PROGETTO CONFIGURATORE

MODELLAZIONE E GESTIONE DELLA CONOSCENZA

docente: **Michele Loreti** e **Lorenzo Rossi**

Università degli studi di Camerino

Giorgio Della Roscia

Informatica per la Comunicazione Digitale

Scuola di Scienze e Tecnologie Informatiche

Anno Accademico 2023/2024

Camerino, MC

Indice

Immagini

Tabelle

Codici

Ambienti

Sezione 1

ONTOLOGIA

In questa prima sezione andrò a presentare la mia proposta di configuratore. Ho deciso di spaziare dalle varie alternative proposte – rispettivamente nel campo delle automobili, degli arredi e dei computer – per concentrarmi su un'idea venutami durante una serata in compagnia di amici.

Prima però di annunciare il tema, vorrei esaminare quelli rimossi per svariate motivazioni.

tabella 1.1: possibili ontologie scartate

CONFIGURATORE	MOTIVO DELLA RINUNCIA
Shoes	scarsa conoscenza del settore
Library	distanza dal concetto di configuratore
PC	opzione proposta dal docente, volevo differenziare
Keyboard	alternativa valida, ma assenza di decisioni esclusive

Vediamo ora su quale dominio ho effettivamente impostato lo sviluppo: si tratta di un **configuratore di kebab**. Ammetto di aver optato per tale insieme in modo da affidarmi, almeno in parte, alla struttura di `pizza.owl`[1]. Ciò mi ha permesso di iniziare con maggiori certezze mantenendo però una garantita variazione di tema.

Come intuibile, la scelta resta prettamente personalizzata e sono andato a sviluppare l'ontologia da zero. Così facendo sono stato in grado di comprendere in profondità ogni dettaglio implementato durante la realizzazione, mantenendo quindi il controllo sul progetto.

Lo strumento adoperato per la creazione dell'ontologia è quello presentato a lezione: Protégé[2].

Nota bene 1 (abbreviazione prefissi)**.** Negli estratti di codice ho sostituito i prefissi degli URI coi rispettivi namespace per garantire maggiore leggibilità.

`https://www.unicam.it/giorgiodellaroscia/KebabConfigurator#` ↓ `kbb:`

`https://dbpedia.org/ontology/` ↓ `dbo:`

`http://www.w3.org/2001/XMLSchema#` ↓ `xsd:`

⋮

[1] Hor16.
[2] Bio16.

1.1 Entities

La struttura dell'ontologia si poggia su un tre macroconcetti: cibo, persona e restrizioni alimentari. In questo modo si vengono a creare delle buone interconnessioni fra le classi realizzate. L'intento è quello di permettere all'utente di configurare il panino secondo le proprie esigenze.

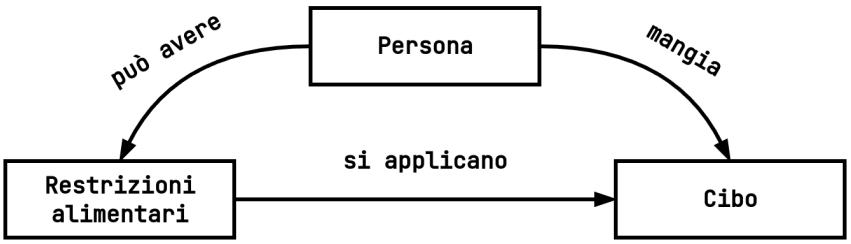

figura 1.1: bozza della struttura delle connessioni dell'ontologia

L'approccio alla realizzazione è stato di tipo top-down poiché mi sono appoggiato ai tre elementi evidenziati per iniziare lo sviluppo.

Dichiaro da subito che ho deciso di rispettare la notazione standard di Protégé che prevede *PascalCase* per le entità e i loro individui, e *camelCase* per le proprietà. Anche se sarebbe stato sicuramente più tematico adoperare il *kebab-case*.

Mostro ora l'albero delle classi che sono andato a realizzare.

```
owl:Thing
├── FoodRelated
│   ├── FoodCategory
│   │   ├── Animal
│   │   ├── dbo:Animal
│   │   │   ├── Meat
│   │   │   │   ├── dbo:Bird
│   │   │   │   └── dbo:Mammal
│   │   │   └── MeatDerivate
│   │   └── Vegetal
│   │       └── Cereal
│   ├── FoodCertification
│   └── dbo:Food
│       └── Kebab
│           ├── Dressing
│           ├── Bread
│           └── Filling
│               ├── Doner
│               └── Falafel
├── Person
└── Restriction
    ├── Diet
    │   ├── OmnivorousDiet
    │   ├── VeganDiet
    │   └── VegetarianDiet
    ├── dbo:Disease
    │   └── Intolerance
    └── Religion
        ├── Buddhism
        ├── Islam
        └── Judaism
```

1.1.1 Visualizzazione albero

Nel software utilizzato è possibile installare diversi plugin appositi per la schematizzazione grafica dell'ontologia realizzata. I tre principali sono OWLViz[3], OntoGraf[4] e VOWL[5].

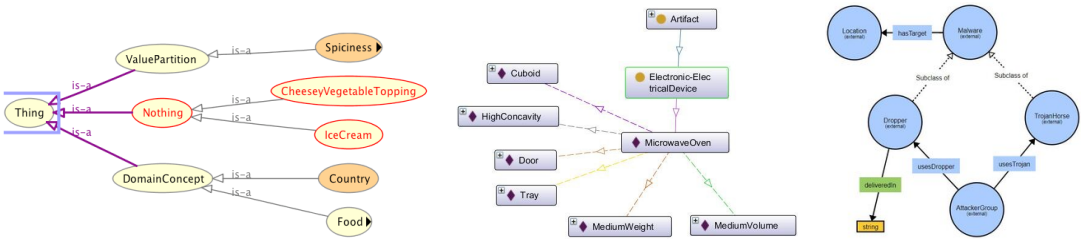

figura 1.2: OWLViz, OntoGraf e VOWL

Gli ultimi due coinvolgono le proprietà fra entità e gli individui. Per questa motivazione ho preferito il primo come strumento capace di far percepire la struttura del progetto senza appesantirla.

figura 1.3: visualizzazione dell'ontologia tramite OWLViz

Occorre specificare che ho aggiunto esclusivamente gli elementi che ritenevo utili al dominio. Ecco spiegato il motivo per cui sono presenti poche religioni, che sono poi le uniche che impongono dei limiti sul cibo; non tutte le diete, ho per esempio tralasciato quelle pescetariana e crudista; ed ancora soltanto alcuni tipi di carne animale, proiettati a fungere da riempimento del panino. Ciò vale allo stesso modo per le intolleranze, le certificazioni alimentari, le salse, e così via.

[3] Hor.
[4] Fal.
[5] BLN.

1.1.2 Riutilizzo ontologie esterne

La qualità primaria delle ontologie sta nel potersi appoggiare a vocabolari già presenti in rete in modo da poterle connettere e rendere fruibili in quanto parte di un sistema. Esistono numerose collezioni di vocabolari: due fra tante sono LOV[6] e LOD[7], che ne raccolgono molteplici.

Parlando ora del mio ambito applicativo, ho commesso l'errore di effettuare una scelta troppo restringente. Ciò mi ha portato, nella fase di importazione di ontologie esterne, a scontrarmi con insiemi troppo vasti rispetto alle mie effettive esigenze.

Ho quindi deciso di ridimensionare gli assiomi importati in delle copie locali. Mi rendo conto che non è l'ideale, ma essendomi trovato in tale situazione a lavoro inoltrato mi risultava impossibile variare la scelta del tema.

Com'è possibile notare dagli alberi gerarchici delle entità, ho importato alcune ontologie: `dbo:Animal`[8], `dbo:Disease`[9] e `dbo:Food`[10]. Tutte e tre provengono da DBPedia[11], per selezionarle sono infatti andato ad esaminare la mappatura[12] presente nel sito.

> **Nota bene 2** (legame fra Wikipedia e DBPEDIA)**.** Le entità di DBPedia sono estrapolate direttamente da Wikipedia. Esiste infatti una corrispondenza uno a uno fra i due siti:
>
> `https://dbpedia.org/page/` = `https://en.wikipedia.org/wiki/`

Attenzione però che in questo caso si parla di *page* e non *ontology*! Il termine "pagina" fa riferimento all'entità e non all'ontologia per intero.

Analizzando nello specifico `dbo:Animal`, ho pensato che una possibile imprecisione organizzativa delle entità stesse nel non aver disposto `Person` come sua sottoclasse. Ho dunque valutato la possibilità di effettuare lo spostamento, ma ciò risultava stravagante.

```
<owl:Class rdf:about="kbb:Meat">
    <rdfs:subClassOf rdf:resource="dbo:Animal"/>
</owl:Class>
<owl:Class rdf:about="kbb:Person">
    <rdfs:subClassOf rdf:resource="kbb:Meat"/>
</owl:Class>
```

codice 1.1: `Person` subClassOf `Meat`

1.1.3 Vincoli di proprietà

Alcune delle entità realizzate sono vincolate.

```
<owl:Restriction>
    <owl:onProperty rdf:resource="kbb:canBeCertified"/>
    <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">0</owl:minCardinality>
    <owl:onClass rdf:resource="kbb:FoodCertification"/>
</owl:Restriction>
```

codice 1.2: restrizione sulla classe `Kebab`

Come specificato, un kebab può non avere certificazioni alimentari. Al contempo, grazie all'OWA[1], non è sbagliato immaginare che possa essere approvato congiuntamente secondo le normative `Halal` e `Kosher`.

> **Definizione 1** (*Open-World Assumption*)**.** L'assenza di informazione non ne implica la falsità.

[6] Gro24.
[7] Dat07.
[8] Vira.
[9] Vird.
[10] Vire.
[11] Virb.
[12] Virc.

1.2 Proprietà

L'ontologia, salvata in formato RDF (*Resource Description Framework*), è essenzialmente un insieme di triple composte da soggetto, predicato e oggetto.

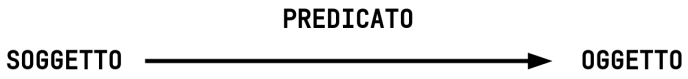

figura 1.4: tripla soggetto-predicato-oggetto

Appreso ciò, andiamo ora a focalizzarci sul secondo elemento: le funzioni.

1.2.1 Object properties

I predicati che connettono i componenti dell'ontologia sono unidirezionali, perciò il grafo creato è orientato. Ovviamente è possibile impostare la simmetria in una proprietà.

```
:Class1 rdf:type owl:ObjectProperty ,
                 owl:SymmetricProperty ;
        rdfs:domain :Class1 ;
        rdfs:range :Class2 .
```

codice 1.3: proprietà simmetrica

Le object properties che ho deciso di inserire sono:

```
owl:topObjectProperty
├── allowsToEat
├── believes
├── belongsTo
├── canBeCertified
├── contains
├── doNotAllowToEat
├── doNotDigest
├── eats
├── follows
├── hasDisease
├── hasIngredient
├── hasIntolerance
├── hasKebab
├── isMoreRestrictiveThan
└── onlyAllows
```

Di quest'elenco, solamente alcune hanno delle relazioni:

- `isMoreRestrictiveThan` è `owl:IrreflexiveProperty` dato che coinvolge due componenti necessariamente differenti;
- `believes`, `belongsTo`, `follows` e `onlyAllows` sono `owl:FunctionalProperty` per la mutua esclusività dei possibili range.

1.2.2 Data properties

Analizziamo ora quelle proprietà che associano ad un componente dell'ontologia un literal.

```
owl:topDataProperty
├── kcal
├── name
├── price
└── surname
```

Come si può immaginare `name` e `surname` sono associate a `Person`, mentre `kcal` e `price` a `Kebab`.

Ho deciso infatti di fornire delle informazioni riguardo ad ogni panino configurabile. I dati citati sono rispettivamente di tipo intero e double.

1.3 Individuals

Passiamo ora agli individui, ossia le istanze delle classi. In questo caso ho applicato dei compromessi in quanto molto spesso mi sarebbe tornato utile collegare tramite una object property degli individui a delle classi. Ne sono un esempio le tre diete che avevo inizialmente inserito come individui per poi essere promosse ad entità. Così come le religioni.

Inoltre, per quanto riguarda la celiachia, rappresentata come `CeliacDisease`, ho inteso un'estensione dell'intolleranza al glutine trattandosi di una vera e propria malattia. Mi sono concesso la libertà di ignorare ciò per semplificarne la comprensione, ma in effetti sarebbe stato più corretto definirla come `GlutenIntolerance`.

Sono poi andato ad implementare tanti individui per la classe `Kebab` quante le possibili combinazioni di scelte effettuabili dall'utente. Qui avvengono delle illogicità.

> **Esempio 1** (individui uguali)**.** Le possibilità alimentari dall'istanza `LactoseVeganHinduism` non differiscono da quelle di `Vegan`. L'intolleranza al lattosio è ininfluente in quanto la dieta non permette di mangiare latticini, e la religione che vieta la carne di vacca ha valenza nulla rispetto alla restrizione vegana sugli animali in generale.

Come questa relazione ne esistono molte altre.

tabella 1.2: kebab e valori associati

LABEL	PRICE €	KCAL
CeliacOmnivorous	4.2	1500
CeliacOmnivorousHinduism	4.2	1300
CeliacOmnivorousIslam	4.2	1500
CeliacOmnivorousJudaism	4.2	1500
CeliacVegan	3.7	500
CeliacVeganHinduism	3.7	500
CeliacVeganIslam	3.7	500
CeliacVeganJudaism	3.7	500
CeliacVegetarian	3.9	1000
CeliacVegetarianHinduism	3.9	1000
CeliacVegetarianIslam	3.9	1000
CeliacVegetarianJudaism	3.9	1000
LactoseCeliacOmnivorous	4.1	1400
LactoseCeliacOmnivorousHinduism	4.1	1200
LactoseCeliacOmnivorousIslam	4.1	1400
LactoseCeliacOmnivorousJudaism	4.1	1400
LactoseCeliacVegan	3.7	500
LactoseCeliacVeganHinduism	3.7	500
LactoseCeliacVeganIslam	3.7	500
LactoseCeliacVeganJudaism	3.7	500
LactoseCeliacVegetarian	3.8	900
LactoseCeliacVegetarianHinduism	3.8	900
LactoseCeliacVegetarianIslam	3.8	900
LactoseCeliacVegetarianJudaism	3.8	900

LABEL	PRICE €	KCAL
LactoseOmnivorous	3.9	1400
LactoseOmnivorousHinduism	3.9	1200
LactoseOmnivorousIslam	3.9	1400
LactoseOmnivorousJudaism	3.9	1400
LactoseVegan	3.5	500
LactoseVeganHinduism	3.5	500
LactoseVeganIslam	3.5	500
LactoseVeganJudaism	3.5	500
LactoseVegetarian	3.6	900
LactoseVegetarianHinduism	3.6	900
LactoseVegetarianIslam	3.6	900
LactoseVegetarianJudaism	3.6	900
Omnivorous	4	1500
OmnivorousHinduism	4	1300
OmnivorousIslam	4	1500
OmnivorousJudaism	4	1500
Vegan	3.5	500
VeganHinduism	3.5	500
VeganIslam	3.5	500
VeganJudaism	3.5	500
Vegetarian	3.7	1000
VegetarianHinduism	3.7	1000
VegetarianIslam	3.7	1000
VegetarianJudaism	3.7	1000

La variazione di prezzo e importo calorico non è casuale! Si parte infatti da una base di €4 per l'onnivoro, €3.7 per il vegetariano e €3.5 per il vegano. Le ulteriori differenze si applicano con un alleggerimento di -€0.1 per la rimozione della salsa bianca, contenente yogurt quindi lattosio, e +€0.2 per il pane senza glutine. Le due direttive esposte possono coesistere, andando così a modificare il prezzo di +€0.1. Le restrizioni religiose non variano il costo del panino.

Per quanto riguarda invece le kilocalorie si ha una base di 1500kcal per l'onnivoro, 1000kcal per il vegetariano e 500kcal per il vegano. A queste ne vanno poi rimosse 100 se intolleranti al lattosio, poiché il panino perderà la salsa bianca, e 200 se induisti, che comporterà un riempimento di carne composto solamente da pollo e agnello.

Esempio 2 (direttive non rispettate)**.** Il costo e l'importo calorico di `Vegan` è il medesimo di `LactoseVeganHinduism`. Ciò, in continuità con l'esempio[1], è dato dal fatto di non aver tenuto conto degli accorgimenti appena esposti nel caso in cui gli individui siano identici.

1.3.1 Struttura logica in vista dell'applicazione

La trovata è stata quella di allestire già internamente all'ontologia degli individui specifici per ogni possibile combinazione di scelte effettuate in fase di configurazione. Così facendo avrei potuto poi attingere ad un elemento già pronto e predisposto con delle apposite interrogazioni presenti nell'applicativo Java.

Tale scelta di basare l'intero progetto sulla creazione di tanti individui di panini quante le possibili combinazioni mi è costata la rimozione di una sottoclasse di `Restriction`: `Preference`.

Essa andava ad adempiere eventuali preferenze legate al gusto della persona, ciò veniva garantito da due suoi individui: `Garlicky` e `Spicy`. Essi si legavano rispettivamente alle salse `WhiteSauce` e `SpicySauce`. In tal modo se l'utente avesse inserito una delle due preferenze, o entrambe, le relative salse sarebbero state rimosse dalla composizione del kebab.

Ma qual è quindi il motivo della rimozione? La questione è strettamente pratica in quanto se l'avessi mantenuta avrei moltiplicato il numero di individui da realizzare per quattro. Infatti le possibili combinazioni si aggregano tramite moltiplicazione.

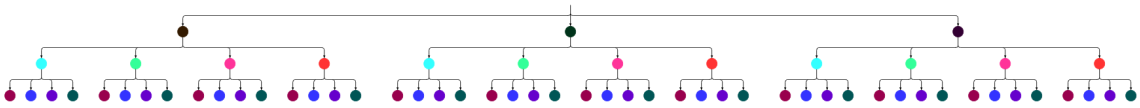

figura 1.5: diramazioni delle possibili scelte del configuratore

Le tre diete hanno selezione unica obbligata, le due intolleranze possono anche coesistere o non essere presenti, delle tre religioni c'è da considerare l'eventualità della persona atea.

Si intuisce quindi che se le due preferenze rimosse fossero state lasciate si avrebbe ottenuto un numero di individui ingestibile:

$$3_{\texttt{Diet}} \cdot 4_{\texttt{Intolerance}} \cdot 4_{\texttt{Religion}} = 48 \text{ combinazioni}$$

$$3_{\texttt{Diet}} \cdot 4_{\texttt{Intolerance}} \cdot 4_{\texttt{Religion}} \cdot 4_{\texttt{Preference}} = 192 \text{ combinazioni}$$

Quadruplicato appunto, poiché oltre ai due individui riportati c'è anche da considerare le casistiche in cui il cliente non abbia preferenze o le selezioni entrambe.

Mi rendo conto che probabilmente non si tratta della corretta struttura logica su cui basarsi per realizzare un progetto simile, al contempo sono purtroppo consapevole di non aver ideato in tempo alternative altrettanto valide per la mia casistica.

Controllando gli ultimi particolari mi sono accorto di come giustamente avvenisse un conflitto nella richiesta di panini senza intolleranze né religione. Ciò era dovuto all'aver assegnato la stessa etichetta alle diete ed ai panini `Omnivorous`, `Vegetarian` e `Vegan`. Ho quindi rimediato aggiungendo il suffisso `Diet` di seguito alle diete.

1.4 Sviluppi futuri

Le possibili estensioni riguardano l'approfondire un altro campo interessante ed in linea con l'argomento. Sto parlando dell'importazione dell'ontologia realizzata in Neo4j[13] per poterne visualizzare un grafo relativo ed accedere ai campi tramite query SPARQL.

[13] Cyp10.

Sezione 2

OOP

Eccoci arrivati alla parte di codice. L'obiettivo è quello di sfruttare l'ontologia in modo tale da ottenere dati utili all'applicazione.

figura 2.1: comunicazione applicazione e ontologia

La prima classe, avviata dal pacchetto `it.unicam.cs.mgc.kebabConfigurator`, è `App.java`.

```java
import it.unicam.cs.mgc.kebabConfigurator.viewer.ConsistencyChecker;
import it.unicam.cs.mgc.kebabConfigurator.viewer.chooser.ChoicesLauncher;
import it.unicam.cs.mgc.kebabConfigurator.viewer.resulter.ResultKebab;

public class App {
    public static void main(String[] args) {
        if (new ConsistencyChecker().getConsistency())
            new ResultKebab().kebabComposing(new ChoicesLauncher().kebabCodeGeneration());
    }
}
```

codice 2.1: classe `App.java` da cui viene lanciato il programma

Vediamo come vengono creati tre oggetti di tre classi differenti. Dai nomi si intuisce di già come prima si richiede la verifica della consistenza dell'ontologia per poi, in caso sia valida, avviare il processo di configurazione del panino.

2.1 Responsabilità

Il progetto richiede la configurazione di un prodotto, nel mio caso un kebab. Prima di iniziare a costruire l'applicazione ho individuato le necessità da adempiere, esse sono:

1. costruire il modello dell'ontologia;
2. verificare la consistenza dell'ontologia;
3. permettere la configurazione del proprio kebab;
4. effettuare delle interrogazioni SPARQL sull'ontologia;
5. convertire i dati ottenuti dalle query in formato accessibile;
6. stampare il kebab risultante accompagnato da informazioni utili.

Per andarle ad implentare ho suddiviso il progetto in sotto-pacchetti, ognuno avente un incarico.

```
it.unicam.cs.mgc.kebabConfigurator
├── model
│   ├── builder
│   ├── controller
│   ├── parser
│   ├── sparql
│   └── util
└── viewer
    ├── chooser
    ├── cleaner
    └── resulter
```

2.1.1 Model

Per sviluppare ciò è necessario innanzitutto agganciare l'applicazione Java all'ontologia RDF. Fortunatamente esiste un framework che permette tale connessione: Apache Jena[1].

2.1.1.1 Builder

Viene costruito il modello RDF[1].

Ho implementato l'interfaccia `ModelBuilder.java` che viene implementata dal costruttore standard `DefaultModelBuilder.java`. Mi è sembrato utile predisporre inoltre un'estensione per il modello inferito: `InferredModelBuilder.java`. Questo eredita le funzionalità della classe base.

2.1.1.2 Controller

Si assicura, tramite l'unica classe `Controller.java`, che l'ontologia sia consistente[2].

Per svolgere tale compito è necessario uno strumento apposito, un reasoner: come per Protégé ho selezionato Pellet[2].

2.1.1.3 Sparql

Sono andato a realizzare l'interfaccia `DataQuery.java`, implementata poi da `QueryContainer.java` per ottenere la query con tanto dei prefissi necessari per il corretto funzionamento.

Anche l'interfaccia `Executor.java`, ripresa da `QueryExecutor.java` per eseguire le query[4]. Quest'ultima classe contiene anche le interrogazioni usate per ottenere informazioni sull'ontologia.

```
1 SELECT ?label
2 WHERE {
3     ?kebab rdf:type kbb:Kebab .
4     BIND(?kebab AS ?label).
5 }
```

codice 2.2: query SPARQL per ottenere la lista di tutti i kebab

Questa prima richiesta, seppur non sfruttata in definitiva nell'accesso all'ontologia, mi è stata fondamentale per estrarre la lista di tutti gli individui dell'entità `Kebab`.

```
1 SELECT ?label ?value
2 WHERE {
3     ?kebab rdf:type kbb:Kebab ;
4            rdfs:label \"%s\"@en .
5     BIND(?kebab AS ?label) .
6     BIND(?kebab AS ?value) .
7 }
```

codice 2.3: query SPARQL per ottenere la label del kebab

[1] Fou11.
[2] Gal16.

Si inizia poi ad estrarre le informazioni utili all'utente in fase finale. Per iniziare prelevo l'etichetta del panino, che funge da nome vero e proprio, e riprende le caratteristiche dalle opzioni selezionate durante la configurazione.

```
SELECT ?label ?value
WHERE {
    ?kebab rdf:type kbb:Kebab ;
           rdfs:label \"%s\"@en ;
           kbb:price ?price .
    BIND(?kebab AS ?label) .
    BIND(?price AS ?value) .
}
```

codice 2.4: query SPARQL per ottenere il costo del kebab

Avviene lo stesso, basandosi sempre sulla stessa etichetta per quanto riguarda il costo associato al panino ed anche l'importo calorico.

```
SELECT ?label ?value
WHERE {
    ?kebab rdf:type kbb:Kebab ;
           rdfs:label \"%s\"@en ;
           kbb:kcal ?kcal .
    BIND(?kebab AS ?label) .
    BIND(?kcal AS ?value) .
}
```

codice 2.5: query SPARQL per ottenere le kilocalorie del kebab

Le quattro interrogazioni appena esposte sono inefficienti se private dei prefissi necessari a riconoscere gli URI per intero.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX kbb: <https://www.unicam.it/giorgiodellaroscia/KebabConfigurator#>
PREFIX dbo: <http://dbpedia.org/ontology/>
```

codice 2.6: prefissi per le query da eseguire nell'ontologia

Si ha infine la classe `QueryCaller.java` che richiama il lancio delle query già predisposte nell'enumerazione `QueryContainer.java`.

2.1.1.4 Parser

Abbiamo ora il risultato prodotto dalle interrogazioni sul file RDF. Risulta però necessario convertirlo in un formato comprensibile[5].

Ho disposto due interfacce: `DataParser.java` e `ParsedData.java`. Esse vengono implementate rispettivamente da `JSONParser.java` e `JSONData.java`. Proprio quest'ultima consente di ottenere i dati come una raccolta di stringhe.

```
@Override
public Collection<String> getAllValues() {
    return this.data.values();
}
```

codice 2.7: metodo `getAllValues()` della classe `JSONData.java`

2.1.1.5 Util

Pacchetto standard contenente codice comodo. `URIs.java` è un'enumerazione degli URI presenti.

2.1.2 Viewer

Vengono gestite tutte le interazioni di input/output da terminale fra l'utente e l'applicazione.

La classe `ConsistencyChecker.java` stampa la consistenza, o l'incosistenza, dell'ontologia.

2.1.2.1 Chooser

Ci si imbatte innanzitutto nella classe `ChoicesLauncher.java` che richiama la fase di selezione per andare a costruire una tripla di interi a cui associare un kebab.

Per la configurazione del panino[3] si ha la classe `RestrictionChooser.java` che implementa `Chooser.java` e fornisce i metodi per stampare le alternative predisposte.

L'unica cosa da tenere a mente è la possibile opzionalità delle restrizioni evidenziate.

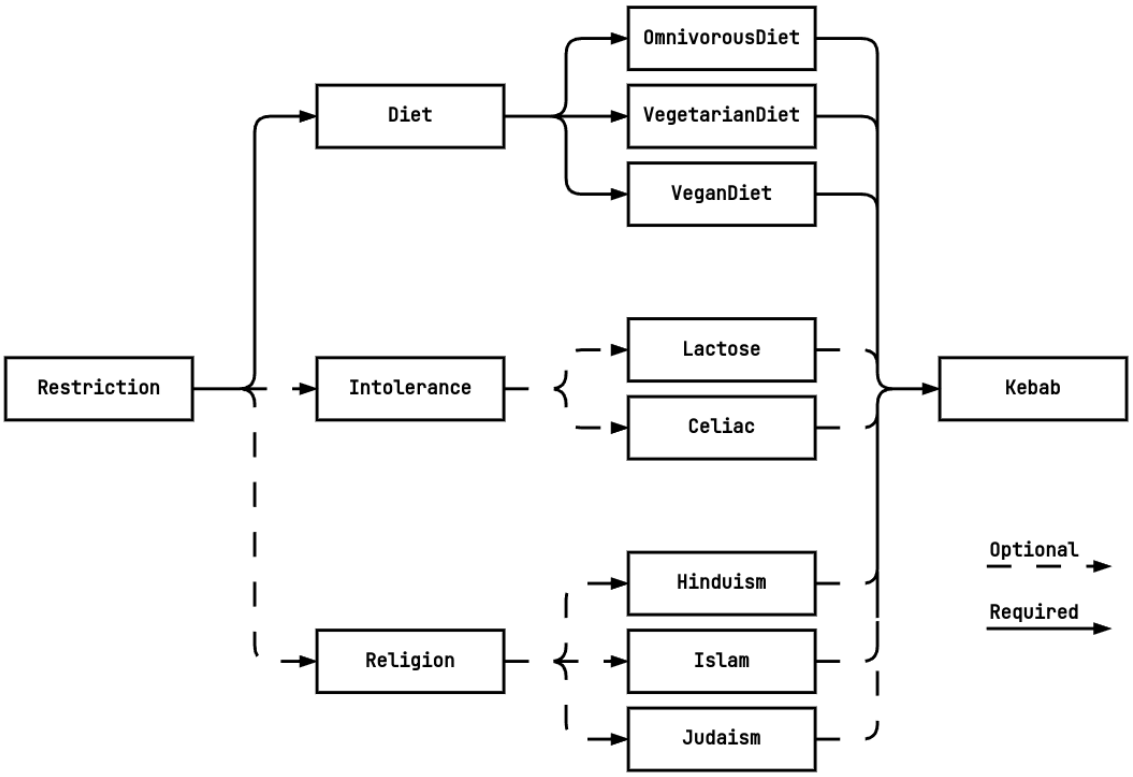

figura 2.2: opzionalità di alcune restrizioni alimentari

2.1.2.2 Cleaner

Semplice modulo per gestire la ripulitura del pannello. L'interfaccia `Cleaner.java` è ripresa in `CleanTerminal.java`

2.1.2.3 Resulter

In questa sezione vengono richiamate tutte le informazioni necessarie per poi andare a mostrare il risultato all'utente.

La classe `ResultKebabConstructor.java` associa alla tripla di interi un'etichetta, che è poi quella ripresa nelle query per ricavare i valori delle due data property `kbb:price` e `kbb:kcal`.

Infine, viene stampato un breve timer d'attesa con `WaitTimer.java` a cui segue il risultato[6] da `ResultKebab.java`.

2.2 Test

Oltre al ramo del progetto Java appena terminato di visionare, sono andato a realizzare delle classi di testing[3] utili a verificare la corretta esecuzione di alcuni moduli cardine dell'applicativo.

Ho ritenuto principalmente utile garantire la ricezione di dati dall'esecuzione delle query. Per questo sia `ControllerTest.java` che `QueryCallerTest.java` sono indirizzate a tale compito.

Per concludere, `ResultKebabTest.java` mi ha aiutato ad accertarmi che le informazioni ottenute fossero quelle che mi aspettavo. Ho infatti inserito manualmente tre dati, relativi alla label e ai due valori di costo e kilocalorie, per poi confrontarli con quelli estrapolati dall'ontologia.

In particolare, mi è stato utile controllare anche la presenza dell'etichetta nel file RDF come nome associato all'individuo ricercato.

```
1 @Test
2 void testGetKebabLabelResult() {
3     assertEquals(kebabLabel, resultKebab.getKebabLabelResult(kebabLabel));
4 }
```

codice 2.8: test etichetta kebab

2.3 Sviluppi futuri

Mi suona scontato dire che il più diretto miglioramento tecnico riguarda il rimpiazzamento della scarna visualizzazione da CLI (Command Line Interface) ad una da GUI (*Graphical User Interface*) basata chiaramente su JavaFX[4].

[3] Tea21.
[4] Ora08.

Bibliografia

[Bio16] Stanford Center for Biomedical Informatics Research. Protégé. 2016. URL: https://protege.stanford.edu.

[BLN] David Bold, Steffen Lohmann, and Stefan Negru. VOWL. URL: https://protegewiki.stanford.edu/wiki/VOWL.

[Cyp10] Cypher. Neo4j. 2010. URL: https://neo4j.com/.

[Dat07] Insight Centre for Data Analytics. The Linked Open Data Cloud Diagram. 2007. URL: https://lod-cloud.net.

[Fal] Sean Falconer. OntoGraf. URL: https://protegewiki.stanford.edu/wiki/OntoGraf.

[Fou11] The Apache Software Foundation. Apache Jena. 2011. URL: https://jena.apache.org.

[Gal16] Galigator. Openllet. 2016. URL: https://github.com/Galigator/openllet.

[Gro24] Ontology Engineering Group. Linked Open Vocabularies. 2024. URL: https://lov.okfn.org/dataset/lov.

[HKR10] Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. Foundations of Semantic Web Technologies. 1st ed. Taylor & Francis group, 2010. ISBN: 9780429143472.

[Hor] Matthew Horridge. OWLViz. URL: https://protegewiki.stanford.edu/wiki/OWLViz.

[Hor16] Matthew Horridge. Pizza Ontology. 2016. URL: https://github.com/owlcs/pizza-ontology/blob/master/pizza.owl.

[Ora08] Oracle. JavaFX. 2008. URL: https://openjfx.io.

[Tea21] The JUnit Team. JUnit 4. 2021. URL: https://junit.org/junit4.

[Vira] OpenLink Software Virtuoso. Animal ontology. URL: https://dbpedia.org/ontology/Animal.

[Virb] OpenLink Software Virtuoso. DBPedia. URL: https://dbpedia.org.

[Virc] OpenLink Software Virtuoso. DBPedia mappings. URL: https://mappings.dbpedia.org/server/ontology/classes.

[Vird] OpenLink Software Virtuoso. Disease ontology. URL: https://dbpedia.org/ontology/Disease.

[Vire] OpenLink Software Virtuoso. Food ontology. URL: https://dbpedia.org/ontology/Food.