Répássy Adrienn

Komplex MI alkalmazások
Kereső zenei ontológiában
Házi feladat

Bevezetés

A zenei ontológiai kereső egy olyan webalkalmazás, amely egy zenei ontológia példányainak böngészését teszi lehetővé. Ehhez egy keresőfelületet nyújt, amelybe egy példány nevét kell beírni, hogy lássuk róla az adatokat, ill. a kapcsolódó példányok neveit. Ha olyan nevet írunk be a keresőbe, amilyen nevű példány nincs az ontológiában, akkor az applikáció kilistázza az elérhető előadó példányok neveit.

Az alkalmazás egy REST alapú Java EE 8-as Maven applikáció, amely szerverként Wildfly 12-t használ. A Maven által megfogalmazott függőségek között megtalálható a Wildfly, a JBoss a Wildfly használatához, valamint az OWL API. Az alkalmazás a HermiT következtetőt használja, amelynek jar fájlja megtalálható a projektben.

Ontológia

Az alkalmazáshoz használt ontológia az interneten talált Music Ontology (http://musicontology.com/), amely a zenei fogalmakat és az azok közötti kapcsolatokat rendszerezi. Az ontológiában lévő fogalmak között előadó, együttes, tagság, aktivitás, album, dal, szerző, dalszöveg, hangmérnök, fellépés, kiadó, különböző kiadások, lemezformátumok, stílus, hangszer stb. találhatók. Az ontológiában szerepelnek még a fogalmak közötti kapcsolatok és a fogalmak adat property-jei is. Konkrét példányokat azonban nem tartalmaz az ontológia a fogalmakból. Az ontológia eredeti RDFS forrása a motools GitHub repository-jából tölthető le (https://github.com/motools/musicontology/blob/master/rdf/musicontology.rdfs).

A Music Ontology több külső ontológiát is beimportál, hogy azok fogalmait használja, és még azoknak is vannak függőségeik. Az importok között vannak az időt, az eseményeket, a kapcsolatokat, a hasonlóságot, az alkotást stb. leíró ontológiák.

Módosítása

A feladat megoldásához azonban módosítanom kellett a Music Ontology-t. Ugyanis, az egyik indirekt függőség (függőség függősége), a *http://purl.org/ontology/similarity* nem volt elérhető, sem a Protégé, sem a Java alkalmazás nem tudta betölteni az internetről, miközben betöltötték a Music Ontology-t. Ezért a zenei ontológia ettől függő direkt importját, a *http://purl.org/ontology/ao/core* ontológiát el kellett távolítanom a függőségek közül. Szerencsére ez nem volt egy fontos függőség, csak egy genre property-t használt belőle a zenei ontológia, amelynek a subproperty-jének definiálták a saját genre object

property-jüket, amely önmagában is megállja a helyét, hiszen leírja, hogy mely fogalmak között teremt kapcsolatot.

Voltak azonban további problémás függőségek is, amelyeket szintén el kellett távolítanom ahhoz, hogy a Java alkalmazás be tudja tölteni a zenei ontológiát. Ugyanis, a *http://purl.org/vocab/bio/0.1/* és a *http://purl.org/vocab/frbr/core* függőségeket nem tudta a Java alkalmazás beparszolni, szintaktikai hibát talált bennük, bármely ontológia parszolóval is próbálkozott (ezek együtt sokféle ontológia formátumot feltételeznek). A bio ontológiából csak az esemény property-t használta más property-k őseinek, míg a frbr függőség fontosabb volt, mert az alkotással kapcsolatos ősosztályokat és ősproperty-ket hozta be, és több osztály és property volt ezeknek a leszármazottai, és egyéb property-knek is a tárgyai vagy tulajdonosai voltak. Azonban a fő fogalmakat és kapcsolatokat ennek a két függőségnek az eltávolítása sem érintette.

A megmaradt függőségek a következők: a *http://purl.org/NET/c4dm/keys.owl*, az időt leíró *http://www.w3.org/2006/time* ontológia, az eseményekkel kapcsolatos *http://purl.org/NET/c4dm/event.owl*, valamint az emberi kapcsolatokat leíró *http://xmlns.com/foaf/0.1/*.

Használata

Néhány függőség eltávolítása miatt nem működött az a megoldás, hogy egy új ontológiát hozzak létre, amely függőségként használja a Music Ontology-t, mert ebben az esetben az interneten található verziót próbálja betölteni a Java alkalmazás, hiába importáltam be az általam módosított Music Ontology forrásfájlt a link helyett, akkor is a linket jelölte meg függőségként az új ontológia forrása. És természetesen a fenti hibákba ütközött a betöltése.

Ezért közvetlenül a megnyirbált Music Ontology-ba vettem fel a fogalmak konkrét példányait. Az egyszerűség kedvéért kézzel hoztam létre néhányat Protégé-ben.

Példányok

Először felvettem egy együttes példányt, amelynek a neve *PinkFloyd*. Típusa **MusicGroup**, amely a MusicArtist és a Group osztályok leszármazottja. A MusicGroup-hoz tartozik egy **member** object property, amelynek a tulajdonosa a Group, és a tárgya az Agent. Az Agent osztálynak minden olyan osztály a leszármazottja, amely valamilyen személyiséggel ruházható fel, pl. szervezetek, köztük vannak az előadók (MusicArtist) is. Tehát egy csapatnak lehetnek előadók is a tagjai, ezért felvettem az ontológiába négy **MusicArtist** példányt: *DavidGilmour*, *NickMason*, *RickWright* és *RogerWaters*. A *PinkFloyd* együtteshez négyszer vettem fel a member object property-t ezzel a négy előadóval.

Az ontológiában **singer** object property is van, amelynek nem az Agent vagy valamelyik leszármazottja a tulajdonosa, hanem a Performance, amelynek semmi köze az Agent-ökhöz, mégis működött a *PinkFloyd*-hoz a singer property felvétele (nem tudja, hogy lehet-e Performance a *PinkFloyd*, vagy nem, és az Agent és a Performance osztályok sem zárják ki egymást). Tudván, hogy az említett tagok közül hárman is énekeltek a bandában, ezért háromszor vettem fel a singer property-t a PinkFloyd-hoz, azzal a három taggal.

ProgressiveRock néven egy **Genre** osztály példányt is felvettem, és a *PinkFloyd* példány **genre** property-jéhez rendeltem hozzá. A genre valóban az Agent-ökhöz rendeli a Genre-t.

A *PinkFloyd*-hoz data property-ket is felvettem: az általános **name** property-t, amelynek tulajdonosa a Thing, az értéke „Pink Floyd". Ezen kívül az **activity start** és az **activity end** data property-ket is felvettem, amelyek MusicArtist-ra vonatkoznak, és értékük egy dátum. Én évszámokat rendeltem ehhez a két property-hez.

Ezután létrehoztam egy albumot is a csapatnak, a neve *DarkSideOfTheMoon*, a típusa pedig **Record**. Ezt az albumot a csapat **published** object property-jéhez rendeltem hozzá, amely Agent-ökhöz rendel MusicalManifestation-öket. Szerencsére a Record a MusicalManifestation leszármazottja. A *DarkSideOfTheMoon*-hoz 9 darab **Track** példányt hoztam létre, majd 9-szer vettem fel az album **track** object property-jét a 9 dallal. A track ténylegesen Record-okhoz rendel Track-eket, és ekvivalens a has_track object property-vel. A *DarkSideOfTheMoon*-hoz ezen kívül a name és a **year** data property-ket is felvettem, ahol a year egy általános dátum, bármihez hozzárendelhető maximum egy darab év adat.

Az eredeti ontológiához képest változás, hogy felvettem egy új object property-t, amelynek neve **track_of**, és ez az inverz tulajdonsága a track-nek. Azért hoztam létre, hogy majd a következtető a dalokhoz is meg tudja majd állapítani, hogy mely albumhoz tartoznak.

A dalokhoz, vagyis Track példányokhoz csak a name és a **track number** data property-ket vettem fel. A track number a Track-hez tartozó nemnegatív integer, és azt mondja meg, hogy az albumon hányadik dalról van szó. A name alatt a dalok címét adtam meg.

Végül térjünk rá az előadókra, vagyis az együttes tagjaira. A típusuk a általában MusicArtist, de a *PinkFloyd* tagjai speciális esetek, mert mindegyikük folytat(ott) szólókarriert, ezért a MusicArtist-on belül a típusuk a **SoloMusicArtist**. Hozzájuk is megadtam a name data property értékét, vagyis a nevüket. Ezen kívül hangszer, vagyis **Instrument** példányokat is létrehoztam nekik (*Bass*, *Drums*, *Guitars*, *Synthetizator*), amelyeknek egyedül a name tulajdonságukat adtam meg (ugyanaz, mint a példány neve). A hangszereket a művészekhez az Agent **primary_instrument** object property-jén keresztül rendeltem.

Program

A fentiekben módosított Music Ontology forrásfájlja a music.ontology.search Eclipse projekt gyökerében van. A HermiT következtető jar fájlja a lib mappában található meg. A leglényegesebb fájlok a src/main könyvtárban vannak, azon belül a webapp alatt az index.jsp a Wildfly általi sikeres futtatáskor megjelenített HTML oldalt tartalmazza. A home.html pedig az az oldal, amely a kereső felületet nyújtja az alkalmazáshoz. A webapp/WEB-INF mappában lévő web.xml pedig beállítja azt, hogy a kérések URL-je az „api" szócskával kezdődjön. A src/main/java/music/ontology/search útvonal pedig a szerver alkalmazás fő fájljait tartalmazza.

A **home.html** csak egy formot tartalmaz, amelyben van egy textbox és egy Search gomb, és a form submitteléskor egy get kérést küld az api/music/search linkre, a kérésben tartalmazva a term nevű adatot a textboxból, vagyis a keresett példány nevét.

Az **IMusicService.java** fájl tartalmazza a szolgáltatás interfészét, amely definiálja, hogy a music/search linkkel lehessen elérni a search függvényét, amely a kérés term nevű paramétereként egy Stringet vár.

A **MusicService.java** valósítja meg az IMusicService interfészt, vagyis a szolgáltatást. Pontosabban, ő hajtatja végre ezt a MusicSearcher osztállyal, amelyet létrehoz, majd meghívja az ő searchTerm(term) függvényét, átadva neki a term paramétert, amelyet a kérésben megkapott. Ha a konstruktor meghívásakor Exception keletkezik, akkor ő kezeli. Fontos, hogy a searchTerm egy Stringgel tér vissza a hívójához, és ez a String fog megjelenni az oldalon a sikeres kérés után. Ha pedig kivétel keletkezett, akkor null-lal tér vissza.

A program lényeges működése a **MusicSearch** osztályban található. A **konstruktora** végzi el az OWL API használatához szükséges objektumok inicializálását. Az OWLOntology inicializálásakor a projektben található ontológia forrásból kell betölteni a módosított zenei ontológiát. Ekkor töltődik be az ontológia előtt annak összes függősége, itt keletkeznek kivételek, ha valamelyik függőséggel vagy magával az ontológiával probléma van. Ha az ontológiát sikerült betölteni, akkor kiírja ennek sikerességét a konzolra. Miután a HermiT következtető (reasoner) is létrejött, a testSatisfiable(reasoner) függvényt hívja meg a konstruktor, amely az ontológia konzisztenciáját ellenőrzi. Ha nem konzisztens, akkor a getUnsatisfiableClasses függvény dob kivételt.

A MusicSearch osztály **searchTerm(String term)** függvénye végzi az érdemi logikát a programban. Paraméterként a keresett példány nevét várja, visszatérési értéke az a String, amelyet a weboldalon meg fog jeleníteni az alkalmazás a sikeres kérés után. A String egy ontológiában létező példány kérése esetén a példány nevét és adatait tartalmazza, ha pedig egy nem létező példány neve a paraméter, akkor egy „Nincs ilyen ontológia!” üzenetet és az ontológiában létező előadók neveit tartalmazza, hogy legyen honnan kiindulni kereséskor.

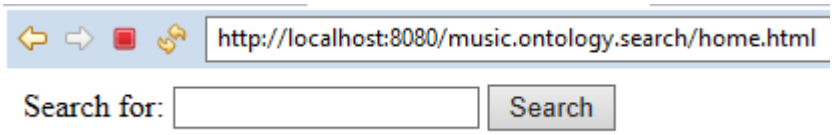
A függvény először megvizsgálja a beírt szó IRI-je (ontológia prefixszel együtt a példány neve) alapján, hogy ilyen nevű példány létezik-e a betöltött ontológiában. Ha nem, akkor a válasz szöveget a következőképpen építi fel: elején a „Nincs ilyen ontológia! Próbálkozzon a következőkkel:” mondatok lesznek, utána bejárja az ontológia összes példányát, és ha a következtető megállapítja, hogy az adott példány a MusicArtist osztály példánya, akkor annak a példánynak a nevét beveszi a szövegbe. A név a példány IRI-jének getFragment() függvényéből derül ki. Az enter karakterek is benne vannak a szövegben. Az így felépített szöveggel a függvény visszatér, ezzel véget ér a nem létező példányra vonatkozó ág.

Ha pedig létezik ilyen nevű példány, akkor lekéri az OWLDataFactory-tól az ilyen nevű példányt (individual), majd az eredményszövegbe először ennek a neve kerül. Ezután a program bejárja az ontológia összes object property-jét, közben a reasoner kikövetkezteti, hogy a kérdéses példánynak van-e ilyen object property-je, és ha igen, akkor összeszedi az ahhoz kapcsolódó példányokat egy NodeSet-ben. Ezután ezen a NodeSet-en jár végig, és fűzi hozzá az eredménysztringhez az object property és a NodeSet-ben lévő aktuális példány nevét is, entereket is beletéve. Ha bejárta az object property-ket, akkor a data property-k bejárása következik. Minden egyes data property-re megállapítja a következtető, hogy van-e ilyen tulajdonsága az adott egyednek, ha igen, akkor annak értékeit kérdezi le, vagyis

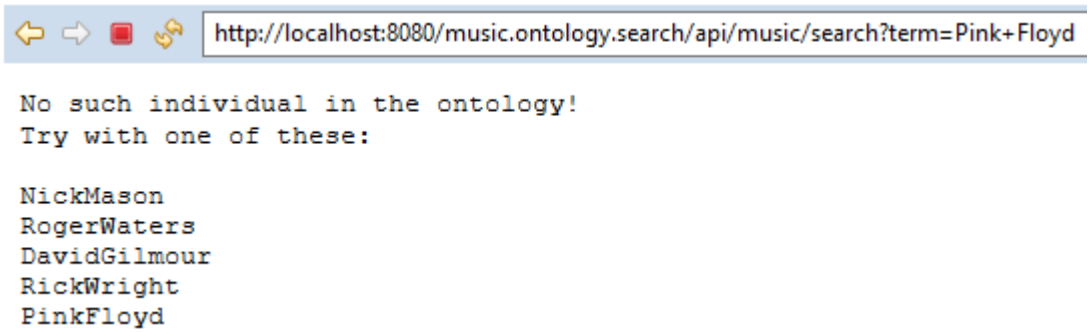
literálok halmazát. Ezután a literálokat járja végig, és írja ki őket az eredménysztringbe a data property nevével együtt. Az OWLLiteral.getLiteral() segítségével nyerhetjük ki a literálok értékeit. Ezután a függvény visszatér a kialakított eredménnyel.

Tesztelés

Eclipse Java EE-ben a projekten a Run As/Run On Server opciót kell választani az indításhoz. Ha jól sikerült a deploy, akkor megjelenik előttünk a „Hello world!” felirat. Ezután a megnyíló link végére a home.html-t kell írni, és ez a felület jelenik meg:



Ha beírjuk pl., hogy „Pink Floyd”, akkor a következő képernyő jelenik meg, és egyébként is minden olyan szóra, amely nem egy példány neve az ontológiában:



Ha egybeírjuk, akkor megjelennek róla az object property-jeinek és a data property-jeinek az értékei is, olyanok is, amelyeket még csak be sem írtunk, mert a következtető azt kikövetkeztette. A keresőbe az object property-kben megjelölt egyedek is beírhatók.

```
PinkFloyd

member: RickWright
member: RogerWaters
member: NickMason
member: DavidGilmour
performer: RickWright
performer: RogerWaters
performer: DavidGilmour
singer: RickWright
singer: RogerWaters
singer: DavidGilmour
agent: RickWright
agent: RogerWaters
agent: DavidGilmour
genre: ProgressiveRock
published: DarkSideOfTheMoon

activity_end: 2015
activity_start: 1965
```

```
MoneySong

track_of: DarkSideOfTheMoon

track_number: 5

RogerWaters

isAgentIn: PinkFloyd
performed: PinkFloyd
primary_instrument: Guitars
primary_instrument: Bass
member_of: PinkFloyd

NickMason

primary_instrument: Drums
member_of: PinkFloyd
```