Università degli Studi di Camerino

Informatica per la comunicazione digitale

ST1414 Modellazione e Gestione della Conoscenza

Tartufoli Alessio, matricola 119054

Configuratore di Scooter

Progettazione di un’ontologia e implementazione Software

Descrizione dell’ambito applicativo e del dominio

L'applicazione è un configuratore interattivo per scooter, progettato per consentire agli utenti di personalizzare e configurare un modello di scooter in base alle proprie preferenze e necessità. L'obiettivo è fornire un'interfaccia che guidi l'utente nella scelta di componenti e accessori, garantendo che le opzioni siano coerenti e compatibili tra loro.

Si tratta di un’applicazione Jena basata su JavaFX, che utilizza il dataset di un’ontologia RDF per rappresentare la conoscenza sui componenti e sulle loro relazioni. Tale approccio garantisce la capacità di gestire configurazioni complesse, assicurando che le scelte effettuate dall’utente rimangano sempre valide e nel rispetto dei vincoli predefiniti.

L'idea alla base del configuratore è di permettere agli utenti, sia essi consumatori finali, venditori, consulenti o produttori di assemblare un dispositivo che rispecchi precisamente le loro esigenze. Gli utenti possono selezionare e combinare vari componenti, visualizzando in tempo reale le modifiche e ottimizzando così le performance del veicolo finale in base alle specifiche desiderate.

Concetti e proprietà modellate nell’ontologia

Tale ontologia si concentra strettamente sul dominio degli scooter da 50 a 125 cc (ed equivalenti elettrici), rimanendo però aperta ad estensioni.

Ogni componente è descritto in termini di proprietà, relazioni e funzionalità, permettendo così di rappresentare in modo accurato le interazioni tra le diverse parti del veicolo. Inoltre, l'ontologia è progettata per essere scalabile, consentendo di includere future evoluzioni tecnologiche (ad esempio innovazioni nei sistemi di batterie, tecnologie per la guida autonoma, ecc.).

Nell’ontologia i concetti principali sono i seguenti:

Scooter: Rappresenta l'entità principale, il prodotto finale che sarà configurato dall'utente.

Motore: rappresenta la classe del componente preposto a mettere in moto il veicolo e ad avviarne il movimento. Esso può essere a combustione interna o elettrico.

Sistema d’alimentazione: rappresenta la classe che indica l’insieme di componenti e meccanismi che forniscono l'energia necessaria al motore per funzionare. I motori a combustione interna hanno bisogno dell’energia derivata dalla combustione di un carburante, mentre i motori elettrici hanno bisogno dell’energia elettrica ottenuta tramite una batteria.

Accessori: tale classe include l’aggiunta di componenti opzionali che possono essere integrati nella configurazione dello scooter per aumentarne le funzionalità o i gadget, in base ai gusti e/o alle necessità degli utenti finali. Possibili opzioni sono il bauletto posteriore e il paravento, ma tale lista di accessori può essere ridefinita o espansa in futuro.

Freni: tale classe rappresenta i componenti utilizzati per garantire la sicurezza del veicolo e arrestarlo in caso di necessità.

Luci: tale classe indica i componenti essenziali per garantire la visibilità e la sicurezza durante la guida, soprattutto di notte o in condizioni di scarsa visibilità.

Ruota: definisce i componenti cruciali per stabilità e manovrabilità del veicolo, costituendo il punto di contatto diretto con la strada.

Targa: rappresenta la classe che indica la placca identificativa obbligatoria, generalmente fissata sul retro del veicolo, la quale riporta un codice alfanumerico unico allo scopo di identificare il veicolo.

Sellino: indica la classe che rappresenta la parte dello scooter su cui il conducente si siede per guidare il veicolo.

Colore: corrisponde alla colorazione della scocca del veicolo.

Guidatore: tale classe rappresenta la persona responsabile della guida dello scooter da configurare e di cui se ne descrivono i dati più importanti in relazione alla configurazione finale.

Patente: rappresenta la classe che indica la specifica patente che deve possedere il guidatore per poter guidare un certo veicolo, in base alla cilindrata del veicolo e alla potenza del motore. La patente AM serve per guidare ciclomotori per 50cc (ed equivalenti elettrici) e deve essere conseguita dai 14 anni. La patente A1 invece consente di guidare motocicli superiore ai 50 cc fino a 125 cc (ed equivalenti elettrici) e deve essere ottenuta a partire dai 16 anni.

Nell'ontologia, i concetti sono delineati tramite l'uso di data property e object property.

Le **Data Property** specificano attributi quantitativi o descrittivi dei componenti. Queste proprietà consentono di attribuire valori numerici o testi che dettagliano le caratteristiche essenziali dei componenti dello scooter. Segue l'elenco delle proprietà riportate nel file RDF:

- HasAccessoryDescription
- HasBatteryCapacity
- HasBatteryChargingTime
- HasBrakeDescription
- HasColourName
- HasEngineDisplacement
- HasFuelName
- HasHeight
- HasLenght
- HasLightDescription
- HasMaximumSpeed
- HasMaximumWeight
- HasMotorPower
- HasRange
- HasScooterPlateNumber
- HasWheelDescription
- IsMadeOf

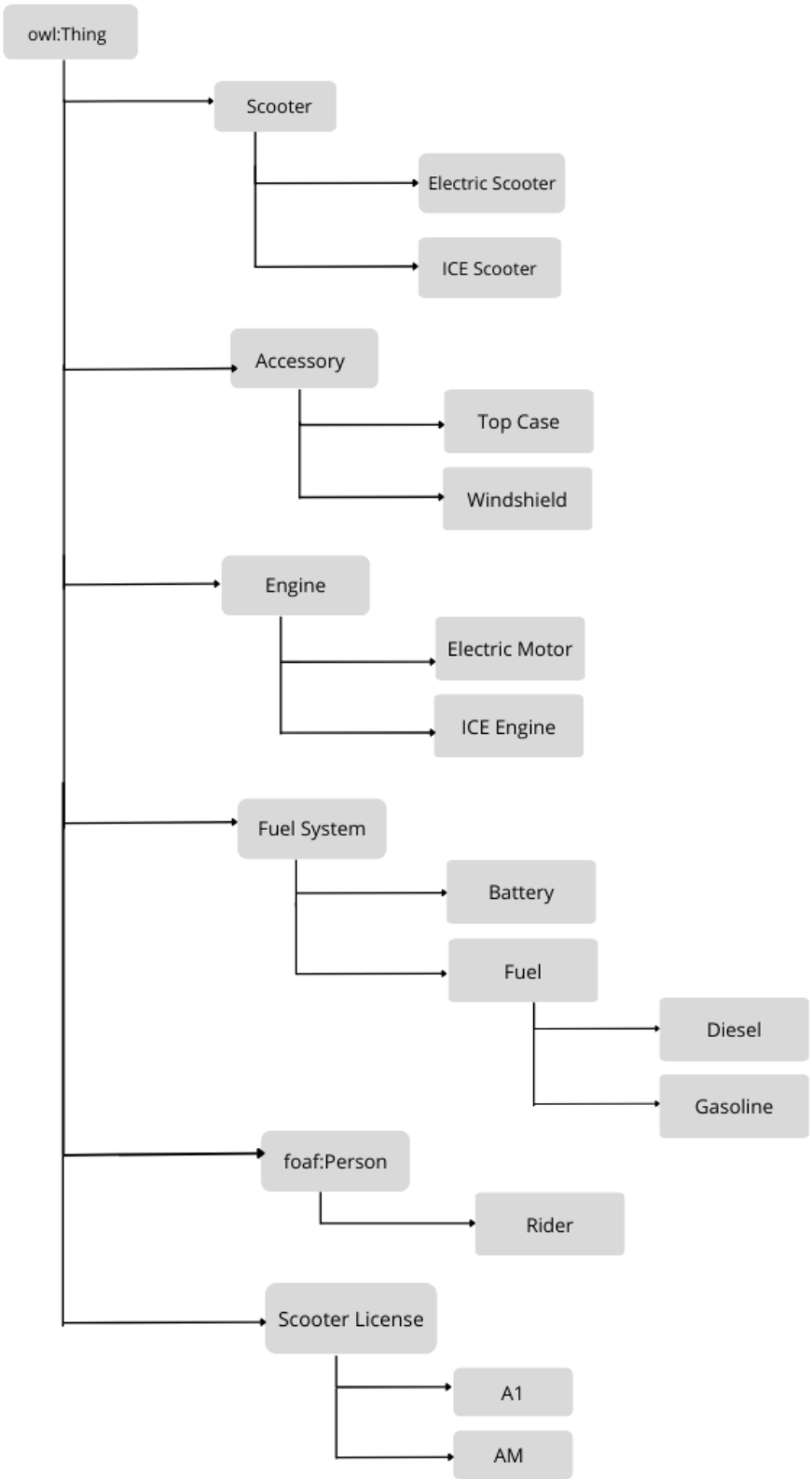
Le **Object Property** nell'ontologia collegano i vari componenti e caratteristiche dello scooter, stabilendo relazioni tra essi. Tali proprietà permettono di definire connessioni strutturali e funzionali all'interno dell'ontologia, facilitando l'organizzazione delle informazioni in modo logico e interconnesso. Segue l'elenco delle proprietà riportate nel file RDF:

- RequiresScooter
- RiddenBy
- RidesScooter

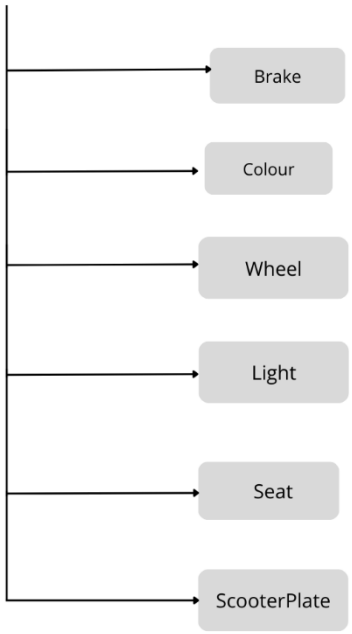
Per integrare l'ontologia creata con vocabolari già presenti online, è stato riutilizzato **FOAF:** acronimo di **Friend of a Friend**, indica una specifica semantica che fornisce un vocabolario per descrivere informazioni personali e relazioni sociali su di sé e sulle altre persone.

L'applicazione di tale vocabolario è stata utile per rappresentare e descrivere la classe Rider all'interno del file RDF.

Di seguito è riportato uno schema complessivo della gerarchia delle classi complesse presenti nell'ontologia.



ICE = Internal Combustion Engine



La struttura allegata riporta le restanti classi semplici presenti nell'ontologia relativa al progetto.

Scenari di utilizzo:

• **Configurazione di un nuovo scooter**: L'utente ha la possibilità di scegliere tra diversi componenti per assemblare uno scooter che soddisfi le proprie necessità.

• **Personalizzazione estetica**: L'utente sceglie il colore dello scooter e altri dettagli visivi in modo da averne un'anteprima.

• **Configurazione tecnica**: permette all'utente di configurare gli aspetti tecnici dello scooter (sistema d'alimentazione, motore, ecc..). Inoltre, mentre l'utente seleziona le opzioni disponibili, l'applicazione valida le regole di validazione secondo la logica implementativa nel viewController.

• **Aggiunta di accessori**: Configurazione di bauletti, paravento, o altri optional. L'applicazione consente all'utente di deselezionare gli accessori selezionando l'opzione "None".

• **Conferma della configurazione**: Esportazione o visualizzazione della configurazione finale. Dopo aver completato le selezioni, l'utente clicca su un pulsante per visualizzare il riepilogo della configurazione.

L'applicazione mostra una finestra di dialogo con:

- Una lista dettagliata di tutti i componenti e accessori selezionati.

- Un messaggio di conferma che indica se la configurazione è valida.

L'utente può rivedere le scelte ed eventualmente tornare indietro per apportare modifiche

Query SPARQL

```
private String buildQuery(String componentType, String property) {
    String sparqlQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
            "PREFIX scooter: <http://www.semanticweb.org/aless/ontologies/2024/ScooterConfigurator#>" +
            "SELECT ?component ?property WHERE { " +
            "?component rdf:type scooter:" + componentType + " . " +
            "?component scooter:" + property + " ?property . " +
            "}";

    return sparqlQuery;
}
```

Questa porzione di codice costruisce una query SPARQL dinamica, in modo generico, per recuperare le informazioni richieste in base al tipo della classe (a cui appartiene l'individuo da ricercare) definita nell'ontologia ("componentType") e alla relativa proprietà ("property") a cui tale classe è associata.

Per ogni componente è incluso un metodo dedicato per recuperare informazioni in base al tipo di oggetto RDF e alla data property specifica associata nel dataset.

Alcuni esempi di query SPARQL per il recupero di informazioni dall'ontologia sono i seguenti:

```
public List<Battery> getBatteryComponents(Scooter scooter) {
    String Battery = "Battery";
    String HasBatteryCapacity  = "HasBatteryCapacity ";
    String sparqlQuery= this.buildQuery(Battery,HasBatteryCapacity );
    return getComponents(scooter, componentType: "Battery", sparqlQuery);
}
1 usage  turtle
public List<ElectricMotor> getElectricMotorComponents(Scooter scooter) {
    String Engine  = "Engine ";
    String HasMotorPower = "HasMotorPower";
    String sparqlQuery= this.buildQuery(Engine ,HasMotorPower  );
    return getComponents(scooter, componentType: "ElectricMotor", sparqlQuery);
}

1 usage  turtle
public List<InternalCombustionEngine> getInternalCombustionEngineComponents(Scooter scooter) {
    String Engine  = "Engine ";
    String HasEngineDisplacement ="HasEngineDisplacement ";
    String sparqlQuery= this.buildQuery(Engine ,HasEngineDisplacement   );
    return getComponents(scooter, componentType: "InternalCombustionEngine", sparqlQuery);
}
```

Le altre query SPARQL e il resto del codice relativo si trovano all'interno del file QueryService.

Responsabilità considerate nello sviluppo dell'applicazione

• **Caricamento delle risorse necessarie:** tale responsabilità riguarda il caricamento dei file necessari (ontologia e file FXML) in fase di avvio tramite l'OntologyLoader e tramite il metodo 'launch' nel controller principale.

• **Inferenza ontologica**: tale responsabilità ha lo scopo di applicare deduzioni logiche sui dati dell'ontologia per arricchire la conoscenza disponibile, abilitando nuove correlazioni e garantendo la coerenza delle informazioni attraverso operazioni di inferenza.

• **Esecuzione di query SPARQL**: tale responsabilità si intende consentire l'accesso a informazioni specifiche dell'ontologia tramite l'esecuzione di query SPARQL, permettendo di estrarre dati e relazioni in modo mirato e dinamico, in base alle esigenze dell'applicazione.

• **Popolamento dinamico dei dati**: inizializzazione dinamica dei dati dell'interfaccia, in base ai servizi e al modello ontologico caricato.

• **Configurazione e visualizzazione**: tale responsabilità si occupa di fornire un'interfaccia utente intuitiva che permetta agli utenti di configurare i componenti dello scooter tramite selezioni guidate e di visualizzare chiaramente la configurazione finale.

Interfacce e classi utilizzate

L'intero progetto è stato implementato secondo il pattern MVC, con i seguenti attori:

MODEL: Il modello è rappresentato dall'ontologia RDF e dalle relative classi che rappresentano il modello di dati.

VIEW: La parte GUI è definita nel file FXML e gestita dalla classe ScooterViewController, che si occupa anche del popolamento dinamico dei dati.

CONTROLLER: sono stati utilizzati due controller che collaborano tra loro per gestire le funzionalità del sistema. Il primo, il ScooterViewController, gestisce l'interfaccia utente e le interazioni; tale controller è dichiarato esplicitamente come il gestore del file FXML associato. Il secondo, ScooterConfiguratorController, rappresenta l'altro controller che funge da coordinatore per la configurazione inziale del sistema.

Di seguito viene presentato l'elenco delle classi utilizzate, con una descrizione delle responsabilità associate a ciascuna di esse.

ScooterConfiguratorApplication: Tale file gestisce l'inizializzazione e l'avvio dell'applicazione e ne definisce il punto di ingresso. Gestisce eventuali errori durante il lancio dell'app e coordina l'avvio del sistema passando il controllo alla classe ScooterConfiguratorController;

ScooterConfiguratorController: configura i servizi e il controller della vista, gestendo il caricamento delle risorse (ontologia e file FXML). Coordina le configurazioni iniziali e globali, delegando al controller della vista le impostazioni specifiche e collaborando alla gestione del file app e delle risorse globali.

ScooterViewController: interagisce con l'utente, gestendo la logica di selezione e visualizzazione, tramite il file FXML. È il controller specifico della vista per le configurazioni dinamiche: collega il modello dei dati con il popolamento e la visualizzazione nella GUI.

È stato aggiunto tale controller in modo tale da poterlo dichiarare in modo esplicito all'interno del file FXML.

OntologyLoader:

La classe OntologyLoader è incaricata dell'importazione del file RDF, utilizzando il framework Jena per caricare e inizializzare la struttura ontologica nel sistema. Fornisce il modello di ontologia utilizzato dall'intera applicazione.

• **PelletInference:**

La classe PelletInference si occupa di eseguire operazioni di inferenza sull'ontologia caricata, utilizzando il reasoner Pellet. La sua responsabilità principale è applicare regole logiche e deduzioni automatiche sui dati ontologici per arricchire la conoscenza estraibile. Questo consente di derivare nuove informazioni implicite dai dati esistenti, supportando query avanzate e garantendo la coerenza delle configurazioni.

Questa classe è stata implementata per ampliare il dominio dell'ontologia, pur non avendo attualmente un utilizzo diretto nel codice. Tuttavia, è stata progettata per essere potenzialmente utile in sviluppi futuri.

• **SPARQLQueryExecutor:**

La classe SPARQLQueryExecutor si occupa di gestire l'interazione tra ontologia e applicazione, eseguendo query SPARQL per recuperare dati.

• **QueryService:**

è un servizio che fornisce un'interfaccia per eseguire query SPARQL in un contesto di configurazione di uno scooter basato su un'ontologia RDF. La classe utilizza un esecutore di query (SPARQLQueryExecutor) per comunicare con il database RDF, recuperare i risultati e mapparli a oggetti Java specifici.

Considerazioni sull' estendibilità del codice prodotto

Il codice è impostato in modo tale che possa essere riutilizzato anche per altri modelli RDF: ciò che cambia riguarda esclusivamente il popolamento dinamico dei dati, in base al modello di riferimento caricato.

Il viewController, che gestisce il file FXML, mantiene sempre lo stesso funzionamento: variano solo la nomenclatura dei componenti lato UI e il popolamento dei di dati, diversificato per il modello caricato. Il controller gestisce la collaborazione tra i controller GUI e il file FXML sempre allo stesso modo, mentre il QueryService viene inizializzato con metodi relativi alla configurazione. Questa struttura consente al codice di adattarsi a diversi file RDF, rendendolo applicabile ad altre possibili configurazioni di prodotti.

Il codice risulta, così, estendibile anche grazie alla gestione del popolamento dei dati basata su eventi e sull'implementazione di metodi generici: tale approccio è stato concepito considerando che le operazioni eseguite seguono sempre lo stesso schema, differenziandosi unicamente per il popolamento specifico dei dati. In questo modo si tende ad utilizzare una logica unificata, riducendo il codice ripetuto e includendo qualche controllo per la gestione dei conflitti e di compatibilità delle specifiche configurazioni.

L'inserimento di nuovi elementi è strutturato in modo da richiedere solo poche modifiche, evitando stravolgimenti globali e la dipendenza da un'architettura rigida e monolitica.

Inoltre, sono stati implementati dei test per favorire il debug del codice durante lo sviluppo dell'applicazione.

Il progetto è eseguibile digitando nel terminale i comandi "**gradle build**" e "**gradle run**".

Applicazioni del configuratore di scooter nel settore

Il configuratore trova applicazione in diversi ambiti del settore motociclistico e commerciale. Grazie alla sua flessibilità, il sistema può essere adottato in vari contesti per migliorare l'esperienza del cliente, ottimizzare i processi di vendita e agevolare la personalizzazione del prodotto.

Alcune applicazioni possono essere le seguenti:

1) Vendita al Dettaglio nei Concessionari: Il configuratore può essere utilizzato nei concessionari per offrire ai clienti un'esperienza interattiva nella scelta del proprio scooter. Grazie all'interfaccia grafica, i clienti possono personalizzare in tempo reale il veicolo, visualizzando le opzioni disponibili e le relative configurazioni. Tale approccio migliora la qualità dell'esperienza utente, riducendo i tempi necessari per illustrare le configurazioni da adattare alle specifiche del cliente.

Esempio di utilizzo: Un cliente visita un concessionario e utilizza l'applicazione per configurare uno scooter, scegliendo tra diverse tipologie di motore, colori e accessori. Il venditore guida il cliente nella selezione e mostra un riepilogo della configurazione finale.

2) Vendita Online e E-Commerce: Il configuratore può essere integrato in piattaforme di e-commerce per consentire ai clienti di configurare e acquistare scooter personalizzati direttamente online, offrendo ai clienti un'esperienza personalizzata senza la necessità di visitare fisicamente un concessionario.

Esempio di utilizzo: Un cliente visita un sito web dedicato alla vendita di scooter, configura il proprio modello utilizzando il configuratore online e finalizza l'acquisto con pochi clic.

3) Produzione e Assemblaggio Personalizzati: Il configuratore può essere integrato nei processi produttivi per supportare la produzione su misura. I dati delle configurazioni vengono trasferiti direttamente ai sistemi di gestione della produzione, garantendo che ogni scooter venga assemblato secondo le specifiche richieste dal cliente, facilitando la produzione di scooter personalizzati, ottimizzando i tempi di produzione e migliorando la soddisfazione del cliente.

Esempio di utilizzo: Le configurazioni effettuate dai clienti nei concessionari vengono inviate automaticamente agli stabilimenti di produzione, dove vengono assemblati scooter unici basati sulle scelte personalizzate.

Considerazioni finali

Per i vincoli stabiliti nell' ontologia, questo configuratore può essere adatto per gli scooter. Tuttavia, modificando le specifiche dei vincoli relativi al motore e alla potenza della batteria, lo stesso strumento può essere adattato per motocicli più grandi, per i quali potrebbe essere necessario indicare ulteriori dettagli per la guida, come il possesso di una patente specifica e l'età minima per la guida.

In questo caso, l'utilizzo della classe 'Rider' consente di mantenere informazioni rilevanti per questi aspetti, anche se non viene direttamente utilizzata nel contesto di configurazione, ma offre un grado maggiore di chiarezza per questo caso implementato e potrebbe diventare più rilevante in futuro, in seguito a successivi sviluppi ed estensioni del sistema.