

Progetto di Web Semantico

Ontologia Credite Agricole Italia

Zandoli Silvia

950299

silvia.zandoli2@studio.unibo.it

25 febbraio 2023

Indice

1	Introduzione	1
2	Tecnologie e linguaggi adottati	1
2.1	Descrizione Owl	1
2.2	Java Spring Boot	2
2.3	Apache Jena	2
2.4	Pellet	2
3	Le classi modellate	2
4	Le object properties modellate	9
5	Le data properties modellate	15
6	Esempi di Inferenze	15
7	SWRL	19
7.1	Altre regole SWRL	21
8	SPARQL	22
8.1	Descrizione di altre query	26
9	Applicazione finale	28
9.1	Modalità di esecuzione	32
10	Sviluppi Futuri	32

1 Introduzione

Questa relazione si pone come obiettivo quello di voler descrivere come è stata realizzata l'ontologia Credite Agricole Italia che modella in maniera semplificata della conoscenza relativa all'attività di una banca, concentrandosi sull'aspetto più commerciale. Tale ontologia non importa alcuna ontologia esterna, pertanto è stata realizzata interamente da me.

Ci si concentra sulle attività necessarie per la gestione del denaro di singoli e organizzazioni a partire dai depositi in conto corrente, che poi danno l'accesso alle diverse operazioni ordinarie tra le quali si ricorda l'emissione di assegni. Viene gestita anche la gestione dei prestiti, a privati o imprese.

Con i fondi così raccolti, su richiesta dei clienti, le banche commerciali possono operare investimenti sui mercati finanziari come l'acquisto di titoli.

In questo progetto si modellano tali aspetti legati alla banca dove sono cliente, la Credite Agricole Cariparma. Essa negli anni ha inglobato dentro di sé tre banche, la banca Cassa Di Risparmio di Cesena, la Cassa di Risparmio di Rimini (ex Carim) e la banca di San Miniato, e le loro rispettive filiali.

2 Tecnologie e linguaggi adottati

- RDF: linguaggio utilizzato per la definizione del modello e delle triple che descrivono il dominio di riferimento ;
- RDF Schema: linguaggio utilizzato per estendere il vocabolario di RDF;
- OWL: linguaggio utilizzato per aumentare l'espressività dell'ontologia;
- SWRL, un linguaggio a regole basato su OWL ;
- SPARQL: linguaggio utilizzato per definire le query sulle istanze dell'ontologia ;
- Angular: framework scelto per la realizzazione del front-end dell'applicativo ;
- Spring Boot: framework scelto per la realizzazione del back end dell'applicativo e dei servizi REST;
- Apache Jena: framework Java usato per estrarre dati dall'ontologia ;

2.1 Descrizione Owl

Si è visto come sia possibile con RDF definire delle relazioni tra risorse differenti. Tuttavia RDF non è in grado di definire un vocabolario e una semantica, cioè fornire della conoscenza in più alle risorse e ai predicati. Per risolvere questo problema viene utilizzato RDFS (RDF Schema), come linguaggio ontologico. Con RDFS è possibile definire classi e proprietà e si introduce la cosiddetta T-Box (Terminological Knowledge), costruita sopra la A-Box (Assertional Knowledge) definita da RDF.

La A-Box rende disponibile e accessibile la rappresentazione formale del modello concreto di un frammento di realtà (dati, istanze, ...), mentre la T-Box rende disponibile ed accessibile la rappresentazione formale del modello concettuale di un frammento di realtà (schemi, classi, ...)

Per poter ottenere una maggiore espressività entra in gioco OWL (Ontology Web Language). Con OWL è possibile definire ulteriori concetti, tra cui:

- Vincoli di Cardinalità
- Tipi di proprietà
- Restrizioni d'uso di una specifica proprietà
- Combinazione tra classi
- Esprimere il concetto che due istanze diverse fanno riferimento allo stesso oggetto
- Esprimere il concetto che due classi diverse fanno riferimento allo stesso insieme
- Esprimere il concetto che due proprietà diverse fanno riferimento alla stessa relazione binaria

2.2 Java Spring Boot

Java Spring Framework (Spring Framework) è un popolare framework open source di livello enterprise per la creazione di applicazioni autonome a livello di produzione che vengono eseguite sulla Java Virtual Machine (JVM).

Spring Boot è un framework per lo sviluppo di applicazioni web basate su codice Java (precisamente su stack Java). Esso semplifica e velocizza lo sviluppo di applicazioni web e microservizi con Spring Framework attraverso tre funzionalità principali:

- 1) Configurazione automatica
- 2) Un approccio persistente alla configurazione
- 3) La capacità di creare applicazioni autonome

Queste funzioni cooperano per fornire uno strumento che consente di configurare un'applicazione basata su Spring con una configurazione e un'installazione minime.

2.3 Apache Jena

Jena è un framework Java per lo sviluppo di applicazioni orientate al web semantico. Esso fornisce delle API per leggere e scrivere grafi RDF (Resource Description Framework). I grafi sono rappresentati come un "modello" astratto. Un modello può provenire da fonti come un file, un database, un URL o una combinazione di questi. Il modello può anche essere interrogato con query SPARQL.

Jena supporta anche OWL. Il framework include inoltre vari motori di deduzione logica, ed è possibile impostare il motore Pellet per lavorare con Jena. Jena supporta la serializzazione dei grafi RDF attraverso vari metodi come i database relazionali, RDF/XML, Turtle etc.

2.4 Pellet

Pellet è il reasoner Java open source utilizzato nella presente ontologia. Può essere usato con Jena e o nelle librerie OWL-API. Ha funzionalità per verificare la coerenza delle ontologie, spiegare le inferenze e rispondere alle query SPARQL.

3 Le classi modellate

In figura si riportano tutte le classi che sono state modellate per rappresentare il dominio in esame. In figura anche alcune rappresentazioni delle classi.

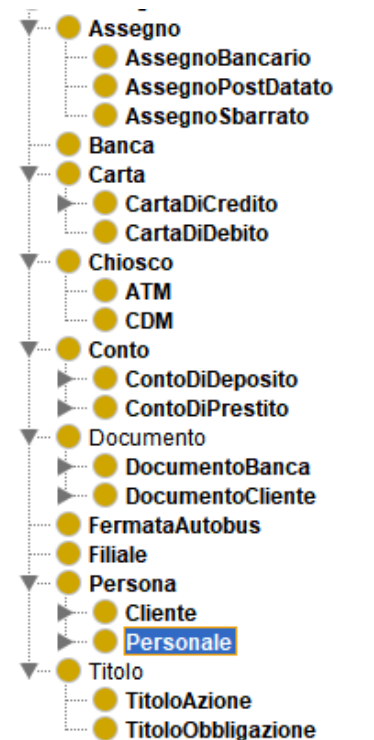


Figura 1: Le classi dell'ontologia

Banca

L'entità banca rappresenta la figura dell'istituto pubblico centrale. Ha delle filiali e ospita gran parte del personale manageriale.

Chiosco

Modella l'entità chiosco, un centro vicino alle filiali o in altre zone dove ci sono terminali che permettono di fare il lavoro che svolgerebbe l'impiegato di qualsiasi filiale come prelevare o versare denaro. Si suddivide nelle sottoclassi ATM e CDM. La CDM rappresenta la Cash Deposit Machine che serve per accettare depositi su un conto bancario di un cliente, L'ATM invece è il classico bancomat che serve per ritirare contanti dal conto.

FermataAutobus

Rappresenta la fermata di autobus, con un numero di fermata .

Filiale

Concetto che modella la filiale di una banca, che si trova in una specifica zona della città. Ospita tutto il personale non manageriale e il direttore di filiale .

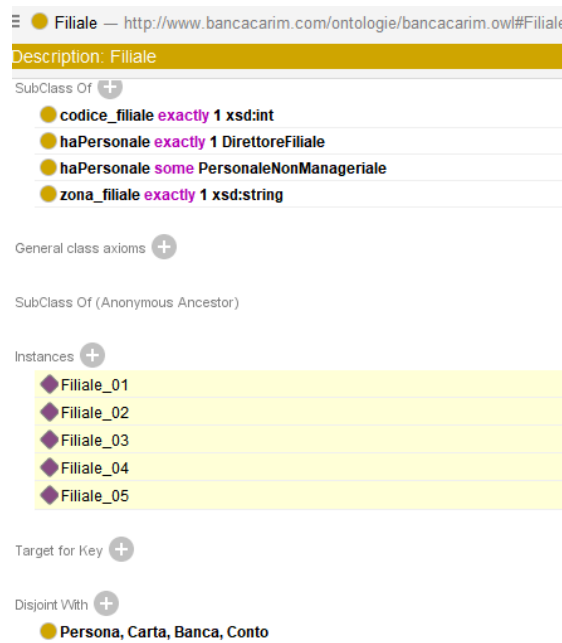


Figura 2: Rappresentazione classe filiale

Documento

Concetto che modella l'entità Documento e si suddivide in due sottoclassi, una modella tutta la varia documentazione della banca gestita dal personale, l'altra quella del cliente, utile da sottomettere per richiedere conti, prestiti, carte, investimenti.

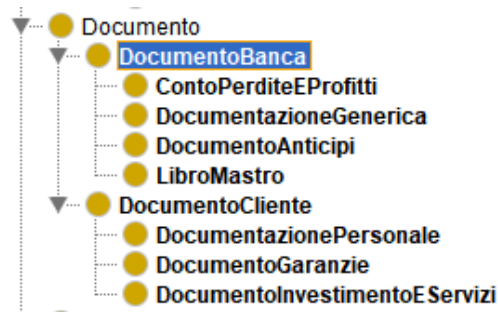


Figura 3: Gerarchia Classe Documento

Titolo

Il titolo rappresenta qualcosa che può essere acquistato o venduto. E' di tipo azionario o obbligazionario.

Conto

La classe Conto esprime il concetto di conto. Come si vede in figura ha come sottoclassi il conto di deposito e il conto di prestito. Il conto di deposito è un generico deposito di risparmio che permette di fare operazioni semplici, come versamenti e prelievi. Ha un numero di conto, un intestatario e un saldo. Bisogna sottomettere della specifica documentazione per aprire un conto di qualsiasi tipo. Si suddivide in:

- Conto Corrente, strumento per deposito e gestione del denaro. Ha una carta associata .
- Deposito Fisso, uno strumento finanziario che offre agli investitori un tasso di interesse più elevato rispetto a un normale conto di risparmio, fino alla data di scadenza indicata. Non si possono fare prelievi durante il periodo vincolato a pena di perdere l'interesse agevolato.
- Deposito di Risparmio che corrisponde al classico libretto di risparmio e in base all'età del cliente può essere di diversi tipi e può avere diverse agevolazioni, come si vede in fig. Può avere una carta associata.

Il prestito, per semplicità di modellazione, è stato modellato come un conto. La cui finalità è quella di richiedere un prestito, appunto. Il suo saldo rappresenta la quantità di somma da restituire. Si suddivide in prestito a imprese o a privati, quest'ultima ha specifiche sottoclassi. Deve essere approvato da parte del personale manageriale.

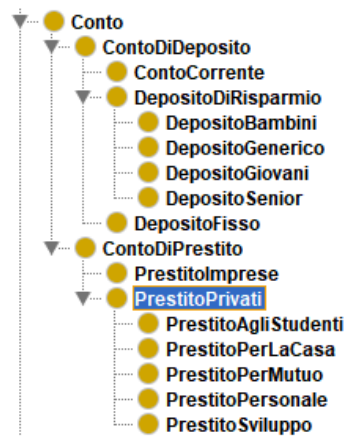


Figura 4: Gerarchia classe conto

Si riportano in figura solo alcune rappresentazioni.

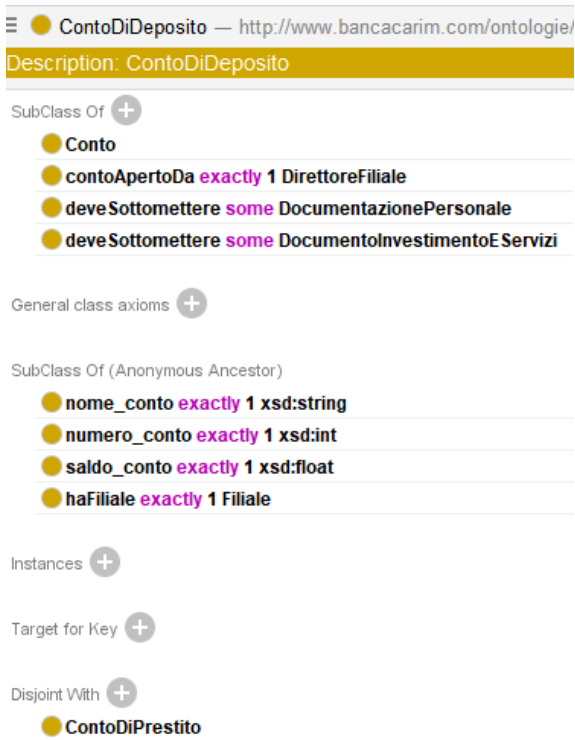


Figura 5: Rappresentazione classe Conto Di Deposito

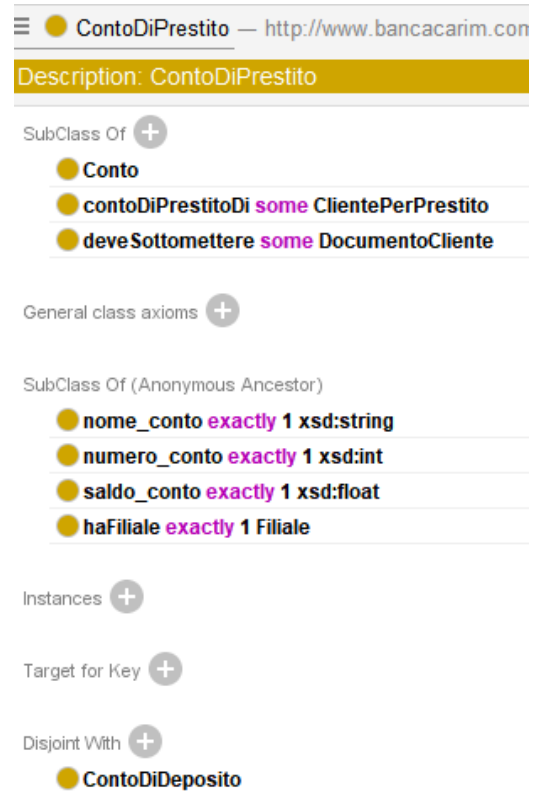


Figura 6: Rappresentazione classe Conto di Prestito

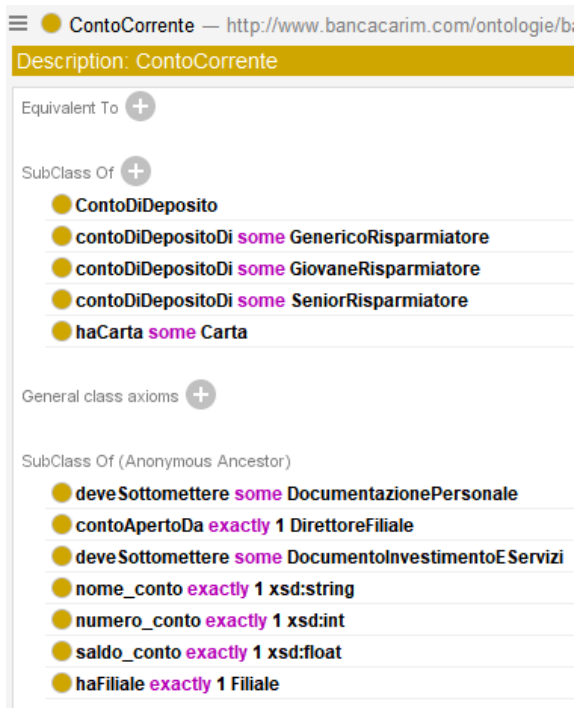


Figura 7: Rappresentazione classe Conto Corrente

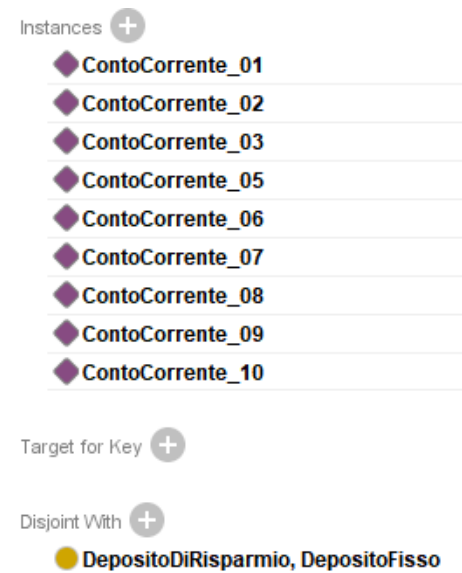


Figura 8: Rappresentazione classe Conto Corrente -2

Assegno

La classe modella l'entità Assegno, uno strumento di pagamento sostitutivo del contante con il quale il titolare del conto corrente (traente) ordina alla propria banca (trattario) di versare una determinata somma di denaro a favore di un'altra persona (beneficiario). Si suddivide in tre sottoclassi: bancario, postdatato (su cui è apposta una data futura) e sbarrato (assegno non trasferibile).

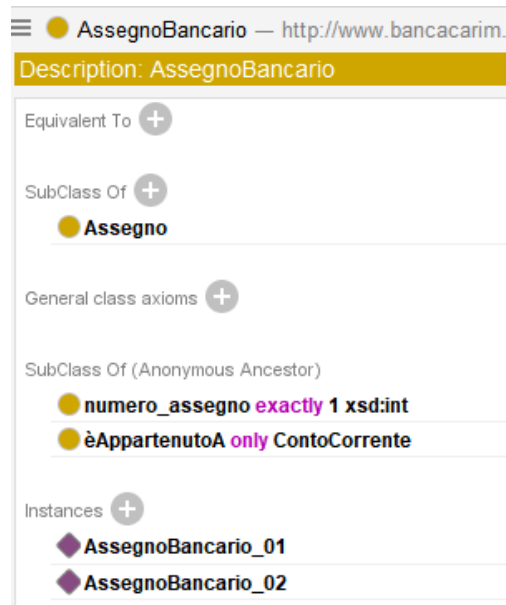


Figura 9: Rappresentazione classe Assegno Bancario

Carta

La classe modella il concetto di Carta, utile per fare versamenti o prelievi. Si suddivide in carta di credito o di debito. Una carta di credito è approvata se è valida e non ha raggiunto la data di scadenza.

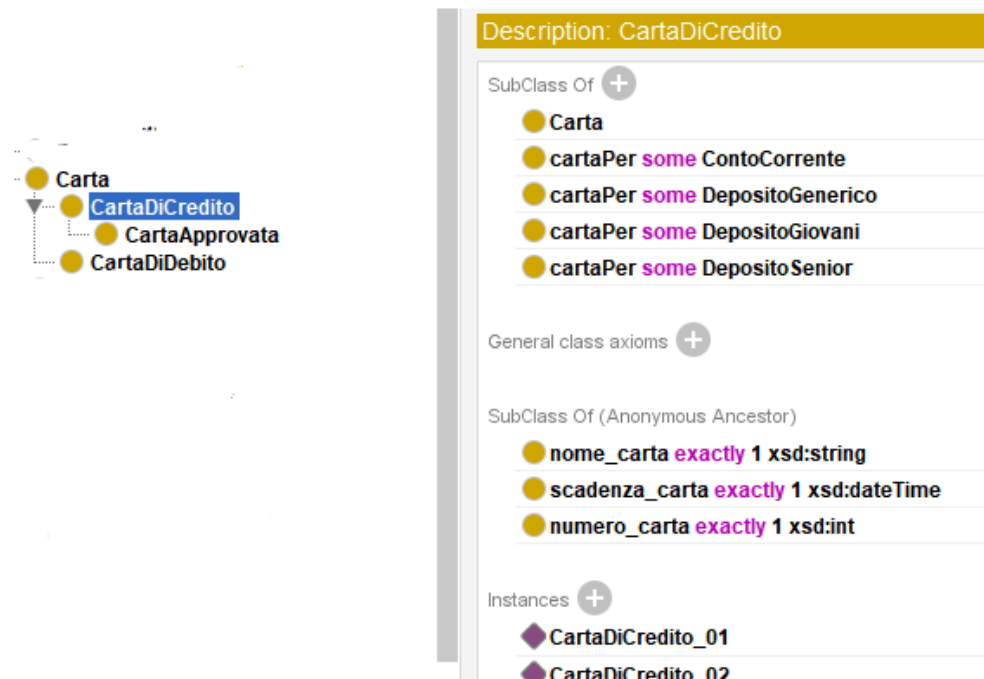


Figura 10: Rappresentazione classe Carta di Credito

Persona

Modella il concetto di Persona che si suddivide nelle sottoclassi Cliente e Personale e contiene tutti quegli individui che all'interno di una banca hanno un ruolo o come personale o come clienti.

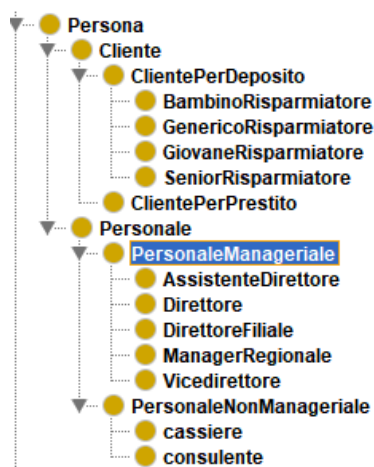


Figura 11: Gerarchia classe Persona

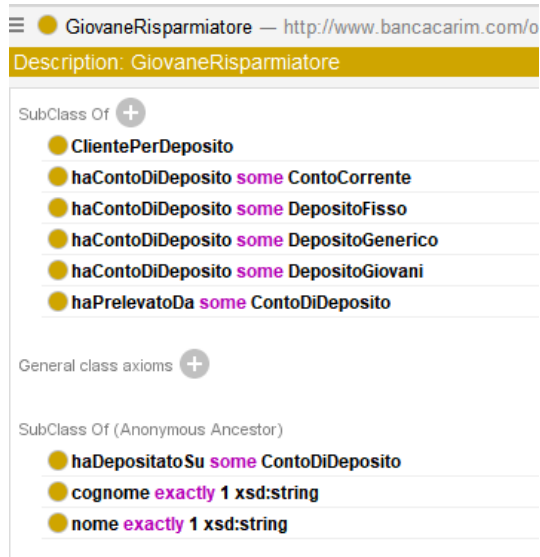


Figura 12: Rappresentazione classe Giovane Risparmiatore

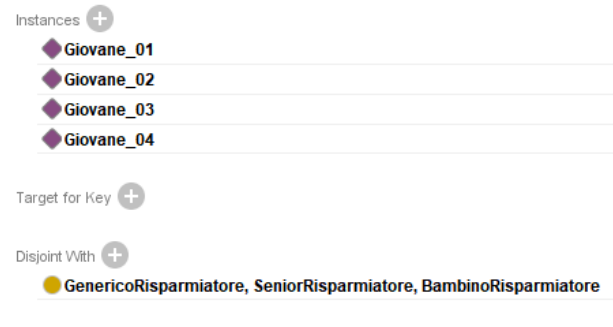


Figura 13: Rappresentazione Giovane Risparmiatore - 2

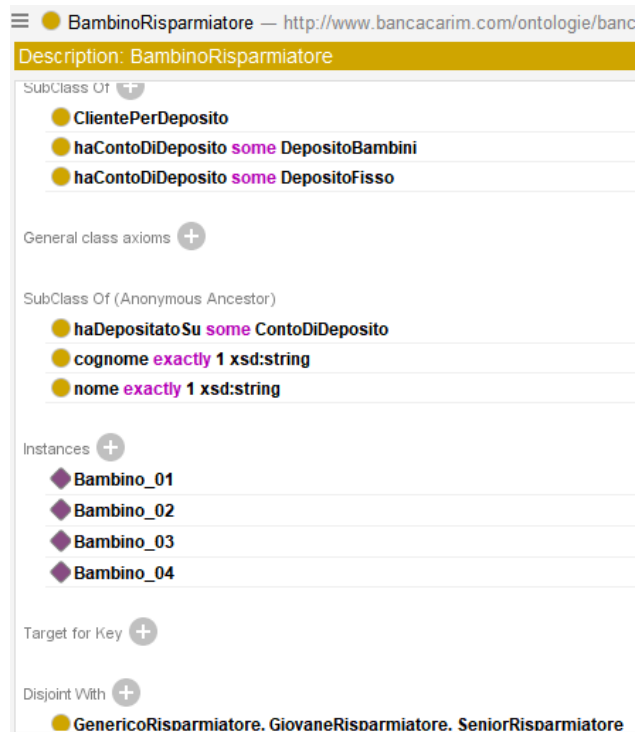


Figura 14: Rappresentazione classe Bambino Risparmiatore

4 Le object properties modellate

Vengono riportate le object properties realizzate, le quali consentono di stabilire una relazione tra un soggetto e un oggetto.

- acquistoTitolo

- descrizione: proprietà che mette in relazione un cliente di un conto di deposito con il titolo azionario o obbligazionario che acquista.
- dominio: ClientePerDeposito
- range: Titolo
- caratteristiche: Si tratta di una proprietà asymmetric. Con questa proprietà si afferma che se il dominio è legato al range allora il range non è collegato al dominio con la stessa proprietà.
- **proponeTitolo**
 - descrizione: proprietà che relaziona un consulente con un titolo che vuole proporre
 - dominio: consulente
 - range: Titolo
 - caratteristiche: E' una proprietà asymmetric
- **èAppartenutoA:**
 - descrizione: proprietà che mette in relazione un assegno con un conto corrente.
 - dominio: Assegno .
 - range: ContoCorrente .
 - caratteristiche: Si tratta di una proprietà inverse functional in quanto dato un assegno appartiene solo a un conto corrente.
- **cartaPer:**
 - descrizione: proprietà che specifica la relazione di una carta con il suo specifico conto di deposito.
 - dominio: Carta .
 - range: ContoDiDeposito .
 - caratteristiche: è una proprietà asymmetric.
 - proprietà inversa: haCarta .
- **checkCarta:**
 - descrizione: questa proprietà lega un ATM o un cassiere con una carta, e indica l'atto di controllare che la carta non sia soggetta a frodi, non abbia avuto transazioni sospette e che abbia un fondo .
 - dominio: ATM or cassiere
 - range: Carta
 - caratteristiche: è una proprietà asymmetric.
- **checkSaldo:**
 - descrizione: questa proprietà esprime una relazione tra un membro del personale non manageriale e un conto di deposito e esprime l'atto di controllare il saldo del conto, ovvero che l'importo da prelevare o l'importo minimo per acquistare un titolo ci siano.
 - dominio: PersonaleNonManageriale.
 - range: ContoDiDeposito.
 - caratteristiche: è una proprietà asymmmetric.
- **checkSaldoATM:**
 - descrizione: questa proprietà lega un ATM con un conto corrente e esprime l'atto di controllare il saldo del conto, ovvero che l'importo da prelevare ci sia.
 - dominio: ATM.

- range: ContoCorrente.
- caratteristiche: è una proprietà asymmetric.
- consisteDi:
 - descrizione: proprietà che lega una banca con una delle sue filiali
 - dominio: Banca.
 - range: Filiale.
 - caratteristiche: è una proprietà asymmetric.
- contoApertoDa:
 - descrizione: proprietà che mette in relazione il conto da aprire con un membro del personale manageriale.
 - dominio: ContoDiDeposito.
 - range: PersonaleManageriale.
 - caratteristiche: è una proprietà functional in quanto ogni conto è aperto da una sola persona, che deve far parte del personale manageriale.
- contoDi:
 - descrizione: relazione che esprime l'appartenenza di un conto.
 - caratteristiche: è una proprietà asymmetric .
 - proprietà inversa: haConto .
- ContoDiDepositoDi:
 - descrizione: relazione che esprime l'appartenenza di un conto di deposito e relaziona un conto di deposito con il suo proprietario.
 - dominio: ContoDiDeposito .
 - range: ClientePerDeposito .
 - caratteristiche: è proprietà SubProperty Of di contoDi.
 - proprietà inversa: haContoDiDeposito .
- ContoDiPrestitoDi:
 - descrizione: relazione che esprime l'appartenenza di un conto di prestito e relaziona il conto di prestito con il suo proprietario.
 - dominio: ContoDiPrestito .
 - range: ClientePerPrestito .
 - caratteristiche: è proprietà SubProperty Of di contoDi.
 - proprietà inversa: haContoDiPrestito .
- ContoGestitoDa:
 - descrizione: relazione che esprime la gestione di un conto da parte di un membro del personale.
 - dominio: Conto.
 - range: Personale .
 - caratteristiche: è una proprietà asymmetric.
- deveSottomettere:

- descrizione: proprietà che relaziona un conto a un documento del cliente e esprime la necessità di sottomettere della documentazione .
 - dominio: Conto .
 - range: DocumentoCliente .
 - caratteristiche: Questa proprietà è asymmetric.
- docGestitaDa:
 - descrizione: proprietà che esprime la gestione di documentazione da parte del personale.
 - dominio: DocumentoBanca .
 - range: Personale .
 - caratteristiche: Questa proprietà è asymmetric .
- emetteAssegno:
 - descrizione: proprietà che esprime l'emissione di un assegno e lega un cliente con l'assegno da emettere.
 - dominio: ClientePerDeposito .
 - range: Assegno .
 - caratteristiche: Questa proprietà è asymmetric .
- haCarta:
 - descrizione: proprietà che indica la relazione tra un conto di deposito e una carta.
 - dominio: ContoDiDeposito
 - range: Carta
 - caratteristiche: è una proprietà asymmetric
 - proprietà inversa: cartaPer
- haChiosco:
 - descrizione: proprietà che specifica la presenza di un chiosco.
 - dominio: Banca
 - range: Chiosco
 - caratteristiche: è una proprietà asymmetric
- haATM:
 - descrizione: proprietà che specifica la presenza di un ATM.
 - dominio: Banca
 - range: ATM
 - caratteristiche: è la proprietà SubProperty Of di haChiosco.
- haCDM:
 - descrizione: proprietà che specifica la presenza di un CDM.
 - dominio: Banca
 - range: CDM
 - caratteristiche: è la proprietà SubProperty Of di haChiosco.
- haConto

- relazione che esprime il possedere un conto
- caratteristiche: è una proprietà asymmetric.
- proprietà inversa: contoDi
- haContoDiDeposito
 - descrizione: proprietà che esprime il possedere un conto di deposito e lega un cliente per deposito con il suo conto di deposito.
 - dominio: ClientePerDeposito
 - range: ContoDiDeposito
 - caratteristiche: è una proprietà SubProperty Of di haConto
 - proprietà inversa: contoDiDepositoDi
- HaContoDiPrestito
 - descrizione: proprietà che esprime il possedere un conto di prestito e lega un cliente per prestito con il suo conto di prestito.
 - dominio: ClientePerPrestito
 - range: ContoDiPrestito
 - caratteristiche: è una proprietà SubProperty Of di haConto
 - proprietà inversa: contoDiPrestitoDi
- haDepositatoSu
 - descrizione: proprietà che lega un cliente con un suo conto di deposito e identifica l'atto di depositare denaro
 - dominio: ClientePerDeposito
 - range: ContoDiDeposito
 - caratteristiche: è una proprietà asymmetric
- haFiliale
 - descrizione: proprietà che relaziona il conto con una filiale
 - dominio: Conto
 - range: Filiale
 - caratteristiche: e' una proprietà functional perchè un conto ha una sola filiale
- haPagatoPer
 - descrizione: proprietà che relaziona un cliente con un prestito e indica l'atto di chiedere un prestito in denaro
 - dominio: ClientePerPrestito
 - range: ContoDiPrestito
 - caratteristiche: è una proprietà asymmetric
- haPersonale
 - descrizione: proprietà che relaziona una filiale con un membro del personale
 - dominio: Filiale
 - range: Personale
 - caratteristiche: è una proprietà asymmetric

- proprietà inversa: èPersonaleDi
- haPersonaleManageriale
 - descrizione: proprietà che relaziona una banca con un membro del personale manageriale
 - dominio: Banca
 - range: PersonaleManageriale
 - caratteristiche: è una proprietà asymmetric
 - proprietà inversa: èPersonaleManagerialeDi
- haPrelevatoDa
 - descrizione: proprietà che descrive l'azione del prelevare e collega un cliente con un suo conto di deposito
 - dominio: ClientePerDeposito
 - range: ContoDiDeposito
 - caratteristiche: è una proprietà asymmetric
- prestitoApprovatoDa
 - descrizione: proprietà che lega un prestito con un membro del personale manageriale, descrive l'atto di approvare un prestito richiesto da un cliente.
 - dominio: ContoDiPrestito
 - range: PersonaleManageriale
 - caratteristiche: è una proprietà asymmetric
- vicinoA
 - descrizione: proprietà che esprime la vicinanza di una fermata di autobus ad una certa filiale
 - dominio: FermataAutobus
 - range: Filiale
- èPersonaleDi
 - descrizione: proprietà che relaziona un membro del personale con una filiale
 - dominio: Personale
 - range: Filiale
 - proprietà inversa: haPersonale
- èPersonaleManagerialeDi
 - descrizione: proprietà che relaziona un membro del personale manageriale con una banca
 - dominio: PersonaleManageriale
 - range: Banca
 - proprietà inversa: haPersonaleManageriale

5 Le data properties modellate

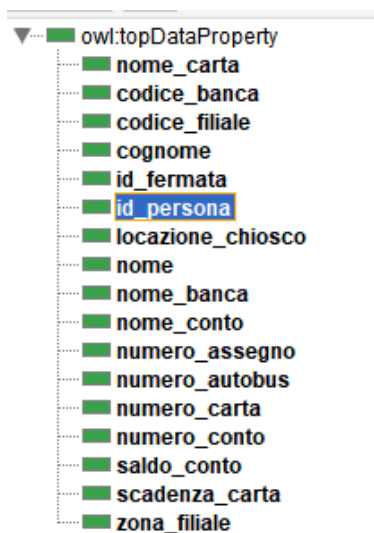


Figura 15: Data Properties modellate

6 Esempi di Inferenze

Uno dei vantaggi delle tecnologie del Web Semantico sta nel fatto che sono in grado di inferire informazioni tramite deduzioni logiche. Il motore di inferenza usato è Pellet.

Si mostrano alcune inferenze dall'ontologia

Filiale_01 — http://www.bancacarim.com/ontologie/bancacarim.owl#Filiale_01

Annotations Usage

Annotations: **Filiale_01**

Annotations

Description: Filiale_01

Types

Filiale

Same Individual As

Different Individuals

Property assertions: Filiale_01

Object property assertions

- haPersonale** Consulente_01
- haPersonale** Consulente_02
- haPersonale** DirettoreFiliale_01
- haPersonale** Cassiere_02
- haPersonale** Cassiere_01

Data property assertions

- codice_filiale** "01"^^xsd:int
- zona_filiale** "Miramare"^^xsd:string

Figura 16: Rappresentazione di una filiale e delle deduzioni sul personale che ci lavora

CartaDiCredito_06 — http://www.bancacarim.com/ontologie/bancacarim.owl#CartaDiCredito_06

Annotations Usage

Annotations: **CartaDiCredito_06**

Annotations

Description: CartaDiCredito_06

Types

CartaDiCredito

Same Individual As

Different Individuals

Property assertions: CartaDiCredito_06

Object property assertions

- cartaPer** DepositoGiovani_03

Data property assertions

- nome_carta** "GoldCard"^^xsd:string
- numero_carta** "0006"^^xsd:int

Negative object property assertions

Negative data property assertions

Figura 17: Rappresentazione di una carta di credito e della deduzione del conto a cui appartiene

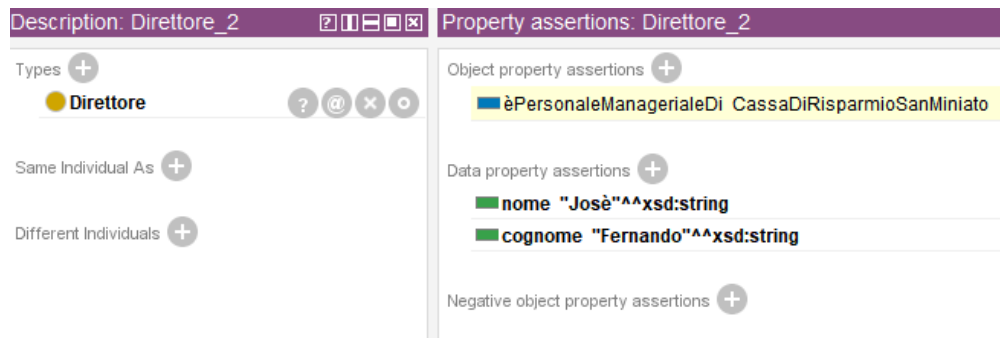


Figura 18: Rappresentazione di un direttore e della deduzione sulla banca per la quale lavora

Si possono dedurre nuove informazioni anche grazie alle **regole swrl**. Come nei prossimi esempi. La deduzione sottolineata in rosso è dedotta dalla regola swrl numero 7, riportata nel paragrafo successivo.



Figura 19: Rappresentazione di un conto corrente e deduzione sul consulente che gestisce quel conto

La deduzione sottolineata in verde è dedotta dalla regola swrl n 3 , mentre la deduzione sottolineata in viola è dedotta dalla regola swrl n 5.

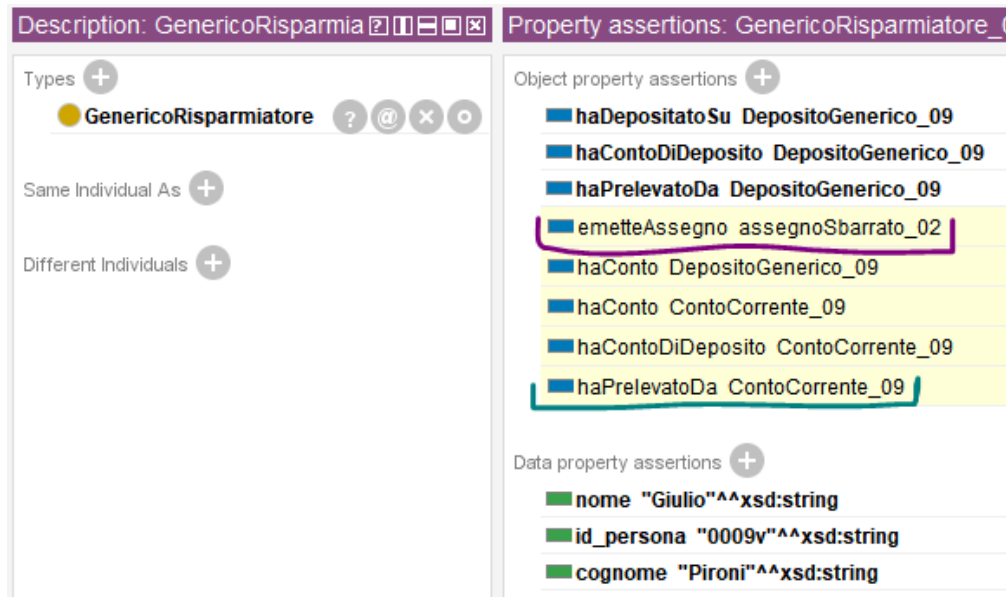


Figura 20: Rappresentazione di un generico risparmiatore con alcune deduzioni

La deduzione sottolineata in verde chiaro è dedotta dalla regola swrl numero n 6

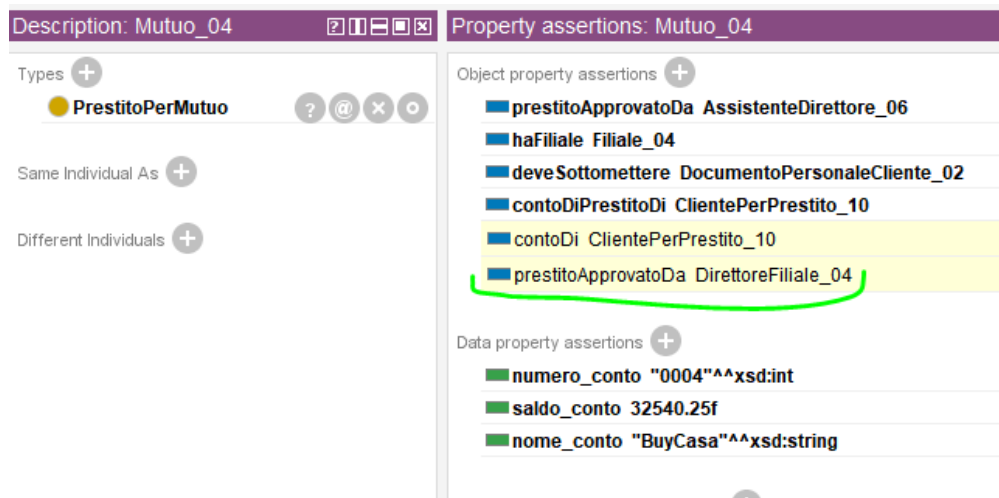


Figura 21: Rappresentazione di un mutuo con la deduzione che è stato approvato anche dal direttore della filiale, quindi è approvato in via definitiva

La deduzione sottolineata in rosso è dedotta dalla regola swrl n 5, mentre la deduzione sottolineata in blu è dedotta dalla regola n 1

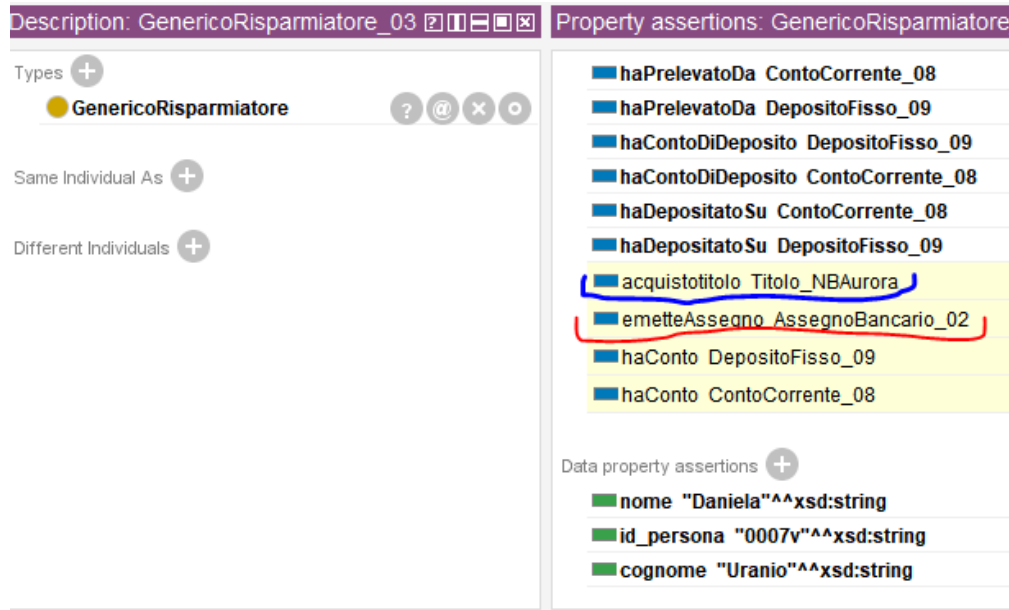


Figura 22: Rappresentazione di un generico risparmiatore e deduzione sul fatto che ha acquistato un titolo

7 SWRL

Il Semantic Web Rule Language (SWRL) è un linguaggio proposto per il Semantic Web che può essere utilizzato per esprimere regole oltre che logiche. Le regole hanno la forma di un'implicazione tra un antecedente (corpo) e il conseguente (capo). Il significato previsto può essere letto come: ogni volta che le condizioni specificate nell'antecedente valgono, devono valere anche le condizioni specificate nella conseguente. Esempio di sintassi leggibile dall'uomo.

```
hasParent(?x1,?x2) ^ hasBrother(?x2,?x3) -> hasUncle(?x1,?x3)
```

SWRL consente agli utenti di scrivere regole che possono essere espresse in termini di concetti OWL per fornire capacità di ragionamento deduttivo più potenti rispetto al solo OWL.

- Le regole vengono salvate come parte dell'ontologia.
- Funziona con motori di regole e reasoners (ad es. Pellet)

Per permettere al reasoner di inferire maggiori informazioni riguardanti l'ontologia sono state realizzate alcune regole SWRL, sotto riportate.

Regola n 1) Regola che specifica che l'acquisto di un titolo sia azionario che obbligazionario deve essere fatto solo dal conto corrente e da un consulente che gestisce quel conto che inoltre deve prima controllare il saldo.

```
bancacarim:ContoCorrente(?currentaccount) ^
```

```
bancacarim:haContoDiDeposito(?custom, ?currentaccount) ^
```

```
bancacarim:Titolo(?tit) ^ bancacarim:contoGestitoDa(?currentaccount, ?consulent) ^
```

```
bancacarim:proponeTitolo(?consulent, ?tit) ^
```

```
bancacarim:deveSottomettere(?currentaccount, ?docPersonal) ^
```

bancacarim:checkSaldo(?consulent, ?currentaccount) ->

bancacarim:acquistotitolo(?custom, ?tit)

Regola n 2)Regola che esprime il fatto che l'ATM prima di autorizzare il prelievo del denaro dalla carta deve controllare che la carta non abbia avuto transazioni sospette (checkCarta) e che sul conto corrente ci sia un saldo sufficiente.

Si è aggiunta la condizione haCarta perchè per fare il prelevamento dal conto presso un ATM è necessaria che ci sia associata una carta di credito.

bancacarim:ContoCorrente(?currentaccount) ^

bancacarim:haContoDiDeposito(?customer, ?currentaccount) ^

bancacarim:ATM(?atmmachine) ^ bancacarim:haCarta(?currentaccount, ?card) ^

bancacarim:checkCarta(?atmmachine, ?card) ^

bancacarim:checkSaldoATM(?atmmachine, ?currentaccount)

-> bancacarim:haPrelevatoDa(?customer, ?currentaccount)

Regola n 3) Si ripete la stessa regola ma con autorizzazione di prelevamento dal conto corrente presso il cassiere. Non c'è bisogno della carta ma preleva direttamente dal conto.

bancacarim:ContoCorrente(?currentaccount) ^

bancacarim:haContoDiDeposito(?customer, ?currentaccount) ^

bancacarim:cassiere(?clerk) ^ bancacarim:checkSaldo(?clerk, ?currentaccount) ->

bancacarim:haPrelevatoDa(?customer, ?currentaccount)

Regola n 4)Regola che esprime l'autorizzazione di prelevamento dal Deposito di Risparmio.

bancacarim:DepositoDiRisparmio(?deposit) ^

bancacarim:haContoDiDeposito(?customer,?deposit) ^

bancacarim:cassiere(?clerk) ^ bancacarim:checkSaldo(?clerk, ?deposit) ->

bancacarim:haPrelevatoDa(?customer, ?deposit)

Regola n 5)Regola che esprime le condizioni per emettere un assegno

bancacarim:ContoCorrente(?currentaccount) ^

bancacarim:haContoDiDeposito(?customer, ?currentaccount) ^

bancacarim:èAppartenutoA(?cheque, ?currentaccount) ->

bancacarim:emetteAssegno(?customer, ?cheque)

Regola n 6) Regola che esprime le condizioni per ottenere un prestito per privati. Deve essere approvato anche dal direttore della filiale, che prima deve venire a conoscenza delle condizioni personali del richiedente, leggendo l'apposita documentazione fornita dal cliente.

```

bancacarim:PrestitoPrivati(?leading) ^ bancacarim:DirettoreFiliale(?dir) ^
bancacarim:haFiliale(?leading, ?filiale) ^ bancacarim:haPersonale(?filiale, ?dir) ^
bancacarim:deveSottomettere(?leading, ?docPers) ->
bancacarim:prestitoApprovatoDa(?leading, ?dir)

```

Regola n 7) Regola che stabilisce il fatto che un conto corrente deve essere gestito solo da un consulente che lavora nella filiale alla quale il conto appartiene

```

bancacarim:ContoCorrente(?conto) ^ bancacarim:haFiliale(?conto, ?filiale) ^
bancacarim:consulente(?consulent) ^ bancacarim:haPersonale(?filiale, ?consulent) ->
bancacarim:contoGestitoDa(?conto, ?consulent)

```

7.1 Altre regole SWRL

Una delle caratteristiche più importanti di SWRL sono le built-in. Una built-in è un predicato che prende uno o più argomenti e restituisce true se gli argomenti soddisfano il predicato. Le built-in possono aumentare di molto l'espressività e per essere aggiunte nelle regole devono essere importate dalle librerie specifiche.

Per convenzione, una delle built-in cuore di SWRL viene espressa preceduta dal qualificatore `swrlb:`. Di seguito si riportano alcuni esempi dell'uso di tale built-in sull'ontologia in esame.

Si vuole esprimere il controllo sulla validità della Carta di credito, utile quando si vuole fare una transazione o prelevamento. La carta viene approvata e ritenuta valida se non si è raggiunta la data di scadenza e se supera i controlli anti frode (quest'ultima condizione è racchiusa dalla proprietà `checkCarta`)

```

bancacarim:CartaDiCredito(?card) ^ bancacarim:checkCarta(?clerk, ?card) ^
swrlb:dateTime(?presentTime) ^ swrlb:lessThan(bancacarim:scadenza_carta, ?presentTime)
-> bancacarim:CartaApprovata(?card)

```

Un altro esempio, si vuole esprimere che un risparmiatore, per essere definito come giovane risparmiatore e quindi poter aprire un deposito giovani, deve avere più di 14 anni e meno di 30 anni.

```

bancacarim:GiovaneRisparmiatore(?p) ->
swrlb.greaterThan(età, 14) ^ swrlb.lessThan(età, 30)

```

Lo stesso ragionamento si può applicare per i bambini e per i senior.

Un'altra built-in importante è la `sqwrl`. Viene utilizzata per interrogare l'ontologia OWL.

E' difficile eseguire il debug sulle regole SWRL perchè esplorare tutte le potenziali e complesse dipendenze tra le regole può essere complesso. SQWRL può essere usato per facilitare questo tipo di debugging.

I risultati dell'interrogazione vengono trovati attraverso l'antecedente che può essere visto come pattern matching.

Un esempio, si vogliono contare tutti i conti (esattamente i conti corrente, conti di deposito, risparmio) che hanno i giovani nella Credite Agricole

```

bancacarim:GiovaneRisparmiatore(?p) ^ bancacarim:haContoDiDeposito(?p, ?c) ->
sqwrl:count(?c)

```

Risposta: 8

Un altro esempio, per ogni giovane cliente della Credite Agricole si vogliono contare tutti i suoi conti (esattamente conti corrente, conti di deposito, risparmio)

```
bancacarim:GiovaneRisparmiatore(?p) ^ bancacarim:haContoDiDeposito(?p, ?c) ->
sqwrl:select(?p) ^ sqwrl:count(?c)
```

S6		bancacarim:GiovaneRisparmiatore(?p) ^ bancacarim:haContoDiDeposito(?p, ?c) -> sqwrl:select(?p) ^ sqwrl:count(?c)	
SQWRL Queries	OWL 2 RL	S6	
p		count(?c)	
bancacarim:Giovane_02		"2"^^xsd:int	
bancacarim:Giovane_01		"3"^^xsd:int	
bancacarim:Giovane_04		"1"^^xsd:int	
bancacarim:Giovane_03		"2"^^xsd:int	

8 SPARQL

SPARQL è un linguaggio di interrogazione per dati rappresentati tramite il Resource Description Framework (RDF). SPARQL è uno degli elementi chiave delle tecnologie legate al Web Semantico, e consente di estrarre informazioni dalle basi di conoscenza distribuite sul web. In una query SPARQL molto importante è la definizione dei prefissi (PREFIX), che non sono altro che delle scorciatoie per fare riferimento ai diversi namespace utilizzati nella query (le ontologie di riferimento).

SPARQL trova i risultati dell'interrogazione attraverso il pattern matching all'interno del grafo che deve soddisfare ciò che è stato specificato nella clausola WHERE della query. Le parole chiave di SPARQL sono: SELECT, FROM, WHERE, GROUP BY, LIMIT, OPTIONAL, FILTER, ORDER BY.

UNION serve per specificare pattern alternativi nella clausola WHERE quando si vogliono ottenere risultati che soddisfino una path-expression oppure un'altra (pattern in OR).

Si riportano alcune query per l'ontologia in esame

1) Si vogliono trovare tutte le fermate dei bus vicini alle filiali della Banca Carim

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT ?nomefiliale ?IDfermata
WHERE{
  ?banca banca:consisteDi ?filiale .
  ?fermata banca:vicinoA ?filiale .
  ?fermata banca:id_fermata ?IDfermata .
  ?filiale banca:zona_filiale ?nomefiliale .
  FILTER(?banca = banca:CassaDiRisparmioDiRiminiCarim) .
}
ORDER BY ASC (?nomefiliale)
```

Risposta: vicino alla filiale di Miramare c'è la fermata n 682, mentre vicino alla filiale di bellariva c'è la fermata n 678

2) Si trova un assegno bancario rubato. A partire dal numero di assegno si vuole risalire al legittimo proprietario.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT ?nome ?cognome ?nomebanca ?nomefiliale
WHERE{
```

```
?assegno banca:èAppartenutoA ?conto ;
rdf:type banca:AssegnoBancario ;
banca:numero_assegno ?numbercheque .
?utente banca:haContoDiDeposito ?conto .
?conto banca:haFiliale ?filiale .
?nomebanca banca:consisteDi ?filiale .
?filiale banca:zona_filiale ?nomefiliale .
?utente banca:nome ?nome ;
                banca:cognome ?cognome .
FILTER(?numbercheque = 0001) .
}
```

Risposta: L'assegno bancario che ha numero di assegno "0001" appartiene a Raffaele Samith, della filiale di Miramare affiliata alla Banca Carim di Rimini.

3) Si vuole sapere il conteggio dei prestiti richiesti per Filiale presso la Banca di Cesena

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT ?filiale (COUNT(?prestito) AS ?numeroprestiti)
WHERE{
?banca banca:consisteDi ?filiale .
?utente banca:haPagatoPer ?prestito .
?prestito banca:haFiliale ?filiale .
FILTER (?banca = banca:CassaRisparmioCesena) .

}
```

GROUP BY ?filiale

Risultati:

Cesena Barriera: 5 prestiti richiesti
Cesena Centro : 6 prestiti richiesti

4) Si vogliono visualizzare tutti i conti correnti aperti presso la Credite Agricole con i nomi dei proprietari. Se è indicato, si vuole sapere anche il nome del consulente che segue quel conto

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
```

```
SELECT DISTINCT ?nomefiliale ?nome ?cognome ?consulente
WHERE{
```

```
?conto banca:haFiliale ?filiale .
?utente banca:haContoDiDeposito ?conto .
?conto rdf:type banca:ContoCorrente .
?utente banca:nome ?nome ;
           banca:cognome ?cognome .
?filiale banca:zona_filiale ?nomefiliale .
OPTIONAL{
?conto  banca:contoGestitoDa ?consulent .
?consulent banca:cognome ?consulente .
}

}
ORDER BY ASC (?nomefiliale)
```

Risposta

nomefiliale	nome	cognome	consulente
"Cesena Centro"	Daniela	Uranio	
"Cesena Centro"	Carlotta	Sandella	
"Miramare"	Lidia	Lorenzetti	Mendis
"Miramare"	Raffaele	Samith	
"Miramare"	Samanta	Cristoforo	Mendis
"Miramare"	Sonia	Maddi	
"Miramare"	Ricky	Martin	

5) Su tutti i prestiti presso la Credite Agricole, si vuole conoscere la percentuale dei mutui

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT (100*COUNT(DISTINCT ?mutuo)/(COUNT(DISTINCT ?prestito)) AS ?percentualeMutui)
WHERE{
{
?user banca:haContoDiPrestito ?mutuo .
?mutuo rdf:type banca:PrestitoPerMutuo .
}UNION{
?user banca:haContoDiPrestito ?prestito .
}
}
```

Risposta. Le richieste di mutuo sono il 17 percento.

6) Si vogliono visualizzare tutte le carte e i loro proprietari appartenenti alla banca di Cesena, e, se è indicata, la data di scadenza

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT ?nomecarta ?nome ?cognome ?scadenza
WHERE{
```

```

?banca banca:consisteDi ?filiale .
?conto banca:haFiliale ?filiale .
?utente banca:haContoDiDeposito ?conto .
?conto banca:haCarta ?carta .
?utente banca:nome ?nome ;
        banca:cognome ?cognome .
?carta banca:nome_carta ?nomecarta .
OPTIONAL{
?carta banca:scadenza_carta ?scadenza .
}
FILTER(?banca=banca:CassaRisparmioCesena) .

}

```

Risposta:

nomecarta	nome	cognome	scadenza
"VisaCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Lorenzo" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Leonardi" ^{^^} <http://www.w3.org/2001/XMLSchema#	
"GoldCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Carlotta" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Sandella" ^{^^} <http://www.w3.org/2001/XMLSchema#	
"TitaniumCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Daniela" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Uranio" ^{^^} <http://www.w3.org/2001/XMLSchema#	
"SilverCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Giuseppina" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Nardi" ^{^^} <http://www.w3.org/2001/XMLSchema#	"2020-01-01T00:00:00" ^{^^} <http://www.w3.org/2001/XMLSchema#
"MasterCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Carlotta" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Sandella" ^{^^} <http://www.w3.org/2001/XMLSchema#	
"VisaCard" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Daniela" ^{^^} <http://www.w3.org/2001/XMLSchema#	"Uranio" ^{^^} <http://www.w3.org/2001/XMLSchema#	

7) Si vogliono sapere i primi due conti più diffusi all'interno della Credite Agricole

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>
SELECT ?nomeconto (COUNT( ?nomeconto) AS ?frequenza)
WHERE{

?user banca:haContoDiDeposito ?conto .
?conto banca:nome_conto ?nomeconto .

}
GROUP BY (?nomeconto)
ORDER BY DESC (?frequenza)
LIMIT 2

```

Risposta. Il primo conto più diffuso è "TopItalia" di tipologia Conto Corrente. Segue il conto "CiaoItalia" che è di tipo conto a deposito fisso .

Al posto di SELECT si può usare ASK quando si vuole restituire un risultato booleano che indica se vi è stata almeno un'unificazione con successo.

Esempio:

8) Si vuole sapere se nella Cassa Di Risparmio di Cesena sono presenti depositi bimbi

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX banca: <http://www.bancacarim.com/ontologie/bancacarim.owl#>

```

```

ASK
{
?banca banca:consisteDi ?filiale .
?conto banca:haFiliale ?filiale .
?user banca:haContoDiDeposito ?conto .
?conto rdf:type banca:DepositoBambini .
FILTER(?banca= banca:CassaRisparmioCesena) .
}

```

Risposta: True

8.1 Descrizione di altre query

In questa sezione sono riportate altre query di estrazione dati, scritte in linguaggio SPARQL. Il codice di tutte le query che seguono può essere visualizzato ed eseguito direttamente dall'applicativo CrediteAgricole-App (alcuni screen sono riportati in sezione "Applicazione Finale").

Queste query sono riportate nel Controller dell'applicazione, Spring Boot permette di definire anche le relative chiamate REST API.

9) Attraverso la seguente query SPARQL dati in input la banca, la filiale, il tipo di conto, si vogliono sapere tutti i clienti di quella banca, di quella filiale, che hanno il tipo di conto richiesto.

```

//Get Customers By Account and Branch and Bank
@CrossOrigin
@RequestMapping(method = RequestMethod.POST, value="/getbybankbranchaccount")
public @ResponseBody String getByBankAndBranchAndAccount(@RequestBody BankDetails bank) {

    String query="PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n" +
        "PREFIX bank: <http://www.bancacarim.com/ontologie/bancacarim.owl#>\r\n" +
        "\r\n" +
        "\r\n" +
        "SELECT DISTINCT ?user ?firstName ?lastName\r\n" +
        "WHERE { \r\n" +
        "  ?account rdf:type bank:"+bank.accountType+".\r\n" +
        "  ?account bank:haFiliale ?branch.\r\n" +
        "  bank:"+bank.bankName+" bank:consisteDi ?branch.\r\n" +
        "  ?branch bank:zona_filiale+"+"\""+bank.branchLocation+"\"." +
        "  {\r\n" +
        "    { ?account bank:contoDiPrestitoDi ?user.}\r\n" +
        "    union\r\n" +
        "    { ?account bank:contoDiDepositoDi ?user.}\r\n" +
        "  }\r\n" +
        "  \r\n" +
        "  \r\n" +
        "  \r\n" +
        "  ?user bank:nome ?firstName;\r\n" +
        "    bank:cognome ?lastName\r\n" +
        "  \r\n" +
        "}" ;

    return OwlHelper.OwlToJson(query);
}

```

Figura 23: Query eseguita in Figura 27

10) Attraverso la seguente query si vogliono visualizzare tutti i clienti di una banca

```

//get Customers Only By Bank Name
@CrossOrigin
@RequestMapping( value="/getbybank",method = RequestMethod.POST)
public @ResponseBody String getByBankName(@RequestBody BankDetails bank) {

    String query="PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n" +
        "PREFIX bank: <http://www.bancacarim.com/ontologie/bancacarim.owl#>\r\n" +
        "\r\n" +
        "SELECT DISTINCT ?user ?firstName ?lastName\r\n" +
        "WHERE { \r\n" +
        "    bank:"+bank.bankName+" bank:consisteDi ?branch.\r\n" +
        "    ?account bank:haFiliale ?branch.\r\n" +
        "    { \r\n" +
        "        {\r\n" +
        "            ?user bank:haContoDiDeposito ?account.\r\n" +
        "        }\r\n" +
        "        union\r\n" +
        "        {\r\n" +
        "            ?user bank:haContoDiPrestito ?account.\r\n" +
        "        }\r\n" +
        "        ?user bank:nome ?firstName.\r\n" +
        "        ?user bank:cognome ?lastName.\r\n" +
        "    }\r\n" +
        "}" ;

    System.out.println(bank.bankName);

    return OwlHelper.OwlToJson(query);
}

```

Figura 24: Query eseguita in Figura 28

11) Attraverso la seguente query dato in input un utente si vogliono sapere tutte le sue informazioni, anche relative ai conti che possiede

```

@CrossOrigin
@RequestMapping(method = RequestMethod.GET, value="/getByCustomerId/{id}")
public @ResponseBody String getByCustomerId(@PathVariable String id) {

    String query="PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\r\n" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\r\n" +
        "PREFIX bank: <http://www.bancacarim.com/ontologie/bancacarim.owl#>\r\n" +
        "\r\n" +
        "\r\n" +
        "SELECT ?firstName ?lastName ?bankName ?branchCity ?accountName ?accountNo ?accountBalance \r\n" +
        "WHERE { \r\n" +
        "\r\n" +
        " ?customer bank:id_persona+"\""+id+"\".\r\n" +
        " ?customer bank:nome ?firstName;\r\n" +
        "          bank:cognome ?lastName\r\n" +
        " {\r\n" +
        " { ?account bank:contoDiPrestitoDi ?customer.}\r\n" +
        " union\r\n" +
        " { ?account bank:contoDiDepositoDi ?customer.}\r\n" +
        " }\r\n" +
        " \r\n" +
        " ?account bank:numero_conto ?accountNo;\r\n" +
        "          bank:saldo_conto ?accountBalance;\r\n" +
        "          bank:nome_conto ?accountName;\r\n" +
        "          bank:haFiliale ?branchName.\r\n" +
        " ?branchName bank:zona_filiale ?branchCity.\r\n" +
        " ?bank bank:consisteDi ?branchName;\r\n" +
        "       bank:nome_banca ?bankName.\r\n" +
        " }\r\n" +
        "};

    return OwlHelper.OwlToJson(query);
    System.out.println(id);
    return id;
}

```

Figura 25: Query eseguita in Figura 31

Altre query eseguite, che per similarità non riporto, sono:

- trovare tutti i clienti di una banca e di una specifica filiale
- trovare tutti i clienti della Credite Agricole che hanno un determinato tipo di conto
- trovare tutti i clienti di una banca che hanno un determinato tipo di conto

9 Applicazione finale

L'implementazione dell'applicazione ha avuto le seguenti fasi:

- Back end implementato con Spring Boot .
- Inserimento di Apache Jena per eseguire le query con SPARQL e recuperare i dati delle ontologie .
- Front end implementato con Angular .

Per inserire facilmente le dipendenze di Apache Jena all'interno del progetto è stato inserito il tool Apache Maven. Dichiarando così la dipendenza core di Jena dentro il file pom.xml si aggiungono automaticamente tutto l'insieme di librerie di cui Jena ha bisogno.

Con Apache Jena da una stringa viene creata una query usando *QueryFactory*. La query e il modello vengono poi passati a una *QueryExecutionFactory* per produrre un'istanza dell'esecuzione della query. I risultati vengono gestiti in un try catch e infine l'esecuzione della query viene terminata.

```

1 public static String OwlToJson(String sparqlQuery ) {
2
3     String r = null;
4
5     FileManager.get().addLocatorClassLoader(OwlHelper.class.getClassLoader());
6     Model model=FileManager.get().loadModel("inserisci percorso rdf file");
7
8     String queryString = sparqlQuery;
9
10    Query query=QueryFactory.create(queryString);
11    QueryExecution quexec=QueryExecutionFactory.create(query,model);
12    try {
13        ResultSet results=quexec.execSelect();
14
15        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
16
17        ResultSetFormatter.outputAsJSON(outputStream, results);
18
19        String json = new String(outputStream.toByteArray());
20
21        r=json;
22    }catch(Exception e) {
23        System.out.println(e);
24    }
25
26    finally {
27
28        quexec.close();
29
30    }
31    return r;
32 }
33
34 }

```



Figura 26: Home Page sito Credite Agricole

Cerca dall'ontologia

Seleziona la banca

Banca Carim Rimini

Seleziona la zona della filiale

Miramare

Seleziona il tipo di conto

Deposito Bancomat

Deposito Generico

Deposito Giovani

Deposito Senior

Conto Corrente

Cerca

Risultati dall'ontologia

Link dell' Utente	Nome	Cognome
http://www.bancacarim.com/ontologie/bancacarim.ow#GenericoRipartitore_01	Raffaele	Semith
http://www.bancacarim.com/ontologie/bancacarim.ow#Senior_Citizen_01	Sonia	Maddi
http://www.bancacarim.com/ontologie/bancacarim.ow#GenericoRipartitore_02	Samantha	Cristoforo
http://www.bancacarim.com/ontologie/bancacarim.ow#Giovane_02	Ricky	Martin
http://www.bancacarim.com/ontologie/bancacarim.ow#Senior_Citizen_02	Lidia	Lorenzetti

Figura 27: Sezione Esplora Clienti: Esempio di ricerca di utenti selezionando la banca, la filiale e il tipo di conto -> si vogliono sapere tutti i clienti della banca Carim e della filiale di Miramare che hanno un Conto Corrente

Cerca dall'ontologia

Seleziona la banca
 Cassa di Risparmio Cesena

Seleziona la zona della filiale

Seleziona il tipo di conto
 Deposito Bambini
 Deposito Generico
 Deposito Giovani
 Deposito Senior

Cerca

Risultati dall'ontologia

Link dell' Utente	Nome	Cognome
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_09	Filo	Ripa
http://www.bancacarim.com/ontologie/bancacarim.owl#GenericoRisparmiatore_03	Deniele	Uranio
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_07	Odesa	Maresi
http://www.bancacarim.com/ontologie/bancacarim.owl#Giovane_03	Carlotta	Sandella
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_08	Azzurra	Nina
http://www.bancacarim.com/ontologie/bancacarim.owl#Senior_Citizen_03	Giuseppina	Nardi
http://www.bancacarim.com/ontologie/bancacarim.owl#Bambino_03	Nicolino	Cioccolini
http://www.bancacarim.com/ontologie/bancacarim.owl#Senior_Citizen_04	Lorenzo	Leonardi
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_10	Lorella	Senti
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_11	Deniele	Lolli
http://www.bancacarim.com/ontologie/bancacarim.owl#Bambino_04	Tsunai	Perera

Figura 28: Sezione Esplora Clienti: Esempio di ricerca di utenti selezionando solo la banca -> Si vogliono sapere tutti i clienti della banca Carim

Seleziona la banca
 Banca Carim Rimini

Seleziona la zona della filiale

Seleziona il tipo di conto
 Prestito per la casa
 Prestito per mutuo
 Prestito Personale
 Prestito per Sviluppo

Cerca

Link dell' Utente	Nome	Cognome
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_02	Nipun	Generico
http://www.bancacarim.com/ontologie/bancacarim.owl#ClientePerPrestito_08	Roberto	Neri

Figura 29: Sezione Esplora clienti: Esempio ricerca di utenti selezionando solo la banca e il tipo di conto-> Si vogliono sapere i clienti della Banca Carim che hanno richiesto un prestito per mutuo

Seleziona la banca
 Cassa di Risparmio Cesena

Seleziona la zona della filiale
 Cesena Centro

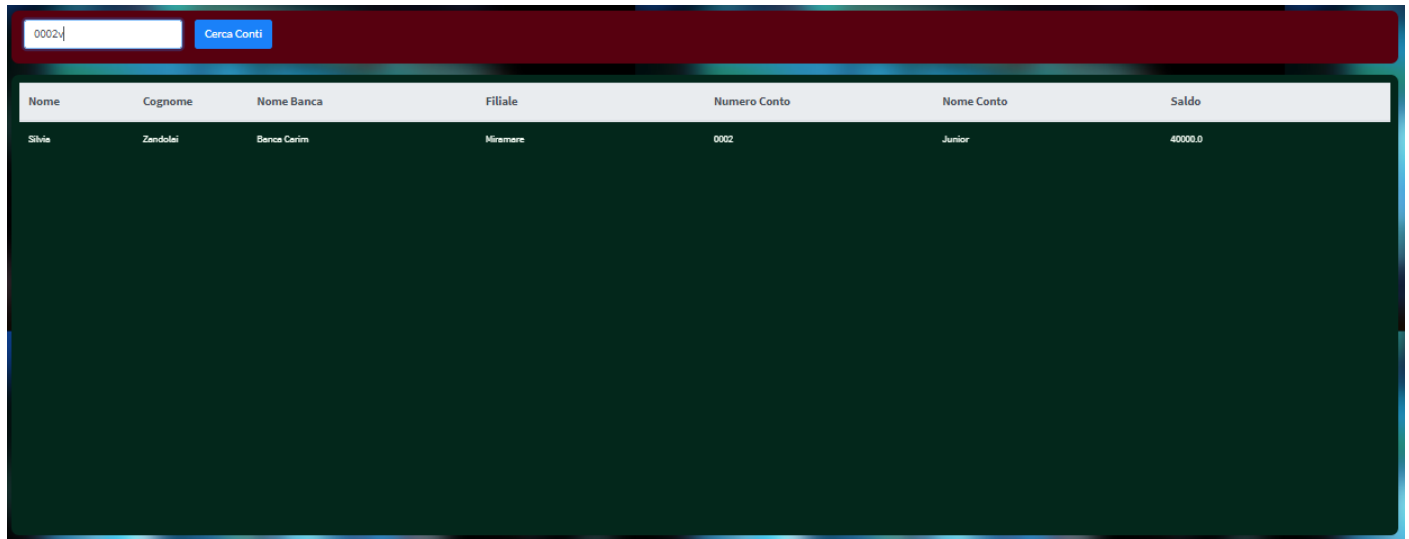
Seleziona il tipo di conto
 Deposito Bambini
 Deposito Generico
 Deposito Giovani
 Deposito Senior

Cerca

Link dell' Utente	Nome	Cognome
http://www.bancacarim.com/ontologie/bancacarim.owl#Giovane_03	Carlotta	Sandella

Figura 30: Si vogliono sapere tutti i clienti della Cassa di Risparmio di Cesena, della filiale Cesena Centro, che hanno un deposito Giovani

In Esplora Clienti si possono fare altre interrogazioni, Si veda la fine del paragrafo 8.1.



Nome	Cognome	Nome Banca	Filiale	Numero Conto	Nome Conto	Saldo
Silvia	Zandolai	Banca Carim	Miramare	0002	Junior	40000.0

Figura 31: Sezione Dettaglio Conti clienti: Ricerca tramite Id utente

9.1 Modalità di esecuzione

1) Front end: Mettersi sul giusto percorso e eseguire da terminale "npm start". Mettersi poi in ascolto sulla porta 4200.

2) Back end: Mettersi sul giusto percorso e eseguire da terminale "mvn spring-boot:run". Mettersi poi in ascolto sulla porta 8080.

Ora si può usare l'applicazione.

10 Sviluppi Futuri

Il modello implementato è una versione semplificata di una banca commerciale, pertanto un naturale sviluppo del progetto potrebbe essere quello di ampliare e estendere l'ontologia.

Si potrebbero anche esplorare altri aspetti oltre a quello puramente commerciale.

Per esempio, una banca potrebbe anche avere bisogno di un'ontologia finalizzata alla descrizione del contenuto dei documenti storici bancari e di parte dell'attività svolta dalla banca, soprattutto verso soggetti terzi (finanziamenti, beneficenza, sequestri e confische...), che comporta l'istruzione di pratiche o la produzione di documenti.

Si potrebbe quindi procedere alla creazione di questa nuova ontologia e integrarla insieme a quella già realizzata oppure estendere quella esistente.

Il punto focale del modello descrittivo sarebbe la banca - soggetto che istruisce delle pratiche di diversa natura - e il rapporto con la documentazione realizzata e le informazioni in essa contenute. L'ontologia così si baserebbe quindi anche sui dati raccolti dai documenti d'archivio che descrivono vari processi e attività eseguiti dagli istituti bancari: punto di partenza per la sua costruzione potrebbero essere gli inventari e le banche dati della documentazione conservata dall'archivio, prodotta dalle diverse banche che nel tempo si sono via via fuse in Credite Agricole. L'ontologia avrebbe l'obiettivo di fornire una rappresentazione e un modello sufficientemente astratto per la descrizione del processo di diverse attività bancarie da cui deriva la produzione di documentazione - dalla richiesta di finanziamento da parte di un'impresa e il suo esito, alla disposizione di pratiche di sequestro, confisca e restituzione di beni, fino alle erogazioni di beneficenza, per citare qualche esempio.

Con questo lavoro si porrebbe un primo fondamentale tassello per la descrizione del dominio sulle pratiche archivistiche bancarie nella loro complessità, varietà e interezza, e sui dati in esse contenuti.

Bibliografia principale

- [Ang] Angular. *Angular tutorial*. <https://angular.io/docs>.
- [Bae] Baeldung. *Spring Boot tutorial*. <https://www.baeldung.com/spring-boot-start>.
- [Gro] Grosov. *SWRL rules*. <https://coherentknowledge.com/wp-content/uploads/2013/05/talk-prelim-aaail3-rules-tutorial.pdf>.
- [Jen] Apache Jena. *Apache Jena Tutorial*. <https://jena.apache.org/documentation/index.html>.
- [Pap] Fabio Papacchini. *Protège tutorial*. https://cgi.csc.liv.ac.uk/~frank/teaching/comp08/protege_tutorial.pdf.