. Po prohledání známých slovníků jsem pro každou definici mapování doplnil požadovaný datový typ. V dalším kroku se porovnaly přeložené a nepřeložené výstupy Metadata Extraktoru u každého tagu a rozhodl jsem o volbě výstupu a transformaci. Nakonec se přidaly do definice odkazy na uzel, na který mají být metadata navázána, pokud to není uzel obrazu.

Ve druhé fázi byla vytvořena ontologie pro tagy, které se nepodařilo namapovat a proběhlo mapování na tyto tagy. Poté byly opět nadefinovány datové typy, typ čteného výstupu z knihovny a metoda transformace.

Ve třetí fázi byly pro některé tagy mapované na existující ontologie vytvořeny vlastnosti i v mé ontologii využívající kontrolované slovníky. Byla doplněna definice daných prvků, aby odkazovala na další definici.

5.5.2 Pojmová ontologie

Tato ontologie definuje třídy *Image*, *ICC*, *Camera* a *Resolution* sloužící k vymezení domény definovaných vlastností. Kromě těchto tříd je definovaná třída CodeList a její podtřídy, jejichž instance tvoří kontrolované slovníky vybraných vlastností. Jde o vlastnosti, které v původním navrženém mapování vyžadují ukládat číselníkové kódy. Aby bylo možné zapsat údaje i ve tvaru čitelném pro člověka je vytvořeno další mapování na vlastnosti této ontologie.

Ontologie obsahuje objektové vlastnosti *createdWithCamera*, *hasIccProfile* a *hasResolution*, kterými lze propojit instanci *Image* s instancemi ostatních tříd zde definovaných. Vlastnost *hasResolution* a vlastnosti s ní související jsou zde definovány i přesto, že v NEXIF ontologii jsou dostupné. Důvodem je potřeba vytvořit strukturu, do které půjde zapsat kombinace hodnot tagů pro rozlišení na ose x, y a měrnou jednotku. V NEXIF jsou všechny vlastnosti vázány přímo na uzel obrazového souboru a může dojít ke konfliktu, pokud je například v Exif tagu definována sada hodnot *72, 72, DPI* a v makernote bude sada hodnot *1, 1, proporcionálně*. Obě sady jsou sami o sobě správné, ale při jejich mapování na stejnou NEXIF vlastnost se stanou strojově, a v některých případech i pro člověka, nepoužitelné. Zavedením třídy *Resolution* je možné sady hodnot od sebe oddělit.

Datové vlastnosti jsou rozdělené do dvou skupin na společné pro více tagů a odpovídající pouze jednomu tagu. Ty, které odpovídají pouze jednomu tagu, byly skriptem sesbírány ze zdrojových kódů knihovny. Skript načetl popisek odpovídající názvu tagu v knihovně a komentář, pokud byl nějaký nad definicí tagu. U vlastností byla definována doména *Image*. Společné tagy byly vybrány na základě analýzy nenamapovaných tagů. Tyto vlastnosti byly vytvářeny manuálně a mají definovaný i obor hodnot a detailnější popis v anotaci.

6 Diskuze výsledku

6.1 Výkonnostní metriky

Plugin jsem otestoval s verzí MetaMed 2.0.0-r201504231326. Použil jsem obrazové soubory, které volně poskytuje autor Metadata Extractoru. Testoval jsem na stroji Intel Core i3-2100 3,1 GHz s 8 GB pamětí RAM a operačním systému Windows 7, 64 bitová verze. Na stejném stroji jsem poté spustil virtuální stroj s operačním systémem Debian GNU/Linux 6 s přidělenými 4 GB RAM a otestoval plugin i zde.

V tabulce 6.1 uvádím srovnání naměřených časů při zpracování 277 obrazových souborů pomocí MetaMedu na stroji s OS Windows a na virtuálním stroji s OS Debian. Dále v posledním sloupci uvádím dobu čtení stejných souborů samotnou knihovnou na stejném stroji. V tabulce je čas prvního běhu a průměrný čas tří následujících běhů. Na obou operačních systémech dokázal MetaMed zpracovat soubory za přibližně 16 vteřin, což odpovídá průměrně 58 ms na jeden soubor. Tato rychlost se jeví jako přijatelná, V případě, kdy soubory již byly načteny v paměti z předchozího běhu, se doba zpracování na OS Windows zkrátila na polovinu. Důvodem, proč tomu tak nebylo i na OS Debian, může být omezení dostupné paměti na 4 GB nebo samotná virtualizace.

	Windows	Debian	Metadata Extractor
Čas prvního běhu	16 376 ms	16 222 ms	10 747 ms
Čas ostatních běhů	8 423 ms	15 882 ms	909 ms

Tabulka 6.1: Doba zpracování testovací sady programem MetaMed a samotnou extrahovací knihovnou.

Statistiky získané ze zpracování celé obrazové sady jsou v tabulce 6.2. Sleduji počet zapsaných trojic, prázdných či neznámých hodnot, neexistujících definic a prázdných definic. Neexistující definice znamená, že pro daný tag nebyla v ontologii nalezena žádná definice. Prázdná definice znamená, že záměrně není definováno mapování pro daný tag a v ontologii je jen prázdná instance. Vysoký počet neexistujících definic (11,2 %) by mohl indikovat špatně získanou sadu čtených tagů při tvorbě ontologie. Po prozkoumání ladícího logu se ukázalo, že 91 % nenalezených tagů jsou makernote metadata, která

nejsou vyjmenována ve zdrojových kódech tříd **MakernoteDirectory* a není zaručeno jejich správné načtení. Zbylých 9 % případů jsou Exif tagy, které rovněž nejsou k nalezení v příslušných třídách. V tomto ohledu tedy funguje plugin správně.

	Počet	Relativně
Zapsané trojice	21786	81,1 %
Prázdná či neznámá hodnota	865	3,2 %
Neexistující definice	3013	11,2 %
Prázdná definice	1193	4,4 %

Tabulka 6.2: Statistiky zpracovaných metadat.

V ideálním případě by měla testovací sada obsahovat každý tag alespoň jednou, aby bylo ověřeno, že je hodnota z tagu správně čtena. Najít sadu obrázků pokrývající všechny tagy je velmi těžké. V tabulce 6.3 porovnávám kolik ze všech známých tagů Metadata Extractoru je obsaženo v použité testovací sadě obrázků. Neúplné pokrytí mi dělalo problémy při analýze metadat a hledání možností kam je namapovat. Takové tagy jsem většinou nechal namapované na vlastnost mojí ontologie, protože jsem nedokázal odhadnout, v jakém tvaru je bude knihovna poskytovat.

	Známých tagů	Ověřených tagů	Podíl ověřených
Exif Thumbnail	20	20	100 %
XMP	15	15	100 %
JIF	9	9	100 %
Adobe JPEG	4	4	100 %
JFIF	4	4	100 %
Png	24	23	95,8 %
Exif IFD0	22	21	95,5 %
Makernote	722	600	83,1 %
Exif Interoperability	5	4	80 %
Exif GPS	31	24	77,4 %
Exif SubIFD	111	69	62,2 %
ICC	65	36	55,4 %
Photoshop	43	21	48,8 %
IPTC	77	26	33,8 %

Tabulka 6.3: Srovnání počtu známých tagů a tagů obsažených v testovací obrazové sadě.

6.2 Srovnání s řešením jiných autorů

Jak již bylo zmíněno v kapitole 3.1.4, není žádný aktivní projekt, který by se zabýval čtením obrazových metadat a jejich zápisem do RDF. Jediný významný projekt, Aperture framework, byl v roce 2011 zastaven z důvodu ztráty zájmu hlavního vývojáře o RDF.

Aperture u souborů JPEG rozpoznává 20 základních Exif tagů a 3 GPS tagy. Můj plugin definuje mapování 314 tagů na existující slovníky, z toho je 107 tagů z Exif Makernote adresáře. V tomto ohledu je plugin výrazným pokrokem, protože zpracovává i další Exif IFD a metadatové formáty. Navíc pracuje i se soubory TIFF a PNG. Aperture mapuje všechny tagy na vlastnosti ontologie NEXIF. Já také vycházím z této ontologie. Společně s ní dále používám Dublin Core, Friend of a Friend, Geo a další slovníky projektu NEPOMUK. Také jsem vytvořil vlastní slovník pro tagy, které nelze přiřadit žádné jiné RDF vlastnosti. Slovník kromě pojmů pro jednotlivé tagy definuje i 51 společných pojmů.

Jistým způsobem by za podobný projekt šel označit metadatový formát XMP, který funguje na principu RDF. Dle jeho specifikací a výpisu známých XMP tagů na stránkách ExifTool bych odhadoval, že svým rozsahem předčí mojí práci. Překážkou v užívání je však nemožnost získat serializaci XMP slovníku a ontologií.

Při zaměření se pouze na extrakci obrazových metadat, je významným nástrojem ExifTool, jehož autorem je Phil Harvey. Nástroj rozeznává více tagů než mnou používaný Metadata Extractor.

6.3 Problémy řešení

Hlavním problémem řešení bych označil malý počet rozeznávaných XMP tagů. Použitá knihovna je v tomto ohledu slabá a v budoucnu by stálo za prozkoumání, zda-li by nebylo možné s XMP tagy pracovat jiným způsobem. Ideální by bylo využít faktu, že se jedná o RDF model a pracovat s ním pomocí nástrojů Jena frameworku.

Nikde v procesu zpracování získaných metadat a jejich zapsání do RDF modelu nekontroluji, zda-li vlastnost použitá při mapování existuje. Řešení funguje i v případě, že vlastnost neexistuje v daném slovníku. Uživatel nemá

zpětnou vazbu o případné chybě vzniklé při upravování ontologické definice mapování. Ověřování během extrakce by celý proces výrazně zpomalilo. Vhodnější řešením by bylo vytvoření samostatného nástroje pro kontrolu definice ontologie.

V časových údajích z Exif adresáře není korektně řešena časová zóna. U neznámých časových pásem předpokládá program GMT+0. Řešení by mohlo spočívat v odhadu časové zóny dle GPS lokace, nebo přečtením některého tagu jiného adresáře obsahující časovou značku a zónu.

Hodnoty některých makernote tagů by mohly být lépe mapované na existující vlastnosti. Bohužel nemám dostatek testovacích souborů k ověření platnosti takového mapování.

Závěr

Téma práce vyvstalo z potřeby výzkumné skupiny Medicínské informační systémy rozšířit stávající aplikaci MetaMed o možnost extrahovat metadata z obrazových souborů a zapsat je do RDF modelu.

Rozšíření má formu pluginu k aplikaci MetaMed v jazyce Java. Celá definice způsobu zapsání získaných hodnot do RDF modelu je uchována v samostatné ontologii pro tento účel navržené. Je vytvořena i další ontologie definující pojmy pro metadata, která nešlo namapovat na stávající ontologie.

Práce je členěna do šesti částí. V první části se zabývám problematikou RDF, ontologiemi a principy sémantického webu. Ve druhé části popisuji obrazové soubory typu JPEG, PNG, TIFF a metadata v nich obsažená. Ve třetí kapitole jsou představené vybrané existující nástroje pro práci s obrazovými metadaty a RDF slovníky, které se těchto metadat týkají. V další části zdůvodňuji výběr knihovny využité k extrahování metadat ze souborů. V páté kapitole popisuji implementaci rozšíření pro MetaMed. Na závěr hodnotím výsledné softwarové dílo a srovnávám ho s obdobnými projekty jiných autorů.

Všechny body zadání byly v práci splněny a softwarové řešení vyhovuje požadavkům kladeným na projekt.

Použité zkratky

EXIF	Exchangeable Image File Format	str. 13
FOAF	Friend of a Friend	str. 25
ICC	International Color Consortium	str. 16
IFD	Image file directory	str. 13
IIM	Information Interchange Model	str. 15
IPTC	International Press Telecommunications Council	str. 15
IRI	Internationalized Resource Identifier	str. 3
JFIF	JPEG File Interchange Format	str. 13
JIF	JPEG Interchange Format	str. 12
JPEG	Joint Photographic Experts Group	str. 12
MIME	Multipurpose Internet Mail Extension	str. 13
NEPOMUK	Networked Environment for Personal, Ontology-based Management of Unified Knowledge	str. 27
NEXIF	NEPOMUK EXIF ontologie	str. 27
NIE	NEPOMUK Information Element	str. 27
OWL	Web Ontology Language	str. 8
PNG	Portable Network Graphics	str. 17
PSIR	Photoshop Image Resources	str. 16
RDF	Resource Description Framework	str. 2
RDFS	RDF Schema	str. 6
TIFF	Tagged Image File Format	str. 19
W3C	World Wide Web Consortium	str. 2
XMP	Extensible metadata platform	str. 14

Literatura

[1] SCHREIBER, G. – RAIMOND, Y. *RDF 1.1 Primer* [online]. 2014. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[2] *W3C Mission* [online]. 2015. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/Consortium/mission.

[3] DUERST, M. – SUIGNARD, M. RFC 3987: Internationalized Resource Identifiers (IRIs). IETF (January 2005). Dostupné z: http://www.ietf.org/rfc/rfc3987.txt.

[4] CYGANIAK, R. – WOOD, D. – LANTHALER, M. *RDF 1.1 Concepts and Abstract Syntax* [online]. 2014. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[5] CHEN, L. et al. Blank Nodes in RDF. *Journal of Software*. 2012, 7, 9, s. 1993–1999.

[6] BERNERS-LEE, T. *Linked Data - Design Issue* [online]. 2009. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/DesignIssues/LinkedData.html.

[7] HITZLER, P. et al. *OWL 2 Web Ontology Language Primer (Second Edition)* [online]. 2012. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[8] BRICKLEY, D. – GUHA, R. *RDF Schema 1.1* [online]. 2014. [cit. 22.4.2015]. Dostupné z: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[9] MAZZOCCHI, S. *Closed World vs. Open World: the First Semantic Web Battle* [online]. 2005. [cit. 22.4.2015]. Dostupné z: http://www.betaversion.org/~stefano/linotype/news/91/.

[10] *Raster Graphic Definition* [online]. 2015. [cit. 28.4.2015]. Dostupné z: http://techterms.com/definition/rastergraphic.

[11] *Vector Graphic Definition* [online]. 2015. [cit. 28.4.2015]. Dostupné z: http://techterms.com/definition/vectorgraphic.

[12] MATTHEWS, R. *Digital image file types* [online]. [cit. 28.4.2015]. Dostupné z: `http://users.wfu.edu/matthews/misc/graphics/formats/formats.html`.

[13] ISO-IEC, I. 10918. *Digital compression and coding of continuous–tone still images*. 1992.

[14] WALLACE, G. K. The JPEG still picture compression standard. *Consumer Electronics, IEEE Transactions on*. 1992, 38, 1, s. xviii–xxxiv.

[15] ISO-IEC, I. 10918-1. *Digital compression and coding of continuous–tone still images*. 1992.

[16] NHU, T. *The Metadata in JPEG files - Exiv2* [online]. 2015. [cit. 28.4.2015]. Dostupné z: `http://dev.exiv2.org/projects/exiv2/wiki/The_Metadata_in_JPEG_files`.

[17] FREED, N. – BORENSTEIN, N. Multipurpose internet mail extensions (MIME) part two: Media types. Technical report, rfc 2046, November, 1996.

[18] HAMILTON, E. JPEG file interchange format. *C-Cube Microsystems*. 1992.

[19] *Exchangeable image file format for digital still cameras: Exif Version 2.3* [online]. 2012. [cit. 28.4.2015]. Dostupné z: `http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf`.

[20] HUGGEL, A. *Exiv2 - Image metadata library and tools* [online]. 2014. [cit. 28.4.2015]. Dostupné z: `http://www.exiv2.org/makernote.html`.

[21] *Adobe XMP Specification Part 3: Storage in Files* [online]. 2014. [cit. 28.4.2015]. Dostupné z: `http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2014-12/XMPSpecificationPart3.pdf`.

[22] *Extensible Metadata Platform (XMP) Specification: Part 1, Data Model, Serialization, and Core Properties* [online]. 2012. [cit. 28.4.2015]. Dostupné z: `http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2014-12/XMPSpecificationPart1.pdf`.

[23] *Adobe XMP Specification Part 2: Additional Properties* [online]. 2014. [cit. 28.4.2015]. Dostupné z: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2014-12/XMPSpecificationPart2.pdf.

[24] *Information Interchange Model Version 4* [online]. 2014. [cit. 28.4.2015]. Dostupné z: http://www.iptc.org/std/IIM/4.2/specification/IIMV4.2.pdf.

[25] *IIM Schema for XMP* [online]. 2008. [cit. 28.4.2015]. Dostupné z: http://www.iptc.org/std/IIM/4.1/specification/IPTC-IIM-Schema4XMP-1.0-spec_1.pdf.

[26] *Guidelines for Handling Image Metadata* [online]. 2010. [cit. 28.4.2015]. Dostupné z: http://metadataworkinggroup.com/pdf/mwg_guidance.pdf.

[27] *Adobe Photoshop File Formats Specification* [online]. 2013. [cit. 28.4.2015]. Dostupné z: http://www.adobe.com/devnet-apps/photoshop/fileformatashtml/.

[28] *Image technology colour management — Architecture, profile format, and data structure* [online]. 2010. [cit. 28.4.2015]. Dostupné z: http://www.color.org/specification/ICC1v43_2010-12.pdf.

[29] *Portable Network Graphics (PNG) Specification (Second Edition)* [online]. 2003. [cit. 28.4.2015]. Dostupné z: http://www.w3.org/TR/2003/REC-PNG-20031110/.

[30] RANDERS-PEHRSON, G. *Converting TIFF/EP or Exif files to and from the PNG format (Draft 0.20, December 20, 2000)* [online]. 2000. [cit. 28.4.2015]. Dostupné z: http://pmt.sourceforge.net/exif/drafts/history/d020.html.

[31] *TIFF Revision 6.0* [online]. 1992. [cit. 28.4.2015]. Dostupné z: http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

[32] HUGGEL, A. *Exiv2 - Image metadata library and tools* [online]. 2014. [cit. 30.4.2015]. Dostupné z: http://www.exiv2.org/.

[33] HARVEY, P. *ExifTool by Phil Harvey* [online]. 2015. [cit. 30.4.2015]. Dostupné z: http://owl.phy.queensu.ca/~phil/exiftool/.

[34] NOAKES, D. *drewnoakes/metadata-extractor · GitHub* [online]. 2015. [cit. 30.4.2015]. Dostupné z: `https://github.com/drewnoakes/metadata-extractor`.

[35] *Aperture Framework* [online]. 2015. [cit. 30.4.2015]. Dostupné z: `http://aperture.sourceforge.net/`.

[36] MYLKA, A. *Aperture / Feature Requests / #118 Advice please on extracting embedded files from RTF docs* [online]. 2014. [cit. 30.4.2015]. Dostupné z: `http://sourceforge.net/p/aperture/feature-requests/118/#e875`.

[37] *DCMI Metadata Terms* [online]. 2012. [cit. 30.4.2015]. Dostupné z: `http://dublincore.org/documents/2012/06/14/dcmi-terms/`.

[38] BRICKLEY, D. – MILLER, L. *FOAF Vocabulary Specification 0.99* [online]. 2014. [cit. 30.4.2015]. Dostupné z: `http://xmlns.com/foaf/spec/20140114.html`.

[39] KANZAKI, M. *Exif data description vocabulary* [online]. 2007. [cit. 30.4.2015]. Dostupné z: `http://www.kanzaki.com/ns/exif`.

[40] MYLKA, A. et al. *Nepomuk Information Element Ontology (NIE)* [online]. 2012. [cit. 30.4.2015]. Dostupné z: `http://www.semanticdesktop.org/ontologies/2007/01/19/nie/`.

[41] MYLKA, A. et al. *Nepomuk File Ontology (NFO)* [online]. 2013. [cit. 30.4.2015]. Dostupné z: `http://www.semanticdesktop.org/ontologies/2007/03/22/nfo/`.

[42] MYLKA, A. et al. *Nepomuk EXIF Ontology (NEXIF)* [online]. 2012. [cit. 30.4.2015]. Dostupné z: `http://www.semanticdesktop.org/ontologies/2007/05/10/nexif/`.

[43] BRICKLEY, D. *W3C Semantic Web Interest Group: Basic Geo (WGS84 lat/long) Vocabulary* [online]. 2006. [cit. 30.4.2015]. Dostupné z: `http://www.w3.org/2003/01/geo/`.

Přílohy

Struktura přiloženého DVD

Na přiložením DVD jsou následující složky a soubory:

- **latex** obsahující zdrojové kódy dokumentace.
- **MetaMed** obsahující aplikaci MetaMed s pluginem pro zpracování metadat z obrazových souborů. Ve složce jsou tři .bat soubory sloužící k ukázce práce MetaMedu a extraktorů.
- **metamed-extractor-image** se zdrojovými kódy pluginu ke zpracování metadat obrazových souborů.
- **srovnani** obsahující obrazové soubory, kterými byla testována schopnost nástrojů pracovat s obrazovými metadaty v kapitole 4.2. Pro každý nástroj je vytvořena složka se seznamy nalezených metadat v jednotlivých souborech. V adresáři je tabulka se zaznamenanými počty získaných metadat.
- **dip.pdf** s textem diplomové práce.

Výpis nalezených metadat v souboru Sanyo SR662.jpg knihovnou Metadata Extractor.

```
[Exif IFD0 - 0x010e] Image Description = SANYO DIGITAL CAMERA
[Exif IFD0 - 0x010f] Make = SANYO Electric Co.,Ltd.
[Exif IFD0 - 0x0110] Model = SR662
[Exif IFD0 - 0x0112] Orientation = Top, left side (Horizontal / normal)
[Exif IFD0 - 0x011a] X Resolution = 72 dots per inch
[Exif IFD0 - 0x011b] Y Resolution = 72 dots per inch
[Exif IFD0 - 0x0128] Resolution Unit = Inch
[Exif IFD0 - 0x0131] Software = V662U-71
[Exif IFD0 - 0x0132] Date/Time = 2001:09:15 18:11:27
[Exif IFD0 - 0x0213] YCbCr Positioning = Datum point
[Exif SubIFD - 0x829a] Exposure Time = 0.25 sec
[Exif SubIFD - 0x829d] F-Number = F2.8
[Exif SubIFD - 0x9000] Exif Version = 2.00
[Exif SubIFD - 0x9003] Date/Time Original = 2001:09:15 18:11:27
[Exif SubIFD - 0x9004] Date/Time Digitized = 2001:09:15 18:11:27
[Exif SubIFD - 0x9101] Components Configuration = YCbCr
[Exif SubIFD - 0x9102] Compressed Bits Per Pixel = 3.5 bits/pixel
[Exif SubIFD - 0x9204] Exposure Bias Value = 0 EV
[Exif SubIFD - 0x9205] Max Aperture Value = F2.8
[Exif SubIFD - 0x9207] Metering Mode = Multi-segment
[Exif SubIFD - 0x9208] White Balance = Unknown
[Exif SubIFD - 0x9209] Flash = Flash did not fire
[Exif SubIFD - 0x920a] Focal Length = 6.0 mm
[Exif SubIFD - 0xa000] FlashPix Version = 1.00
[Exif SubIFD - 0xa001] Color Space = Undefined
[Exif SubIFD - 0xa002] Exif Image Width = 1024 pixels
[Exif SubIFD - 0xa003] Exif Image Height = 768 pixels
[Exif SubIFD - 0xa004] Related Sound File =
[Exif SubIFD - 0xa300] File Source = Digital Still Camera (DSC)

[Exif Thumbnail - 0x0103] Thumbnail Compression = JPEG (old-style)
[Exif Thumbnail - 0x011a] X Resolution = 72 dots per inch
[Exif Thumbnail - 0x011b] Y Resolution = 72 dots per inch
[Exif Thumbnail - 0x0128] Resolution Unit = Inch
[Exif Thumbnail - 0x0201] Thumbnail Offset = 878 bytes
[Exif Thumbnail - 0x0202] Thumbnail Length = 4562 bytes

[File - 0x0001] File Name = Sanyo SR662.jpg
[File - 0x0002] File Size = 18526 bytes
[File - 0x0003] File Modified Date = Sun Jan 27 02:31:16 GMT 2013

[JFIF - 0x0005] Version = 1.1
[JFIF - 0x0007] Resolution Units = inch
[JFIF - 0x0008] X Resolution = 72 dots
[JFIF - 0x000a] Y Resolution = 72 dots

[JPEG - 0xfffffffd] Compression Type = Baseline
[JPEG - 0x0000] Data Precision = 8 bits
[JPEG - 0x0001] Image Height = 225 pixels
[JPEG - 0x0003] Image Width = 300 pixels
[JPEG - 0x0005] Number of Components = 3
[JPEG - 0x0006] Component 1 = Y component: Quantization table 0, Sampling factors 1 horiz/1 vert
[JPEG - 0x0007] Component 2 = Cb component: Quantization table 1, Sampling factors 1 horiz/1 vert
[JPEG - 0x0008] Component 3 = Cr component: Quantization table 1, Sampling factors 1 horiz/1 vert

[Sanyo Makernote - 0x0200] Special Mode = 33554691 1114897 3502768191
[Sanyo Makernote - 0x0201] Sanyo Quality = Normal/Medium High
[Sanyo Makernote - 0x0f00] Data Dump = [20 longs]
```

Výpis nalezených metadat v souboru Sanyo SR662.jpg knihovnou Exiv2.

```
Exif.Image.ImageDescription                 Ascii       21  SANYO DIGITAL CAMERA
Exif.Image.Make                             Ascii       24  SANYO Electric Co.,Ltd.
Exif.Image.Model                            Ascii        6  SR662
Exif.Image.Orientation                      Short        1  top, left
Exif.Image.XResolution                      Rational     1  72
Exif.Image.YResolution                      Rational     1  72
Exif.Image.ResolutionUnit                   Short        1  inch
Exif.Image.Software                         Ascii        9  V662U-71
Exif.Image.DateTime                         Ascii       20  2001:09:15 18:11:27
Exif.Image.YCbCrPositioning                 Short        1  Co-sited
Exif.Image.ExifTag                          Long         1  294
Exif.Photo.ExposureTime                     Rational     1  1/4 s
Exif.Photo.FNumber                          Rational     1  F2.8
Exif.Photo.ExifVersion                      Undefined    4  2.00
Exif.Photo.DateTimeOriginal                 Ascii       20  2001:09:15 18:11:27
Exif.Photo.DateTimeDigitized                Ascii       20  2001:09:15 18:11:27
Exif.Photo.ComponentsConfiguration          Undefined    4  YCbCr
Exif.Photo.CompressedBitsPerPixel           Rational     1  3.5
Exif.Photo.ExposureBiasValue                SRational    1  0 EV
Exif.Photo.MaxApertureValue                 Rational     1  F2.8
Exif.Photo.MeteringMode                     Short        1  Multi-segment
Exif.Photo.LightSource                      Short        1  Unknown
Exif.Photo.Flash                            Short        1  No flash
Exif.Photo.FocalLength                      Rational     1  6.0 mm
Exif.Photo.MakerNote                        Undefined  142  (Binary value suppressed)
Exif.Photo.FlashpixVersion                  Undefined    4  1.00
Exif.Photo.ColorSpace                       Short        1  Uncalibrated
Exif.Photo.PixelXDimension                  Long         1  1024
Exif.Photo.PixelYDimension                  Long         1  768
Exif.Photo.RelatedSoundFile                 Ascii       13
Exif.Photo.FileSource                       Undefined    1  Digital still camera
Exif.Thumbnail.Compression                  Short        1  JPEG (old-style)
Exif.Thumbnail.XResolution                  Rational     1  72
Exif.Thumbnail.YResolution                  Rational     1  72
Exif.Thumbnail.ResolutionUnit               Short        1  inch
Exif.Thumbnail.JPEGInterchangeFormat        Long         1  878
Exif.Thumbnail.JPEGInterchangeFormatLength  Long         1  4562
```

Výpis nalezených metadat v souboru Sanyo SR662.jpg knihovnou Exiftool.

```
ExifTool Version Number         : 9.93
File Name                       : Sanyo SR662.jpg
Directory                       : .
File Size                       : 18 kB
File Modification Date/Time     : 2015:02:15 14:26:23+01:00
File Access Date/Time           : 2015:05:13 01:06:26+01:00
File Creation Date/Time         : 2015:05:13 01:06:26+01:00
File Permissions                : rw-rw-rw-
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Little-endian (Intel, II)
Image Description               : SANYO DIGITAL CAMERA
```

```
Make                            : SANYO Electric Co.,Ltd.
Camera Model Name               : SR662
Orientation                     : Horizontal (normal)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Software                        : V662U-71
Modify Date                     : 2001:09:15 18:11:27
Y Cb Cr Positioning             : Co-sited
Exposure Time                   : 1/4
F Number                        : 2.8
Exif Version                    : 0200
Date/Time Original              : 2001:09:15 18:11:27
Create Date                     : 2001:09:15 18:11:27
Components Configuration        : Y, Cb, Cr, -
Compressed Bits Per Pixel       : 3.5
Exposure Compensation           : 0
Max Aperture Value              : 2.8
Metering Mode                   : Multi-segment
Light Source                    : Unknown
Flash                           : No Flash
Focal Length                    : 6.0 mm
Special Mode                    : 0 0 0
Sanyo Quality                   : Normal/Medium High
Data Dump                       : (Binary data 152 bytes, use -b option to extract)
Flashpix Version                : 0100
Color Space                     : Uncalibrated
Exif Image Width                : 1024
Exif Image Height               : 768
Related Sound File              :
File Source                     : Digital Camera
Compression                     : JPEG (old-style)
Thumbnail Offset                : 908
Thumbnail Length                : 4562
Image Width                     : 300
Image Height                    : 225
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:4:4 (1 1)
Aperture                        : 2.8
Image Size                      : 300x225
Megapixels                      : 0.068
Shutter Speed                   : 1/4
Thumbnail Image                 : (Binary data 4562 bytes, use -b option to extract)
Focal Length                    : 6.0 mm
```

Výpis nalezených metadat v souboru Sanyo SR662.jpg frameworkem Aperture.

```
<file:/C:/Users/Keiras/Desktop/New%20folder%20(6)/Sanyo%20SR662.jpg> <http://www.semanticdesktop.
    org/ontologies/2007/05/10/nexif#height> "768" ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#width> "1024" ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#exposureBiasValue> "0" ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#exposureTime> "0.25" ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#flash> "0" ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#flashpixVersion> "48 49 48 48
            " ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#make> "SANYO Electric Co.,Ltd
            ." ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#dateTime> "2001-09-15
            T18:11:27.000+01:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#dateTimeDigitized> "
            2001-09-15T18:11:27.000+01:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
        <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#dateTimeOriginal> "2001-09-15
            T18:11:27.000+01:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
        a <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#Photo> .
```

Zápis nalezených metadat v souboru Sanyo SR662.jpg do RDF modelu pomocí MetaMed aplikace obsahující můj plugin.

```
@prefix mre:      <http://mre.kiv.zcu.cz/ontology/mre.owl#> .
@prefix nihss:    <http://mre.kiv.zcu.cz/ontology/nihss.owl#> .
@prefix mreid:    <http://mre.kiv.zcu.cz/id/> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dscl:     <http://mre.kiv.zcu.cz/ontology/dscl.owl#> .
@prefix fslmre:   <http://mre.kiv.zcu.cz/id/fresnel#> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix mreb:     <http://mre.kiv.zcu.cz/> .
@prefix ds:       <http://mre.kiv.zcu.cz/ontology/dasta.owl#> .
@prefix dc:       <http://purl.org/dc/elements/1.1/> .
@prefix geo:      <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix nfo:      <http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#> .
@prefix acl:      <http://www.w3.org/ns/auth/acl> .
@prefix nie:      <http://www.semanticdesktop.org/ontologies/2007/01/19/nie#> .
@prefix dc10:     <http://purl.org/dc/elements/1.0/> .
@prefix image:    <http://mre.kiv.zcu.cz/ontology/image.owl#> .
@prefix dcm:      <http://mre.kiv.zcu.cz/ontology/dicom.owl#> .
@prefix fsl:      <http://www.w3.org/2004/09/fresnel#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:      <http://www.w3.org/2002/07/owl#> .
@prefix skos:     <http://www.w3.org/2004/02/skos/core#> .
@prefix icd10:    <http://purl.bioontology.org/ontology/ICD10/> .
@prefix nexif:    <http://www.semanticdesktop.org/ontologies/2007/05/10/nexif#> .

<http://mre.kiv.zcu.cz/id/JPEG/39c02bd1735fb8aadfa178ef6478c768146d95a4>
      image:fileSource image:fileSource_3 ;
      image:hasResolution <http://mre.kiv.zcu.cz/id/870769a0dd5508596ffcf189785e703f451977a1> ;
      image:imageQuality "Normal/Medium High"^^xsd:string ;
      image:jfif_0x0005 "1.1"^^xsd:float ;
      image:jpeg_0x0006 "Y component: Quantization table 0, Sampling factors 1 horiz/1 vert"^^
          xsd:string ;
```

```
      image:jpeg_0x0007 "Cb component: Quantization table 1, Sampling factors 1 horiz/1 vert"^^
          xsd:string ;
      image:jpeg_0x0008 "Cr component: Quantization table 1, Sampling factors 1 horiz/1 vert"^^
          xsd:string ;
      image:jpeg_0xffffffd "Baseline"^^xsd:string ;
      image:orientation image:orientation_1 ;
      image:yCbCrPositioning image:yCbCrPositioning_2 ;
      mre:filePath "single-image\\Sanyo SR662.jpg" ;
      nfo:fileLastModified "2015-02-15T14:26:23"^^xsd:dateTime ;
      nfo:fileName "Sanyo SR662.jpg" ;
      nfo:fileSize "18526" ;
      nfo:hasHash <http://mre.kiv.zcu.cz/id/597f830a96945db95ec532f85c26bff0dd5eeea6> ;
      nexif:bitsPerSample 8 ;
      nexif:componentsConfiguration "YCbCr"^^xsd:string ;
      nexif:compressedBitsPerPixel "3.5"^^xsd:float ;
      nexif:dateTime "2001-09-15T18:11:27"^^xsd:dateTime ;
      nexif:dateTimeDigitized "2001-09-15T18:11:27"^^xsd:dateTime ;
      nexif:dateTimeOriginal "2001-09-15T18:11:27"^^xsd:dateTime ;
      nexif:exifVersion "2.0"^^xsd:float ;
      nexif:exposureBiasValue 0 ;
      nexif:exposureTime "0.25"^^xsd:float ;
      nexif:fNumber "2.8"^^xsd:float ;
      nexif:fileSource 3 ;
      nexif:flash "Flash did not fire"^^xsd:string ;
      nexif:flashpixVersion "1.0"^^xsd:float ;
      nexif:focalLength 6 ;
      nexif:imageDescription "SANYO DIGITAL CAMERA"^^xsd:string ;
      nexif:imageLength 300 ;
      nexif:imageWidth 225 ;
      nexif:make "SANYO Electric Co.,Ltd."^^xsd:string ;
      nexif:maxApertureValue  3 ;
      nexif:meteringMode "Multi-segment"^^xsd:string ;
      nexif:model "SR662"^^xsd:string ;
      nexif:orientation 1 ;
      nexif:pixelXDimension 1024 ;
      nexif:pixelYDimension 768 ;
      nexif:resolutionUnit 2 ;
      nexif:samplesPerPixel 3 ;
      nexif:software "V662U-71"^^xsd:string ;
      nexif:xResolution 72 ;
      nexif:yCbCrPositioning 2 ;
      nexif:yResolution 72 ;
      foaf:thumbnail <http://mre.kiv.zcu.cz/id/4311d6a46dc1df946ee9159d9518f8ae57ea3a37> .

<http://mre.kiv.zcu.cz/id/4311d6a46dc1df946ee9159d9518f8ae57ea3a37>
      a       foaf:Image ;
      image:compression image:compression_6 ;
      image:resolutionUnit image:resolutionUnit_2 ;
      nexif:compression 6 ;
      nexif:jpegInterchangeFormat 878 ;
      nexif:jpegInterchangeFormatLength 4562 ;
      nexif:resolutionUnit 2 ;
      nexif:xResolution 72 ;
      nexif:yResolution 72 .

<http://mre.kiv.zcu.cz/id/597f830a96945db95ec532f85c26bff0dd5eeea6>
      a       nfo:FileHash ;
      nfo:hashAlgorithm "SHA1" ;
      nfo:hashValue "c8be1a92dce8347e0547f0a13f542a86d6b83409" .

mreid:e6180e894cda9a9f377fbe63ae2858030c449cfa
      a       mre:Dataset ;
```

```
      mre:hasData <http://mre.kiv.zcu.cz/id/JPEG/39c02bd1735fb8aadfa178ef6478c768146d95a4> ;
      mre:hasDataSource <http://mre.kiv.zcu.cz/id/50d8b4a941c26b89482c94ab324b5a274f9ced66> ;
      dc:title "unknown 2015-02-15" .

<http://mre.kiv.zcu.cz/id/50d8b4a941c26b89482c94ab324b5a274f9ced66>
      a       mre:DataSource ;
      dc:title "unknown" .

<http://mre.kiv.zcu.cz/id/870769a0dd5508596ffcf189785e703f451977a1>
      a       image:Resolution ;
      image:resolutionUnit image:resolutionUnit_2 ;
      image:xResolution 72 ;
      image:yResolution 72 .
```