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

Bakalářská práce

Správce hudebních sbírek založený na technologiích sémantického webu

Plzeň 2012 Jan Šmucr

Prohlášení

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

V Plzni dne 10. května 2012

Jan Šmucr

Abstract

The main goal of this bachelor thesis is to become acquainted with basics of the semantic web, related technologies and toolkits and to use this knowledge to create a user-friendly multimedia collection manager application. The application was tested on a set of real data. The achieved results are discussed and shown in graphs and tables.

Obsah

1 Úvod **1**

2 Sémantický web **2**
2.1 Motivace 2
2.2 Základní principy 2
2.2.1 Obecný popis dat 2
2.2.2 Resource Description Framework 4
2.2.3 RDF Schema 5
2.2.4 SPARQL 6
2.3 Sémantický web v praxi 8
2.4 Další důležité pojmy 8

3 Analýza problému **10**

4 Výběr prostředků pro tvorbu aplikace **13**
4.1 Použité knihovny třetích stran 13
4.1.1 Sesame 13
4.1.2 JAudioTagger 14
4.1.3 QtJambi 14
4.1.4 Apache Commons Collections 14
4.1.5 Apache Commons Pool 14
4.1.6 JKeyMaster 15
4.2 Výběr vhodné ontologie 15
4.2.1 Užití pojmů a jejich URI z ontologií uvnitř aplikace . . 16

5 Struktura a fungování aplikace **17**
5.1 Dělení kódu 17
5.2 Přehled nejdůležitějších tříd 17
5.3 Popis stěžejních funkcí 19
5.3.1 Získání individua z repozitáře 19
5.3.2 Manipulace s hodnotami atributů 20

5.3.3 Vytvoření individua v repozitáři 22
5.3.4 Odstranění individua z repozitáře 23
5.3.5 Řazení, vyhledávání a vazba na audio soubory 23
5.3.6 Import metadat 24
5.4 Úložiště dat 26

6 Uživatelská příručka k aplikaci 27
6.1 Hardwarové požadavky 27
6.2 Softwarové požadavky 27
6.3 Sestavení 28
6.4 Instalace a spuštění 28
6.5 Ovládání 28
6.5.1 Import RDF metadat 30
6.5.2 Import audio metadat 30
6.5.3 Export metadat 32
6.5.4 Prohlížení 32
6.5.5 Editace 33
6.5.6 Přehrávání 34
6.5.7 Ikona v trayi 35
6.6 Známé problémy a nedostatky 36

7 Dosažené výsledky 39

8 Závěr 44

A Užití aplikace 52

1 Úvod

Cílem této práce je seznámit čtenáře se základními stavebními kameny Sémantického webu, který je považován za budoucnost webu současného. Ten byl od počátku budován jako úložiště dokumentů pro lidského návštěvníka, nikoliv jako elektronicky zpracovatelný zdroj dat [9] a to je v dnešní době již ne zcela vyhovující vlastnost. Principy Sématického webu ji odbourávají a to přitom při zachování výhod, které současný web má (např. decentralizace). [9, 10]

Práce se následně zaměří na využití tohoto konceptu v uživatelsky přívětivé aplikaci pro běžné osobní počítače, která bude mít za úkol usnadnit uživateli správu a procházení jeho multimediální sbírky, což je v tomto případě sbírka hudebních děl. Podobné sbírky obvykle nabývají rozměrů několika desítek až set GiB a aplikace by měla zvládnout informace o takovém množství dat bez problémů zpracovávat a umožnit uživateli jejich plynulé prohlížení.

V závěru práce přijde na řadu diskuse dosažených výsledků a možností dalších rozšíření aplikace.

2 Sémantický web

2.1 Motivace

Jak bylo zmíněno v úvodu, web v současné podobě není elektronicky zpracovatelným zdrojem dat, který by bylo možné prohledávat s tím, že předem očekáváme relevantní výsledek. Člověk tak může strávit hledáním požadované informace klidně několik hodin a jeho úspěch závisí z velké části na štěstí a umění správně se zeptat.

Co dokáže člověk, nemusí umět stroj, který jako odpověď na dotaz zpravidla očekává buďto přímo hledanou informaci nebo alespoň hlášení o její nedostupnosti.

Jednu z cest, jak lze tento problém řešit, nabízí právě Sémantický web.

Principy Sémantického webu by nicméně mohly najít uplatnění i v aplikacích pro osobní počítače. Tato práce si klade za cíl ověřit možnosti jejich využití v aplikaci fungující jako správce lokální hudební sbírky.

2.2 Základní principy

Hlavní myšlenkou Sémantického webu je zaměřit se na klasifikaci dat, jejich popis a definování jejich vztahu k okolí a umožnit tak jejich strojové zpracování.

Daty jsou zde myšleny *zdroje* (angl. *resources*), což může být dle [8] cokoliv, co lze jednoznačně identifikovat pomocí URI.

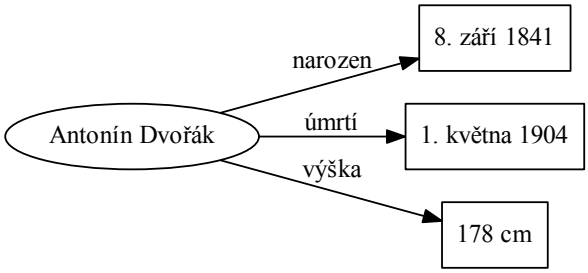
2.2.1 Obecný popis dat

Zdrojem tedy může být i člověk. Nechť je to (s ohledem na zaměření práce) třeba hudební skladatel. Bylo o něm napsáno mnoho publikací, učí se o něm na školách a útržky jeho tvorby jsou známé lidem po celém světě.

My bychom jej nicméně rádi popsali a zaznamenali jeho život a tvorbu strukturovaně – tak, aby byl výsledek počítačově zpracovatelný a mohl se stát součástí větší znalostní databáze, kde by byla každá informace okamžitě zpětně dohledatelná.

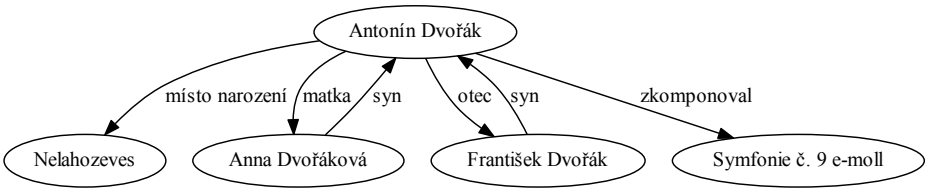
Datům, která takto popisují jiná data, se říká *metadata*. [6]

Začít s tvorbou metadat můžeme vyjádřením jejich vlastností jako na obrázku 2.1.



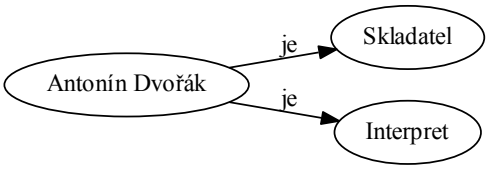
Obrázek 2.1: Vyjádření vlastností dat.

Druhým způsobem, jak o datech něco říci, je vyjádřit jejich vztah k jiným datům. U našeho skladatele to může být vztah k jeho tvorbě, k městu, kde se narodil nebo k lidem kolem něj. Právě jako na na obr. 2.2.



Obrázek 2.2: Vztah dat k okolí.

Poslední způsob je klasifikace, což je zařazení objektu do nějaké předem definované třídy. Obrázek 2.3 ukazuje, že můžeme předmět našeho zkoumání zařadit rovnou do více tříd najednou.



Obrázek 2.3: Klasifikace dat.

Na obrázcích 2.1, 2.2 a 2.3 je vidět, že každý ze zmíněných způsobů popisu lze vyjádřit orientovaným grafem. Ten je tvořen takzvanými *trojicemi*, tedy počátečním uzlem, hranou a koncovým uzlem. V rámci Sémantického webu se počátečnímu uzlu se říká *subjekt*, hraně *predikát* a koncovému uzlu *objekt*. [20]

Každá trojice formuje právě jedno *tvrzení* (angl. *statement*) reprezentující elementární *znalost* (angl. *knowledge*). [19]

Systém využívající způsob popisu dat pomocí trojic, se nazývá *Resource Description Framework* (česky *systém pro popis zdrojů*).

2.2.2 Resource Description Framework

RDF je soubor standardů W3C pro popis zdrojů se zaměřením na následné zpracování počítačovými aplikacemi. [24] V RDF má každý zdroj svůj jednoznačný identifikátor a je popsán pomocí trojic (tvrzení), kde na straně subjektu stojí popisovaný zdroj, predikát identifikuje jeho vlastnost a objekt přestavuje hodnotu této vlastnosti.[24] Objektem může být jiný zdroj nebo *literál*[1]. [26]

RDF poskytuje jen úplně základní množinu pojmů (tříd a vlastností), které slouží převážně jako základ pro odvození pojmů jiných. [25]

[1]Prostá data – např. číslo či řetězec.

2.2.3 RDF Schema

RDF Schema je množina pojmů přinášející do RDF základ pro tvorbu *RDF slovníků*, které definují hierarchii tříd a jejich vlastností použitelných ke klasifikaci a popisu zdrojů. [24]

RDF Schema dále rozšiřují slovníky pro tvorbu *ontologií*[2] (např. OWL).

Pojďme se nyní podívat, jak by vypadal náš skladatel částečně popsaný pomocí RDF ve formátech RDF/XML (kód 2.1) a Notation3 (kód 2.2) s využitím existujících slovníků a fiktivní ontologie „Music“.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:music="http://foo.com/music#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">

  <music:Composer rdf:about="artists://dvorak/antonin">
    <!-- Vyjadreni vlastnosti -->
    <foaf:name>Antonin Dvorak</foaf:name>
    <!-- Vztah k jinym datum -->
    <music:composed>
      <music:Piece>
        <dc:title>Symfonie c. 9 e-moll</dc:title>
      </music:Piece>
    </music:composed>
    <!-- Klasifikace -->
    <rdf:type rdf:resource="http://foo.com/music#Interpret />
  </mo:MusicArtist>

</rdf:RDF>
```

Kód 2.1: Ukázka RDF/XML.

Takto vytvořený graf nyní dokáže zpracovat jakákoliv aplikace podporující zvolené RDF slovníky. Sám o sobě samozřejmě příliš smysl nemá, ale – jak bylo řečeno úvodem kap. 2.2.1 – může být součástí větší znalostní databáze.

[2]Datový model explicitně definující pojmy a vztahy mezi nimi. [23]

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix music: <http://foo.com/music#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<artists://dvorak/antonin>
  foaf:name "Antonin Dvorak" ;
  music:composed _:9thsymph ;
  a music:Composer .

_:9thsymph
  a music:Piece ;
  dc:title "Symfonie c. 9 e-moll" .
```

Kód 2.2: Ukázka Notation3.

2.2.4 SPARQL

SPARQL je jazyk pro tvorbu dotazů nad RDF grafy, resp. úložišti trojic (angl. *triplestore*). Svou syntaxí připomíná kombinaci SQL a Notation3. [19]

Psaním SPARQL dotazu nesestavujeme celý graf (jako v kap. 2.2.2), ale pouze jeho šablonu, kde jsou uzly či hrany, které neznáme, nahrazeny proměnnými. Výsledek poté odpovídá tomu, co lze za tyto proměnné dosadit, aby výsledný text popisoval podgraf dotazovaného grafu. [27]

Výsledek může nabývat čtyř forem stanovených klíčovým slovem, kterým dotaz začíná:

- `SELECT` - Bere si za parametr seznam proměnných (příp. `*` namísto všech proměnných) z dotazu a výsledkem je tabulka typu proměnná-hodnota. [27]
- `ASK` - Variace na `SELECT`, kde nejde o konkrétní hodnoty proměnných, ale pouze o to, zda nějaký výsledek vůbec existuje. Vrací `true` nebo `false`, resp. `yes` nebo `no` v závislosti na implementaci.[27]
- `CONSTRUCT` - Vrací graf představující sjednocení všech podgrafů vyhovujících dotazu. [27]
- `DESCRIBE` - Bere si za parametr seznam proměnných (příp. `*` namísto

všech proměnných) z dotazu a pro každou dosaditelnou hodnotu vrací graf, který ji popisuje (podobně, jako ukazuje např. kód 2.2). [27]

Namísto seznamu proměnných lze dosadit také rovnou identifikátor zdroje a `DESCRIBE` pak vrátí jeho popis přímo. [27]

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:alice foaf:name  "Alice" .
_:alice foaf:knows _:bob .
_:alice foaf:knows _:clare .
_:bob   foaf:name  "Bob" .
_:clare foaf:name  "Clare" .
_:clare foaf:nick  "CT" .
```

Kód 2.3: Data pro SPARQL dotaz 2.4.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE
{
  ?bob foaf:name "Bob" .
  ?clare foaf:name "Clare" .
  ?person
    foaf:knows ?bob ;
    foaf:knows ?clare ;
    foaf:name ?name .
}
```

Kód 2.4: Ukázka SPARQL dotazu nad daty 2.3.

Kódy 2.3[3] a 2.4 demonstrují užití SPARQL a jeho `SELECT` varianty. První zmíněný kód definuje graf a druhý představuje dotaz hledající v grafu jméno společného známého Boba a Clare. Výsledkem je samozřejmě řetězec `"Alice"`.

Rozšíření jazyka SPARQL

Ve chvíli psaní tohoto dokumentu (10. května 2012), je SPARQL stále ve verzi 1.0 a v této formě umožňuje pouze čtení. SPARQL 1.1, který je aktuálně ve fázi schvalování [28], přináší mimo jiných také rozšíření *SPARQL Update Language* umožňující provádět změny v úložišti.

[3] Převzato z [27].

2.3 Sémantický web v praxi

Vyvstává otázka, proč je Sémantický web – přestože byly jeho základní myšlenky vysloveny již v roce 2001 [12] – i o 11 let později stále poměrně okrajovou záležitostí.

Některé důvody nastiňuje např. Michal Černý v [12]. Zmiňuje především nechuť uživatelů data popisovat a neochotu vývojářů technologie Sémantického webu využívat – ať už z důvodu malého povědomí o jejich možnostech, nutnosti dodržovat XML standardy či obecně z důvodu chybějící motivace.

Proto se se semantizací dat v dnešní době setkáváme hlavně v oblastech, které nemají s webem zas tak moc společného. Například:

- **Generální rada soudní moci (Španělsko)**

  Využívá technologie Sémantického webu ve znalostním systému pomáhajícím čerstvě jmenovaným soudcům hledajícím odpovědi na často kladené otázky týkající se výkonu spravedlnosti či nějaké precedenční rozhodnutí. [11]

- **Renault (Francie)**

  Znalostní systém zaměřený na diagnostiku a opravy automobilů. Jeho repozitář modeluje postup zjištění a opravy pro každou možnou vadu. [18]

- **Vodafone Group Research & Development (Španělsko)**

  Provozuje korporátní síť shromažďující informace, které zaměstnanci firmy považují na hodné uchování a užitečné s ohledem na další technologický rozvoj. Řešení umožňuje pokládání otázek v přirozeném jazyce. [14]

2.4 Další důležité pojmy

Následuje přehled několika dalších užitých pojmů, které zatím nebyly definovány.

- **Odvozování (angl. *inferring*)** - Proces, při kterém dochází k indukci nových faktů na základě faktů již existujících. Pro příklad je-li třída B potomkem třídy A, pak lze logicky odvodit, že třída A je rodičem třídy B.
- **Turtle** - Podmnožina jazyka Notation3 (viz kód 2.2) určená výhradně pro použití s RDF. [29]
- **Individuum** - Konkrétní instance třídy – zdroj mající alespoň jednu vlastnost `rdf:type`, jejíž hodnota je identifikátor nějaké známé třídy. [21]
- **Atribut** - Vlastnost zdroje. [25] Uvnitř trojice představuje predikát.
- **Inverzní atribut** - Opačný atribut. Pro příklad – má-li třída B atribut říkající, že je potomkem třídy A, pak atribut k němu inverzní bude atribut třídy A říkající, že má tato třída potomka B. [22]
- **Formát** - Způsob organizace dat za účelem jejich uložení či zobrazení. [2]

3 Analýza problému

Výsledkem práce by měla být aplikace pro osobní počítač použitelná pod operačními systémy Windows a GNU/Linux. Jejím cílem bude poskytnout uživateli komfort při organizaci a prohledávání jeho hudební sbírky.

Tato kapitola zmiňuje některé základní problémy, které budou během vývoje řešeny:

- **Výběr prostředků pro tvorbu aplikace:**

  Protože je na aplikaci kladen požadavek přenositelnosti mezi platformami, je potřeba tomu přizpůsobit výběr prostředků pro její tvorbu. Programovací jazyk bude nutné vybrat podle toho, zda jsou pro něj dostupné kompilátory (resp. interprety) pro obě podporované platformy a zároveň zda jsou pro něj dostupné knihovny, které budou aplikaci zaručovat alespoň základní funkce pro

  - práci s RDF metadaty,
  - tvorbu grafického uživatelského rozhraní,
  - práci s multimédii.

  Protože bude vývoj probíhat na obou platformách současně, bude vhodné zvolit také multiplatformní IDE.

- **Vytvoření systému pro ukládání metadat:**

  Základ pro tento systém bude muset tvořit ontologie zaměřená na popis zdrojů s hudební tématikou a datové úložiště, se kterým by měla přijít přímo knihovna pro práci s RDF metadaty. Aby šlo ukládaná data případně využít i v jiných aplikacích, bude upřednostněna možnost použít ontologii již existující.

  Dále bude třeba vyřešit způsob, jakým budou RDF metadata reprezentována uvnitř aplikace. Ideální by bylo rovnou převádět RDF zdroje na instance tříd z aplikace napsané v objektově orientovaném jazyce (načtení) a ty zpět na RDF zdroje (uložení).

  Metadata bude nutné do systému nejprve importovat. Aplikace by měla zvládat import dvou typů metadat – metadata ve formátu RDF/XML a metadata získaná z hudebních souborů. Minimálně ve druhém případě

se dá očekávat nutnost funkčnost implementovat ručně, neboť žádná knihovna třetí strany požadovanou funkčnost nenabízí.

Export metadat by měla aplikace rovněž zvládat dvojí – jednak export jednotlivých individuí (např. samotného interpreta a jeho tvorby) a jednak export celého úložiště (např. pro potřeby zálohy nebo využití obsahu úložiště v jiné aplikaci).

- **Tvorba uživatelského rozhraní:**

  S ohledem na očekávaný způsob využití aplikace (tedy jako „chytřejší" audio přehrávač) by mělo být uživatelské rozhraní spíše minimalistického charakteru, oproštěné od nepotřebných informací a nemělo by zbytečně reflektovat způsob, jakým aplikace funguje – tedy např. to, že není seznam skladeb uložen v tabulce ale v grafu.

  Důraz by měl být kladen na rychlost použití. Má-li běžný uživatel – posluchač hudby – těžit z jejího využití, pak je naprosto klíčové, aby měl svou hudbu vždy v dosahu maximálně několika kliknutí. Chce-li ale namísto toho daty pouze procházet, může aplikace využít toho, že jsou uvnitř data reprezentována grafem a nechat uživatele tímto grafem „cestovat".

  Bude samozřejmě nutné implementovat i rozhraní pro přímou editaci metadat – tedy např. úpravu vlastností individuí nebo vztahů mezi nimi. Nejedná se ale o věc, kterou by musel mít uživatel neustále po ruce.

  Neustále po ruce by naopak měl mít možnost zobrazená individua okamžitě filtrovat podle názvu nebo jeho části.

- **Implementace vyhledávání a filtrace dat:**

  SPARQL je pro aplikaci v tomto ohledu velice mocný nástroj. Zobrazovaný rozsah dat bude možné omezovat například pouhým přidáváním dalších trojic do dotazu před jeho vyhodnocením. Zbývá tedy vyřešit, jak bude sestavování dotazu fungovat v rámci aplikace tak, aby do něj mohla každá její část bezpečně zasahovat.

- **Implementace interního audio přehrávače:**

  Prvně bude nutné vyřešit převod individuí na existující audio soubory, které by následně šly přehrát. Protože je nesmysl, aby měla všechna individua vlastnost popisující cesty k souborům, které k nim patří, bude nutné nalézt ontologii, která by dokázala kromě jiného popsat i audio soubory a umožňovala propojit individua, která je reprezentují,

se zbytkem grafu. Problém se následně zjednoduší na tvorbu takového SPARQL dotazu, který zajistí cestu od individua, které chce uživatel „přehrát“ až k individuu, které reprezentuje soubor, co skutečně přehrát jde.

Tyto soubory bude následně potřeba umístit do playlistu[1], který by měl být exportovatelný do některého z běžných formátů podporovaných jinými audio přehrávači.

Několik problémů s sebou přináší interakce přehrávače a playlistu, kdy bude nutné definovat chování přehrávače v případě, že dojde během přehrávání k manipulaci s playlistem. Přehrávač by se měl také umět vyrovnat s nepodporovanými nebo nenalezenými soubory.

Ke zmíněným problémům se nabízí řešení dalších, jako je například spolupráce aplikace s online databázemi hudebních děl nebo automatická aktualizace informací o sbírce v případě její změny provedená bez zásahu uživatele.

Graf ukazující základní scénáře užití aplikace naleznete v příloze A na obrázku A.1.

[1]Seznam položek pro přehrání.

4 Výběr prostředků pro tvorbu aplikace

Nejvýznamnějšími kritérii pro výběr programovacího jazyka byla multiplatformnost výsledné aplikace a dostupnost knihoven třetích stran umožňujících práci s RDF a hudebními soubory (resp. s uvnitř uloženými metadaty) a tvorbu grafického uživatelského rozhraní.

První dvě kritéria učinila jasným vítězem jazyk Java, ačkoliv byly ve hře i další – např. C#, který by měl oproti Javě výhodu např. v podpoře struktur. [15] U třetího kritéria jsem váhal, jelikož jsem pro Javu nenalezl SDK pro tvorbu GUI, který by mi plně vyhovoval (tj. aby měl intuitivní layout manager, širokou paletu komponent k okamžitému použití, aby dobře vypadal...).

Jako IDE jsem si na základě předchozích zkušeností zvolil Eclipse. [1]

4.1 Použité knihovny třetích stran

4.1.1 Sesame

Sesame je framework pro zpracování RDF dat. Umožňuje jejich parsování (formáty RDF/XML, Turtle, N-Triples, TriG a TriX), odvozování (nad vlastním úložištěm pouze základní dle RDF Schema) a dotazování (SPARQL, SeRQL). Sesame disponuje vlastním proprietárním úložištěm[1], které lze provozovat buď'to na lokální stanici nebo jako webový server, ke kterému je přímo k dispozici také obslužná webová aplikace pro Apache. Kromě svého úložiště přímo podporuje také některá úložiště třetích stran (např. MySQL) a nabízí rozhraní pro implementaci dalších.

K dispozici na: `http://www.openrdf.org/`

[1]RDF úložišti se jinak říká také *repozitář*. [5]

4.1.2 JAudioTagger

Java knihovna pro práci s audio soubory se zaměřením na parsování audio metadat. Kromě běžných formátů, jako je MP3, OGG či WMA, zvládá i práci s bezztrátovým formátem FLAC. Disponuje plnou podporou kódování Unicode.

K dispozici na: `http://www.jthink.net/jaudiotagger/`

4.1.3 QtJambi

Řešit GUI jsem se rozhodl prostřednictvím knihovny Qt [4], pro kterou existuje Java wrapper jménem QtJambi. S jistými omezeními poskytuje vše, co je s Qt možné v jazyce C++. Výhodou Qt je nejen neustávající vývoj a velké množství dostupných komponent, ale také fakt, že je psané v C++ a na pozadí tedy využívá standardních nativních knihoven, což může mít příznivý dopad na rychlost aplikace. [13] Nevýhodou je obtížnější ladění.

Qt také poskytuje vlastní prostředek pro lokalizaci v něm napsaných aplikací – QtLinguist.

K dispozici na: `http://www.qt-jambi.org/`

4.1.4 Apache Commons Collections

Knihovna obsahující mimo jiné třídu `ReferenceMap`, jejíž instance mohou sloužit jako asociativní cache nebo je lze použít při snaze zabránit tvorbě redundantních objektů.

K dispozici na: `http://commons.apache.org/collections/`

4.1.5 Apache Commons Pool

Knihovna pro tvorbu *Object-pooling API*. Slouží jako základ pro implementaci třídy, jejíž instance doslova zapůjčuje aplikaci požadované objekty. Pokud

již nejsou potřeba, lze je vrátit a uchovat pro další použití, což opět pomáhá uspořit paměť a čas.

K dispozici na: `http://commons.apache.org/pool/`

4.1.6 JKeyMaster

Nabízí možnost ovládat aplikaci pomocí globálních klávesových zkratek napříč platformami.

K dispozici na: `https://github.com/tulskiy/jkeymaster`

4.2 Výběr vhodné ontologie

Aplikace bude pro popis audio souborů využívat pojmů z ontologie Music Ontology, která umožňuje popsat mnoho aspektů týkajících se hudby od její tvorby, přes produkci až po záznam a publikaci.[17]

Původně jsem zamýšlel, že bude aplikace využívat svou vlastní ontologii, která by byla více přizpůsobena pro účely aplikace, ale k tomu nakonec nedošlo, protože by poté nebyla sebraná data využitelná jinou aplikací. Naproti tomu s sebou Music Ontology přináší nevýhodu v podobě velkého množství redundantních dat, která by nebyla pro popis hudební sbírky vůbec potřeba a vývoj aplikace to značně zkomplikuje. Přitom se dá očekávat, že možnosti, které to přináší, využije jen opravdu malé procento uživatelů.

Pro příklad: Aplikace importuje audio metadata v podobě ID3 tagu z MP3 souboru. Omezíme-li se na položky popisující název stopy, její číslo, příslušnost k albu, rok vydání a autora, pak to v řeči Music Ontology znamená vytvořit pojmy pro autora, jeho skladbu, akt kompozice této skladby, konkrétní interpretaci skladby, událost interpretace (objekt obsahující čas, kdy k ní došlo a jak dlouho trvala), záznam této interpretace, stopu jakožto publikovanou formu záznamu, nahrávku, kam stopa patří, album, ve kterém nahrávka vyšla a událost jeho vydání.

Předpokládá se proto, že budou některé méně podstatné pojmy z aplikace vypuštěny.

4.2.1 Užití pojmů a jejich URI z ontologií uvnitř aplikace

Protože není vhodné ve zdrojovém kódu aplikace výslovně uvádět hodnoty, které se mohou v průběhu života aplikace měnit (patří sem například i lokalizovatelné texty), rozhodl jsem se umístit veškeré používané pojmy definované v ontologiích a jejich příslušné URI do externího souboru. Ten bude aplikace zpracovávat prostřednictvím speciální třídy, která dokáže soubor přečíst a namapovat získané URI na pojmy užívané uvnitř aplikace. Protože může každá z částí aplikace využívat jiný soubor pojmů, je tato třída abstraktní a potomci musí explicitně uvést všechny pojmy, které chtějí poskytovat své části aplikace.

5 Struktura a fungování aplikace

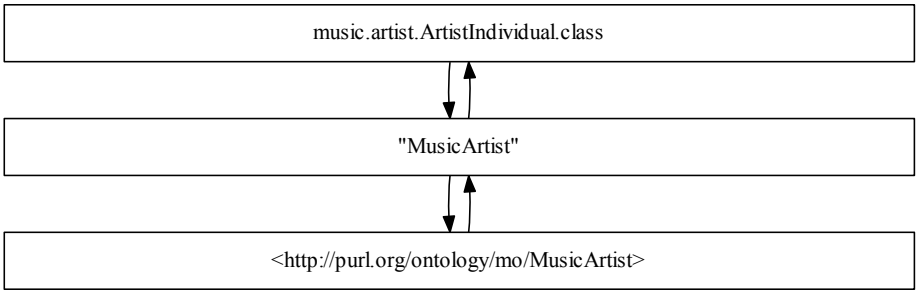
5.1 Dělení kódu

Zdrojový kód aplikace je rozdělen do čtyř balíků podle funkčnosti, na kterou se obsažené kódy orientují.

- `core` - Třídy, které nepatří jinam, protože jsou využitelné univerzálně.
- `gui` - Celé grafické uživatelské rozhraní a příslušné třídy a struktury.
- `music` - Balík obsahující zdrojové kódy aplikační vrstvy pracující s hudebními daty.
- `rdf` - Nejnižší vrstva aplikace a třídy sloužící jako základ pro balík `music` a výhledově i pro další balíky zaměřující se na jiný druh multimédií.

5.2 Přehled nejdůležitějších tříd

- `rdf.DataAgent` - Usnadňuje komunikaci aplikace s repozitářem Sesame. Veškerý datový tok mezi aplikací a repozitářem vede přes instanci této třídy. Příchozí data dělí dle kontextů (např. hudba, výhledově fotografie nebo jiný multimediální obsah) a přeparsované dotazy a jejich výsledky (odchozí data) ukládá do vyrovnávacích pamětí.
- `rdf.Module` - Abstraktní třída, jejíž potomci představují modul s konkrétním multimediálním zaměřením. V základu poskytuje funkce pro získávání objektů z repozitáře a tvorbu jejich Java ekvivalentů. Zároveň zamezuje tomu, aby v aplikaci existoval více jak jeden objekt odkazující na jeden konkrétní objekt z repozitáře.
- `rdf.Namespaces` - Abstraktní třída. Mapuje aplikací používané pojmy (krátké řetězce – např. `"MusicArtist"` nebo `"audiofile_encoding"`) na skutečné identifikátory pojmů načtené z externího souboru a zároveň k nim přiřazuje odpovídající Java třídy (viz obr. 5.1). Každému modulu by měla náležet jedna instance potomka této třídy, který musí užívané pojmy výslovně uvést.



Obrázek 5.1: Převod mezi Java třídou a pojmem z ontologie.

- `rdf.Individual` - Naprostý základ pro třídy fungující jako Java ekvivalenty tříd z využívaných ontologií. Jejich instance představují individua z repozitáře.
- `rdf.TypedIndividual` - Abstraktní potomek `rdf.Individual`, jehož instance patří pod konkrétní modul. Podporuje prázdné uzly, čtení a zápis atributů, dokáže v repozitáři vytvořit duplikát svého RDF protějšku či jej exportovat do RDF/XML. Využívá přitom – mimo jiné – služeb `rdf.DataAgent`. Pro správné fungování očekává aplikace u potomků této třídy anotaci[1] říkající, kterou třídu z užívaných ontologií tento potomek reprezentuje.
- `rdf.properties.Property` - Instance potomků této abstraktní třídy představují právě jeden atribut zdroje. Pro správné fungování očekává aplikace u potomků této třídy anotaci[2] říkající, který predikát z užívaných ontologií tento potomek představuje. Podporovány jsou rovněž inverzní atributy[3].
- `rdf.properties.PropertyValueConverter` - Potomci této abstraktní třídy umožňují provádět konverzi mezi datovými typy užívanými v repozitáři a datovými typy Javy.
- `rdf.filters.AbstractFilter` - Potomci této abstraktní třídy mohou reprezentovat části SPARQL dotazů a umožňují sestavovat dotaz po

[1] Třída `rdf.annotations.RDFInternalClassName`.
[2] Třída `rdf.annotations.RDFInternalPropertyName`.
[3] S využitím anotace `rdf.annotations.RDFPropertyInverseOf`.

částech v různých místech aplikace předtím, než dojde k jeho vyhodnocení. Přidáním instance potomka reprezentujícího např. trojici do dotazu lze filtrovat výsledky, z čehož vychází pojmenování této třídy.

- `music.generic.MusicIndividual` - Abstraktní třída sloužící jako základ pro všechny Java ekvivalenty RDF tříd z ontologií využívaných hudebním modulem. Disponuje rozšířenou funkčností. Např. podporuje výpis svého vlastního popisu do HTML využívajíc předpřipravené šablony načtené z externího souboru, což umožňuje hypertextovou navigaci mezi různými instancemi této třídy.
- `music.importing.MusicScanner` - Načítá do speciální struktury metadata z předaných audio souborů a připraví je k zobrazení ve formě stromu před samotným importem.
- `music.importing.MusicImporter` - Provádí import předpřipravených metadat do repozitáře.

5.3 Popis stěžejních funkcí

5.3.1 Získání individua z repozitáře

Za předpokladu, že je znám identifikátor, pod kterým lze individuum v repozitáři najít, pak stačí pouze vytvořit instanci příslušné Java třídy a v konstruktoru jí tento identifikátor předat. Od této chvíle je individuum v repozitáři reprezentováno právě vytvořeným objektem. Pokud již takový objekt existoval, je vyvolána výjimka jako v kódu 5.1.

Pokud identifikátor znám není, lze využít metody `getClassInstances()`, které stačí předat parametrem Java třídu, která bude (stejně jako v předchozím odstavci) datovým typem objektů reprezentujících v aplikaci individua z repozitáře vrácená v návratové hodnotě (viz kód 5.2).

Zavolána jako v kódu 5.2 vrací metoda `getClassInstances()` všechna individua odpovídající třídy, která v repozitáři nalezne. Výběr lze dále zúžit například pomocí filtrů, jak ukazuje kód 5.3.

Metoda `getClassInstances()` nejprve zjistí, jaký je RDF identifikátor třídy, kterou předaná Java třída v aplikaci představuje a to tak, že se podívá, zda k ní náleží anotace `RDFInternalClassName`. Pokud ano, pomocí

```
URI artistURI = /* ... */ ; // Identifikator
ArtistIndividual artist = null;

try
{
    artist = new ArtistIndividual(artistURI);
}
catch (OnlyOneInstancePerURIAllowedException e)
{
    artist = e.individual;
}
```

Kód 5.1: Propojení Java objektu s individuem z repozitáře.

```
MusicModule module = /* ... */ ; // Hudebni modul

ArrayList<ArtistIndividual> artists =
    module.getClassInstances(ArtistIndividual.class);
```

Kód 5.2: Získání individuí z repozitáře (nejjednodušší způsob).

`rdf.Namespaces` převede její hodnotu na identifikátor a začne sestavovat dotaz. Ten pak zašle instanci `rdf.DataAgent` a na oplátku očekává seznam identifikátorů individuí, která vyhovují dotazu. Poté (pomocí Java reflexe) vytvoří objekty reprezentující tato individua v rámci aplikace a vrátí je.

Kód 5.3 vytvoří SPARQL dotaz jako v kódu 5.4.

5.3.2 Manipulace s hodnotami atributů

Aplikace disponuje uživatelsky přívětivým rozhraním pro získávání hodnot atributů. Stejně jako zde Java objekty reprezentují objekty z repozitáře, tak i atributy těchto objektů mohou být reprezentovány pomocí atributů Java objektů. Kód 5.5 ukazuje zápis a čtení jednoduchého atributu. U násobných atributů je to obdobné, ale lze s nimi pracovat jako s kolekcemi (implementují rozhraní `java.util.Collection`).

Kód 5.6 ukazuje, jak jsou atributy definovány uvnitř.

Při získávání hodnoty atributu se sestaví SPARQL dotaz (využívajíc při-

```
MusicModule module = /* ... */ ; // Hudebni modul

Collection<AbstractFilter> filters =
    new ArrayList<AbstractFilter>();

filters.add(new AdditionalTripleFilter(
    music.artist.properties.NameProperty.class, "Arakain"));

ArrayList<ArtistIndividual> artists =
    module.getClassInstances(ArtistIndividual.class, filters);
```

Kód 5.3: Získání individuí z repozitáře.

```
SELECT ?resource
FROM <collector://music>
WHERE
{
  ?resource a <http://purl.org/ontology/mo/MusicArtist>.
  ?resource <http://xmlns.com/foaf/0.1/name> "Arakain".
}
```

Kód 5.4: Dotaz pro kód 5.3. Za klíčovým slovem FROM je identifikátor kontextu, ve kterém pracuje modul, který dotaz vygeneroval.

tom případných přednastavených filtrů – např. pro řazení hodnot) a předá jej instanci `rdf.DataAgent`. Ten vrátí raw hodnotu[4], ta je pomocí konverteru[5] převedena na datový typ použitelný aplikací a poté vrácena.

Při zápisu se převod děje v opačném směru, ale namísto sestavování dotazu se raw hodnota přímo předá instanci `rdf.DataAgent` a je zapsána. Zároveň se kontroluje, zda je hodnotou zapisovaného atributu nějaké individuum. Pokud ano a zapisovaný atribut má k sobě definovaný také atribut inverzní[6], pak se tento inverzní atribut u daného individua vyhledá a jako hodnota se mu zapíše individuum, kterému náleží atribut původní („neinverzní“).

Přímo v Java objektech se žádné hodnoty neukládají, vše se zapisuje přímo do repozitáře.

[4]Termín označující uvnitř aplikace hodnotu mající datový typ užívaný v repozitáři.
[5]Třída `rdf.properties.PropertyValueConverter`.
[6]Pomocí anotace `rdf.annotations.RDFPropertyInverseOf`.

```
ArtistIndividual artist = /* ... */ ;
// Nastaveni hodnoty
artist.name.setValue("Arakain");
// Precteni a vypis hodnoty
System.out.println(artist.name.getValue());
```

Kód 5.5: Manipulace s jednoduchým atributem.

```
@RDFInternalClassName("MusicArtist")
public class ArtistIndividual extends MusicIndividual
{
    // ...
    public final NameProperty name = new NameProperty(this);
    // ...
}
```

Kód 5.6: Definice atributů.

5.3.3 Vytvoření individua v repozitáři

Provádí se voláním metody `TypedIndividual.create()` z objektu, který bude nové individuum reprezentovat v aplikaci. Požadavek na vytvoření se předá instanci `rdf.DataAgent`. Ta zjistí, jaká třída reprezentuje nové individuum v aplikaci tak, že pomocí reflexe projde její hierarchii až k základní třídě, po cestě sbírá anotace `RDFInternalClassName` a z nich zjišťuje, jakého typu bude nové individuum vytvářené v repozitáři.

Při zápisu se pak kromě identifikátoru přiřadí novému individuu i všechny zjištěné typy. Správně by stačil pouze úplně první zjištěný – předpokládalo by se, že je-li individuum typu B, jež je potomkem A, pak je individuum zároveň i typu A. Bohužel aby toto vše fungovalo, muselo by nastoupit odvozování a repozitář by musel obsahovat i celou ontologii, která by tuto typovou hierarchii definovala. Odvozování je ale výpočetně velmi náročné a během vývoje bylo zjištěno, že by to aplikaci v některých ohledech znatelně zpomalovalo.

```
// Vytvori individuum s nahodne generovanym identifikatorem
ArtistIndividual artist = new ArtistIndividual(null);
artist.create();
```

Kód 5.7: Vytvoření nového individua s náhodně generovaným identifikátorem.

5.3.4 Odstranění individua z repozitáře

Odstranění individua z repozitáře je o něco složitější, než je jeho vytvoření. Prosté odebrání všech faktů o něm by totiž mohlo narušit souvislost[7] grafu. To by zapříčinilo, že již nebude možné se k některým individuím prostřednictvím aplikace dostat a tedy je ani odstranit. Je proto nutné izolovaná individua najít a odstranit je také.

Aplikace si při odebírání udržuje seznam identifikátorů individuí, která bude potřeba odebrat.

Na začátku se do seznamu vloží individuum, které má být odstraněno. Následující kroky se provádí pro každou položku z tohoto seznamu až do doby, než je úplně prázdný:

1. O každé položce ze seznamu se získají všechny trojice, kde figuruje na pozici subjektu.
2. Poté se z repozitáře odeberou všechny trojice, kde figuruje bud' jako subjekt nebo jako objekt.
3. Seznam se vyprázdní a ze získaných trojic se zjistí, na které zdroje odebrané položky odkazovaly.
4. Tyto zdroje se vloží do seznamu.
5. Seznam se celý projde a zjišťuje se, zda jsou v něm položky, na které odkazují zdroje, co se na něm nenachází. Takové zdroje se ze seznamu vyškrtnou. Pokud na seznamu ještě nějaká individua zbyla, aplikace se vrací k bodu 1.

5.3.5 Řazení, vyhledávání a vazba na audio soubory

Aplikace obsahuje několik statických tříd, které definují, jakým způsobem se s individui provádí operace řazení, rychlého vyhledávání nebo odvození audio souborů k nim náležejících (což umožňuje tvorbu playlistů).

Každá tato operace je definována sadou filtrů, ze kterých se sestavuje dotaz pro úložiště. Tyto filtry jsou vráceny zmíněnými třídami na základě

[7] V souvislém grafu existuje cesta mezi každými dvěma uzly. [30]

požadavku obsahujícího mimo jiné třídu individua, kterého se operace týká – například řazení se totiž provádí jinak pro skladby a jinak pro jejich interpretace.

U řazení ovlivňují vrácené filtry pořadí, v jakém jsou vráceny výsledky z úložiště, u vyhledávání se k řazení přidává ještě filtr zužující výběr podle zadaných parametrů. U „převodu“ na audio soubory zas vrácená sada filtrů definuje cestu od počátečního individua po individua popisující příslušné soubory.

Kód 5.8 ukazuje, jak jednoduše získat z repozitáře seznam umělců seřazených dle výchozích pravidel – tj. podle jejich jména od A do Z – takových, jejichž jméno obsahuje podřetězec `"the"`.

```
1  MusicModule module = /* ... */ ; // Hudebni modul
2  ArrayList<ArtistIndividual> artists;
3  Collection<AbstractFilter> filters;
4
5  filters = DefaultResourcesSortingFilters.getFor(
6     module,
7     // "?resource"
8     AbstractWhereClauseFilter.DEFAULT_SPARQL_SUBJECT_IDENTIFIER,
9     ArtistIndividual.class);
10
11 filters.addAll(DefaultSearchFilters.getFor(
12    module,
13    AbstractWhereClauseFilter.DEFAULT_SPARQL_SUBJECT_IDENTIFIER,
14    ArtistIndividual.class,
15    "the");
16
17 artists = module.getClassInstances(
18    ArtistIndividual.class,
19    filters);
```

Kód 5.8: Řazení a vyhledávání.

5.3.6 Import metadat

Aplikace dokáže importovat metadata dvojího druhu – metadata ve formátu RDF/XML (což zvládá knihovna Sesame sama od sebe) a metadata z audio

souborů.

V druhém případě se import skládá z několika kroků:

1. Prvním krokem je výběr audio souborů, ze kterých se budou získávat metadata.

2. V dalším kroku se ze všech těchto souborů získají metadata pomocí knihovny JAudioTagger a vloží se do struktury složené z několika do sebe vnořených `HashMap`. Užití `HashMap` má svůj smysl v tom, že klíče tvoří např. název autora a tím je zajištěno, že přestože se v načtených metadatech může jméno autora opakovat, ve výsledné struktuře bude pouze jednou.

3. Následně jsou metadata převedena do formy stromu, který nabízí uživateli přehled o tom, co bude vlastně importováno. K dispozici jsou i omezené možnosti úprav (např. přesun alba k jinému autorovi nebo přejmenování stopy).

   Metadata jsou ve stromu organizována podle autora, následně alba, nahrávky[8] a nakonec stopy.

4. Po potvrzení importu uživatelem je strom převeden na seznam položek k importu[9]. Každá položka obsahuje informace ze stromu a k tomu odkaz na původní – neupravená metadata a soubory, odkud tato metadata pochází.

5. Samotný import každé položky znamená nejprve se podívat, zda se individuum reprezentující autora (nebo album, nahrávku, skladbu) nenachází ve vyrovnávací paměti třídy provádějící import. Pokud ne, vyhledání proběhne v repozitáři a teprve pokud není k nalezení ani tam, je nové indiviuum vytvořeno a propojeno s ostatními (tj. např. skladba je přiřazena k autorovi, nahrávka přiřazena k albu).

   Hledání je zde prováděno podle jména, přičemž se nedbá na velikost písmen.

6. Následně jsou pro každý zdrojový soubor vytvořena, propojena a popsána individua reprezentující samotný soubor, interpretaci, audio signál a stopu tak, jak je to dáno použitou ontologií. Ta sice definuje i

[8]Nahrávkou je v ontologii Music Ontology myšleno např. CD, které může a nemusí být součástí větší sady, která byla nakonec vydána jako album.

[9]Seznam instancí `music.importing.MetadataImportItem`.

některé další třídy popisující např. událost vydání alba nebo událost kompozice skladby, ale ty zatím nejsou podporovány a při importu aplikace jejich instance nevytváří.

5.4 Úložiště dat

Knihovna Sesame nabízí různé možnosti, jak a kam ukládat RDF metadata používaná aplikací.

Aplikace využívá jeho úložiště *MemoryStore*, které je na disku tvořeno jedním jediným souborem a je nutné jej při každém startu aplikace celé načíst do operační paměti. [7] Jeho obrovskou výhodou je nicméně rychlost, která je pro tuto aplikaci stěžejní.

O rychlosti se naopak příliš nedá mluvit u úložiště *NativeStore*, které je umístěno na disku i v době běhu aplikace a trojice v něm navíc musí být indexovány, což znásobuje jeho velikost. [7] Aplikace jej ve svých počátcích používala, ale se zvyšujcími-se nároky na složitost a množsví dotazů přestalo svými schopnostmi stačit.

Srování obou druhů úložišť je možné nalézt v kapitole 7.

Složka s úložištěm se nachází v domovském adresáři uživatele aplikace (viz tab. 5.1).

Windows:	`%HOMEDRIVE%%HOMEPATH%\.collector`
GNU/Linux:	`~/.collector`

Tabulka 5.1: Umístění repozitáře.

6 Uživatelská příručka k aplikaci

V této kapitole jsou stručně popsány funkce a způsoby ovládání aplikace. Čtenář, který nečetl předešlé kapitoly, by se měl obeznámit alespoň s některými termíny z kapitoly 2.4.

6.1 Hardwarové požadavky

Aplikace pro svůj běh vyžaduje minimálně 128 MiB RAM, ale doporučen je alespoň 1 GiB. Uživatel tím předejde problémům (nejen) při importu metadat, kdy může při nedostatku paměti docházet k prodlevám či dokonce pádům aplikace.

Uživatelské rozhraní aplikace dosud není zcela uzpůsobeno pro možnost ovládání klávesnicí, takže je vyžadována přítomnost myši nebo podobného polohovacího zařízení.

Pro přehrávání importovaných skladeb je samozřejmostí také přítomnost zvukové karty.

6.2 Softwarové požadavky

Obecně podporovanými platformami jsou Microsoft Windows a GNU/Linux v 32-bitových i 64-bitových verzích s nainstalovaným prostředím Java Runtime Environment 6 nebo kompatibilním.

Funkčnost aplikace byla testována na operačních systémech Microsoft Windows 7 Professional (32b i 64b), Ubuntu 11.10 (32b i 64b, jádro 3.0.4), Debian 6.0.4 (64b, jádro 2.6.31.6) a LinuxMint 12 (32b, jádro 3.0.0.12). S úspěchem byla použita běhová prostředí Java SE Runtime Environment 1.6.0.29 a OpenJDK Runtime Environment IcedTea7 2.0.

Pro sestavení aplikace ze zdrojových kódů je navíc potřeba Java Development Kit kompatibilní s Oracle Java SE 6 Development Kit 1.6 (sestavení bylo úspěšně testováno s OpenJDK 7) a Apache Ant (sestavení bylo úspešně

provedeno pomocí Apache Ant 1.8.3).

Audio přehrávač, který je součástí aplikace, vyžaduje pro svou funkci DirectShow [16] (Windows), resp. GStreamer [3] (GNU/Linux).

6.3 Sestavení

Aplikaci lze sestavit spuštěním dávkového souboru `build.bat` (Windows) nebo `build.sh` (GNU/Linux) umístěného v kořenovém adresáři balíku zdrojových kódů. Po úspěšném sestavení se zde objeví adresář `build`, který obsahuje podadresáře `dist` a `temp` a soubor `collector.zip`.

Adresář `dist` obsahuje sestavenou aplikaci připravenou k použití, soubor `collector.zip` je jeho komprimovaná verze. Adresář `temp` obsahuje přeložené zdrojové kódy a má význam pouze při opětovném překladu.

6.4 Instalace a spuštění

Aplikace se spouští souborem `collector.exe` (Windows), resp. `collector.sh` (GNU/Linux) umístěným v kořenovém adresáři aplikace. Není potřeba ji jakkoliv instalovat.

6.5 Ovládání

Obrázek 6.1 zobrazuje hlavní okno aplikace tak, jak vypadá při běžném používání. Čísla označují jednotlivé prvky popsané níže.

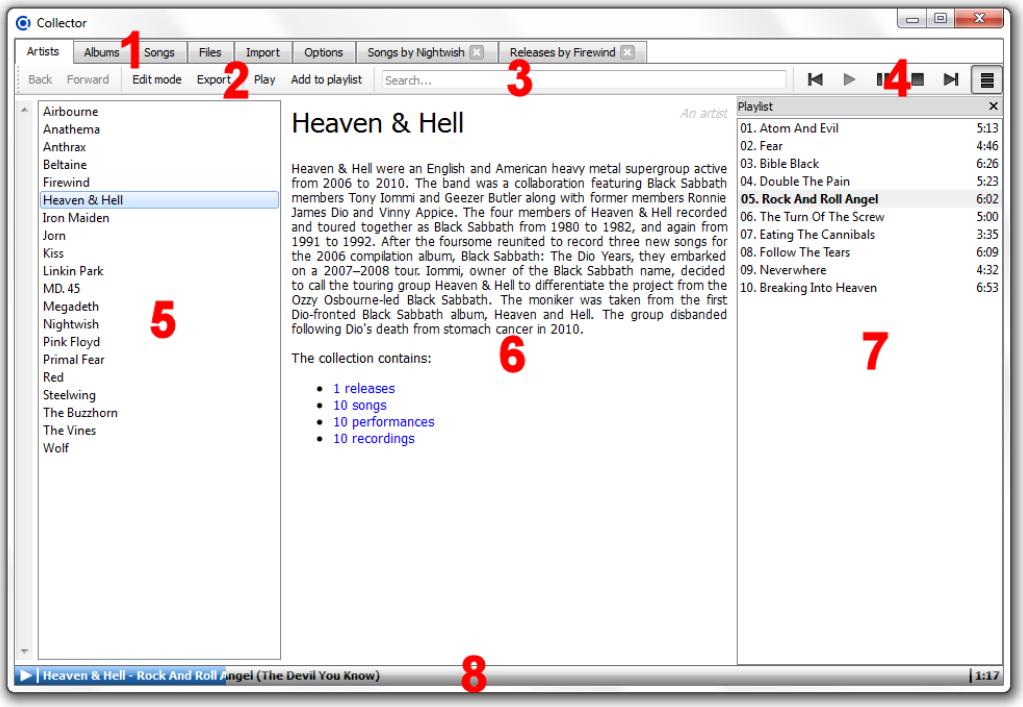
1. Panel inspirovaný moderními webovými prohlížeči. Obsahuje šest výchozích záložek lišících se v tom, co zobrazují:
   - **Artists** - Všechny dostupné umělce a jejich popis.
   - **Albums** - Veškerá alba dostupná v repozitáři.
   - **Songs** - Všechny skladby z repozitáře.

- **Files** - Seznam naimportovaných souborů.
- **Import** - Průvodce importem metadat.
- **Options** - Další možnosti.

Kromě posledních dvou spočívá funkce těchto záložek v tom, že zobrazují obsah repozitáře po aplikaci filtrování – tj. např. záložka *Artists* zobrazuje ze všech individuí z repozitáře právě taková, která popisují umělce.

Výběr se dá určitým způsobem dále zúžit (např. ze všech alb zobrazit pouze alba od jednoho konkrétního interpreta) a otevřít na nové, uzavíratelné záložce.

Záložky *Artists*, *Albums*, *Songs* a *Files* nelze zavřít.



Obrázek 6.1: Hlavní okno aplikace

2. Panel s funkcemi týkajícími se právě vybrané položky a tlačítky pro pohyb v historii hypertextového prohlížení (viz 6.5.4).

3. Políčko pro rychlé vyhledávání v názvech. Lze jej najít i na jiných místech aplikace.

4. Ovládací prvky přehrávače.
5. Seznam filtrovaných individuí z repozitáře. Pro jedno nebo žádné individuum je tento seznam vždy skrytý.
6. HTML popis právě vybraného individua. V pravém horním rohu se zobrazuje jeho typ.
7. Playlist – seznam stop pro přehrávání (ve výchozím stavu skrytý).
8. Panel, který je viditelný pouze, pokud právě probíhá nebo je pozastaveno přehrávání skladby. Zobrazuje stav přehrávání a umožňuje kliknutím přeskočit na libovolnou pozici ve skladbě.

6.5.1 Import RDF metadat

Prvním způsobem, jak vložit do repozitáře aplikace nějaká metadata, je importovat soubor v RDF/XML formátu. Tato funkce je užitečná hlavně při obnově dříve zálohovaného repozitáře nebo nějakého individua z něj.

1. V hlavním okně aplikace vyberte záložku *Import*.
2. Vyberte *Resource Description Framework file* a klikněte na *Next*.
3. Po kliknutí na tlačíko *Browse* vyberte soubor, který budete importovat.
4. Kliknutím na *Next* zahajte import.

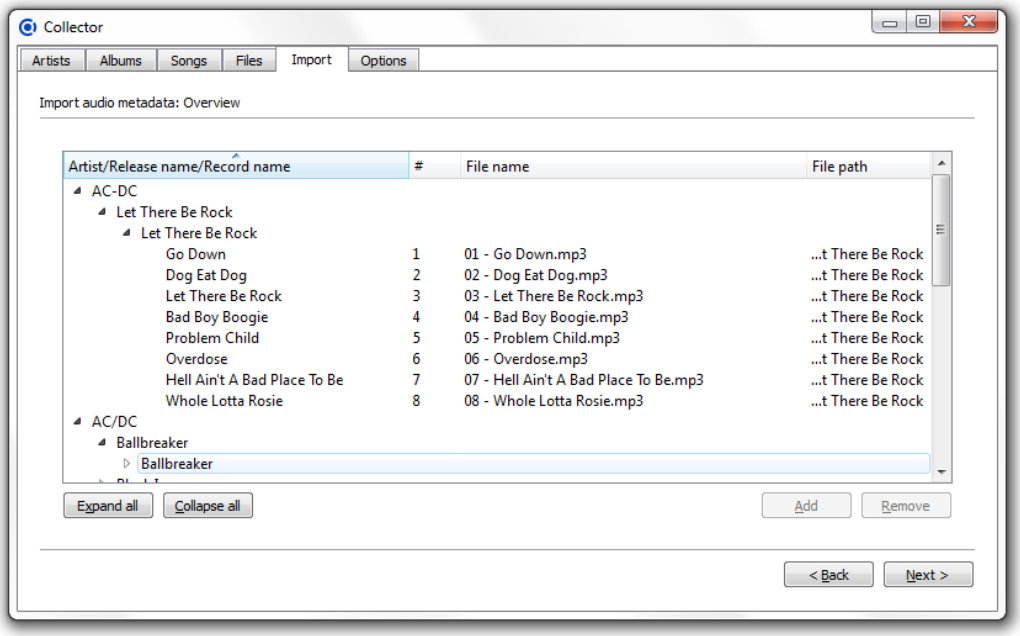
6.5.2 Import audio metadat

Aplikace v současné chvíli podporuje import metadat z těchto audio formátů:

- FLAC
- MP3
- MP4
- Ogg Vorbis

- RA (omezeně)
- WAV (omezeně)
- WMA

Soubory, které aplikace nepodporuje, budou při importu ignorovány.



Obrázek 6.2: Prohlížení a editace metadat před importem.

1. Pro spuštění průvodce importem audio metadat klikněte na záložku *Import* v hlavním okně a po výběru *Audio metadata* klikněte na tlačítko *Next*.

2. Na následující stránce přidejte do seznamu audio soubory, které chcete importovat. Lze přidávat jak jednotlivé soubory nebo jejich skupiny, tak i obsah celých adresářů (vyberou se pouze soubory, o kterých se aplikace domnívá, že je zvládne zpracovat). Pokud jste s výběrem spokojeni, stiskněte *Next*.

3. V dalším kroku aplikace načte informace z vybraných souborů do paměti. Jakmile je proces dokončen, lze stisknout tlačítko *Next* pro pokračování.

4. Nyní si můžete prohlédnout data, která budou po další stisknutí *Next* importována. Povoleny jsou základní úpravy jako editace názvu či čísla stopy (dvojklik na položku), přesuny (táhnutí položky myší) či přidávání a odstraňování položek (pomocí tlačítek *Add* a *Remove*).

5. Poslední obrazovka zobrazuje průběh importu. Do výsledku se promítnou všechny úpravy provedené v předchozím kroku.

6.5.3 Export metadat

Metadata lze z repozitáře exportovat do souborů formátu RDF/XML a to dvěma způsoby. Prvním způsobem je export právě vybraného individua. Stačí kliknout na tlačítko *Export* na panelu pod záložkami. Společně s ním se budou exportovat i všechna s ním propojená. Proto může dojít při exportu umělce rovněž k exportování celé jeho tvorby.

Druhý způsob je export veškerých dat týkajících se hudby – opět do souboru ve formátu RDF/XML. Tlačítko, které má tuto funkci na starosti, se nachází pod záložkou *Options*.

6.5.4 Prohlížení

Prohlížet popis jednotlivých individuí umožňuje aplikace prostřednictvím jednoduchého HTML prohlížeče, jehož největší předností je možnost hypertextové navigace mezi daty, která funguje úplně stejně, jako u webového prohlížeče. Stejně jako tam, i zde jsou k dispozici tlačítka pro pohyb zpět a vpřed (*Back* a *Next*).

Odkazy aktivované na záložkách *Artists*, *Albums*, *Songs* a *Files* se vždy otevírají na nové záložce. Odkaz na uzavíratelné záložce lze otevřít na jiné, pokud jej aktivujete prostředním tlačítkem myši nebo klasicky současně držíc tlačítko `Ctrl`.

Nová záložka se otevírá i po stisknutí klávesy `Enter` v poli rychlého vyhledávání na neuzavíratelné záložce. Na nové (uzavíratelné) záložce se poté zobrazí vyhledávánému řetězci odpovídající individua a původní záložka se vrátí do normálního stavu.

Na uzavíratelné záložce je třeba pro otevření nové záložky potvrdit vyhledávání pomocí `Ctrl+Enter`.

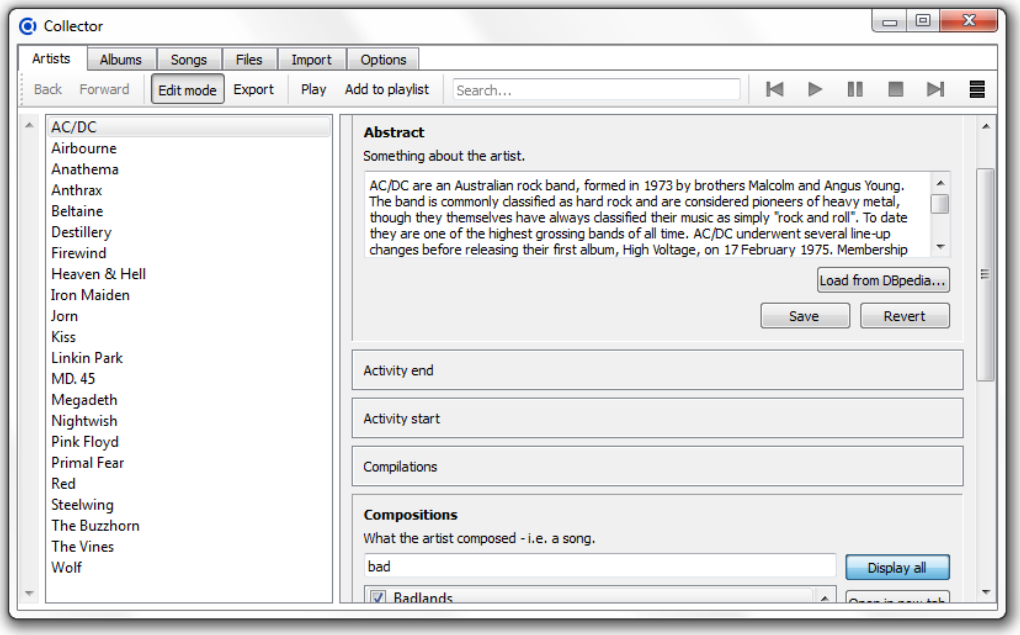
Vyhledání lze okamžitě potvrdit stisknutím `Enter`. Bez potvrzení je prodleva mezi zadáním hledané fráze a vyhledáním 300 milisekund.

6.5.5 Editace

Do režimu úprav lze přepnout tlačítkem *Edit mode* na horním panelu v hlavním okně aplikace. Zmizí HTML popis právě vybrané položky a namísto něj se zobrazí rozhraní pro editaci.

Úpravy se provádí na každém RDF atributu zvlášť a pro načtení a zobrazení hodnoty je každý z nich potřeba nejprve rozkliknout.

Po provedení úprav lze tyto úpravy uložit příslušným tlačítkem *Save* nebo vrátit pomocí tlačítka *Revert*. Veškeré změny se okamžitě projeví i na ostatních otevřených záložkách.



Obrázek 6.3: Editace individua.

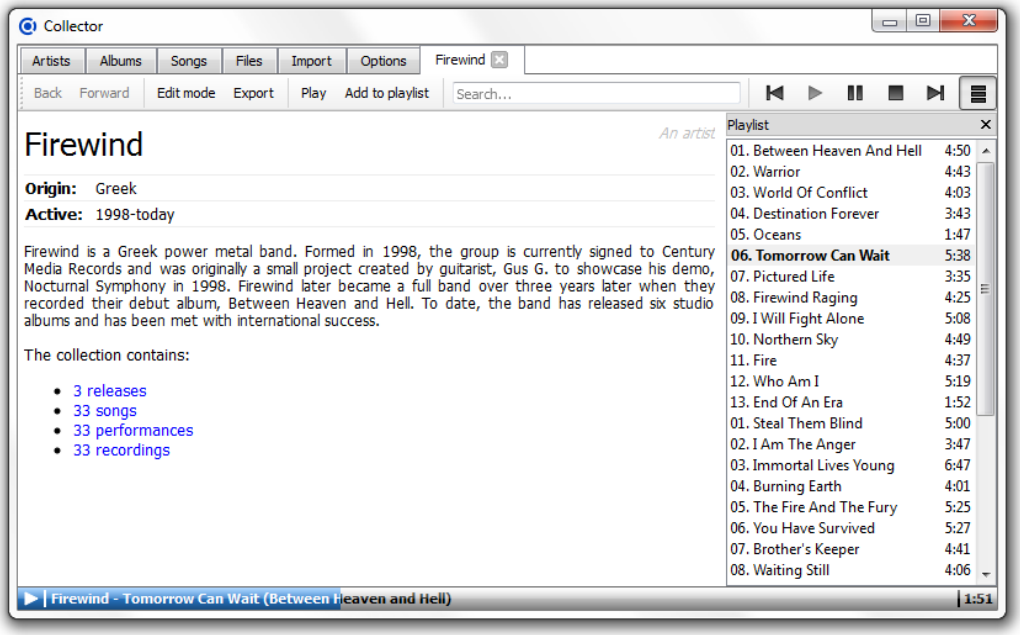
Hodnotu atributu *Abstract* u individuí popisujících umělce lze vyhledat prostřednictvím webového serveru `www.dbpedia.org` pomocí tlačítka *Load*

from DBpedia. Jde o víceméně experimentální funkci. Hledání probíhá pouze podle jména a pokud existuje více stejně se jmenujících umělců, nemusí se stažený abstrakt týkat zrovna umělce, kterého vybrané individuum popisuje.

Kromě úprav atributů lze v tomto režimu individua také mazat, spojovat a duplikovat prostřednictvím tlačítek *Remove*, *Merge* a *Duplicate*.

6.5.6 Přehrávání

Aplikace obsahuje vestavěný přehrávač, kterým lze přehrávat v zásadě veškerý importovaný obsah.



Obrázek 6.4: Přehrávání.

Přehrávají se samozřejmě vždy soubory, odkud byla metadata importována (jejich seznam skrývá záložka *Files*), ale „přehrát“ lze takřka všechna individua[1], která v repozitáři jsou. Aplikace si automaticky odvodí, které

[1]Samozřejmě pouze bavíme-li se o běžném užívání aplikace. Např. pomocí editace nebo importem ručně upravených souborů s metadaty lze mít v repozitáři individua, jejichž „přehrávání“ aplikace nepodporuje nebo nejsou napojena na žádné podporované audio soubory.

soubory k vybranému individuu patří. Přehrát tak lze např. veškerou tvorbu vybraného umělce či všechny dostupné interpretace vybrané sklady.

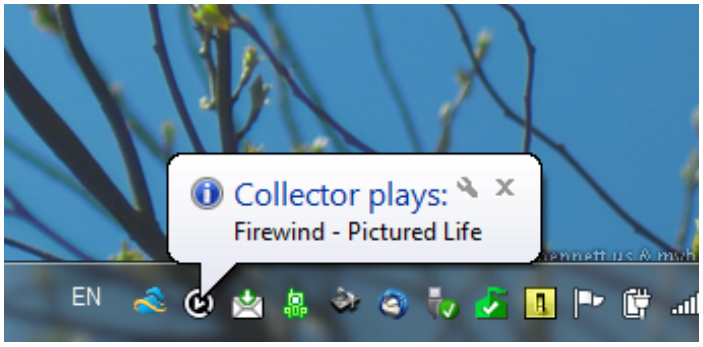
K předání individua přehrávači lze využít dvou tlačítek na panelu. Tlačítko *Add to playlist* pouze přidá soubory (stopy) svázané s vybraným individuem do *playlistu*[2]. Tlačítko *Play* naproti tomu playlist nejprve vyprázdní a po jeho naplnění stopami souvisejícími s vybraným individuem rovnou spustí přehrávání.

Přehrávač samotný má své ovládací prvky na pravé straně panelu a ovládat jej lze i s pomocí multimediálních kláves. Tlačítka mají (až na jedno) ikony známé z jiných přehrávačů a nepovažuji za nutné popisovat zde jejich funkci. Tlačítko úplně vpravo přepíná zobrazení playlistu. Pokud je playlist skrytý a byl do něj přidán nový obsah, ikona tohoto tlačítka se dočasně pozmění a upozorní na to.

6.5.7 Ikona v trayi

Aplikace má vlastní ikonu v *trayi*[3]. Při startu aplikace se zobrazuje jako první a upozorňuje na její probíhající inicializaci (čímž supluje *splash screen*[4]).

Další funkce ikony v trayi se v zásadě týkají pouze přehrávače.



Obrázek 6.5: Ikona v trayi.

Pokud probíhá přehrávání, aplikace se namísto vypnutí do této ikony

[2]Seznam stop přehrání.

[3]Upozorňovací oblast na hlavním panelu grafického rozhraní operačního systému vedle hodin.

[4]Úvodní obrazovka zpravidla ukazující logo aplikace s informací, že tato aplikace právě startuje.

pouze skryje a přehrávání pokračuje. Stav přehrávání je vyjádřen tvarem ikony a během něj zobrazuje ikona bublinu s informacemi o aktuálně přehrávané stopě.

Pro vypnutí aplikace je potřeba přehrávání nejprve zastavit (resp. pozastavit) nebo tak lze učinit prostřednictvím kontextové nabídky tray ikony. Ta obsahuje také ovládací prvky přehrávače a položku *Restore*, která opět zobrazí skryté okno aplikace (lze provést rovněž dvouklikem na ikonu; viditelné okno dvouklik skryje, skryté zobrazí).

6.6 Známé problémy a nedostatky

- **Ikona v trayi se nezobrazuje.**

  Grafické uživatelské rozhraní Unity některých Linuxových distribucí nutí uživatele přidávat na *whitelist*[5] aplikace, u kterých si přeje jim dát možnost zobrazovat svou vlastní ikonu v trayi. Pokud se ikona této aplikace nezobrazuje, bude nutné ji tam přidat také.

- **Přehrávač se chová zvláštně nebo nepřehrává některé soubory.**

  Funkčnost přehrávače z velké míry závisí na možnostech DirectShow (Windows) resp. GStreameru (GNU/Linux). Pokud přehrávač nepřehrává některé formáty audio souborů, pak bude zapotřebí do systému doinstalovat příslušné filtry.

  Aktuálně nainstalované filtry mohou rovněž způsobovat nestandardní chování přehrávače. Během vývoje byla zjištěna nepříjemná vlastnost výchozích filtrů pro zpracování MP3 souborů pomocí DirectShow, kdy byly některé MP3 soubory přehrávačem rovnou odmítnuty a u některých byla špatně odhadnuta délka skladby (což mělo za následek podivné fungování ukazatele průběhu skladby). Tento problém byl úspěšně vyřešen instalací filtrů z balíku LAV Filters[6], které nejen že zmíněné problémy vyřešily, ale přinesly například i podporu FLAC. Jejich instalaci tedy lze uživateli používajícímu aplikaci pod Windows jen doporučit.

- **Aplikace ohlásila, že se nezdařil import RDF metadat, která z ní byla dříve exportována. Po jejím dalším startu je ale vidět,**

[5] Seznam obsahující entity, které jsou nějakým způsobem privilegované.
[6] K dispozici na `http://1f0.de/downloads/`.

že data importována byla – pouze ne správně.

Úspěšně importovat lze pouze RDF/XML soubory, jejichž kódování se shoduje s údajem uvedeným v jejich hlavičce. Bohužel – knihovna Sesame, která má na starosti také export/import do/z RDF/XML, občas vyexportuje soubor kódovaný tak, že jej sama při importu nedokáže správně přečíst.

Jde o problém, který lze zatím vyřešit pouze manuální změnou kódování u vyexportovaného souboru, resp. jeho otevřením a uložením správným způsobem.

- **Editace položky v přehledu před importem nefunguje.**

  Jde o problém některých MP3 souborů, kdy si knihovna JAudioTagger neporadí se stávající strukturou jejich ID3 tagu a nepodaří se jí tag upravit.

  Problém lze dočasně vyřešit manuální editací tagu v jiném editoru ještě před samotným importem.

- **Nelze vyplnit rok vydání alba nebo žánr.**

  Aplikace tyto možnosti zatím nemá. Rok vydání alba je součástí individua popisujícího samotnou událost vydání alba a příslušná třída zatím nebyla v aplikaci implementována. Implementována dosud nebyla ani třída, jejíž instance popisují žánry a ani rozhraní, které by umožňovalo jejich správu.

- **Aplikace si neporadí s větším množstvím dat.**

  Je to nepravděpodobné, ale v případě, že je v repozitáři opravdu obrovské množství dat a aplikace trpí zasekáváním nebo selhávají pokusy o import velkého počtu souborů (řádově tisíce až desetitisíce), může být potřeba povolit aplikaci využívat větší množství paměti.

  Ve výchozím stavu je maximální množství využívané operační paměti nastaveno na 1 GiB.

  Pro změnu maximálního množství využívané operační paměti:

  – **Pod Windows:** V libovolném textovém editoru otevřete soubor `collector.vmparams`. Jeho obsah by měl vypadat takto:

  ```
  -Xms128m -Xmx1g -Djava.library.path=./native/lib
  ```

– **Pod GNU/Linux:** V libovolném textovém editoru otevřete soubor `collector.sh`. Jeho obsah by měl vypadat takto:

```
export LD_LIBRARY_PATH=./native/lib:$LD_LIBRARY_PATH
java -Xms128m -Xmx1g -jar collector.jar
```

Úprava je pro oba systémy stejná. `1g` v parametru `-Xmx1g` stanovuje maximální velikost operační paměti využívané aplikací na 1 GiB. Změnou na např. `2g` navýšíte toto množství na 2 GiB, ale lze použít i např. `1500m` pro navýšení na 1500 MiB.

Po provedení úprav soubor uložte a aplikaci restartujte.

- **Aplikace dlouho startuje.**

  Při startu aplikace je do operační paměti načítán celý repozitář s metadaty. Pokud start aplikace trvá příliš dlouho, pak je to zřejmě způsobeno jeho větší velikostí.

  Nejde o chybu. Repozitář je do paměti načítán celý z důvodu pozitivního vlivu na celkový výkon aplikace (viz kap. 5.4).

7 Dosažené výsledky

Výkon aplikace byl testován na sbírce o velikosti 56 GiB (zhruba 5800 hudebních souborů) na stroji Lenovo Thinkpad R61 s Intel Core 2 Duo T8100 a 2 GiB RAM s operačním systémem Mirosoft Windows 7 Professional v 32-bitové verzi a běhovým prostředím Java SE Runtime Environment 1.6.0.29.

Sbírka byla uložena na pevném disku s rychlostí 7200 otáček za sekundu připojeném přes rozhraní SATA 2.0.

Pro potřeby měření byl z hudební sbírky postupně odebrán vzorek 1250, 2500 a 5000 MP3 souborů a pro každý z nich byly provedeny následující kroky:

1. Vzorek byl importován do prázdného repozitáře, přičemž se měřila doba, po kterou import trval. Do této doby nebyla započítána doba skenování metadat z jednotlivých souborů.

2. Aplikace byla následně vypnuta a bylo zapsáno její hlášení o průměrné době trvání vyhodnocení jednoho SPARQL dotazu. Protože nebyla aplikace po importu dále používána, jde průměrný čas vyhodnocení SPARQL dotazu během importu. Více o importu v kapitole 5.3.6.

3. Poté byla zkontrolována velikost repozitáře.

4. Po opětovném zapnutí aplikace informovala o počtu trojic, které repozitář aktuálně obsahuje a následoval scénář z „běžného“ používání aplikace – listování v seznamu umělců, alb a skladeb a vyhledání skladeb, jejichž název obsahuje řetězec `"the"`. Z nich byla libovolná skladba vybrána a v editačním režimu sjednocena s jinou (tj. ze dvou vybraných individuí vzniklo jedno, jehož vlastnosti jsou sjednocením obou původních).

   Užití tohoto scénáře má svůj smysl – jsou v něm totiž zahrnuty výpočetně poměrně náročné úkony.

5. Po vypnutí aplikace bylo opět zapsáno její hlášení o průměrné době, po kterou trvalo vyhodnocení jednoho SPARQL dotazu.

Uvedený postup byl realizován pro dva podporované druhy úložišť – NativeStore a MemoryStore (viz kap. 5.4).

Jak měření dopadlo, ukazuje tabulka 7.1 a grafy na obrázcích 7.1, 7.2, 7.3, 7.4 a 7.5.

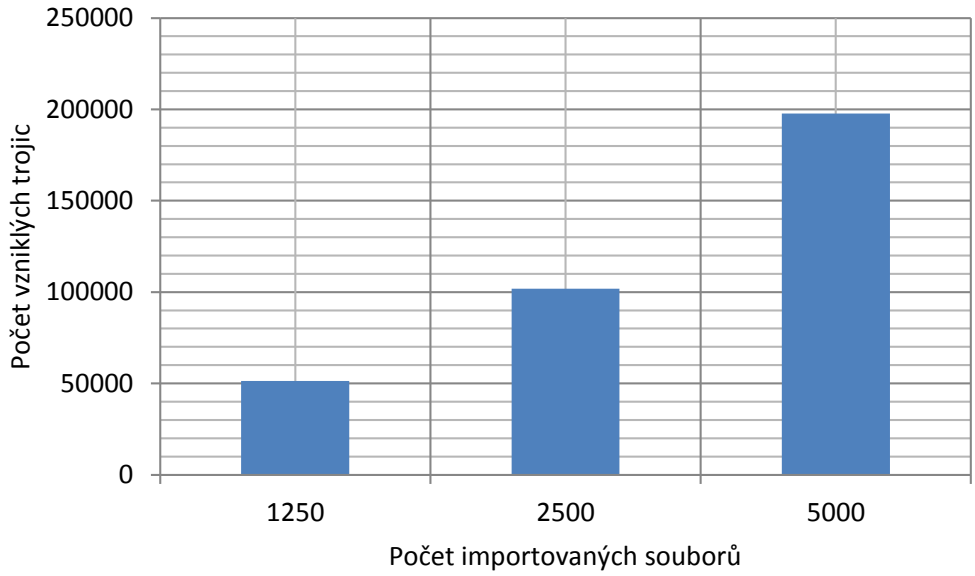
Počet importovaných souborů	1250	2500	5000
Počet vzniklých trojic	51301	101917	197722
Velikost NativeStore [MiB]	4,4	8,9	17,3
Velikost MemoryStore [MiB]	0,45	0,89	1,72
Trvání importu do NativeStore [ms]	69049	123990	299693
Trvání importu do MemoryStore [ms]	9994	19199	41223
Trvání vyhodnocení SPARQL dotazu nad NativeStore během importu [ms]	0,2223	0,213	0,2291
Trvání vyhodnocení SPARQL dotazu nad MemoryStore během importu [ms]	0,0751	0,0558	0,0562
Trvání vyhodnocení SPARQL dotazu nad NativeStore během používání [ms]	1,0096	2,6602	6,4434
Trvání vyhodnocení SPARQL dotazu nad MemoryStore během používání [ms]	0,1237	0,123	0,2424

Tabulka 7.1: Výsledky měření.

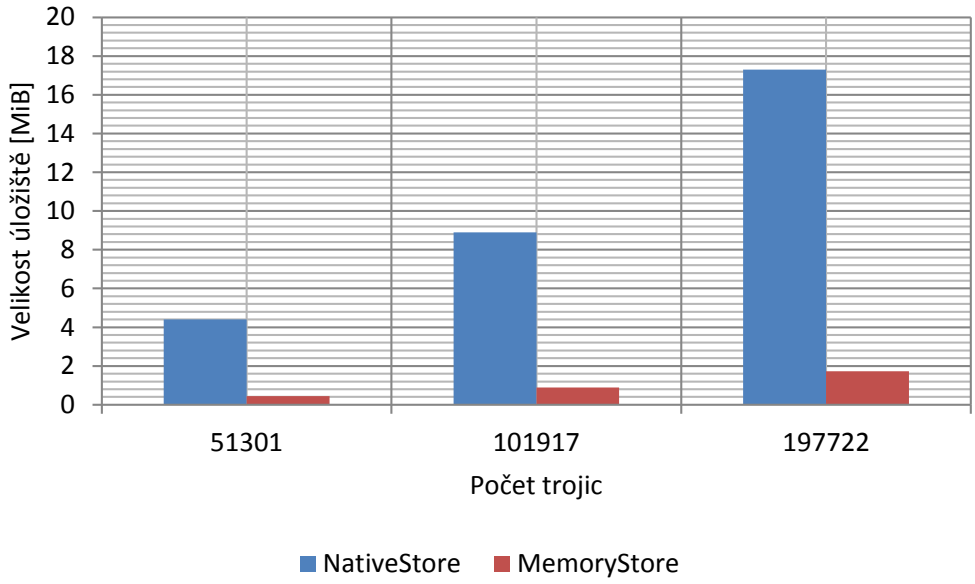
Časy v tabulce 7.1 jsou samozřejmě spíše orientační, neboť výkon aplikace ovlivňovalo mnoho dalších faktorů – např. disková cache a její obsah v době měření. Počet trojic, které se byly během importu přidány do repozitáře je zase silně závislý na obsahu importovaných souborů.

Z tabulky i grafů je nicméně patrné, že je mezi úložišti NativeStore a MemoryStore obrovský výkonostní rozdíl a aplikace je tedy nucena používat MemoryStore i přes jeho nevýhodu spočívající v tom, že je potřeba celé úložiště při startu aplikace nejprve načíst do paměti.

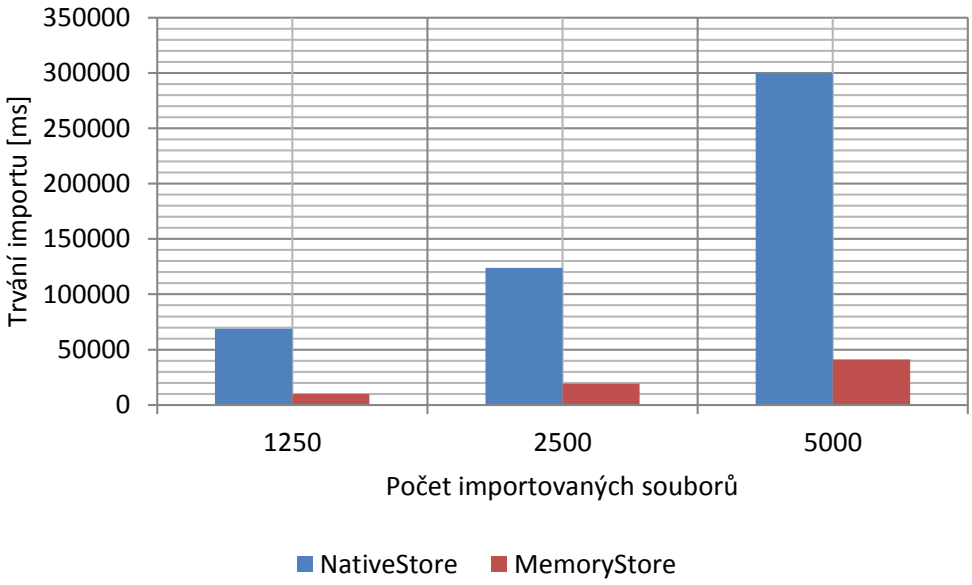
Graf na obrázku 7.4 naznačuje, že se nejvíce SPARQL dotazů vyhodnotí hned zpočátku importu a dále již aplikace využívá vyrovnávací paměti. Rychlost importu proto zůstává na velikosti repozitáře závislá jen minimálně. Toto by se změnilo pouze v případě, že by aplikaci docházela paměť a velikost vyrovnávacích pamětí by byla zredukována.



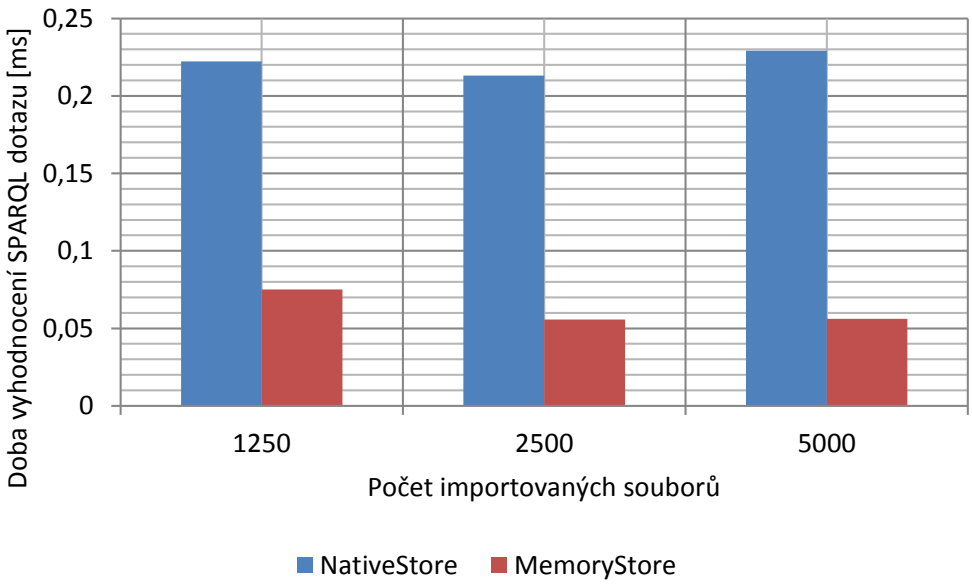
Obrázek 7.1: Závislost počtu trojic vzniklých v repozitáři na počtu importovaných souborů.



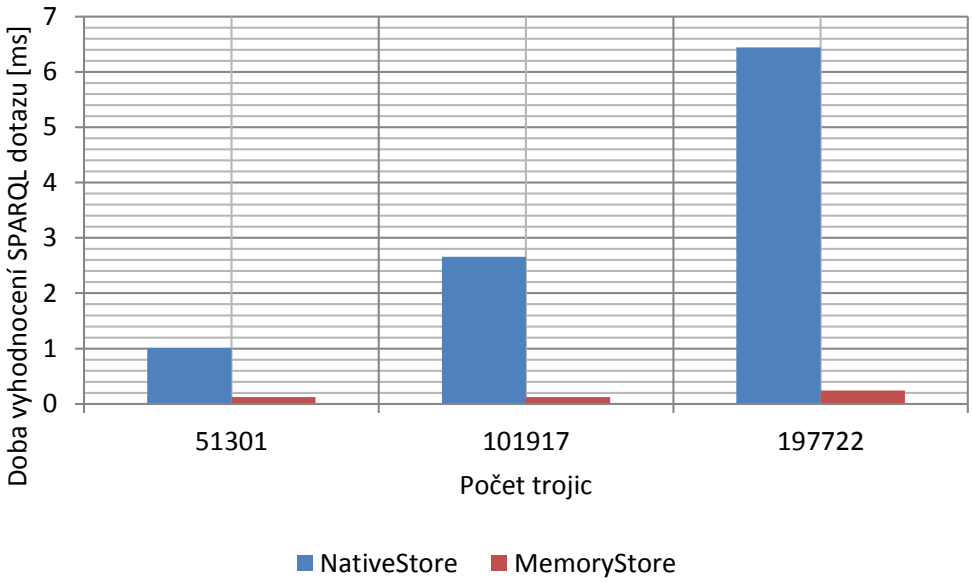
Obrázek 7.2: Závislost velikosti repozitáře na počtu obsažených trojic.



Obrázek 7.3: Závislost doby trvání importu na počtu importovaných souborů.



Obrázek 7.4: Závislost doby vyhodnocení SPARQL dotazu na počtu importovaných souborů (během importu).



Obrázek 7.5: Závislost doby vyhodnocení SPARQL dotazu na počtu trojic v repozitáři (během normálního používání).

8 Závěr

Výsledkem této práce je multiplatformní aplikace použitelná pro správu rozsáhlejší hudební sbírky založená na technologiích Sémantického webu.

Díky mnohým optimalizacím a kompromisům nedocházelo na testovacím stroji (viz kap. 7) k žádným závažným problémům s výkonem. Bohužel i přes veškerou snahu o optimalizaci je znát, že způsob práce s daty, na který se zaměřuje tato práce, není pro zvolenou aplikaci příliš vhodný. Aplikace se totiž snaží vytvořit z uživatelem poskytnutých metadat jakousi znalostní databázi popisující tvorbu jednotlivých interpretů, ale nemá k tomu dostatek podkladů. Metadata z audio souborů jsou velice špatný zdroj informací a aplikace si jich musí během importu mnoho „domýšlet“. V repozitáři tím vzniká mnoho tvrzení, která vůbec nemusejí být pravdivá a v určitých aspektech ani užitečná. Přesto s nimi musí aplikace pracovat a to stojí systémové prostředky.

Během vývoje se také ukázalo, že v uživatelském rozhraní aplikace nachází velké využití návrhový vzor *model-view*, který je ale společně se způsobem, kterým se prostřednictvím SPARQL získávají data z repozitáře, místy obtížně slučitelný. Například v případě, kdy je potřeba v seznamu přemístit kurzor na konkrétní položku, jejíž pozice není dopředu známa, může docházet k citelným prodlevám.

Použití RDF úložiště jako databáze má nicméně i mnohá pozitiva. Aplikace je díky tomu velice dobře rozšiřitelná o další funkce či moduly, nabízí hypertextové procházení obsahu a umožní uživateli nahlížet na jeho hudební sbírku v souvislostech, které by mu jinak zůstaly skryté. Je nicméně stále ve stavu, kdy k plnému využití jejího potenciálu mnoho chybí. Nepříjemná je chybějící možnost doplnit rok vydání alba, zajisté by přišla vhod i podpora dělení tvorby podle žánrů nebo třeba funkce umožňující stáhnout text vybrané skladby z internetu.

Seznam užitých zkratek

FLAC	Free Lossless Audio Codec
HTML	HyperText Markup Language
IDE	Integrated Development Environment
MP3	MPEG-1 Audio Layer III
MP4	MPEG-4 Part 14
OWL	Web Ontology Language
RA	Real Audio
RDF	Resource Description Framework
SDK	Software Development Kit
SeRQL	Sesame Rdf Query Language (vysl. ˈsɛːkl̩)
SPARQL	SPARQL Protocol and RDF Query Language (vysl. ˈspɑːkl̩)
SQL	Structured Query Language
URI	Uniform Resource Identifier
WMA	Windows Media Audio

Literatura

[1] Eclipse. Dostupné z: http://eclipse.org/.

[2] *The American Heritage® Dictionary of the English Language*. Boston, Massachusetts, USA : Houghton Mifflin Harcourt Publishing Company, 4th edition, 2011. Dostupné z: http://ahdictionary.com/word/search.html?q=format.

[3] GStreamer: open source multimedia framework. Dostupné z: http://gstreamer.freedesktop.org/.

[4] Qt - Cross-platform application and UI framework, 2012. Dostupné z: http://qt.nokia.com/.

[5] 2010. Dostupné z: http://www.w3.org/egov/wiki/index.php?title=RDF_Repository&oldid=2081.

[6] *Understanding Metadata*. Bethesda, Maryland, USA : NISO Press, 2004. Dostupné z: http://www.techterms.com/definition/metadata. ISBN 1-880124-62-9.

[7] *User Guide for Sesame 2.3 (Section 7.6.2.: Native store configuration)*. Aduna. Dostupné z: http://www.openrdf.org/doc/sesame2/users/ch07.html#section-repository-config.

[8] BERNERS-LEE, T. – FIELDING, R. – MASINTER, L. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005. Dostupné z: http://www.ietf.org/rfc/rfc3986.txt.

[9] BERNERS-LEE, T. – FISCHETTI, M. *Weaving the Web*. New York : HarperCollins Publishers, 2000. ISBN 0-06-251587-X.

[10] BERNERS-LEE, T. – HENDLER, J. – LASSILA, O. The Semantic Web. *Scientific American Magazine*. 2001.

[11] CASANOVAS, P. Helping New Judges Answer Complex Legal Questions. Dostupné z: http://www.w3.org/2001/sw/sweo/public/UseCases/Judges/. Semantic Web Use Cases and Case Studies, 2007.

[12] ČERNÝ, M. Sémantický web – jak dál? *Ikaros*. 2009, 13, 9. ISSN 1212-5075. Dostupné z: http://www.ikaros.cz/node/5437.

[13] FULGHAM, B. Computer Language Benchmarks Game, 2012. Dostupné z: http://shootout.alioth.debian.org/u32q/which-programming-languages-are-fastest.php.

[14] FÚSTER, J. J. V. WEASEL, Vodafone R&D Corporate Semantic Web. Dostupné z: http://www.w3.org/2001/sw/sweo/public/UseCases/Vodafone-es/. Semantic Web Use Cases and Case Studies, 2007.

[15] *struct*. Microsoft, . Dostupné z: http://msdn.microsoft.com/en-us/library/ah19swz4(v=vs.71).aspx. MSDN Library.

[16] *DirectShow*. Microsoft, . Dostupné z: http://msdn.microsoft.com/en-us/library/ms783323.aspx. MSDN Library.

[17] RAIMOND, Y. et al. *Music Ontology Specification*, 2010. Dostupné z: http://musicontology.com/. Specification Document - 28 November 2010.

[18] SERVANT, F.-P. Semantic Web Technologies in Automotive Repair and Diagnostic. Dostupné z: http://www.w3.org/2001/sw/sweo/public/UseCases/Renault/. Semantic Web Use Cases and Case Studies, 2007.

[19] TAUBERER, J. What is RDF and what is it good for?, 2008. Dostupné z: http://www.rdfabout.com/intro/.

[20] *Resource Description Framework (RDF): Concepts and Abstract Syntax (Section 3.1: Graph Data Model)*. W3C, 2004. Dostupné z: http://www.w3.org/TR/rdf-concepts/#section-data-model. W3C Recommendation 10 February 2004.

[21] *OWL Web Ontology Language Overview (Section 3.1: OWL Lite RDF Schema Features)*. W3C, 2004. Dostupné z: http://www.w3.org/TR/2004/REC-owl-features-20040210/#Individual. W3C Recommendation 10 February 2004.

[22] *OWL Web Ontology Language Reference (Section 4.2.2: owl:inverseOf)*. W3C, 2004. Dostupné z: http://www.w3.org/TR/owl-ref/#inverseOf-def. W3C Recommendation 10 January 2004.

[23] *OWL Web Ontology Language - Overview*. W3C, 2004. Dostupné z: http://www.w3.org/TR/owl-features/. W3C Recommendation 10 February 2004.

[24] *RDF Primer*. W3C, 2004. Dostupné z: `http://www.w3.org/TR/rdf-primer/`. W3C Recommendation 10 February 2004.

[25] *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C, 2004. Dostupné z: `http://www.w3.org/TR/rdf-concepts/#section-triples`. W3C Recommendation 10 February 2004.

[26] *Resource Description Framework (RDF): Concepts and Abstract Syntax (Section 6.1: RDF Triples)*. W3C, 2004. Dostupné z: `http://www.w3.org/TR/rdf-concepts/#section-triples`. W3C Recommendation 10 February 2004.

[27] *SPARQL Query Language for RDF*. W3C, 2008. Dostupné z: `http://www.w3.org/TR/rdf-sparql-query/`. W3C Recommendation 15 January 2008.

[28] *SPARQL 1.1 Query Language*. W3C, January 2012. Dostupné z: `http://www.w3.org/TR/2012/WD-sparql11-query-20120105/`. W3C Working Draft 05 January 2012.

[29] *Turtle - Terse RDF Triple Language*. W3C, 2011. Dostupné z: `http://www.w3.org/TeamSubmission/turtle/`. W3C Team Submission 28 March 2011.

[30] ČADA, R. – KAISER, T. – RYJÁČEK, Z. *Diskrétní matematika*. Plzeň, Česká Republika : Západočeská univerzita v Plzni, 2004. ISBN 80-7082-939-7.

Seznam obrázků

2.1 Vyjádření vlastností dat. 3

2.2 Vztah dat k okolí. 3

2.3 Klasifikace dat. 4

5.1 Převod mezi Java třídou a pojmem z ontologie. 18

6.1 Hlavní okno aplikace 29

6.2 Prohlížení a editace metadat před importem. 31

6.3 Editace individua. 33

6.4 Přehrávání. 34

6.5 Ikona v trayi. 35

7.1 Závislost počtu trojic vzniklých v repozitáři na počtu importovaných souborů. 41

7.2 Závislost velikosti repozitáře na počtu obsažených trojic. . . . 41

7.3 Závislost doby trvání importu na počtu importovaných souborů. 42

7.4 Závislost doby vyhodnocení SPARQL dotazu na počtu importovaných souborů (během importu). 42

7.5 Závislost doby vyhodnocení SPARQL dotazu na počtu trojic v repozitáři (během normálního používání). 43

A.1 Základní scénáře užití funkcí poskytovaných aplikací. 52

Seznam tabulek

5.1 Umístění repozitáře. 26

7.1 Výsledky měření. 40

Seznam výpisů kódu

2.1 Ukázka RDF/XML. 5

2.2 Ukázka Notation3. 6

2.3 Data pro SPARQL dotaz 2.4. 7

2.4 Ukázka SPARQL dotazu nad daty 2.3. 7

5.1 Propojení Java objektu s individuem z repozitáře. 20

5.2 Získání individuí z repozitáře (nejjednodušší způsob). 20

5.3 Získání individuí z repozitáře. 21

5.4 Dotaz pro kód 5.3. Za klíčovým slovem FROM je identifikátor kontextu, ve kterém pracuje modul, který dotaz vygeneroval. . 21

5.5 Manipulace s jednoduchým atributem. 22

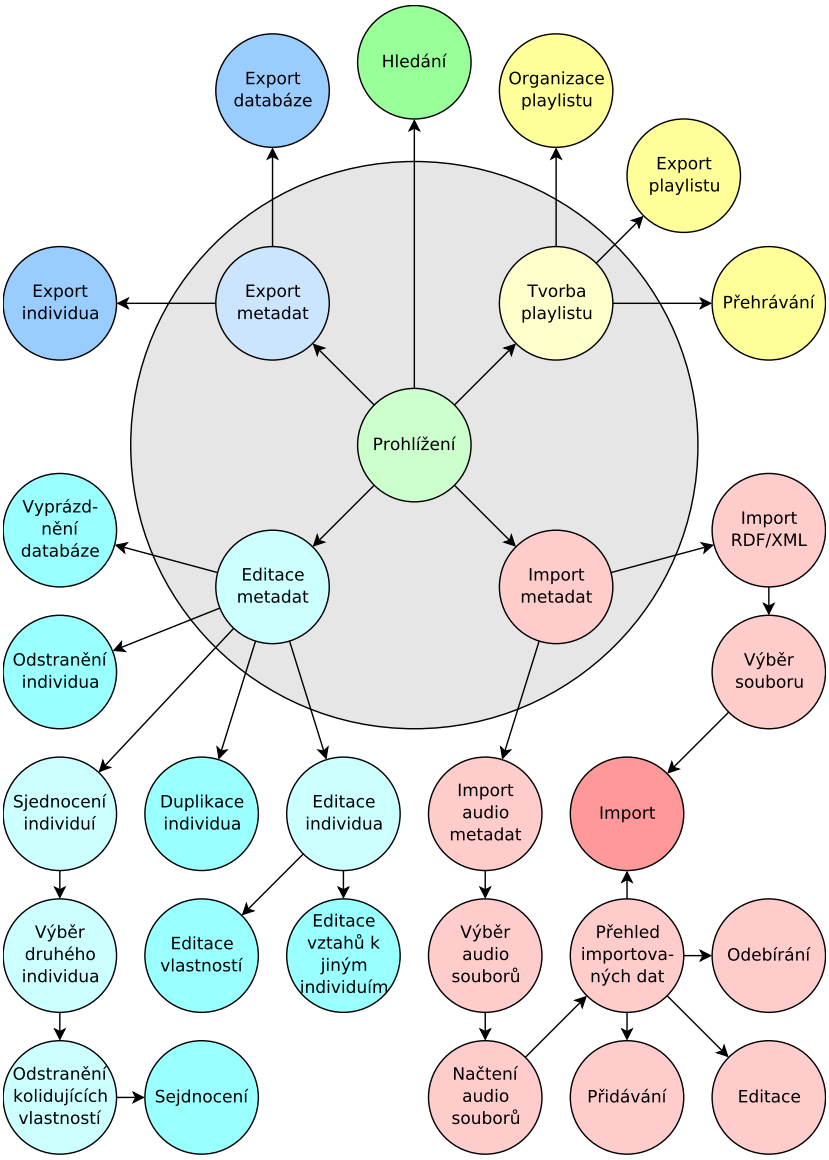
5.6 Definice atributů. 22

5.7 Vytvoření nového individua s náhodně generovaným identifikátorem. 22

5.8 Řazení a vyhledávání. 24

Příloha A

Užití aplikace



Obrázek A.1: Základní scénáře užití funkcí poskytovaných aplikací.