UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

SPRINGLES 2.0

Un sistema di ragionamento a regole su dati semantici

Supervisore
Prof. Kuper Gabriel Mark

Laureando
Joppi Christian

Co-Supervisore
Dott. Bozzato Loris

Anno accademico 2015/2016

Indice

1 Sommario

Con lo sviluppo delle tecnologie del Semantic Web ed il loro utilizzo in diversi ambiti applicativi, emerge la richiesta di un supporto per ragionamenti complessi in presenza di grandi quantità di dati semantici. Spesso questi dati sono estratti automaticamente da sorgenti diverse (testi, file excel, basi di dati, ...), possono contenere errori, essere incompleti, incongruenti e contraddittori. Normalmente questi dati descrivono entità (persone, organizzazioni, luoghi geografici, oggetti, ...) ed eventi (effettivamente successi nel passato, o previsti per il futuro). Partendo da questi dati si possono trarre delle conclusioni tramite metodi di ragionamento automatico. Proprio in questa direzione, lo scopo del lavoro presentato è l'estensione e il miglioramento di una piattaforma per il ragionamento a regole basato su SPARQL, SPRINGLES (SParql-based Rule Inference over Named Graphs Layer Extending Sesame), correntemente sviluppata dall'unità DKM, nelle seguenti direzioni:

- Generalizzazione di SPRINGLES per più motori di inferenza e introduzione di un nuovo motore di inferenza basato su RDFPro[1];
- Realizzazione di un API REST per applicazioni esterne;
- Sviluppo di una nuova interfaccia utente.

Nei prossimi capitoli presenteremo inizialmente gli strumenti e i linguaggi utilizzati (capitolo 2). Nel capitolo 3 descriveremo nel dettaglio il sistema SPRINGLES e tutti i miglioramenti apportati su di esso.

[1] http://rdfpro.fbk.eu

2 Linguaggi e strumenti utilizzati

In questo capitolo analizzaremo i principali strumenti e linguaggi utilizzati nella realizzazione del sistema. Cominceremo presentando il Web Semantico, la sua struttura e i linguaggi ad esso associati. Parleremo in seguito degli strumenti utilizzati per la realizzazione del sistema, Sesame e RDFPro[1].

2.1 Linguaggi

2.1.1 Il Web Semantico

Con il termine *Web Semantico* (termine coniato dal suo ideatore, Tim Berners-Lee[2]), si intende l'associazione del World Wide Web composto di documenti pubblicati (pagine HTML, file, immagini, e così via) con informazioni e dati (metadati) che ne specificano il significato in un formato adatto all'interrogazione e l'interpretazione (es. tramite linguaggi di query) e, più in generale, all'elaborazione automatica.

I contenuti Web al giorno d'oggi sono fatti per essere compresi dall'uomo che riesce a navigare nel Web anche grazie all'esperienza e alla capacità di ragionamento basato sulla capacità comunicativa che hanno parole e immagini. Il Software è in grado di analizzare soltanto le strutture (testuali) delle risorse Web (titoli, collegamenti, immagini, ecc...) ma non di associargli una semantica specifica e quindi un'interpretazione attraverso sistemi automatici. Il Web Sematico aggiunge alle risorse un'insieme di metadati che forniscono la semantica e le relazioni tra di esse, permettendo alle macchine di comprendere tali informazioni.

Il Web Semantico è definito sulla base di una struttura a più strati chiamata *Layer Cake*[1], mostrata in Figura 2.1.

Al primo livello troviamo lo standard URI (*Uniform Resource Identifier*), utilizzato per la definizione degli identificativi delle risorse Web in maniera univoca.

Al secondo livello troviamo XML (*eXtensible Markup Language*), utile per definire il modello di uno scenario applicativo. Questo meta-linguaggio però non riesce a definire in autonomamente la struttura e l'interscambio di informazioni tra le risorse. Per questo è stato creato il linguaggio RDF (*Resource Description Framework*) e RDFS (RDF Schema) adatti a descrivere le risorse e i loro tipi.

Al terzo livello troviamo il livello ontologico (OWL), utilizzo per definire formalmente le descrizioni delle risorse, quali possono essere vincoli di cardinalità o restrizioni di altro tipo.

Al livello superiore troviamo il livello logico, che coinvolge i meccanismi di inferenza per la derivazione di conoscenza implicita.

[1] http://www.w3.org/2001/09/06-ecdl/slide17-0.html

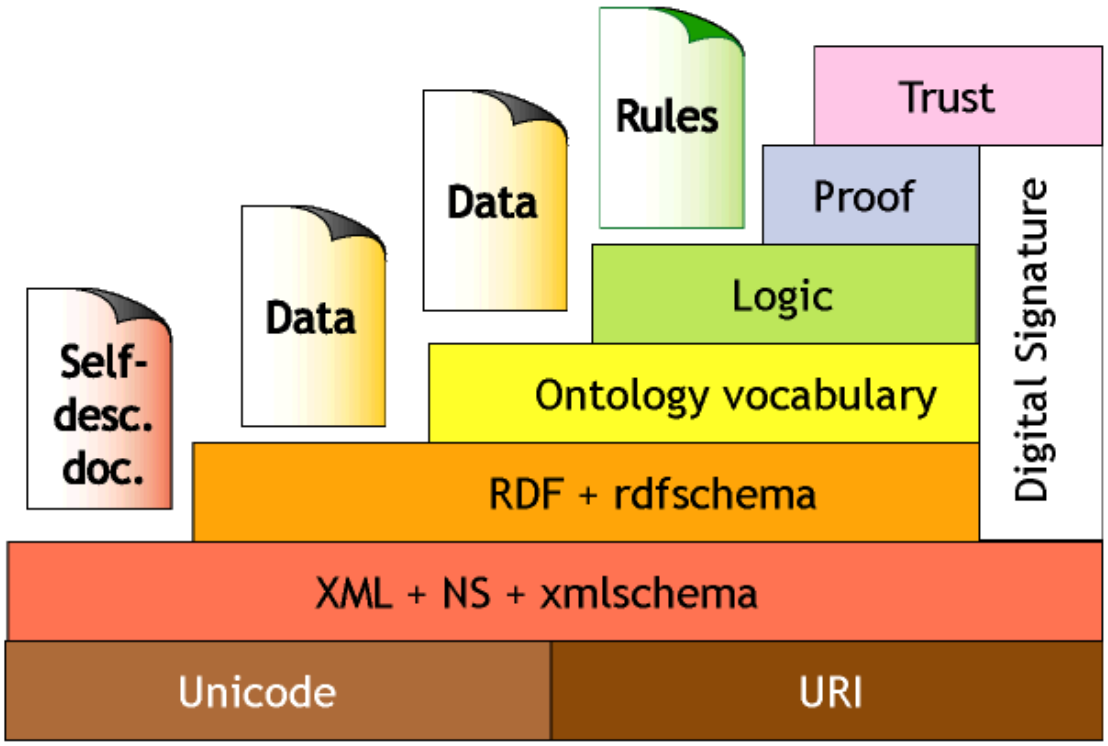

Figura 2.1: Layer Cake del Web Semantico

2.1.2 URI

URI, *Uniform Resource Identifier*[3], è un sistema definito per identificare univocamente qualsiasi risorsa Web. Qualsiasi cosa abbia un URI può essere identificato come oggetto e considerato appartenente al Web. Un esempio possono essere gli URL (Uniform Resource Locator[2]) che identificano in maniera univoca le pagine Web. A differenza da essi però gli URI non sono controllati, possono essere creati da chiunque. Quindi, possiamo definire un URI come il nome di una risorsa e non come il percorso nel Web dove essa si trova.
Esempio: `http://www.example.org/bob#me` identifica l'oggetto BOB.

2.1.3 RDF

RDF, *Resource Description Framework*[4] è un framework definito da W3C come insieme di linguaggi dichiarativi (basati su sintassi XML) per descrivere la struttura di una risorsa Web. Per definire una qualsiasi risorsa (identificata attraverso URI) ci basiamo su un insieme di triple costituite da soggetto, predicato e oggetto.

- Soggetto: identifica la risorsa che deve essere modellata.
- Predicato: identifica la proprietà relativa alla risorsa definita nel soggetto.
- Oggetto: identifica il valore della proprietà relativa alla risorsa.

Ognuna di questi triple può essere a sua volta utilizzata come oggetto di altre asserzioni. Per esempio vogliamo modellare le seguenti asserzioni riguardanti Bob:

- Bob è una persona
- Bob è amico di Alice

[2] `https://url.spec.whatwg.org/`

- Bob possiede un PC

Il soggetto è Bob, i predicati sono *è una*, *è amico di*, *possiede un* mentre gli oggetti sono *persona*, *Alice*, *PC*.

Ad ognuno degli elementi della tripla possono essere assegnati degli identificativi univoci relativi a risorse già definite e modellate. Ad esempio:

Bob (soggetto) a `http://www.example.org/bob#me`.

Possedere (predicato) a `https://it.wiktionary.org/wiki/possedere`

PC (oggetto) a `http://www.buypc.com/products/004320`

Tutte queste triple possono essere identificate attraverso un grafo connesso come mostrato in figura 2.2.

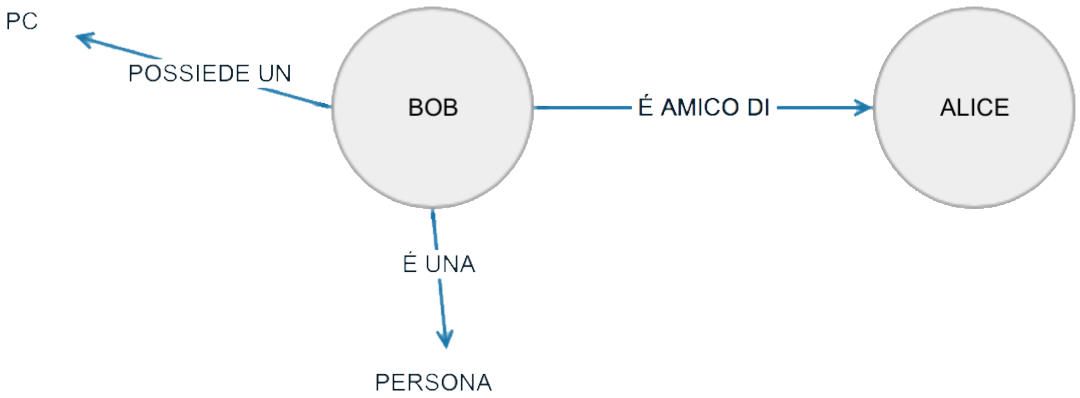

Figura 2.2: Grafo relativo a BOB

Esistono vari formati per serializzare un grafo RDF:

- RDF/XML: basato sulla sintassi XML;

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3. org/1999/02/22-rdf-syntax-ns#"
xmlns:wikidizionario="http://it.wiktionary.org/wiki/">

<rdf:Description
     rdf:about="http://www.example.org/bob#me">
    <wikidizionario:produrre rdf:resource="http://www.buypc.com/products/004320" />
</rdf:Description>
</rdf:RDF>
```

- N-Triples: basato su triple di URI per soggetto-predicato-oggetto;

```
<http://www.example.org/bob#me>
<http://it.wiktionary.org/wiki/possedere>
<http://www.buypc.com/products/004320>
```

- Turtle: è un'estensione di N-Triples con in aggiunta l'utilizzo di prefissi per rendere più facile la scrittura, il parsing e la lettura degli statements;

```
BASE <http://www.example.org/>
PREFIX pc: <http://www.buypc.com/products#>
PREFIX wiki: <http://it.wiktionary.org/wiki#>

<bob#me> wiki:possedere pc:004320
```

- Trig: utilizza la sintassi di Turtle. La differenza è che Turtle supporta le specifiche di soli grafi mentre il formato Trig supporta le specifiche di grafi multipli.

2.1.4 RDF Schema

RDFS, o meglio *RDF Schema*[5], è un estensione di RDF che permette di definire un vocabolario per le proprietà degli oggetti in un dominio. RDF Schema permette di definire significato, caratteristiche e relazioni di un insieme di proprietà, compresi vincoli sul dominio e sui valori delle singole proprietà. Attraverso la proprietà transitiva delle classi e sottoclassi possiamo anche definire un gerarchia su di esse.

Forniamo un esempio (Figura 2.3) considerando una risorsa legata a tutti i PC in vendita (`http://www.buypc.com/products/`).

Definiamo uno schema per l'intero dominio, specificando prima di tutto i prefissi per RDF, RDFS e il contesto globale `http://www.buypc.com/products/` per poi definire classi, proprietà varie e vincoli.

```
<rdf:RDF
xmlns:rdf="http://www.w3. org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.buypc.com/products#">

<rdfs:Class rdf:ID="PC" />

<rdfs:Class rdf:ID="PersonalComputer">
    <rdfs:subClassOf rdf:resource="#PC"/>
</rdfs:Class>

<rdfs:Property ref:ID="Brand">
    <rdfs:domain rdf:resource="#PC" />
    <rdfs:range rdf:resource="#Literal"/>
</rdfs:Property>

</rdf:RDF>
```

Figura 2.3: Esempio RDF Schema

La parte che precede il carattere # indica il nome dello schema al quale è riferito un elemento, mentre la parte successiva indica il nome dello stesso elemento.

In RDFS esistono 3 tipi di oggetti e relazioni:

- **Classi**: ogni risorsa descritta appartiene alla classe *Resource* le cui sottoclassi sono *Literal* (che indicano valori, stringhe o caratteri), *Property* (che definiscono le proprietà) e *Class* (definiscono classi di risorse).

  < *rdfs:Class rdf:ID="PC"*/> è un esempio di risorsa di tipo Class (Figura 2.3).

- **Proprietà**:
  - *rdf:type* : definisce la classe al quale la risorsa appartiene;
  - *rdfs:subClassOf* : indica la relazione tra classe e sottoclasse;
  - *rdfs:subPropertyOf* : indica che una proprietà è specializzazione di un'altra;
  - *rdfs:seeAlso* : indica che una risorsa è anche descritta in altre risorse;

– *rdfs:isDefinedBy* : indica la risorsa soggetto, chi ha definito l'asserzione.

Esempio : < *rdfs:subClassOf rdf:resource="#PC"* />

- **Vincoli**:
  – *rdfs:range* : indica le classi con la quale potrà essere fatta un'asserzione con la proprietà;
  – *rdfs:domain* : definisce a quale classe può essere applicata la proprietà.

  Esempio:

  < *rdfs:domain rdf:resource="#PC"* /> significa che la proprietà *Brand* appartiene alla risorsa PC;

  < *rdfs:range rdf:resource="#Literal"* /> significa che la proprietà *Brand* è di tipo *Literal*.

2.1.5 OWL

OWL, Ontology Web Language[6], è un insieme di linguaggi utilizzati per la creazione di basi di conoscenza. Questi linguaggi sono basati su RDF e XML. I dati ottenuti da queste ontologie sono visti come insiemi di "individui" relazionati uno con l'altro attraverso asserzioni di proprietà. Ogni ontologia è fondata su una serie di assiomi che definiscono dei vincoli su classi di individui, limitandone anche il tipo di relazioni che possono esistere tra di esse. Le ontologie possono a loro volta importare altre ontologie estendone così le informazioni. OWL estende RDF Schema portando nuovi costrutti che permettono di descrivere entità, imporre cardinalità, specificare che due classi sono disgiunte o che due proprietà sono simmetriche. Attraverso OWL i sistemi sono così in grado di dedurre nuove informazioni su dati forniti in maniera esplicita.

Nel frammento di codice sottostante descriviamo un vincolo sul massimo numero di porte USB3.0 che un personal computer può avere.

```
<owl:Class rdf:ID="PersonalComputer">
    <owl:Restriction>
        <owl:onProperty rdf:resource="#NumeroUSB3" />
            <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
                4
            </owl:maxCardinality>
    </owl:Restriction>
</owl:Class>
```

Secondo il primo standard W3C OWL[3], esistono 3 varianti di OWL con diversi livelli di espressività.

- *OWL Lite* : Fornisce all'utente le definizioni più semplici. É utilizzato per definire una gerarchia di classi e imporre vincoli molto semplici come ad esempio la cardinalità su entità che può essere solo "1" o "0". É semplice da implementare ma non è caratterizzato da un forte valore di espressività..
- *OWL DL* : Progettato per gli utenti che necessitano della massima espressività possibile conservando la completezza computazionale (ogni conclusione può essere computata), decidibile (tutte le computazioni terminano in tempo finito) e ha procedure di deduzione con complessità ben conosciute. OWL DL comprende tutti i costrutti del linguaggio OWL, che però possono essere utilizzati soltanto con determinate restrizioni, ad esempio non possiamo impostare restrizioni sul numero su proprietà che vengono dichiarate transitive.

[3] http://www.w3.org/TR/owl-features/

- *OWL Full* : Formulato per fornire agli utenti la massima espressività e la libertà sintattica di RDF senza conservare nessuna garanzia computazionale. In OWL Full una classe può essere trattata come un insieme di individui o un singolo individuo allo stesso tempo. Contro OWL Full nessun software è in grado di svolgere un ragionamento completo.

Nel 2009 viene introdotto OWL2[4] che estende OWL in ogni sua parte. Inoltre questa nuova versione implemente nuove funzioni: alcune come ad esempio l'unione disgiunta di classi, mentre altre offrono maggiore espressività, tra cui chiavi, catene di proprietà, tipi ed intervalli di dati più ricchi, restrizioni di cardinalità qualificate, proprietà assimmetriche, riflessive, disgiuntive e nuove funzionalità di annotazione.

2.1.6 SPARQL(1.1)

SPARQL (SPARQL Protocol and RDF Query Language)[7] è un linguaggio di interrogazione per dati RDF. Si basa sul pattern RDF (soggetto, predicato, oggetto) chiamato anche *Basic Graph Pattern*.

Ad esempio, presentiamo una query di tipo select che interroga i dati che rispondono alla domanda *"Quali sono tutti i PersonalComputer con 2 porte USB 3.0?"*.

```
PREFIX prod:<www.buypc.com/products#>
SELECT ?s WHERE {
    ?s rdf:type prod:PersonalComputer
    ?s prod:NumeroUSB3 2
}
```

La query restituirà una colonna con tutti i prodotti che soddisfano le condizioni, cioè che i termini inseriti nella parte WHERE della query possono essere associati ai termini presenti nello store.

In SPARQL esistono due tipi di interrogazioni: Query (vista sopra) che interrogano i dati e Update che modificano le triple dei dati. Le Query si possono a loro volta suddividere in due sottotipi: *Select* e *Ask*. Le *Select* servono per ottenere delle triple che soddisfano le richieste dell'utente, mentre le *Ask* per verificare la veridicità di un'asserzione e quindi ottenendo un risultato che può essere *true* o *false*. In Figura 2.4 mostriamo anche un esempio di Ask, nella quale viene interrogato lo store verificando l'esistenza di un Personal Computer con 2 porte USB3.0.

```
PREFIX prod:<www.buypc.com/products#>
ASK{
    ?s rdf:type prod:PersonalComputer;
       prod:NumeroUSB3 2
}
```

Figura 2.4: Esempio di interrogazione ASK

Per le interrogazioni di Update, cioè funzioni di inserimento e cancellazione, troviamo la *INSERT DATA* per inserire triple all'interno dello store e la *DELETE* per cancellarle. Vediamo ora un esempio di inserimento di dati

[4]http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/

```
PREFIX prod:<www.buypc.com/products#>
    INSERT DATA {
        prod:asus rdf:type prod:PersonalComputer.
        prod:asus prod:NumeroUSB3 1.
}
```

e un esempio di cancellazione

```
PREFIX prod:<www.buypc.com/products#>
   DELETE DATA{
        prod:asus rdf:type prod:PersonalComputer.
        prod:asus prod:NumeroUSB3 1.
}
```

2.2 Strumenti

2.2.1 Sesame

Sesame è un framework open source per l'interrogazione, l'analisi e il ragionamento logico di dati RDF. É stato sviluppato da Aduna a partire dal 1999. Esso implementa un database scalabile per la memorizzazione di dati e due servlet distinte che vengono utilizzate per gestire e fornire l'accesso al triplestore. In Figura 2.5 possiamo trovare una di queste servlet, chiamata Sesame Workbench, utilizzata per l'interfacciamento con l'utente. Sesame fornisce anche un API per l'input, output e parsing di dati RDF chiamata RIO (RDF Input/Output). Tutti i dati RDF vengono salvati in uno o più repository, archivi utilizzati per la memorizzazione di grandi quantità di metadati. Essi permettono una rapida ricerca grazie alla presenza di tabelle relazionali. Con i repository è possibile interfacciarsi attraverso specifiche API[5]. É possibile creare (fornendo identificativo e descrizione), modificare e cancellare repository. La memorizzazione di questi è possibile sia su File System, fornendo il percorso desiderato, che su server remoto, accessibile attraverso protocollo HTTP. Sesame supporta due linguaggi per le query: SPARQL1.1 e SeRQL. Durante gli anni Sesame è sempre stato in continua evoluzione con il rilascio di versioni sempre più innovative. Da Aprile 2016 è disponibile la versione 4.1.2.

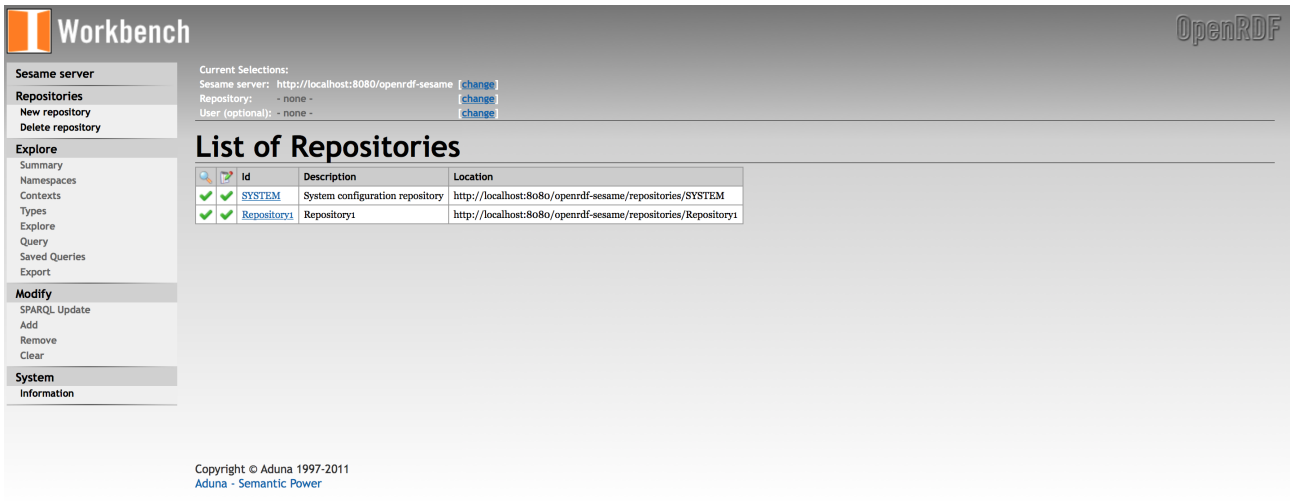

Figura 2.5: Sesame Workbench

[5] http://www.csee.umbc.edu/courses/graduate/691/spring14/01/examples/sesame/openrdf-sesame-2.6.10/docs/users/ch08.html

2.2.2 RDFPro

RDFPro (RDF Processor)[6], è un tool a riga di comando e libreria Java per l'elaborazione di dati RDF. É stato sviluppato presso l'unità di ricerca DKM della Fondazione Bruno Kessler basandosi su Java8 e Sesame3. É nato dalla necessitá di avere uno strumento che fornisse varie funzionalità per la manipolazione di linked data fino a pochi miliardi di triple. Supporta vari formati tra cui: rdf, rj, jsonld, nt, nq, trix, trig, tql, ttl, n3 e brf. Fornisce un tool a riga di comando, un'interfaccia Web[7] e librerie Java[8]. RDFPro offre varie funzionalità per la manipolazione di dati RDF:

- **@read**: legge più file RDF in parallelo, emettendo uno stream di dati;
- **@write**: scrive i dati su uno o più file RDF per permettere la suddivisione di dataset di grandi dimensioni;
- **@download**: crea un flusso di dati in input ricavati da query SPARQL;
- **@upload**: carica i dati in input in un triple store attraverso SPARQL Update;
- **@transform**: trasforma i dati tripla per tripla attraverso espressioni *Groovy*[9];
- **@smush**: effettua lo *smushing*, l'unione di più risorse con URI diversi che identificano la stessa entità, in 2 passaggi: estrae i grafi *owl:sameAs* e poi ne sostituisce gli URI;
- **@infer**: effettua la chisura RDFS sui dati in input;
- **@tbox**: filtra i dati in input sulla base di assiomi RDFS e OWL;
- **@stats**: fornisce statistiche sui dati in input;
- **@unique**: elimina i duplicati sui dati in input;

Esempi:

Trasformo le triple rdfs:label e rimuovo i duplicati
@transform groovy #file1 **@unique**

Carico tutte le triple del file nel triple store
@upload #file1

Scrivo tutte le triple su uno più file RDF
@write #file1

[6] `http://rdfpro.fbk.eu`
[7] `https://knowledgestore2.fbk.eu/rdfpro-demo/`
[8] `http://rdfpro.fbk.eu/rdfprolib.html`
[9] Groovy è un linguaggio per script che utilizza sintassi e librerie Java: `http://groovy.codehaus.org/`

3 SPRINGLES: descrizione del sistema e aggiornamento

In questo capitolo descriveremo in dettaglio il sistema SPRINGLES, il suo funzionamento partendo dalla sua struttura e i suoi limiti iniziali. Successivamente spiegheremo come esso è stato modificato e i dettagli della sua nuova struttura.

3.1 SPRINGLES 1.0: Stato del Sistema

SPRINGLES[8], acronimo di *SParql-based Rule Inference over Named Graphs Layer Extending Sesame* è un sistema sviluppato inizialmente dall'unità di ricerca Data Knowledge and Management dell'FBK[1] di Trento. Esso applica ai dati RDF un insieme di regole (ruleset) codificate come SPARQL Query, ottenendo dei nuovi dati sotto forma di statements che vengono poi salvati nello store.

La prima versione del sistema SPRINGLES (1.0), come si nota in Figura 3.1 è formato da tre componenti:

- SPRINGLES Server e Core: la componente che si occupa di inferire i dati RDF semantici;
- TripleStore (SESAME): l'insieme di repository nel quale vengono inserite tutte le triple RDF;
- SPRINGLES Client e Interfaccia WEB: la componenete grafica del sistema con la quale l'utente può manipolare i dati e l'interfaccia con le componenti.

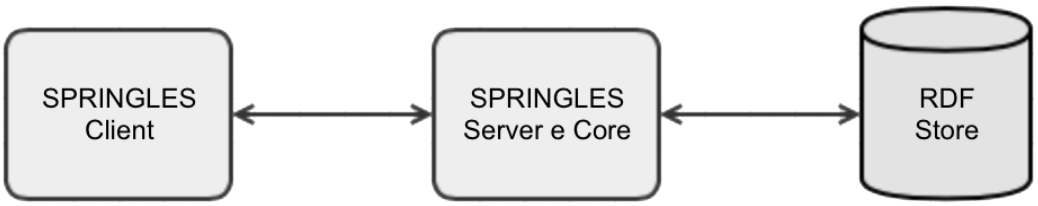

Figura 3.1: Struttura SPRINGLES 1.0

Analizziamo nel seguito ogni componente nel dettaglio.

[1] https://dkm.fbk.eu/

3.1.1 SPRINGLES Server e Core

SPRINGLES si può suddividere in due parti ben distinte:

- SPRINGLES Core: si occupa di aggiungere/rimuovere dati, effettuare la chiusura, interrogare i dati attraverso query SPARQL1.1, gestire le transazioni;
- SPRINGLES Server: si occupa di mettere in collegamento l'esterno (Client) con il core di SPRINGLES.

Regole e strategie di valutazione Una regola, di cui un esempio è riportato in Figura 3.2, è composta principalmente di due parti:

- *spr:head*: la tripla che viene inserita negli statements inferiti nello store;
- *spr:body*: la condizione che deve essere soddisfatta perché la tripla sopra riportata venga inserita.

Analizzando la regola riportata nell'esempio, possiamo dire che l'head, cioè *?x rdf:type ?z*, viene considerato solo se è valida la condizione riportata nel body della regola: se y è sottoclasse di z nel grafo *m1* e se esiste un x che è un y nel grafo *m2*. Inoltre viene controllato che la tripla non sia già stata inserita.

```
:pel-c-subc a spr:Rule ;
  spr:head """ GRAPH ?mx { ?x rdf:type ?z } """ ;
  spr:body """ GRAPH ?m1 { ?y rdfs:subClassOf ?z }
               GRAPH ?m2 { ?x rdf:type ?y }
               GRAPH sys:dep { ?mx sys:derivedFrom ?m1,?m2 }
               FILTER NOT EXISTS {
                 GRAPH ?m0 { ?x rdf:type ?z }
                 GRAPH sys:dep { ?mx sys:derivedFrom ?m0 }
               } """ .
```

Figura 3.2: Esempio di regola

Per valutare le regole logiche un motore di inferenza può seguire varie strategie:

- *Parallel rule evaluation* : le regole vengono valutate contemporaneamente in meniera parallela. Ciò implica indipendenza tra di esse;
- *Sequence* : le regole vengono valutate in sequenza. Alcune possono dipendere da altre e l'ordine può essere fondamentale;
- *Fixpoint* : la valutazione delle regole viene ripetuta fino a quando esse vengono applicate su tutti i dati presenti.

In alcuni casi i ruleset contengono delle variabili al quale deve essere associate un valore. Di questo si occupano i bindings, coppie di *variabile,termineRDF* che ne definiscono l'assegnamento.

Processo di Inferenza Come mostrato in figura 3.3, quando viene richiesta una chiusura (valutazione delle regole), il sistema controlla che il repository sul quale essa è richiesta sia in stato POSSIBLY INCOMPLETE o STALE. Per POSSIBLY INCOMPLETE intendiamo che dopo l'ultima inferenza sui dati il contenuto dello store è stato modificato, e quindi potrebbero esserci nuove inferenze da generare sui dati aggiornati; per STALE intendiamo invece che per qualche motivo un processo di inferenza sui dati non è mai stato completato. In questo caso viene quindi richiesta a SPRINGLES la chiusura, inviandogli un comando SPARQL nella forma *clear graph <springles:update-closure>*. Una volta che SPRINGLES riceve il comando, recupera dal file di configurazione (springles.ttl) i parametri del

repository necessari, tra cui il motore di inferenza scelto, il ruleset utilizzato e l'inference-prefix. Per inference-prefix intendiamo il prefisso dei grafi nel quale tutte le triple generate verranno inserite. A questo punto viene richiesta una nuova sessione al motore di inferenze. Dopo aver terminato l'inferenza sui dati, tutti gli statements generati vengono salvati nel repository, impostando come prefisso del contesto di ognuno di essi l'inference-prefix nominato, per fare in modo che possano essere identificati in caso di cancellazione.

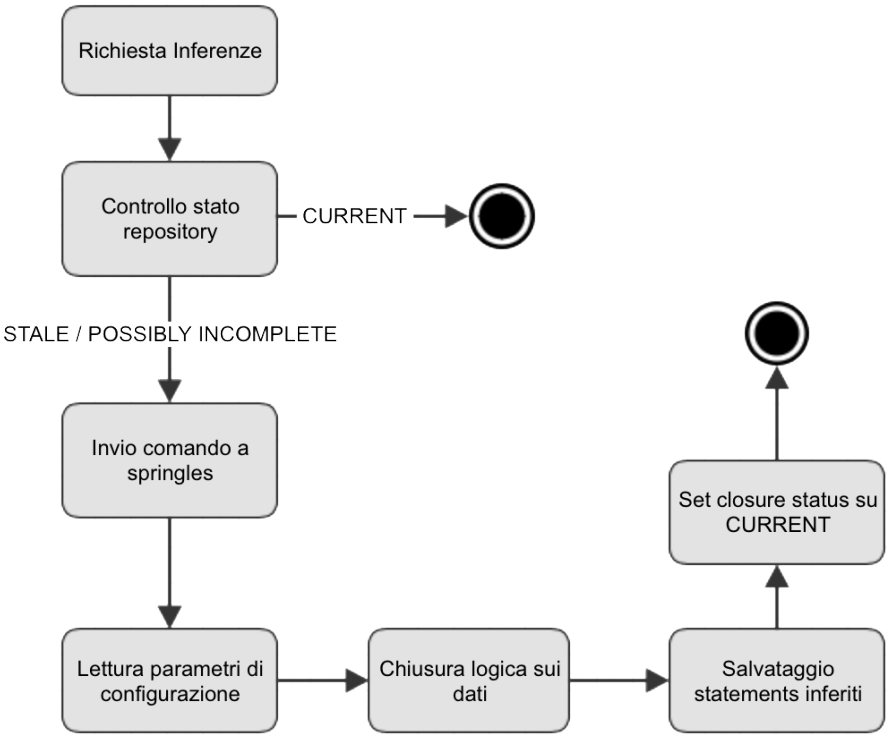

Figura 3.3: Diagramma processo di inferenza

Una volta conclusa l'operazione di inferenza sui dati lo stato del repository viene impostato a CURRENT: se viene richiesta una nuova chiusura logica il sistema non la ripeterà fino a quando non verrà richiesta la cancellazione delle inferenze o vi sarà l'aggiunta di nuovi dati.

3.1.2 SPRINGLES Client

L'interfaccia, che permette all'utente di interagire con il sistema, è un'estensione del tool Sesame Workbench, un'interfaccia grafica di amministrazione per la manipolazione di store di dati (già presentato nel capitolo precedente). L'interfaccia, rispetto a Workbench, offre alcune features aggiuntive, cioè: la richiesta di chiusura logica sui dati (update closure), la richiesta di cancellazione dei dati inferiti (clear closure), la gestione dei ruleset e più informazioni riguardanti il repository dei dati nel sommario. Essa viene mostrata in Figura 3.4.

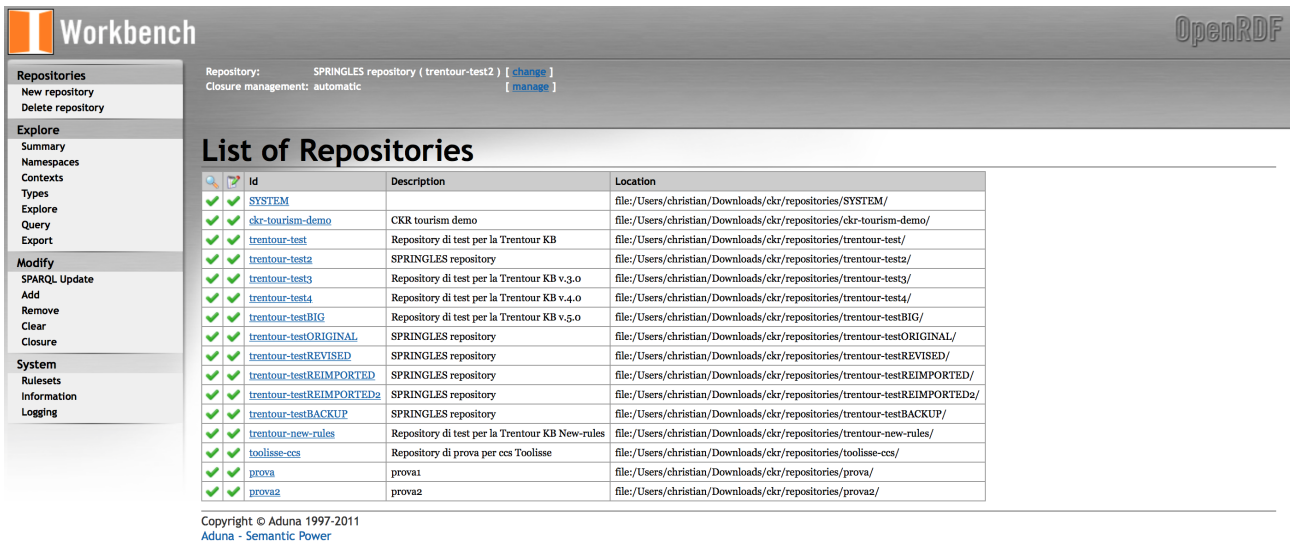

Figura 3.4: Interfaccia WEB per SPRINGLES 1.0

3.1.3 Limitazioni di SPRINGLES 1.0

Abbiamo identificato diverse limitazioni di Springles 1.0:

- L'interfaccia era dipendente da Sesame Workbench[2]: per questo, ad ogni nuova versione del framework Sesame, la sua estensione per SPRINGLES doveva essere adattata e testata sulle nuove caratteristiche;
- era sorta la necessità di sfruttare il sistema SPRINGLES da applicazioni esterne (necessità di una API per l'accesso alle funzionalità del sistema);
- il sistema di inferenza non prevedeva l'utilizzo di regole e motori di inferenza in maniera dinamica: ogni repository era associata ad un motore di inferenza e ad un insieme di regole al momento della creazione e non era possibile cambiarlo;
- il motore di inferenza era limitato al prototipo del NaiveInferencer.

[2]https://sourceforge.net/projects/sesame/files/Sesame 2/2.8.9/

3.2 SPRINGLES 2.0: aggiornamento del sistema

Viste le limitazioni, abbiamo quindi rivisto l'intero sistema, creando una nuova interfaccia API basata su chiamate a un servizio REST e modificando il sistema di inferenze. Abbiamo dunque generalizzato il motore di inferenza, estendendolo a RDFPro.
Sono state create due servlet Java distinte: una per la parte Client (Interfaccia WEB + Servizio REST) e una per la parte di SPRINGLES Server e Core. Entrambe vengono installate e eseguite su Tomcat[3]. Le Servlet sono state pubblicate su GitHub per la distribuzione opensource[4].

Il nuovo sistema, come mostrato anche in figura 3.5, è formato dunque da:

- SPRINGLES Server e Core;
- TripleStore;
- interfaccia WEB;
- servizio REST.

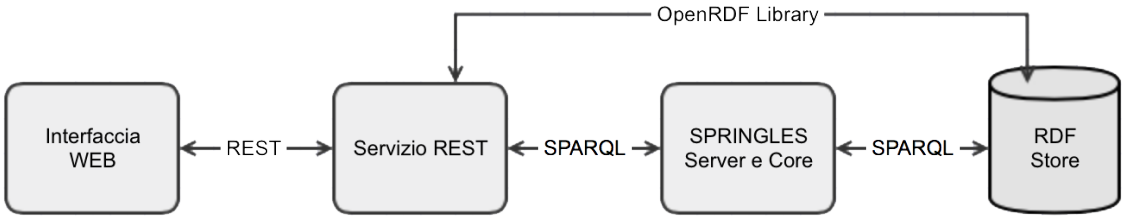

Figura 3.5: Struttura SPRINGLES 2.0

Nel seguito descriveremo in dettaglio tutte le componenti e il loro funzionamento.

[3]http://tomcat.apache.org/index.html
[4]https://github.com/dkmfbk/springlesREST , https://github.com/dkmfbk/springles2.0

3.2.1 Generalizzazione degli inferencer

Come anticipato precedentemente, una delle innovazioni che abbiamo portato al sistema è stata quella di generalizzare l'interfaccia Inferencer e di istanziarla per RDFPro[5]. In questo modo ogni repository può essere assegnato a un inferencer diverso dal Naive Inferencer, motore di inferenza di default di SPRINGLES. Per instanziare questo nuovo motore è stato necessario modificare alcune funzionalità del sistema che erano fortemente dipendenti dall'Inferencer originale. L'obbiettivo è stato proprio quello di dover adattare il sistema all'uso di un secondo motore di inferenze o più. La generalizzazione ha posto il problema della gestione dei Ruleset che vengono gestiti in maniera differente da un inferencer rispetto ad un altro.

Gestione dei Ruleset all'interno di SPRINGLES Un ruleset come abbiamo già visto è un insieme di regole, usate nell'operazione di chiusura e sono specifiche per ogni inferencer. Esse sono inserite in un file di testo in formato .ttl. Questi file vengono inseriti nella cartella delle risorse di springles. Essi vengono poi elencati in un file di testo per ogni inferencer all'interno della sottocartella META-INF.

I due inferencer distinguono i ruleset in maniera differente. Per il motore di default (NaiveInferencer) l'identificativo è l'URI del contesto globale nel quale le regole sono inserite, mentre per il caso di RDFProInferencer, l'identificativo non è altro che il nome del file. Il sistema quindi memorizza per un caso la coppia <URI,Ruleset>, per l'altro <nome File,Ruleset>. La gestione dei rulesets viene garantita rispettivamente dalle classi RulesetsNaive e RulesetsRDFPro che salvano le coppie in una mappa di valori chiamata *REGISTERED_RULESETS*. Queste classi, inoltre, contengono il metodo *load* per l'aggiornamento delle lista, leggendo i file elenco contenuti nella sottocartella delle servlet WEB-INF/classes/META-INF, e richiamando internamente il metodo *register* nel caso venga trovato nel file di testo un nuovo ruleset non registrato in memoria oppure il metodo *unregister* nel caso se ne debba cancellare qualcuno. Esistono poi i metodi *lookup*, per la ricerca di un ruleset partendo dall'identificativo, e il medoto *list* che restituisce la lista dei rulesets in memoria.

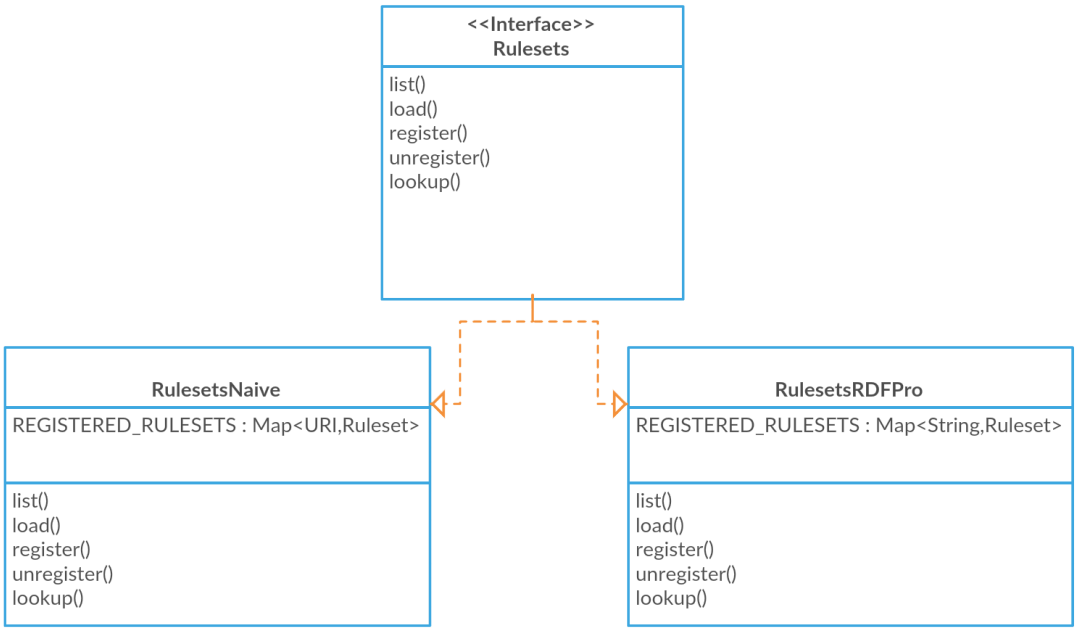

Figura 3.6: Struttura classi per la gestione dei rulesets all'interno di Springles

[5]http://rdfpro.fbk.eu

3.2.2 Inferencer e Ruleset indipendenti

Un'altra limitazione legata a SPRINGLES 1.0 è l'impossibilità di modificare l'inferencer e il ruleset utilizzati per le inferenze una volta creato il repository dei dati. La nostra necessità era quella di assegnare questi due parametri ad ogni richiesta di una chiusura logica, in modo da poterli rendere utilizzabili dinamicamente. Per fare questo abbiamo sfruttato il file di configurazione (springles.ttl) che veniva completato con i parametri al momento della creazione. Con la nuova versione al momento della creazione della struttura dati vengono impostati i valori legati all'inferenza logica ad un valore di default che al momento della chiusura vengono modificati secondo i parametri impostati dall'utente (inferencer, ruleset, binding e inferred context prefix).

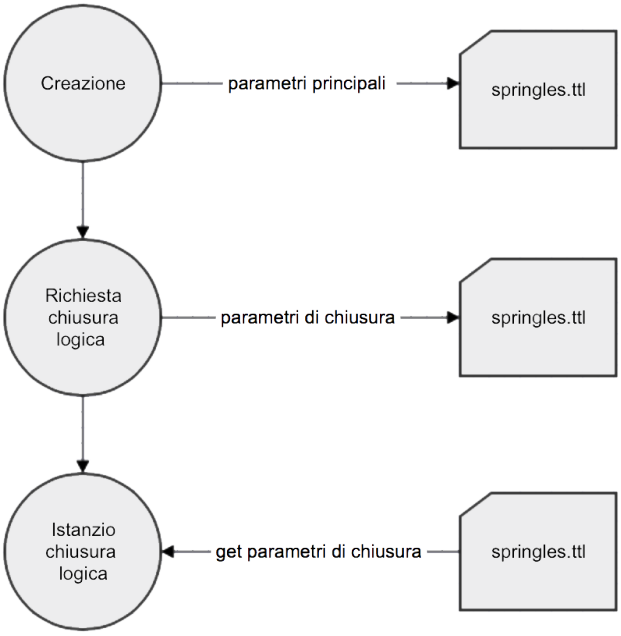

Figura 3.7: Gestione Inferencer e Rulesets

In Figura 3.7, si nota che il nuovo sistema imposta una prima versione del file di configurazione che, al momento della richiesta per la chisura logica, modifica con i nuovi parametri. Un approccio alternativo consiste nel passare tutti i parametri nel comando che il servizio REST invia a SPRINGLES, ma questo non è possibile perché ogni comando deve avere una particolare sintassi (spiegata in dettaglio nel capitolo successivo) per essere riconosciuto da SPRINGLES. In questo modo inoltre sarebbe complesso ricavare tutte le informazioni aggiornate da riportare sul sommario di ogni repository. Ottenuti quindi i parametri aggiornati, la classe Inferencers, che si occupa di gestire l'instanziazione dell'inferencer per la nuova chiusura, ricerca il ruleset utilizzato richiamando il metodo descritto prima *lookup*, per passarlo insieme al resto dei parametri al nuovo inferencer che si occuperà dell'estrazione di nuove conoscenze dai dati RDF espliciti.

3.2.3 Gestione dinamica dei rulesets

Uno dei requisiti dell'aggiornamento riguarda l'aggiunta e la rimozione di rulesets dal sistema. Questo fino alla versione precedente era possibile soltanto manualmente, inserendo il file del nuovo ruleset nella cartella WEB-INF/classes della servlet di SPRINGLES, aggiungendolo alla lista relativa all'inferencer a cui il ruleset è assegnato nella sottocartella META-INF e poi riavviando l'applicazione di

SPRINGLES Server. In SPRINGLES 2.0 l'aggiunta e rimozione dei ruleset viene gestita senza la necessità di riavviare l'applicazione. Ogni volta che l'interfaccia REST richiede la chiusura logica o la lista dei rulesets nel sistema, il sistema richiama il metodo *load* descritto in precendenza, aggiornando la lista dei ruleset per ogni inferencer.

3.2.4 Il servizio REST

Per andare a creare un supporto per la comunicazione con SPRINGLES per applicazioni esterne, tra cui l'interfaccia Web, è stata necessaria la creazione di un'interfaccia API. Questo è stato fatto attraverso un servizio REST scritto in Java. Esso ha l'obiettivo di collegare le applicazioni al server per esporre i servizi dei SPRINGLES. Per tutte le altre funzioni esso si collega direttamente al TripleStore utilizzando la libreria OpenRDF.

In particolare per la parte di inferenza esso gestisce:

- la richiesta di chiusura logica sui dati;
- lo stato di chiusura del repository;
- la richiesta della lista dei rulesets associati all'inferencer di default (NaiveInferencer).

Per richiamare queste funzioni l'interfaccia utilizza chiamate Ajax[6] a degli indirizzi URL specifici per ogni obiettivo passando gli opportuni parametri. Ad esempio per la creazione di un nuovo repository l'indirizzo è

`http://localhost:8080/SpringlesREST/rest/rest/create?springlesrepositoryID=test-springles-5&springlesrepositorytitle=test-springles-5&springlesserverURL=http://localhost:8080/openrdf-sesame`.

Tutti i comandi con i relativi indirizzi sono riportati di seguito presupponendo REST_URL come : http://localhost:8080/SpringlesREST/rest/rest

Richiesta elenco repository nel Database	REST_URL/getRepositories
Creazione Repository	REST_URL/create
Richiesta summary repository	REST_URL/summary
Cancellazione Repository	REST_URL/delete
Caricamento dati	REST_URL/upload
Esportazione Dati	REST_URL/fullexport
Eliminazione Dati	REST_URL/clear
Richiesta chiusura logica	REST_URL/computeclosure
Richiesta grafi di contesto	REST_URL/contexts
Interrogazione dati	REST_URL/querysparql
Richiesta stato chiusura repository	REST_URL/getClosureStatus
Creazione ruleset	REST_URL/create_ruleset
Richiesta lista ruleset nel sistema	REST_URL/list_of_ruleset
Cancellazione ruleset	REST_URL/delete_ruleset
Richiesta contenuto ruleset	REST_URL/content_of_ruleset

Tabella 3.1: Elenco comandi REST

[6]http://api.prototypejs.org/ajax/Ajax/Request/

La comunicazione con SPRINGLES SPRINGLES ha la possibilità di interrogare e manipolare dati attraverso delle Query SPARQL. Alcune query, formate da triple speciali, vengono però filtrare e interpretate come comandi . Ad esempio se viene filtrato il comando “clear graph <springles:update-closure>” il sistema esegue una chiusura logica sui dati; se arriva l’interrogazione “select ?closurestatus {}” il sistema riconosce che deve ricercare il closure status del repository e restituirlo all’utente; mentre se viene riconosciuto il comando ”select ?listofruleset {}” il sistema leggerà tutti i ruleset in memoria per un particolare inferencer che verranno restituiti all’utente. Tutte le altre query vengono gestite normalmente, interrogando o modificando i dati all’interno dello store.

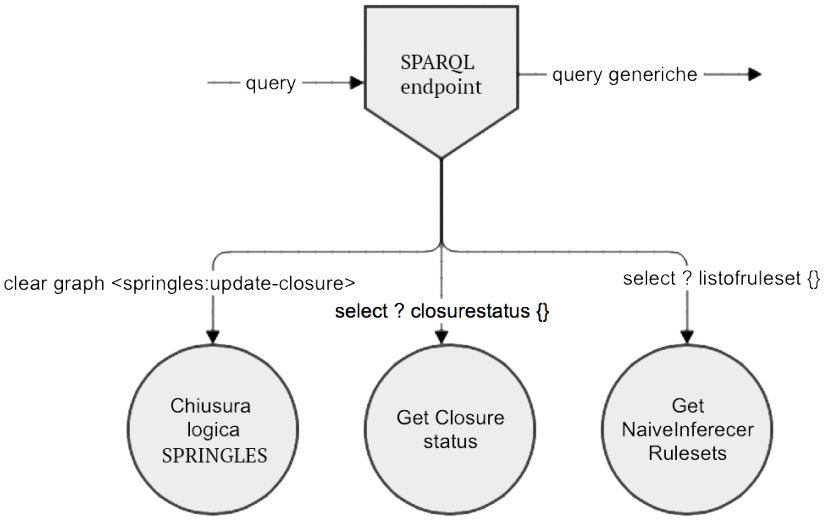

Figura 3.8: Diagramma della comunicazione con Springles

3.2.5 Interfaccia WEB

L'interfaccia WEB, inizialmente un'estensione del framework Sesame Workbench è ora basata unicamente sui servizi REST precedentemente descritti.

Abbiamo utilizzato JQuery[7] per gestire le funzionalità dinamiche dell'interfaccia, come le chiamate Ajax, l'inserimento dei contenuti nella pagina.
Analizzaremo ora ogni singola pagina, descrivendone le funzionalità.

Summary In questa sezione, una volta selezionato il repository desiderato, compariranno le informazioni riguardanti esso in ordine come in figura 3.9: identificativo, titolo, indirizzo repository, inidirizzo server, statements totali, statements non inferiti, statements inferiti, closure status, ultimo inferencer utilizzato, ultimo ruleset utilizzato, prefisso contesto statements inferiti.

Tutte queste informazioni vengono aggiornate ad ogni manipolazione del repository.

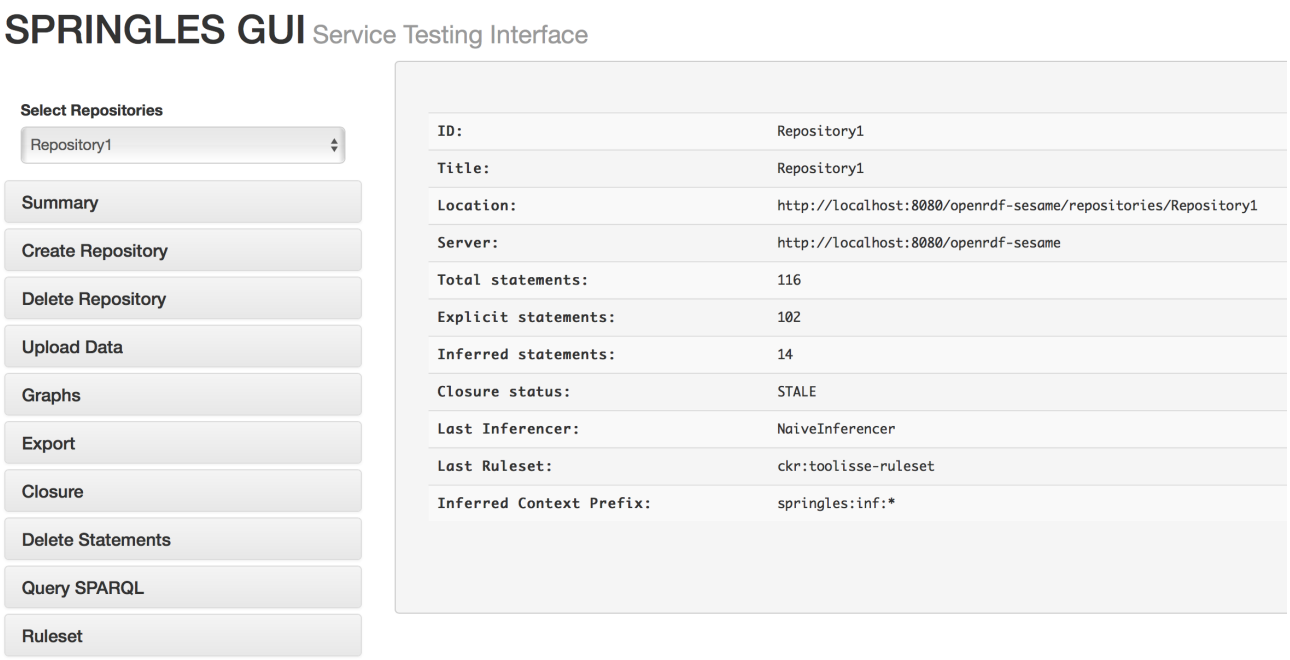

Figura 3.9: Summary

[7] https://jquery.com/

Create Repository Come mostra la figura 3.10, in questa schermata è possibile aggiungere un repository di dati specificandone semplicemente identificativo, che deve essere univoco all'interno dello store, e titolo. Una volta cliccato sul pulsante Create, esso verrà creato e aggiunto alla lista.

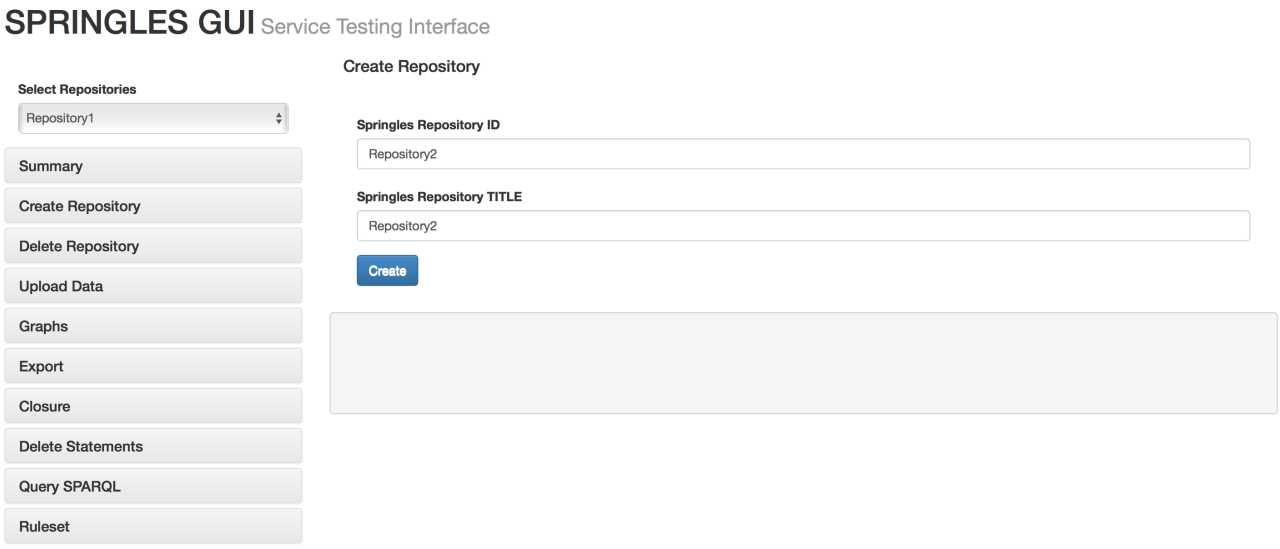

Figura 3.10: Creazione Repository

Delete Repository Come è possibile creare un repository, è possibile anche cancellarlo. Come mostrato in figura 3.11, specificando il repository che si vuole cancellare e premendo il pulsante Delete, esso verrà rimosso dallo store.

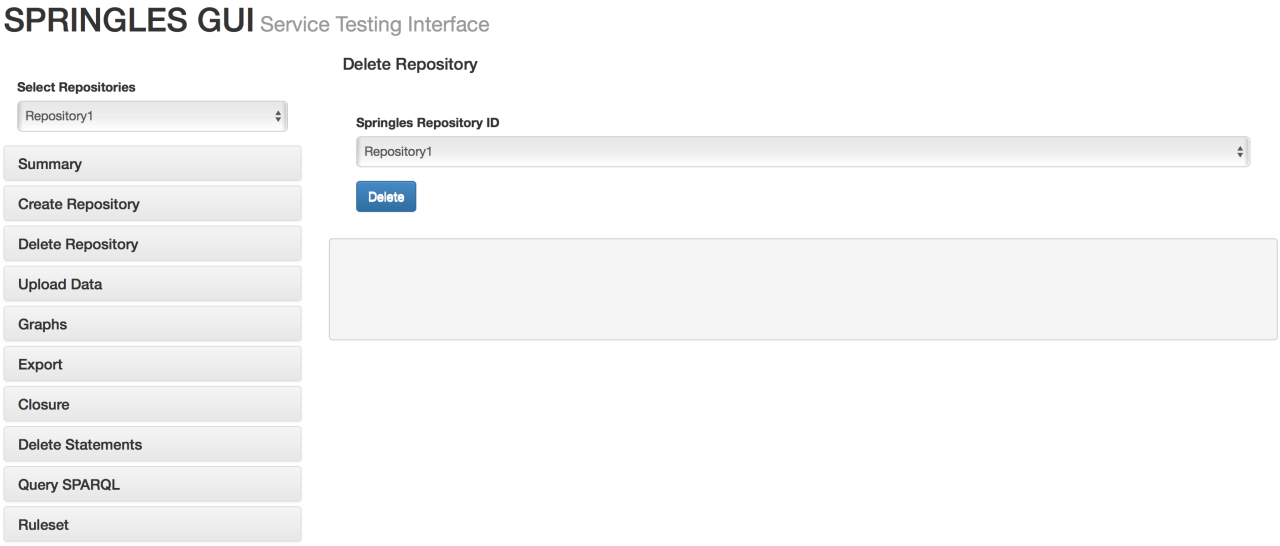

Figura 3.11: Cancellazione Repository

Upload Data In questa sezione, come mostrato in figura 3.12, è possibile caricare i dati in formato RDF specificando il percorso del file all'interno del File System e cliccando Upload.

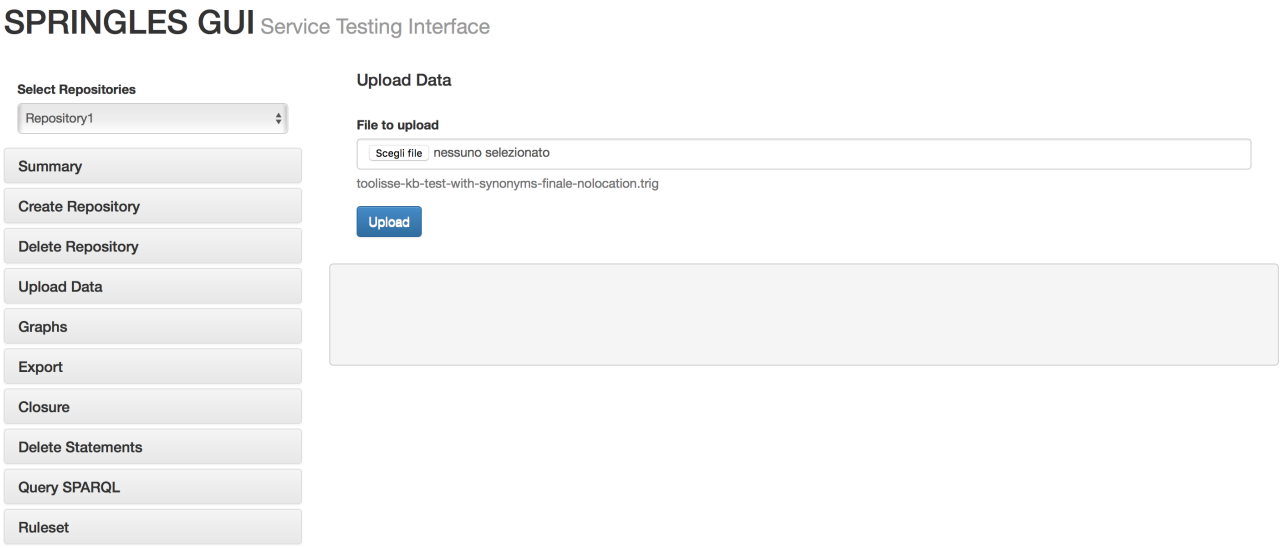

Figura 3.12: Caricamento Dati

Graphs Attraverso questa interfaccia è anche possibile ottenere i vari contesti nel quale i dati sono inseriti. É inoltre possibile filtrare questi ottenendo solo quelli non inferiti oppure tutti quelli presenti nel repository.

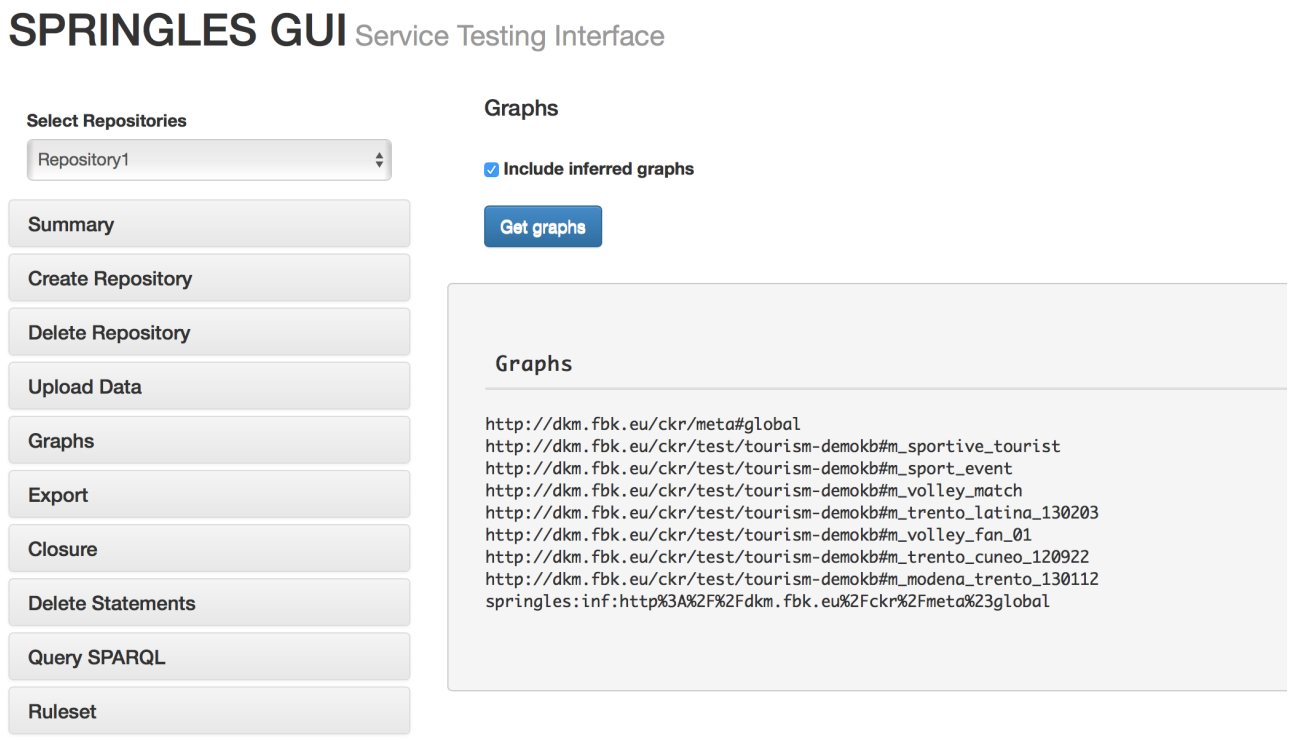

Figura 3.13: Graphs

Export Attraverso questa sezione (figura 3.14), è possibile scaricare i dati di un repository in 3 diversi formati: TRIG, XML o TTL. Anche in questo caso è possibile scegliere se esportare tutti i dati o solo quelli non inferiti.

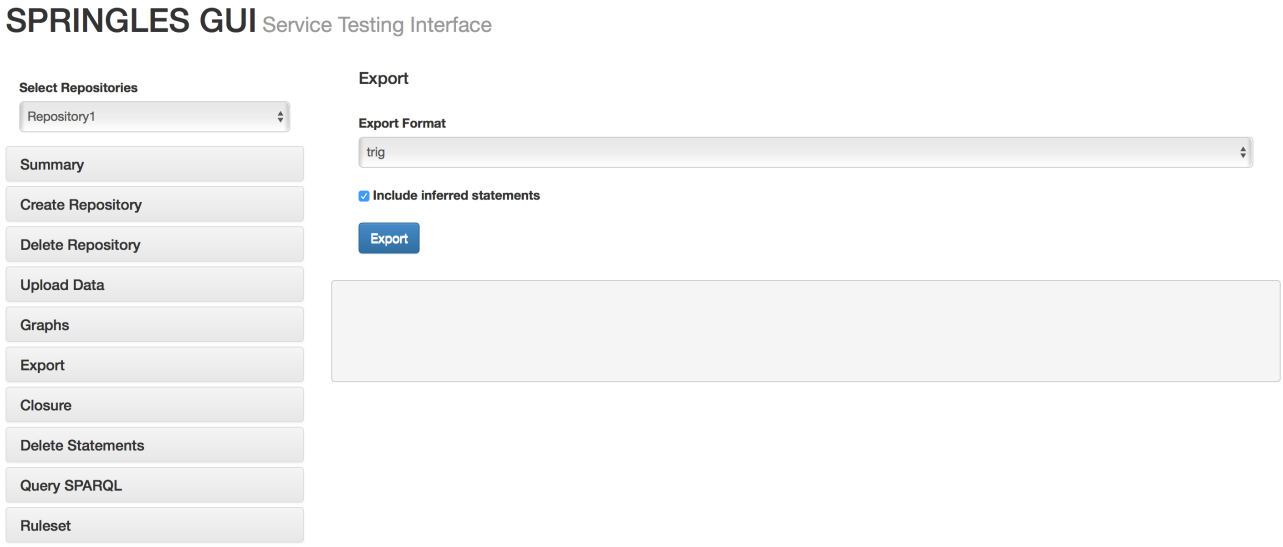

Figura 3.14: Esportazione dati

Closure Come mostrato in figura 3.15, è possibile richiedere la chiusura logica sui dati, specificandone l'inferencer, il ruleset, il binding da applicare alle regole e l'inferred contex prefix. É possibile sia aggiornare la chiusura che farne un clear, cioè cancellare tutti i dati inferiti e riportare lo stato a POSSIBILY_INCOMPLETE.

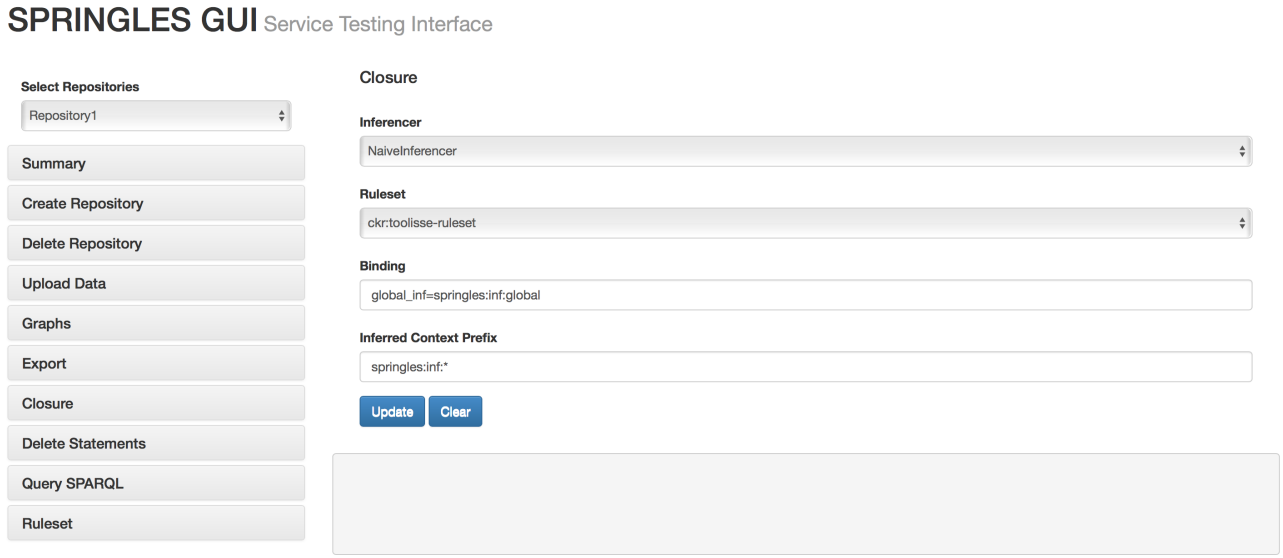

Figura 3.15: Closure

Delete Statements Attraverso questa sezione (figura 3.16) è possibile cancellare completamente il repository dei dati.

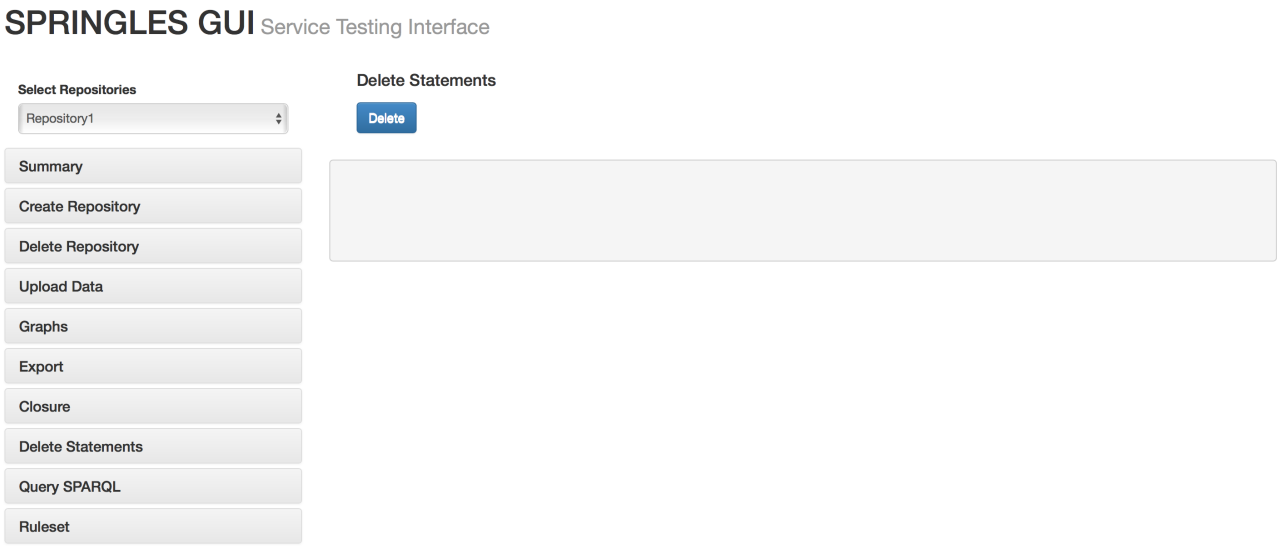

Figura 3.16: Cancellazione statements

Query SPARQL In questa sezione, come mostrato in figura 3.17, si può interrogare il repository attraverso delle query SPARQL, ottenendone dei risultati. Inoltre la parte di editing della query è dotata di autocompletamento, autoimport e segnalazione errori di sintassi utilizzando le librerie YASQUE[8] e YASR[9].

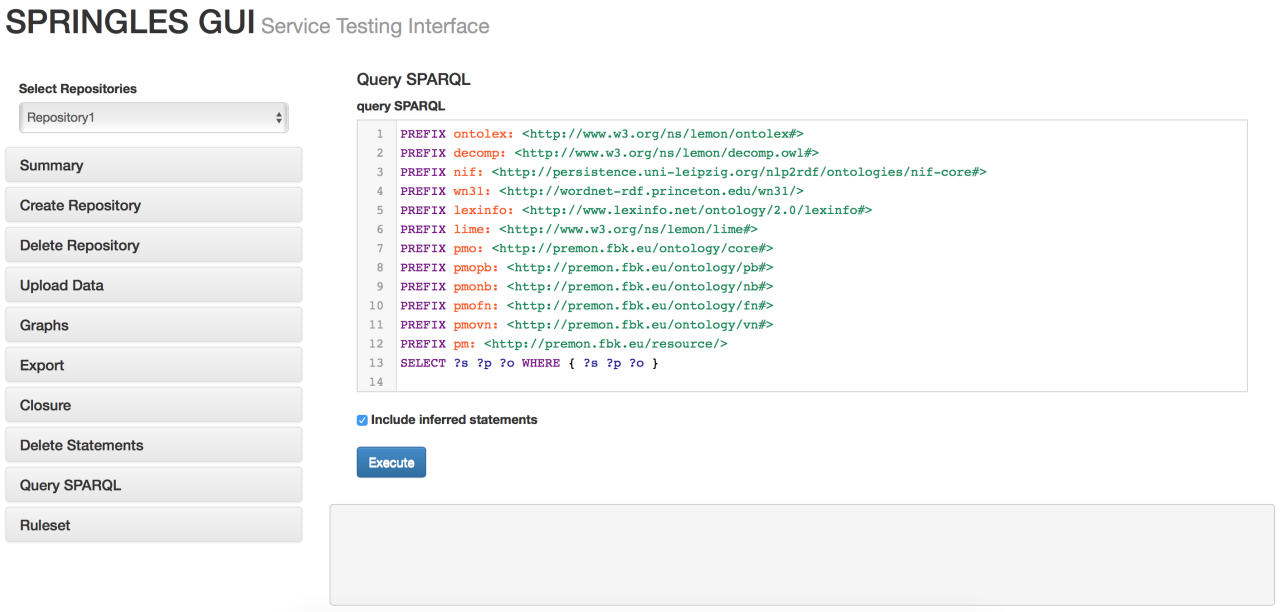

Figura 3.17: Query SPARQL

[8] yasqe.yasgui.org
[9] yasr.yasgui.org

Ruleset L'ultima sezione dell'interfaccia (figura 3.18) è utilizza per la gestione dei Rulesets. Qui è possibile visualizzare i rulesets per ogni inferencer, eliminarli e aggiungerne di nuovi specificandone il nome del file che conterrà le nuove regole.

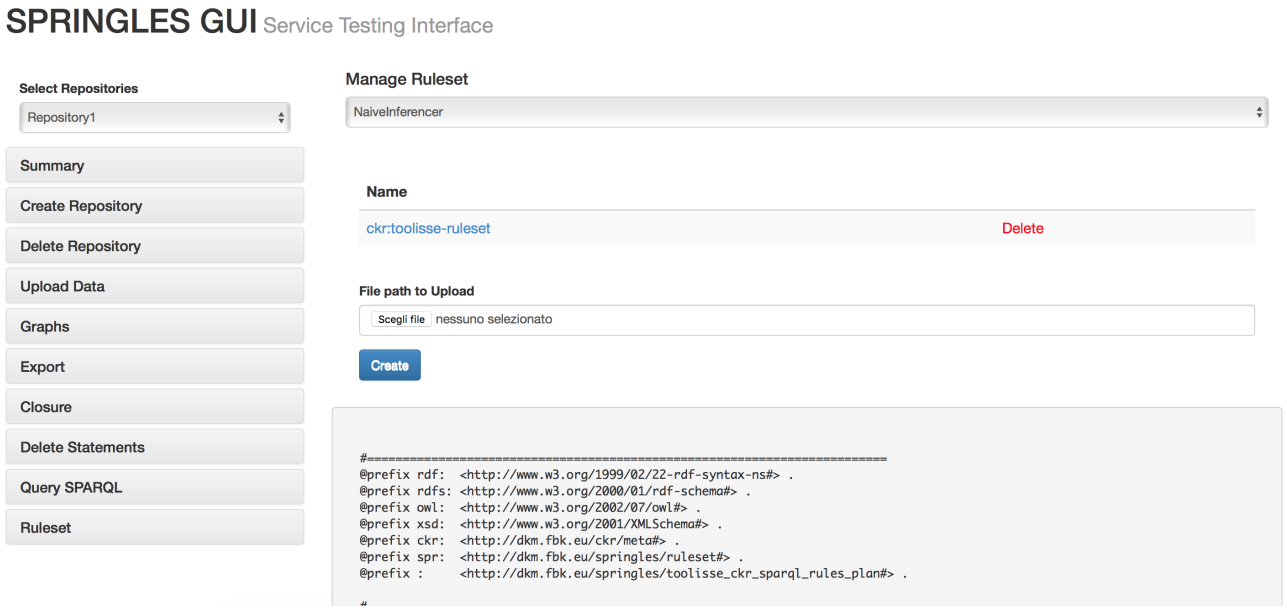

Figura 3.18: Ruleset

4 Conclusioni

Nei capitoli precedenti abbiamo presentato il sistema SPRINGLES, per il ragionamento a regole su triple RDF. Siamo partiti esaminando lo stato iniziale per poi analizzare in dettaglio i miglioramenti fatti fino al rilascio della versione SPRINGLES 2.0. Il sistema ora è utilizzabile attraverso l'interfaccia Web realizzata e descritta. É possibile introdurre nuovi inferencer grazie alla generalizzazione del sistema. Ora, al momento della chiusura, inferencer e ruleset possono essere modificati. Il servizio REST, inoltre, fornisce la possibilità di interfacciarsi con lo store dei dati RDF (aggiunta, cancellazione e proiezione triple, aggiunta e cancellazione repository) e alle funzionalità SPRINGLES (chiusura, gestione rulesets) anche da applicativi esterni attraverso chiamate API.

Ulteriori miglioramenti saranno l'introduzione di un nuovo inferencer chiamato CKRew[9], adattare SPRINGLES alla nuova versione di Sesame 4.0 e all'uso di altri sistemi.

Bibliografia

[1] Francesco Corcoglioniti, Marco Rospocher, Marco Amadori, and Michele Mostarda. RDFPRO: an Extensible Tool for Building Stream-Oriented RDF Processing Pipelines. https://dkm-static.fbk.eu/people/rospocher/files/pubs/2014iswcSemDev01.pdf.

[2] Tim Berners-Lee, Jim Hendler, and Ora Lassila. The Semantic Web, in Scientific American . May 2001. http://www.scienti camerican.com/article/the-semantic-web/.

[3] URI Planning Interest Group and W3C/IETF. URIs, URLs, and URNs: Clarifications and Recommendations 1.0. W3C recommendation, W3C, September 2011. http://www.w3.org/TR/uri-clarification/.

[4] Guus Schreiber. RDF 1.1 Primer. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/.

[5] Dan Brickley and R.V. Guha. RDF Schema 1.1. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/rdf-schema/.

[6] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

[7] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C recommendation, W3C, March 2013. http://www.w3.org/TR/sparql11-query/.

[8] Loris Bozzato and Luciano Serafini. Materialization Calculus for Contexts in the Semantic Web. http://ceur-ws.org/Vol-1014/paper_51.pdf.

[9] Loris Bozzato, Thomas Eiter, and Luciano Serafini. Contextualized Knowledge Repositories with Justifiable Exceptions. http://ceur-ws.org/Vol-1193/paper_74.pdf.