Università degli Studi di Roma "Sapienza"

Facoltà di Ingegneria

Tesina del corso di Seminari di Ingegneria del Software

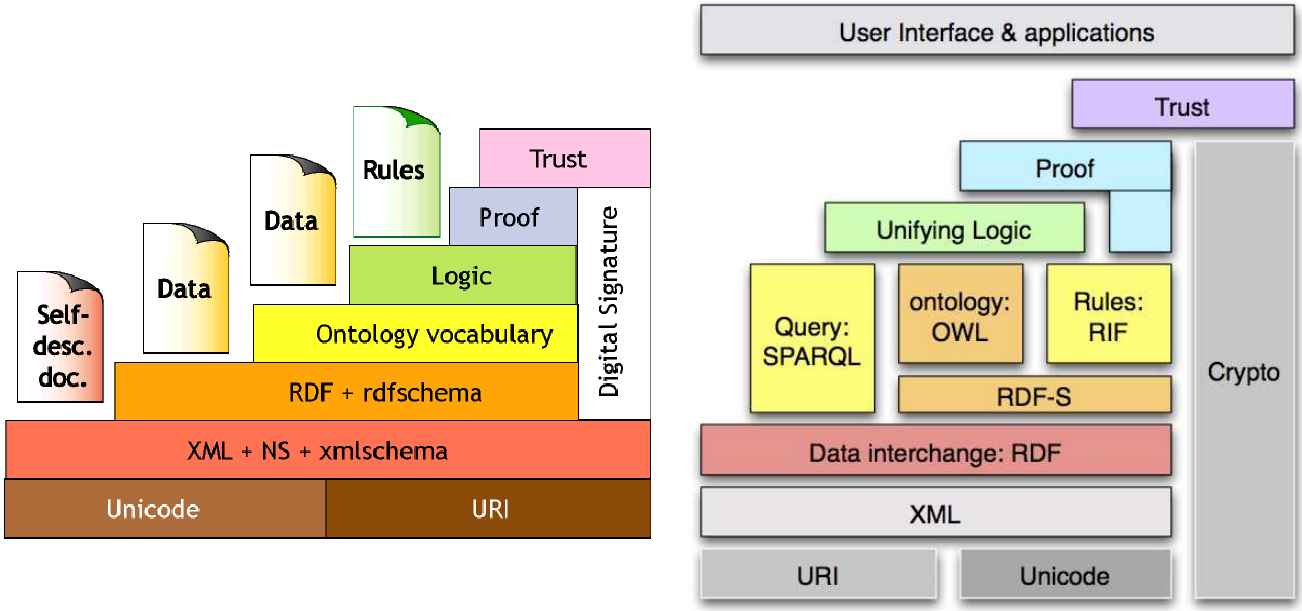
Metamodellazione in RDF

Autore:
Alberto Cerullo

Sommario

Introduzione

Questa tesina ha come scopo lo studio della metamodellazione dei dati definiti in RDF, linguaggio nato per la descrizione delle informazioni delle risorse del Web e come modello per la definizione di affermazioni sul mondo e la gestione dei metadati.
La metamodellazione dei dati è uno degli aspetti più interessanti del Semantic Web.
Con il termine Semantic Web si suole indicare un apparato di tecnologie realizzate per rendere elaborabili e comprensibili le informazioni da parte di programmi. L'obiettivo principale consiste,infatti, nel permettere ad infrastrutture e applicazioni software di creare nuova conoscenza traendo delle conclusioni a partire dalla base di conoscenza iniziale. Si vuole, cioè, passare dal "machine-readable" al "machine-understandable". Usando un'analogia, mentre il Web attuale può essere descritto come una piattaforma decentralizzata utile per presentazioni distribuite di informazioni, il Semantic Web è una piattaforma decentralizzata per una conoscenza distribuita. In questo scenario RDF è lo standard definito dal W3C per "definire" tale conoscenza.
L'interoperabilità tra i sistemi è stabilita proprio tramite la connessione logica delle informazioni.
La struttura del Semantic Web ha uno stack ben preciso formato da diverse componenti poste su più livelli. Ogni livello va inteso come base per gli standard definiti al livello superiore. In questo lavoro ci occuperemo di RDF e dell' RDFSchema (RDFS).



Dopo aver presentato il linguaggio RDF e l'RdfSchema, con relativa sintassi e semantica, vedremo quali sono le tecniche utili per realizzare la metamodellazione

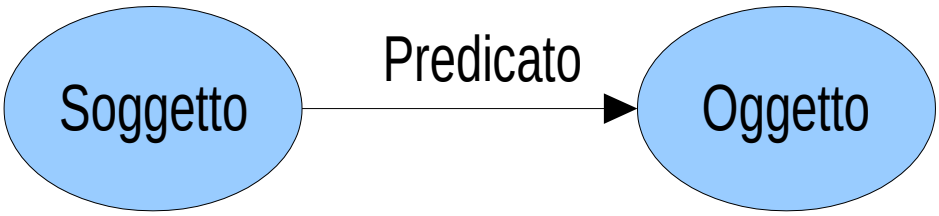
dei dati in questo linguaggio. Inoltre, per completare lo scenario, saranno presentati alcuni esempi di metamodellazione chiarificatori delle tecniche introdotte. Tra questi esempi saranno presentati anche dei pattern ontologici utilizzati come metamodelli per alcuni scenari applicativi.

Capitolo 1 RDF e RDFS

1.1 RDF

Il termine RDF è l'acronimo di Resource Description Framework ed indica uno standard XML definito dal W3C, utile per la rappresentazione dei metadati.
RDF è stato pensato per situazioni in cui l'informazione, che deve essere veicolata, sia processata da applicazioni, piuttosto che visualizzata da persone.
Per capire l'utilità di tale standard facciamo un esempio. Ipotizziamo l'esistenza di tre siti web: il primo presenta una lista di notebook di alcune marche, il secondo pubblica un database di notebook gestito da una rivista on line specializzata; il terzo contiene i commenti dei consumatori relativi all'acquisto dei propri notebook.
Tutti questi siti presentano informazioni sensibili che, qualora descritte in formato RDF ed opportunamente connesse tra loro, permettono di ottenere informazioni più complete rispondenti ad esigenze diverse, come ad esempio aiutare i possibili acquirenti di notebook nella scelta del prodotto che desiderano.

RDF introduce un formalismo per la rappresentazione dei dati: lo "statement" (o asserzione) che viene realizzato per mezzo di triple caratterizzate dai seguenti elementi: "soggetto"-"predicato"-"oggetto". Il soggetto è la risorsa che si vuole definire, il predicato rappresenta la proprietà della risorsa e l'oggetto è il valore di tale proprietà. In questa maniera possono essere definite diverse tipologie di relazione tra elementi, permettendo un'ampia capacità espressiva.



Graficamente, come possiamo vedere anche nell'immagine, un documento RDF è un grafo etichettato ed orientato. I nodi sono i soggetti e gli oggetti di ciascuno statement, mentre gli archi rappresentano i predicati e assumono il verso dal soggetto all'oggetto e l'etichetta uguale al nome della proprietà del soggetto.
Un insieme di triple definisce un grafo RDF, che può essere caratterizzato da proprietà relative ad un'unica risorsa oppure da relazioni tra più risorse. Ogni risorsa viene definita attraverso gli URI (Uniform Resource Identifier) ossia delle stringhe che le identificano in maniera univoca. Gli URI superano le limitazioni degli URL (Uniform Resource Locator) che definiscono risorse specifiche della rete, quali immagini, pagine web, documenti, e che si riferiscono alla loro localizzazione e

protocollo. Infatti tramite gli URI possono essere definiti anche elementi non accessibili in rete o concetti astratti.

RDF è costituito da :

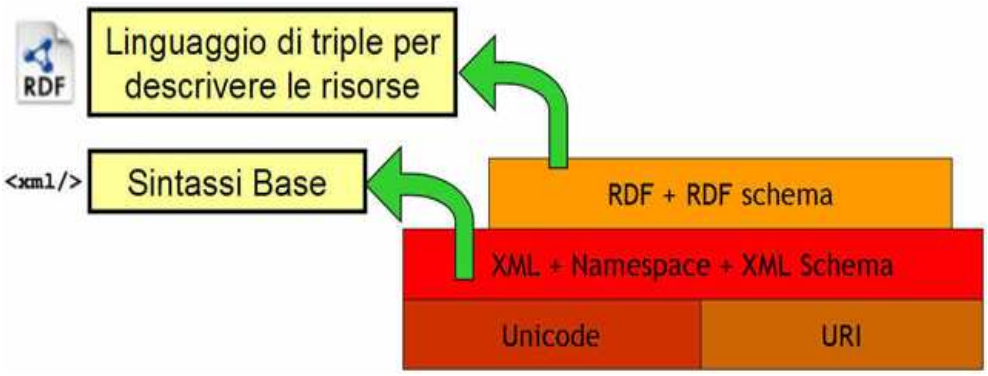
- RDF Model and Syntax: che definisce il data model di RDF e la sua codifica XML
- RDF Schema: che permette la definizione di vocabolari per i metadati.

Le caratteristiche fondamentali di RDF sono:

- Indipendenza: ogni risorsa può essere definita in maniera autonoma, senza riferimenti ad altre risorse preesistenti, definendo le proprie proprietà e le proprie relazioni con altre risorse.

- Interscambio:le proprietà RDF sono espresse attraverso XML quindi sono facilmente interscambiabili.

- Scalabilità: un grafo RDF, per la sua particolare struttura, ossia una serie di triple con tre campi, risulta facile da gestire. Aspetto sicuramente importante visto la mole di informazione, in continua crescita, presente nel Web.



Il linguaggio in cui è scritto RDF è una versione modificata di XML che prende il nome di RDF/XML . La scelta di tale linguaggio come sintassi di base risiede nel fatto che le informazioni RDF possono essere così facilmente scambiate fra macchine che usano diverse piattaforme e differenti linguaggi di programmazione. L' eXtensible Markup Language (XML) è un metalinguaggio con il quale è possibile definire sintatticamente linguaggi di mark-up. Attraverso l'uso di marcatori (mark-up) si esplicita in maniera formale la sintassi di un documento.
Il limite di XML è che il suo scopo è unicamente quello di descrivere i dati, senza interessarsi della semantica relativa ai dati stessi.

L’interpretazione dei dati, infatti, è a discrezione del lettore e non è comprensibile dalla macchina, poiché non è presente un esplicito formalismo di definizione della classificazione. Un documento XML presenta una struttura gerarchica, ad albero, in cui le informazioni sono correlate secondo una relazione di subordinazione. Tuttavia tale subordinazione può essere semanticamente dedotta secondo una logica umana, facendo riferimento ai nomi dei tag e ai livelli d’annidamento degli elementi.
Inoltre, documenti XML diversi potrebbero esprimere informazioni dello stesso tipo, attraverso l’uso di tag diversi, perdendo così la possibilità di integrare tali informazioni.

Mentre un documento XML è essenzialmente un albero con dei tag contente testo, una “description” o asserzione RDF consiste di un semplice statement:soggetto, predicato, oggetto. Più statement possono essere combinati, formando così un grafo, ma ogni statement RDF può essere usato e distribuito da solo.

XML, inoltre, è significativamente meno flessibile nella rappresentazione dei metadati, mentre RDF più semplice e maggiormente flessibile per gli scopi visti.

1.2 RDFS

Il modello di RDF non permette di effettuare la validazione di un valore o la restrizione di un dominio di applicazione di una proprietà. Questo compito è svolto dal RDF Schema. Le proprietà RDF e la loro semantica vengono definite all’interno dello schema RDF, che risulta essere, quindi, una sorta di vocabolario, in cui indicare i vari tipi di risorse che vengono via via descritte.
Dunque, mentre il semplice RDF è utile per descrivere modelli di dati e semplici affermazioni, l’RDF Schema permette di assegnare un significato ai termini utilizzati nelle asserzioni RDF. Una risorsa può essere definita come istanza di una o più classi, le quali possono così descrivere una gerarchia. In questo modo si possono , ad esempio, derivare nuove proprietà per ereditarietà. Inoltre le proprietà possono essere specializzate, attraverso la definizione di vincoli di applicabilità e organizzazione gerarchica.

A differenza di XML Schema o di un DTD, dove viene descritta la struttura sintattica di un documento, l’RDF Schema non vincola la struttura del documento, ma fornisce informazioni utili all’interpretazione del documento stesso. Le differenze tra le due tipologie di schemi possono essere così descritte: XML/XML Schema è un linguaggio per la modellazione dei dati e realizzare un metadato XML è un’attività puramente sintattica. RDF è un linguaggio di meta-modellazione e lo schema RDF richiede parte della semantica dei termini usati.
RDFS viene definito in termini di RDF e si trova ad un livello superiore ad RDF. Infatti, mentre in RDF ogni predicato è astratto, senza connessioni con altri predicati, RDFS permette di esprimere relazioni e vincoli tra predicati e di definire proprietà caratteristiche di un concetto.

RDFS non ha classi predefinite, ma fornisce la piattaforma necessaria per definire nuove classi. Le classi sono simili alle classi di un linguaggio di programmazione. Tuttavia, presentano delle differenze.

A differenza dei linguaggi di programmazione object oriented, non viene descritta una classe con le sue proprietà, ma un insieme di proprietà che possono essere applicate a specifiche classi di risorse tramite l'uso delle proprietà "domain" e "range".
Così mentre in Java, ad esempio, lo scopo di un attributo è ristretto ad una classe, in RDF le proprietà sono di default indipendenti dalla definizione di una classe e quindi possono non avere un dominio specifico cui fare riferimento.
RDF è, perciò, un sistema orientato alle proprietà.
Infine RDFS non è prescrittivo, ma consiste solo in una descrizione di risorse. Se ad esempio in Java quando creiamo un'istanza della classe Libro dobbiamo necessariamente assegnare un valore all'attributo autore (se nella definizione della classe c'è l'attributo autore), in RDF la proprietà autore può anche avere un valore di una classe non specificata.

1.3 Il data model di RDF/RDFS

In questo paragrafo esamineremo brevemente il data model di RDF.
Il data model di RDF è caratterizzato da tre oggetti:

- Resource (Risorsa): qualunque elemento descritto da una tripla RDF. Può essere un elemento presente nel Web come una pagina html o una sua parte o una collezione di pagine e/o suoi elementi. Può rappresentare anche oggetti fisici come ad esempio una persona o una macchina. Le risorse vengono definite attraverso un URI.
- Property (Proprietà): rappresenta un attributo, una caratteristica o una relazione propria di una risorsa. Ogni proprietà definisce i valori che vengono ammessi, il suo significato, le risorse che può descrivere e le eventuali relazioni con altre proprietà. Ogni proprietà ha un nome che la identifica e assume un valore.
- Statement (Asserzione): è l'elemento principale del grafo RDF ed è costituito da una tripla caratterizzata da tre elementi: soggetto, predicato ed oggetto.

Ciascuna tripla, dunque, è caratterizzata da tre elementi , le cui caratteristiche sono le seguenti:

Soggetto	Il soggetto identifica la risorsa che deve essere descritta nello statement. Appartiene a Resource
Predicato	Il predicato identifica la proprietà della risorsa che si sta descrivendo, infatti viene anche chiamata proprietà. Appartiene a Property. Serve per rappresentare attributi,caratteristiche o relazioni di una risorsa.
Oggetto	L' oggetto identifica il valore della proprietà. Appartiene a Resource o Literal.

Il soggetto (S) può essere rappresentato o attraverso un URI reference o un blank node.
Il predicato (P) può essere rappresentato tramite un URI reference.
L'oggetto (O) tramite un URI reference oppure un blank node oppure un Literal.

Riassumendo:

S → URI | blank_node
P → URI
O → URI | blank_node | Literal

Se $s \in S$, $p \in P$, $o \in O$ Una tripla è, dunque, così caratterizzata: $s \quad p \quad o$

Un URI Reference è la congiunzione di un URI (Uniform Resource Idenitifier), cioè una stringa che identifica in maniera univoca una risorsa, e un identificatore opzionale, che è anche esso una stringa, preceduto da #.

Un Literal (Letterale) può essere semplice o tipizzato. Nel primo caso è caratterizzato da una stringa e da un tag di linguaggio opzionale. Nel secondo caso è caratterizzato da una stringa e da un datatype URI, che serve per identificare il tipo di dato che la stringa vuole rappresentare.

Un blank node (chiamato anche « risorsa anonima ») è un nodo che permette la connettività tra le diverse parti del grafo e serve nei casi in cui va descritta una risorsa senza un URI. Viene usato un identificatore per rappresentare il blank node all'interno della tripla e per poterlo usare in altre triple.

Vediamo quanto detto con un semplice esempio. Vogliamo descrivere in RDF l'asserzione: "L'autore di http://www.website.it è Alberto Cerullo".
Il soggetto è http://www.website.it, il predicato è autore, l'oggetto è Alberto Cerullo.

```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:c="http://somewhere#">
        <rdf:Description rdf:about="http://www.website.it">
                <c:autore>Alberto Cerullo</c:autore>
</rdf:RDF>
```

L'elemento *RDF* serve per definire i confini un documento XML degli statement RDF. L'elemento *Description* indica l'inizio della definizione di una risorsa, mentre l'attributo about (che può essere sostituito anche con l'equivalente *id*) indica la risorsa che viene descritta. Se questi due elementi non sono presenti abbiamo una risorsa anonima. All'interno della sezione description c'è l'elenco delle proprietà della risorsa che in questo caso corrispondono alla proprietà *autore*.

Per poter rappresentare relazioni più complesse tra le risorse all'interno degli statement dobbiamo introdurre ulteriori elementi:

rdf:type	Serve per esprimere una relazione del tipo is-member-of. Indica l'appartenenza di una risorsa ad una determinata classe, che presenta, dunque, tutte le caratteristiche previste per un membro di tale classe. Il valore di rdf:type deve essere una risorsa del tipo rdfs:Class. Una risorsa può essere istanza di una o più classi.
rdfs:Class	Le classi RDF rappresentano un concetto simile a quello di classe previsto nei linguaggi object-oriented come ad esempio Java. Quando viene definita una classe, la risorsa che la rappresenta deve presentare la proprietà rdf:type impostata a rdfs:Class
rdfs:subClassOf	Rappresenta una relazione is-a e specifica la relazione di ereditarietà fra classi. Permette di definire una classe come sottoclasse di un'altra o di più classi (ereditarietà multipla). Può essere assegnata solo ad istanze di rdfs:Class
rdfs:subPropertOf	Indica che una proprietà è la specializzazione di un'altra. Ogni proprietà può essere la specializzazione di zero, una o più proprietà
rdfs:domain	E' un' istanza di rdf:Property ed è usata per indicare che una particolare proprietà si applica ad una designata classe. La tripla S rdfs:domain C dichiara che S è istanza della classe rdf :Property, che C è un'istanza della classe rdfs:Class e che la risorsa che presenta come predicato S è istanza della classe C. Una proprietà può avere zero, uno o più domini.

rdfs:range	E' un'istanza di rdf:Property e viene usata per dichiarare che il valore di una proprietà è istanza di una classe. La tripla S rdf:range C asserisce che S è un'istanza della classe rdf:Property, che C è un'istanza della classe rdf:Class e che le risorse presenti nelle terni con oggetto S sono delle istanze di C.

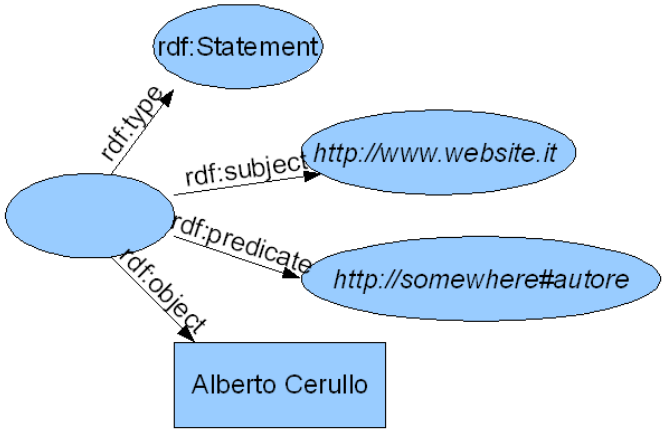
Ultimo importante costrutto del data model di RDF è la *reificazione*.
Oltre alle affermazioni, agli statement su risorse, RDF permette di esprimere anche citazioni, ovvero reificazioni o meta-affermazioni. La reificazione (riduzione ad oggetto) può essere descritto come lo strumento attraverso il quale si possono definire statement RDF tramite altri statement RDF. Ciò equivale a fornire metainformazioni su una metainformazione. Questo meccanismo è stato introdotto nelle specifiche di RDF(S) per aumentarne la capacità di descrizione, poiché rende descrivibile in RDF qualsiasi elemento definito in RDF.
Per realizzare questo costrutto, poiché gli statement sono fatti su risorse, bisogna creare una risorsa con quattro proprietà: subject (il soggetto dello statement) , predicate (la proprietà dello statement), object (il valore della proprietà, nonché oggetto dello statement) e type (che serve per dichiarare la risorsa come un elemento di rdf:Statement). Dopo aver effettuato la reificazione, si possono esprimere ulteriori proprietà sulla risorsa.

Sia r una Risorsa, $s \in S$, $p \in P$, $o \in O$ allora la sintassi della reificazione è del tipo:

r rdf:type rdf:Statement
r rdf:subject s
r rdf:predicate p
r rdf:object o

L'esempio visto in precedenza "L'autore di http://www.website.it è Alberto Cerullo" reificato diventerebbe:

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:c="http://somewhere#">
<rdf:Description>
<rdf:subject rdf:resource=" http://www.website.it "/>
<rdf:predicate rdf:resource=" http://somewhere#autore"/>
<rdf:object>Alberto Cerullo</rdf:object>
<rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdfsyntax-
ns#Statement"/>
</rdf:Description>
```

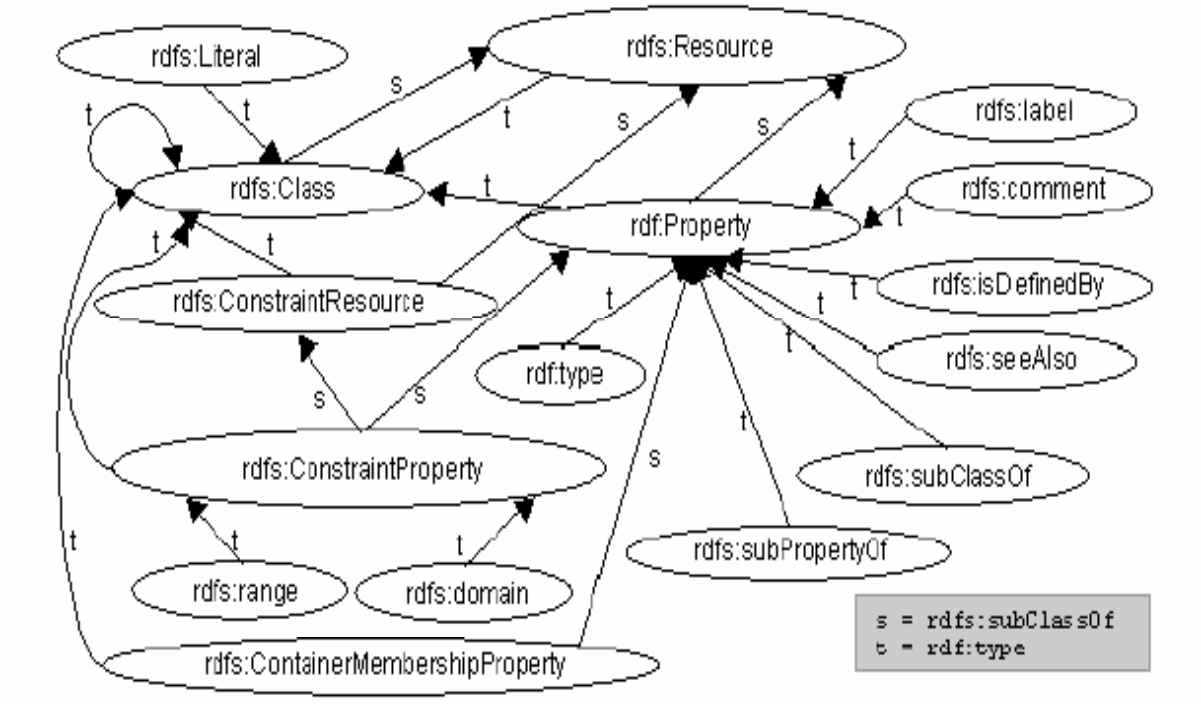


Lo statement reificato può essere usato come oggetto di un altro predicato.

Ad esempio “Luca afferma che l’autore di http://www.website.it è Alberto Cerullo”

```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:c="http://somewhere#">
<rdf:Description>
<rdf:subject rdf:resource=" http://www.website.it "/>
<rdf:predicate rdf:resource=" http://somewhere#autore"/>
<rdf:object>Alberto Cerullo</rdf:object>
<rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdfsyntax-
ns#Statement"/>
<c.affermatoDa>Luca</c:affermatoDa>
</rdf:Description>
```

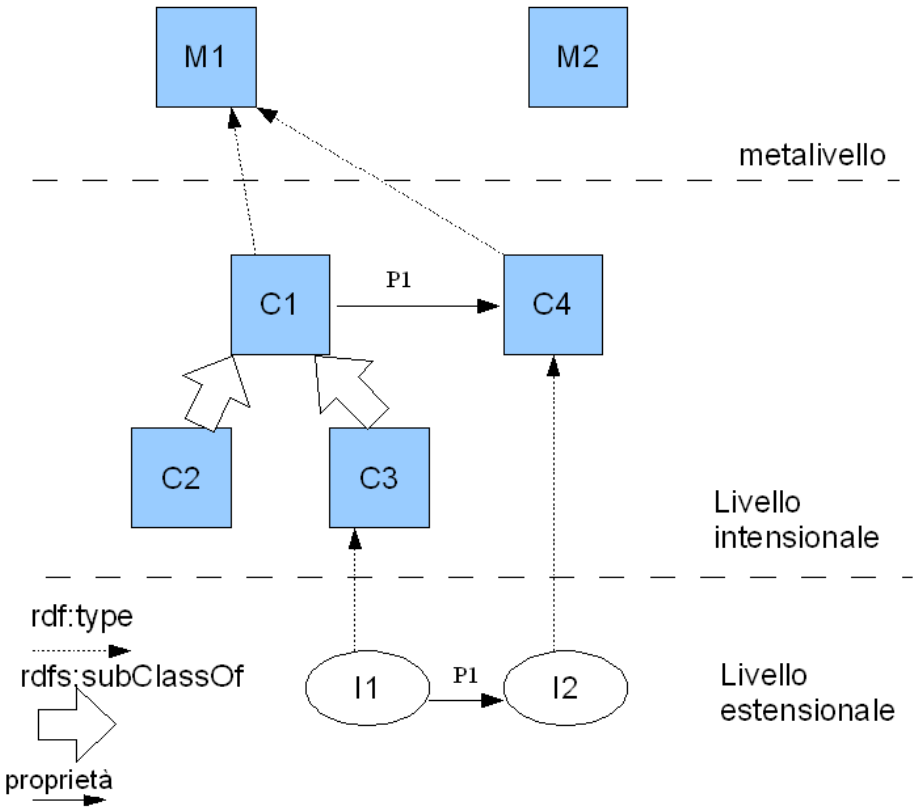
La gerarchia completa delle classi RDFS è la seguente:



Capitolo 2 Metamodellazione

La metamodellazione è la costruzione o modellazione di un insieme di “concetti” relativi ad un certo dominio. Un modello è l’astrazione di elementi del mondo reale; un metamodello è un’ulteriore astrazione relativa a proprietà superiori del modello stesso. Il metamodello, quindi, rappresenta concetti di un livello “più alto” rispetto a quelli del semplice modello.
Per poter descrivere la metamodellazione dobbiamo introdurre i concetti di livello intensionale, estensionale e meta livello.
Con il termine livello intensionale si suole indicare un insieme di concetti, e di regole che le caratterizzano, utili per descrivere la struttura del dominio di interesse. Il livello estensionale, invece, indica un insieme di istanze relative ai concetti descritti al livello intensionale.
Il metalivello, invece, specifica un livello superiore a quello intensionale , caratterizzato da un insieme di categorie più generali di modellazione.

In maniera più lasca possiamo dire che se l’insieme degli elementi del livello estensionale sono un’istanziazione degli elementi del livello intensionale, questi ultimi sono un’istanziazione del meta livello.



Tuttavia il processo può essere iterato, aggiungendo meta livelli su meta livelli. In questo caso si parla di meta-metamodellazione, che consiste in un'ulteriore formalizzazione dei concetti definiti al livello inferiore.

Per gli scopi di questa tesina ci concentriamo sul singolo meta-livello, dato che iterare il procedimento su livelli successivi consiste semplicemente in un riutilizzo delle tecniche che vedremo.

2.1 La metamodellazione in RDF

In questa sezione descriveremo come è possibile realizzare la metamodellazione in RDF/RDFS.

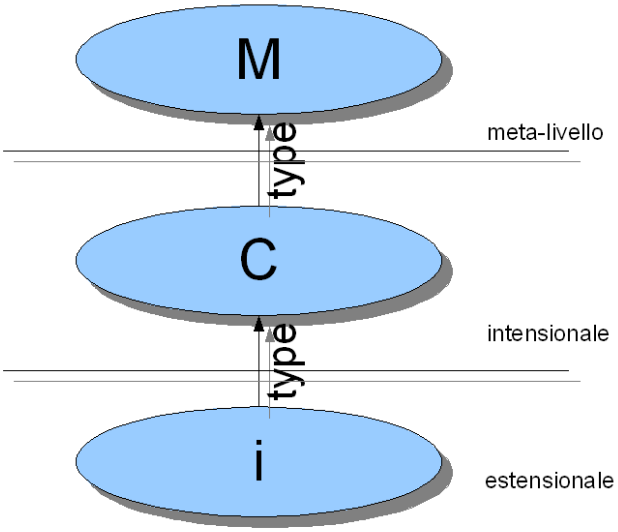
Per creare un metamodello in RDF possiamo usare due strategie: la prima consiste nel definire diversi livelli di risorse, l'altra è quella di far uso della reificazione.
Vediamo, innanzitutto, la prima strategia.

Come detto in precedenza una tecnica di metamodellazione consiste nel definire i diversi concetti da modellare su diversi livelli, a seconda della loro significatività.
Partiamo dal livello estensionale in cui vengono creati degli oggetti che definiamo come istanze di una classe. Dopodiché definiamo la classe stessa come istanza di un'altra classe ancora più generica.

In generale per creare una meta-asserzione adoperiamo i seguenti passi (utilizziamo nel seguito per descrivere una tripla il formato N3, caratterizzato dalla sequenza "soggetto predicato oggetto"):

i rdf:type C ← *i appartiene al livello estensionale*
C rdf:type rdfs:Class ←*C appartiene al livello intensionale*
M rdf:type rdfs:Class
C rdf:type M ←*M appartiene al metalivello*

In questo semplice esempio abbiamo creato un'istanza 'i' che abbiamo definito come istanza della classe 'C'. Quest'ultima, inoltre, è a sua volta un'istanza della classe M.
Ossia abbiamo creato due livelli di classi. Il primo è quello della classe C, che appartiene al tradizionale livello intensionale. Il secondo è quello di M che possiamo definire come una meta-classe, appartenente ad un metalivello superiore al livello della classe C.
Notiamo come la classe C non sia una sottoclasse di M, bensì ne è un'istanza e quindi, ovviamente, non sono sullo stesso piano concettuale.



L'altra strategia perseguibile per descrivere un metamodello consiste nel far uso della reificazione.
Come già visto nel Capitolo 1, la reificazione permette di fare meta-affermazioni attraverso l'uso di statement RDF che definiscono altri statement RDF.

L'esempio visto nella pagina precedente può essere realizzato anche tramite la reificazione. Vedremo come otterremo lo stesso valore per l'esempio anche nella nuova forma.

Utilizzando sempre il formato N3 per le triple otteniamo:

i rdf:type C ← *i appartiene al livello estensionale*
C rdf:type rdfs:Class ←*C appartiene al livello intensionale*
C rdf:type M
x rdf:type rdf:Statement
x rdf:subject M
x rdf:predicate rdf:type
x rdf:object rdfs:Class ←*M appartiene al metalivello*

dove x è una qualsiasi risorsa, anche un blank node che serve per definire la reificazione.
In questo esempio dopo aver definito 'i' come istanza della classe 'C', abbiamo dichiarato che 'C' è un'istanza di 'M'. Lo statement reificato, inoltre, ci dice che la risorsa identificata da M è una classe. Di conseguenza abbiamo ottenuto lo stesso valore espresso nella versione non reificata.
La procedura può essere iterata esprimendo tutti i concetti dell'esempio tramite reificazioni.

Capitolo 3 Esempi di metamodellazione

In questo capitolo vedremo alcuni esempi di metamodellazione in RDF.
Il primo esempio, in cui vengono descritte le relazioni tra animali ed il cibo, è trattato sia nel paragrafo 3.2 che nel paragrafo 3.3 nella versione con la reificazione. In tale maniera vedremo come sia nel meccanismo tradizionale che con la reificazione possiamo ottenere gli stessi risultati di metamodellazione.
A partire dal paragrafo 3.4, inoltre, verranno fatti degli esempi di metamodellazione partendo da alcuni pattern ontologici definiti nello studio [4].

3.1 Scenario applicativo

Per realizzare gli esempi di metamodellazione è stato scelto Jena un framework Java open source sviluppato presso il Semantic Web Research Bristol Lab della HP, che è stato ideato per lo sviluppo di applicazioni orientate al Web Semantico. Jena fornisce una serie di API Java utili per creare, interrogare e gestire in modo adeguato grafi RDF.
Jena usa classi opportune per rappresentare grafi, risorse, proprietà e letterali. Le interfacce che rappresentano risorse, proprietà e letterali sono chiamate rispettivamente “Resource”, “Property” e “Literal”. In Jena un grafo viene chiamato “model” ed è rappresentato attraverso l’interfaccia “Model”.
Per ogni grafo che deve essere definito, va creato un oggetto di tipo Model attraverso, ad esempio, il metodo createDefaultModel() della classe ModelFactory che crea un modello (cioè un grafo RDF) vuoto. Ogni elemento del grafo RDF va inserito nel modello appena creato aggiungendo l’eventuale risorsa o proprietà al modello tramite i metodi createResource() o createProperty().
Per associare poi una proprietà ad una risorsa si fa uso del metodo addProperty() che prende in ingresso due parametri, il primo è la Proprietà che associamo alla risorsa, il secondo può essere un’ altra Risorsa, un RDFNode o una Stringa.
Infine, per poter effettuare la verifica di alcune proprietà sul grafo RDF è stato necessario convertire il Model creato in un Model RDF completo in cui fossero inferite tutte le proprietà. A tale scopo è stato trovato conveniente l’uso del metodo ModelFactory.createRDFSModel che permette di derivare asserzioni RDF addizionali che derivano dal grafo RDF di base. In tal maniera query sul modello creato restituiscono non solo quegli statement già presenti nei dati originali, ma anche statement addizionali. Tali statement possono essere derivati dai dati usando i meccanismi di inferenza implementati dal reasoner di cui fa uso Jena. In particolar modo con ModelFactory.createRDFSModel sono implementate una serie di proprietà derivate RDFS come ad esempio le proprietà transitive e simmetriche di rdfs:subClassOf e rdfs:subProperty Of.

Con Jena è possibile effettuare interrogazioni SPARQL.
SPARQL è un linguaggio di query per RDF ed, in particolare, è un linguaggio di query “graph matching”. Ossia dato un grafo G, una query rappresenta un pattern che

trova un matching in G ed i valori di questo matching vengono processati e restituiti come risultato della query.
A tale scopo è d'aiuto l'application API ARQ le cui classi principali presenti nel package com.hp.hpl.jena.query sono:

- Query – una classe che serve per rappresentare la query dell'applicazione.E' un contenitore di tutti i dettagli dalla query. Oggetti della classe Query vengono normalmente creati chiamando uno del metodi della classe QueryFactory che provvede l'accesso a diversi parsers.
- QueryExecution – rappresenta l'esecuzione di una query
- QueryExecutionFactory – classe utile per ottenere le istanze di una QueryExection
- Per query di tipo SELECT:
  - QuerySolution – rappresenta una singola soluzione della query
  - ResultSet – restituisce tutte le soluzioni della query
  - ResultSetFormatter – permette di ottenere un ResultSet in diverse forme: semplice testo, grafo RDF (o Model RDF seguendo la terminologia di Jena) o in formato XML.

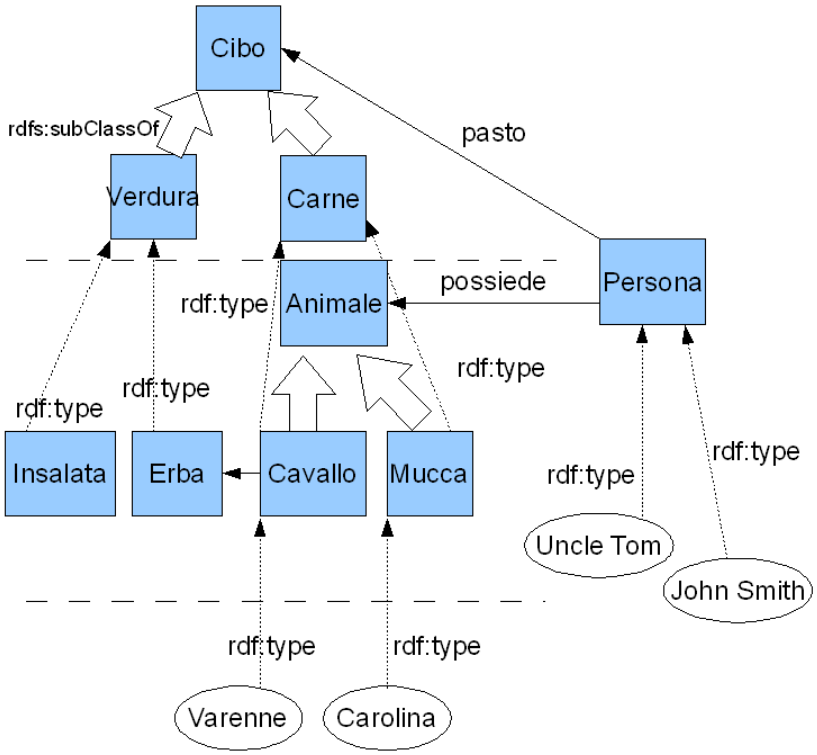
Il succitato ARQ è un modulo che implementa SPARQL nel framework Jena.
E' possibile effettuare tramite questo modulo interrogazioni anche in RDQL e in ARQ, un linguaggio proprietario della macchina. Prevede, inoltre, un motore SPARQL di tipo general purpose e consente l'accesso a motori remoti. E' stato scelto per la nostra implementazione perché è scritto in Java e soprattutto perché è integrato all'interno di Jena.

Per tutti gli esempi che seguiranno, dopo la descrizione del metamodello e delle relazioni tra i vari concetti, verrà presentata la realizzazione dello stesso in Jena e alcune query SPARQL (anche esse realizzate tramite Jena e ARQ) utili per descrivere le proprietà degli elementi del metamodello.

3.2 Animali - Cibo

In questo primo esempio di metamodellazione siamo interessati a descrivere il mondo degli animali ed un suo metalivello che è quello del cibo.
Ogni animale, infatti, può essere visto come fonte di cibo per le persone.
In un primo livello, dunque, descriveremo alcuni "concetti" relativi ad animali, quali ad esempio la mucca, e alcune loro proprietà come ad esempio ciò che mangiano. Nel livello superiore, il metalivello, avremo, invece, il "concetto" cibo di cui i vari tipi di animale ne sono istanza.
A cavallo tra i due livelli avremo il concetto "Persona" e le sue istanze che possono essere proprietari di animali e che mangiano del cibo. Un aspetto importante della metamodellazione consiste nella possibilità di "mischiare" i diversi livelli di modellazione, aspetto che rappresenta effettivamente la realtà, dove mentre alcuni concetti sono separati dal punto di vista semantico altri hanno interazioni con i diversi livelli. Proprio per tal motivo abbiamo realizzato la classe Persona a cavallo tra i due livelli di modellazione.

Per chiarire meglio il quadro disegnato, rappresentiamo i diversi livelli con il seguente grafico (per non appesantire eccessivamente il grafico le proprietà sono rappresentate con una freccia che va dal domain verso il range della proprietà).



Descriviamo ora il codice RDF del metamodello appena descritto:

```
<rdf:RDF
   xmlns:uri="http://somewhere#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
```

In questa porzione di codice, contenuta nel tag <rdf:RDF> sono presenti i prefissi per i namespace degli URI usati nel codice. Il namespace “uri” indica un ipotetico URI dove sono localizzate le risorse descritte nell’esempio.

```
 <rdf:Description rdf:about="http://somewhere#Cibo">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
<rdf:Description rdf:about="http://somewhere#Carne">
   <rdfs:subClassOf rdf:resource="http://somewhere#Cibo"/>
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Verdura">
   <rdfs:subClassOf rdf:resource="http://somewhere#Cibo"/>
```

```
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
```

Qui vengono definiti i metaconcetti Cibo, Carne e Verdura. Ognuno di questi metaconcetti vengono definiti come classi attraverso la proprietà rdf:type. Carne e Verdura, inoltre, sono sottoclassi di Cibo.

```
<rdf:Description rdf:about="http://somewhere#Erba">
  <rdf:type rdf:resource="http://somewhere#Verdura"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
<rdf:Description rdf:about="http://somewhere#Insalata">
  <rdf:type rdf:resource="http://somewhere#Verdura"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
<rdf:Description rdf:about="http://somewhere#Animale">
  <uri:mangia rdf:resource="http://somewhere#Erba"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#Mucca">
  <rdf:type rdf:resource="http://somewhere#Carne"/>
  <rdfs:subClassOf rdf:resource="http://somewhere#Animale"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#Cavallo">
  <rdf:type rdf:resource="http://somewhere#Carne"/>
  <rdfs:subClassOf rdf:resource="http://somewhere#Animale"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
 </rdf:Description>
```

Questa porzione di codice descrive il livello intensionale in cui sono presenti le classi Cavallo e Mucca, sottoclassi di Animale ed inoltre Erba ed Insalata e le loro proprietà. Ognuna di queste classi attraverso la proprietà rdf:type oltre ad essere definita come un'istanza di rdf:Class viene definita come istanza di un metaconcetto. In questo modo riusciamo a realizzare la metamodellazione. Ad esempio Mucca viene dichiarata istanza di Carne; in tal maniera ogni istanza di Mucca avrà come metaclasse di riferimento il metaconcetto Mucca.

```
<rdf:Description rdf:about="http://somewhere#Carolina">
  <rdf:type rdf:resource="http://somewhere#Mucca"/>
  <uri:nome>Carolina</uri:nome>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#Varenne">
  <uri:nome>Varenne</uri:nome>
  <rdf:type rdf:resource="http://somewhere#Cavallo"/>
 </rdf:Description>
```

Nelle righe precedenti abbiamo creato delle istanze delle classi Cavallo e Mucca.

```
<rdf:Description rdf:about="http://somewhere#Persona">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#JohnSmith">
   <uri:pasto rdf:resource="http://somewhere#Carne"/>
   <uri:possiede rdf:resource="http://somewhere#Varenne"/>
   <vcard:FN>John Smith</vcard:FN>
   <rdf:type rdf:resource="http://somewhere#Persona"/>
  </rdf:Description>
<rdf:Description rdf:about="http://somewhere#UncleTom">
   <uri:pasto rdf:resource="http://somewhere#Verdura"/>
   <uri:possiede rdf:resource="http://somewhere#Carolina"/>
   <vcard:FN>Uncle Tom</vcard:FN>
   <rdf:type rdf:resource="http://somewhere#Persona"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#possiede">
   <rdfs:domain rdf:resource="http://somewhere#Persona"/>
   <rdfs:range rdf:resource="http://somewhere#Animale"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#pasto">
   <rdfs:range rdf:resource="http://somewhere#Cibo"/>
   <rdfs:domain rdf:resource="http://somewhere#Persona"/>
  </rdf:Description>
```

In quest'ultima porzione del codice RDF costruiamo il concetto Persona che definiamo come classe e due sue istanze. Inoltre definiamo due proprietà: "possiede" che ha per dominio la classe Persona e come range la classe Animale e "pasto" che come la precedente ha come dominio la classe Persona, mentre come range una risorsa di tipo Cibo.

Per maggiori informazioni su come realizzare un grafo RDF (ed in particolare quello di questo esempio) con le api Jena rimandiamo all'Appendice A.

Vediamo ora alcune query interessanti sull'esempio che stiamo studiando, utili per verificare alcune proprietà dei diversi livelli di metamodellazione.

La prima query che proponiamo verifica se ci sono esseri che mangiano e cosa mangiano.

```
1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX  uri:  <http://somewhere#>
3 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
4
5 SELECT  ?nome ?cibo
6 WHERE
7   {   { ?x  uriFN:FN   ?nome ;
```

```
 8             uri:pasto  ?cibo .
 9       }
10     UNION
11       { ?x     uri:nome     ?nome ;
12                uriRDF:type  ?type .
13         ?type  uri:mangia   ?cibo .
14       }
15   }
```

Con questa query di tipo SELECT verifichiamo se esistono o delle persone che fanno un pasto o degli animali che mangiano del cibo. Il ritorno della select, infatti, coincide con le variabili "?nome" che può corrispondere al nome di una Persona o al nome di un Animale e "?cibo" che può consistere in un elemento del metalivello o del livello intensionale. In tal maniera verifichiamo come elementi di piani diversi siano tra loro connessi.

Il risultato della query è il seguente:

```
Uncle Tom    http://somewhere#Verdura
John Smith    http://somewhere#Carne
Carolina    http://somewhere#Erba
Varenne    http://somewhere#Erba
```

Nella prossima query verifichiamo se esistono delle persone, proprietari di animali, che mangiano del cibo appartenente alla stessa tipologia di quello mangiato dai propri animali.

La query che permette di fare questa verifica è la seguente:

```
 1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX  uri:  <http://somewhere#>
 3 PREFIX  uriRDFS: <http://www.w3.org/2000/01/rdf-schema#>
 4 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
 5
 6 SELECT  ?nome ?nomeAnimale ?cibo
 7 WHERE
 8   { ?x        uri:possiede  ?animale ;
 9               uriFN:FN      ?nome ;
10               uri:pasto     ?cibo .
11     ?animale  uri:nome      ?nomeAnimale ;
12               uriRDF:type   ?tipoAnimale .
13     ?tipoAnimale  uri:mangia  ?ciboAnimale .
14     ?ciboAnimale  uriRDF:type  ?cibo .
15   }
```

Il risultato della query applicata alla nostra base di conoscenza è:

```
Uncle Tom    Carolina    http://somewhere#Verdura
```

Effettivamente Uncle Tom mangia la Verdura, così come il suo animale, la mucca Carolina, mangia l'erba che è un'istanza di Verdura.

Nell'ultima query del nostro esempio ci chiediamo se esistono dei proprietari che teoricamente possano mangiare i propri animali.

```
 1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX  uri:  <http://somewhere#>
 3 PREFIX  uriRDFS: <http://www.w3.org/2000/01/rdf-schema#>
 4 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
 5
 6 SELECT  ?nome ?nomeAnimale
 7 WHERE
 8   { ?x        uri:possiede  ?animale ;
 9               uriFN:FN      ?nome ;
10               uri:pasto     ?cibo .
11     ?animale  uriRDF:type   ?type ;
12               uri:nome      ?nomeAnimale .
13     ?type     uriRDF:type   ?cibo .
14       }
```

Il risultato della query applicata alla nostra base di conoscenza è il seguente:

```
    John Smith    Varenne
```

John Smith, proprietario di Varenne, mangia la carne ed il suo cavallo Varenne è un animale che come cibo è carne. Dunque, teoricamente, John Smith potrebbe mangiare il proprio animale.

3.3 Animali – Cibo realizzato con la reificazione

Come detto nel Capitolo 2 la metamodellazione in RDF può essere realizzata anche attraverso la reificazione. Per tal motivo vediamo il precedente esempio in cui erano metamodellati i concetti relativi al mondo degli animali ed i metaconcetti del cibo attraverso la reificazione.
Ovviamente la struttura dei diversi livelli è la stessa dell'esempio precedente, per cui i grafici dei livelli sono gli stessi di quelli visti nel paragrafo 3.2.

In questo paragrafo vedremo il codice RDF che realizza la reificazione dell'esempio Animali – Cibo.

```
<rdf:RDF
  xmlns:uri="http://somewhere#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#" >
<rdf:Description rdf:nodeID="A0">
   <rdf:subject rdf:resource="http://somewhere#Cibo"/>
   <rdf:predicate rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#type"/>
   <rdf:object rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
   <rdf:subject rdf:resource="http://somewhere#Verdura"/>
   <rdf:predicate rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#type"/>
   <rdf:object rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A2">
   <rdf:subject rdf:resource="http://somewhere#Carne"/>
   <rdf:predicate rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#type"/>
   <rdf:object rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement"/>
  </rdf:Description>
```

Nel codice precedente dopo la definizione dei namespace degli URI usati, vengono definiti tramite la reificazione i metaconcetti Cibo, Carne e Verdura. Notiamo come ogni metaconcetto è il “subject” dello statement reificato e tramite il “predicate” rdf:type definiamo ciascun metaconcetto come una classe. Infatti l’ “object” dello statement è rdfs:Class.

```
   <rdf:Description rdf:about="http://somewhere#Erba">
     <rdf:type rdf:resource="http://somewhere#Verdura"/>
     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Insalata">
     <rdf:type rdf:resource="http://somewhere#Verdura"/>
     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Animale">
     <uri:mangia rdf:resource="http://somewhere#Erba"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#Mucca">
     <rdf:type rdf:resource="http://somewhere#Carne"/>
     <rdfs:subClassOf rdf:resource="http://somewhere#Animale"/>
     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
```

```
    <rdf:Description rdf:about="http://somewhere#Cavallo">
      <rdf:type rdf:resource="http://somewhere#Carne"/>
      <rdfs:subClassOf rdf:resource="http://somewhere#Animale"/>
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Carolina">
      <rdf:type rdf:resource="http://somewhere#Mucca"/>
      <uri:nome>Carolina</uri:nome>
    </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#Varenne">
      <uri:nome>Varenne</uri:nome>
      <rdf:type rdf:resource="http://somewhere#Cavallo"/>
   </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Persona">
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#JohnSmith">
      <uri:pasto rdf:resource="http://somewhere#Carne"/>
      <uri:possiede rdf:resource="http://somewhere#Varenne"/>
      <vcard:FN>John Smith</vcard:FN>
      <rdf:type rdf:resource="http://somewhere#Persona"/>
    </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#UncleTom">
      <uri:pasto rdf:resource="http://somewhere#Verdura"/>
      <uri:possiede rdf:resource="http://somewhere#Carolina"/>
      <vcard:FN>Uncle Tom</vcard:FN>
      <rdf:type rdf:resource="http://somewhere#Persona"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#possiede">
      <rdfs:domain rdf:resource="http://somewhere#Persona"/>
      <rdfs:range rdf:resource="http://somewhere#Animale"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#pasto">
      <rdfs:range rdf:resource="http://somewhere#Cibo"/>
      <rdfs:domain rdf:resource="http://somewhere#Persona"/>
  </rdf:Description>
</rdf:RDF>
```

Il resto del codice mantiene le stesse caratteristiche dell'esempio precedente.

A questo punto applicando le query SPARQL già viste nel paragrafo precedente otteniamo gli stessi risultati. Ciò conferma che le due tecniche viste sono equivalenti nella realizzazione della metamodellazione.
Infatti:

```
1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  2 PREFIX  uri:  <http://somewhere#>
  3 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
  4
  5 SELECT  ?nome ?cibo
  6 WHERE
  7   {   { ?x  uriFN:FN   ?nome ;
```

```
 8            uri:pasto  ?cibo .
 9       }
10     UNION
11       { ?x     uri:nome     ?nome ;
12                uriRDF:type  ?type .
13         ?type  uri:mangia   ?cibo .
14       }
15   }
```

Da come risultato:

```
Uncle Tom    http://somewhere#Verdura
John Smith    http://somewhere#Carne
Carolina    http://somewhere#Erba
Varenne    http://somewhere#Erba
```

come nella versione senza reificazione.

Ed inoltre:

```
 1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX  uri:  <http://somewhere#>
 3 PREFIX  uriRDFS: <http://www.w3.org/2000/01/rdf-schema#>
 4 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
 5
 6 SELECT  ?nome ?nomeAnimale ?cibo
 7 WHERE
 8   { ?x        uri:possiede  ?animale ;
 9               uriFN:FN      ?nome ;
10               uri:pasto     ?cibo .
11     ?animale  uri:nome      ?nomeAnimale ;
12               uriRDF:type   ?tipoAnimale .
13     ?tipoAnimale  uri:mangia  ?ciboAnimale .
14     ?ciboAnimale  uriRDF:type  ?cibo .
15   }
```

restituisce:

```
Uncle Tom    Carolina    http://somewhere#Verdura
```

ed inoltre,

```
 1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX  uri:  <http://somewhere#>
 3 PREFIX  uriRDFS: <http://www.w3.org/2000/01/rdf-schema#>
 4 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
 5
 6 SELECT  ?nome ?nomeAnimale
 7 WHERE
 8   { ?x        uri:possiede  ?animale ;
 9               uriFN:FN      ?nome ;
10               uri:pasto     ?cibo .
11     ?animale  uriRDF:type   ?type ;
12               uri:nome      ?nomeAnimale .
13     ?type     uriRDF:type   ?cibo .
15       }
```

restituisce:

```
John Smith    Varenne
```

esattamente come nel caso non reificato.

3.4 Tipi di entità

In questo e nel seguente esempio vedremo alcuni pattern ontologici definiti nello studio [4].
Il primo di questi pattern intende definire le varie tipologie di entità.

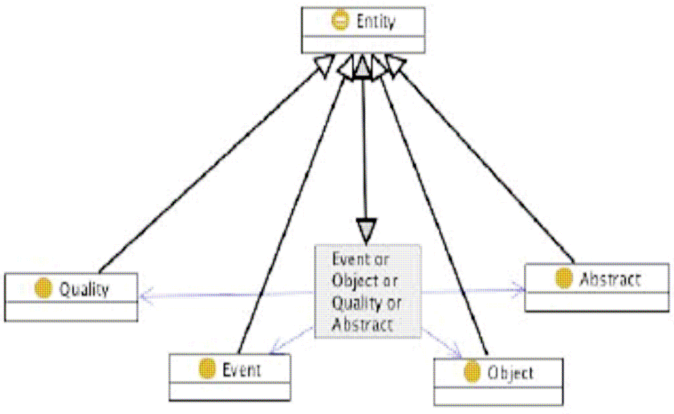
Lo schema del pattern è il seguente:

Nome: Tipi di entità

Scopo: serve ad identificare e categorizzare i tipi più generali degli oggetti che fanno parte del dominio di un discorso.

Requisiti: che tipo di entità è questa? E' un evento o un oggetto? E' un valore astratto o la qualità di un'entità?

Diagramma:



Elementi: i tipi di entità consistono dei seguenti elementi:

- *Abstract*, un'entità che non può essere localizzata nel tempo o nello spazio. Ad esempio entità matematiche, elementi formali della semantica,etc.

- *Event*, qualsiasi processo, evento o stato fisico, sociale o mentale. Gli eventi possono essere classificati in diversi modi, ad esempio basandosi sulla tipologia (ad esempio statico o continuo) o su coloro che vi partecipano (ad esempio persone, elementi astratti,etc.)

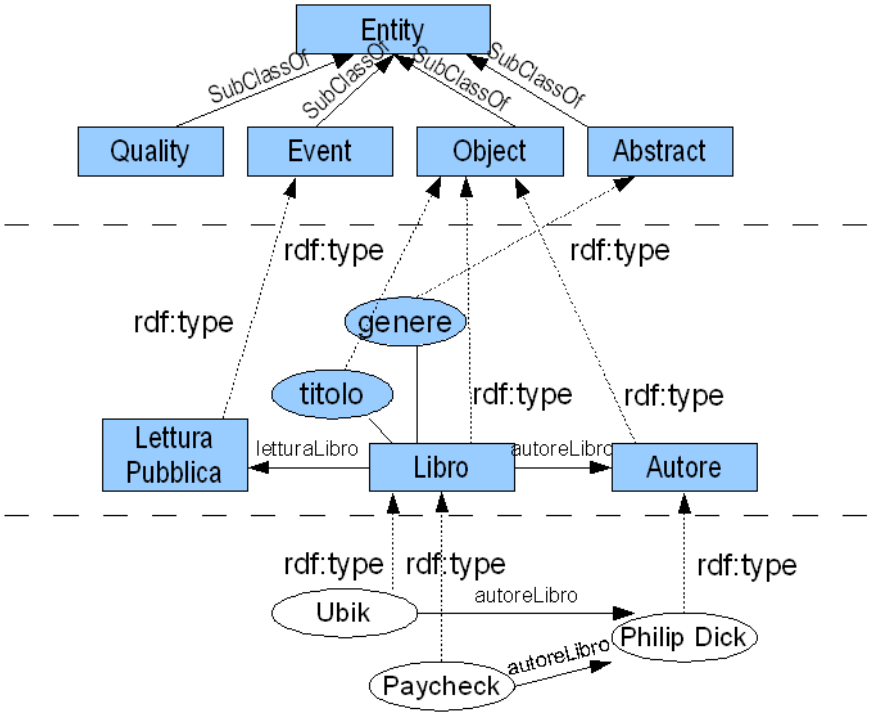
- *Object*, qualsiasi elemento o sostanza fisica.

- *Qualità*, un qualsiasi aspetto di un'entità (ma non una parte di esso) che non può esistere senza tale entità. Ad esempio la superficie di uno specifico oggetto fisico è una entità .

Gli elementi sono definiti come disgiunti gli uni dagli altri. Sono, inoltre, una partizione esaustiva della classe entità

Conseguenze: la tipologia base di ogni elemento della conoscenza è sempre conosciuto.

Il pattern ed i suoi elementi che abbiamo appena descritto fanno parte del metalivello dell'esempio che vogliamo ora descrivere. Al livello intensionale, invece, rappresentiamo il mondo dei libri. Il concetto principale in questo livello è quello di Libro che ha una serie di proprietà quali genere e tipo che lo caratterizzano ed inoltre è relazionato con i concetti Autore (che rappresenta l'autore di un libro) e Lettura Pubblica (che rappresenta l'evento in cui viene fatta una lettura aperta al pubblico del libro in questione).
Possiamo rappresentare il metamodello appena descritto attraverso il seguente grafico.



Nel grafico con i pallini colorati di blu abbiamo rappresentato alcune proprietà significative della classe Libro che hanno come domain Libro e che rappresentano dei concetti istanze dei livelli superiori.

Riportiamo ora il codice RDF del metamodello appena descritto.

```
<rdf:RDF
   xmlns:uri="http://somewhere#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
   xmlns:dc="http://purl.org/dc/elements/1.1/" >
```

Il solito codice relativo ai namespace.

```
<rdf:Description rdf:about="http://somewhere#Entity">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
<rdf:Description rdf:about="http://somewhere#Event">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://somewhere#Entity"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Abstract">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://somewhere#Entity"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Object">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://somewhere#Entity"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Quality">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://somewhere#Entity"/>
  </rdf:Description>
```

In questa porzione di codice abbiamo descritto il metalivello, rappresentando i concetti Entity ed i suoi sottoconcetti (definiti come sue sottoclassi) Event, Abstract,Object e Quality.

```
  <rdf:Description rdf:about="http://somewhere#Libro">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdf:type rdf:resource="http://somewhere#Object"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#genere">
   <rdf:type rdf:resource="http://somewhere#Abstract"/>
   <rdfs:domain rdf:resource="http://somewhere#Libro"/>
  </rdf:Description>
```

```
<rdf:Description rdf:about="http://somewhere#titolo">
  <rdf:type rdf:resource="http://somewhere#Abstract"/>
  <rdfs:domain rdf:resource="http://somewhere#Libro"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#autoreLibro">
  <rdfs:range rdf:resource="http://somewhere#Autore"/>
  <rdfs:domain rdf:resource="http://somewhere#Libro"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#Autore">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdf:type rdf:resource="http://somewhere#Object"/>
  </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#LetturaPubblica">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdf:type rdf:resource="http://somewhere#Event"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#letturaLibro">
  <rdfs:range rdf:resource="http://somewhere#Libro"/>
  <rdfs:domain rdf:resource="http://somewhere#LetturaPubblica"/>
 </rdf:Description>
```

In questa parte di codice abbiamo rappresentato i concetti Libro, con le sue proprietà Autore e LetturaPubblica. Per ognuno di questi elementi abbiamo rappresentato anche la relazione che sussiste con il metalivello corrispondente. In particolare possiamo vedere, ad esempio, come Libro è definita come un'istanza del metaconcetto Object e la sua proprietà genere è, invece, un'istanza di Abstract.

```
 <rdf:Description rdf:about="http://somewhere#Ubik">
  <uri:titolo>Ubik</uri:titolo>
  <rdf:type rdf:resource="http://somewhere#Libro"/>
  <uri:genere>http://somewhere#Fantascienza</uri:genere>
  <uri:autoreLibro rdf:resource="http://somewhere#PDick"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#Paycheck">
  <rdf:type rdf:resource="http://somewhere#Libro"/>
  <uri:genere>http://somewhere#Fantascienza</uri:genere>
  <uri:autoreLibro rdf:resource="http://somewhere#PDick"/>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#lettura Ubik">
  <rdf:type rdf:resource="http://somewhere#LetturaPubblica"/>
  <uri:letturaLibro rdf:resource="http://somewhere#Ubik"/>
  <dc:date>2008-07-25</dc:date>
 </rdf:Description>
 <rdf:Description rdf:about="http://somewhere#PDick">
  <rdf:type rdf:resource="http://somewhere#Autore"/>
  <vcard:FN>Philip Kindred Dick</vcard:FN>
 </rdf:Description>
</rdf:RDF>
```

Quest'ultima porzione di codice del nostro esempio serve per polare il livello estensionale con alcuni elementi istanze dei concetti definiti al livello intensionale.

Una query interessante da realizzare in SPARQL consiste nel verificare quali sono gli elementi presenti nella nostra base di conoscenza e a quale sottocategoria del metaconcetto Entità fanno riferimento.

La query che realizza ciò è la seguente:

```
 1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 2 PREFIX  uri:  <http://somewhere#>
 3 PREFIX  uriRDFS: <http://www.w3.org/2000/01/rdf-schema#>
 4 PREFIX  uriFN: <http://www.w3.org/2001/vcard-rdf/3.0#>
 5
 6 SELECT DISTINCT  ?x ?type
 7 WHERE
 8   { ?x     uriRDF:type         ?type .
 9     ?type  uriRDFS:subClassOf  uri:Entity .
10   }
```

L'esecuzione della query sul grafo RDF descritto in precedenza restituisce il seguente risultato.

```
http://somewhere#PDick    http://somewhere#Object
http://somewhere#Ubik    http://somewhere#Object
http://somewhere#lettura Ubik    http://somewhere#Event
http://somewhere#Paycheck    http://somewhere#Object
http://somewhere#Libro    http://somewhere#Object
http://somewhere#titolo    http://somewhere#Abstract
http://somewhere#Autore    http://somewhere#Object
http://somewhere#genere    http://somewhere#Abstract
http://somewhere#LetturaPubblica    http://somewhere#Event
```

Il risultato restituisce, come prospettato, tutti gli elementi del nostro metamodello sia del livello intensionale (come ad esempio il concetto Autore) che del livello estensionale (come ad esempio il libro Ubik) e la loro relativa tipologia d'entità.

Per poter ottenere questa particolare query è stato necessario introdurre all'interno del codice Java della definizione del grafo RDF la seguente regola di inferenza:

Da ?a rdf:type ?b & ?b rdf:type ?c otteniamo che ?a rdf:type ?c
realizzata con il seguente codice Java.

```
String rules = "[rule1: (?a rdf:type ?b) (?b rdf:type ?c) -> (?a rdf:type ?c)]";
    GenericRuleReasoner reasoner = new
GenericRuleReasoner(Rule.parseRules(rules));
    reasoner.setDerivationLogging(true);
    InfModel inf = ModelFactory.createInfModel(reasoner, model);
```

3.5 Description

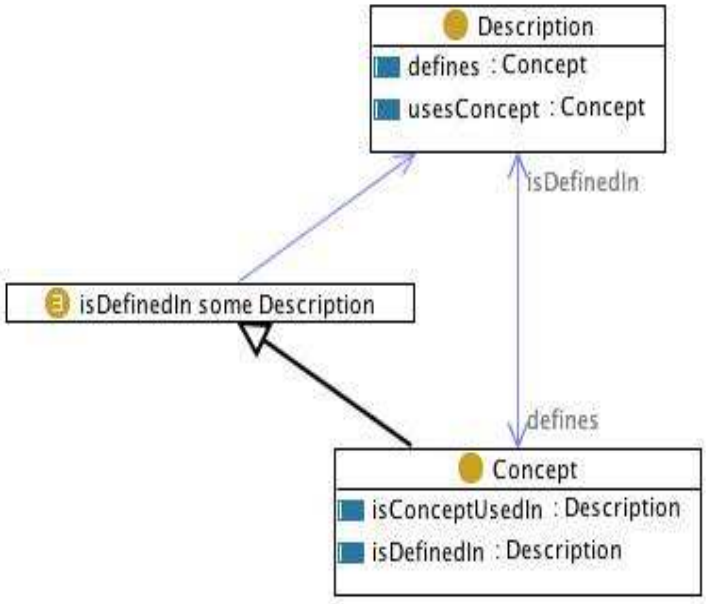
Presentiamo un nuovo pattern ontologico, utile per la descrizione di concetti.

Nome: Description.

Scopo: serve per rappresentare formalmente un concetto o un contesto descrittivo.

Requisiti: Quale sono le assunzioni, per le quali un determinato elemento viene descritto? Quali sono i concetti coinvolti nella descrizione di un certo elemento? Quale è l'interpretazione di questa osservazione?

Diagramma:



Elementi:

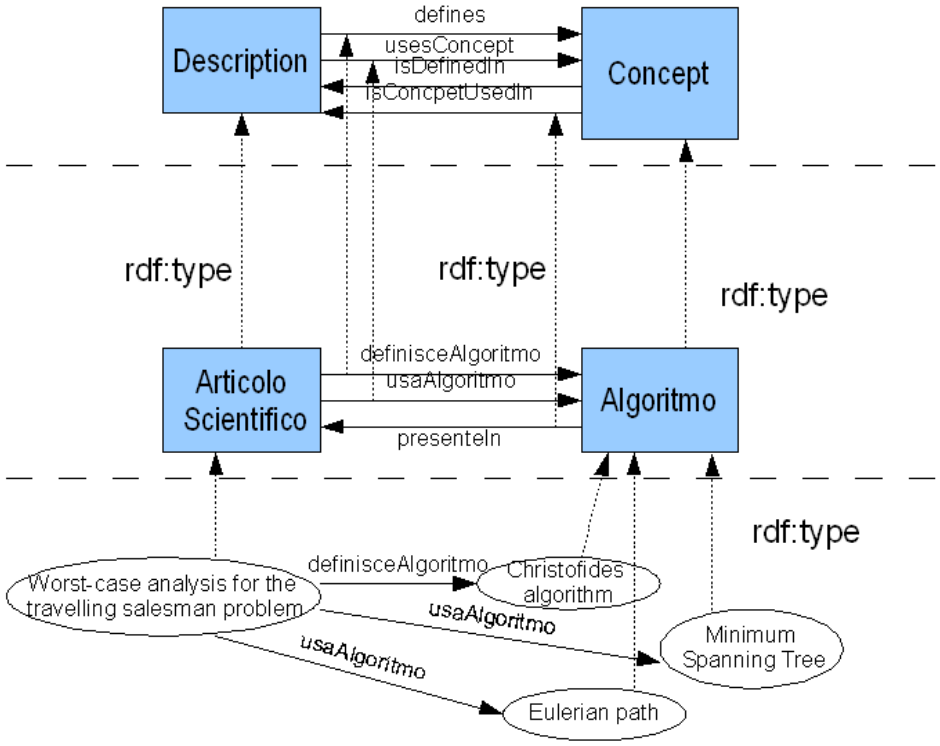
- *Description*, una descrizione rappresenta una concettualizzazione . Può essere pensata anche come un contesto descrittivo che definisce alcuni concetti. Ad esempio un Piano è la descrizione di alcune azioni che devono essere eseguite da agenti in una determinata maniera, con certi parametri,etc.

- *Concept,* può essere un idea,una nozione ed èd definita nella descrizione. Una volta definito, un concetto può essere usato in altre descrizioni.

- *IsDefinedIn*, rappresenta una relazione tra una descrizione ed un concetto.

Conseguenze: questo pattern permette di rappresentare sia un contesto sia gli elementi che lo caratterizzano e sono coinvolti nel contesto stesso.

Partendo dal pattern ontologico appena presentato, che rappresenterà il nostro metalivello, descriviamo ora il livello intensionale. In questo livello rappresenteremo gli Articoli Scientifici e gli Algoritmi presenti in tali articoli.
In ogni articolo scientifico viene definito un algoritmo e vengono usati altri algoritmi.
Possiamo rappresentare il metamodello appena descritto attraverso il seguente grafico.



Riportiamo ora il codice RDF del metamodello.

```
<rdf:RDF
   xmlns:uri="http://somewhere#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description rdf:about="http://somewhere#Description">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Concept">
```

```
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#isDefinedIn">
    <rdfs:domain rdf:resource="http://somewhere#Concept"/>
    <rdfs:range rdf:resource="http://somewhere#Description"/>
  </rdf:Description>
<rdf:Description rdf:about="http://somewhere#usesConcept">
    <rdfs:domain rdf:resource="http://somewhere#Description"/>
    <rdfs:range rdf:resource="http://somewhere#Concept"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#isConceptUsedIn">
    <rdfs:domain rdf:resource="http://somewhere#Concept"/>
    <rdfs:range rdf:resource="http://somewhere#Description"/>
  </rdf:Description>
<rdf:Description rdf:about="http://somewhere#defines">
    <rdfs:domain rdf:resource="http://somewhere#Description"/>
    <rdfs:range rdf:resource="http://somewhere#Concept"/>
  </rdf:Description>
```

In questa sezione del codice RDF vengono definiti i metaconcetti Description e Concept e le proprietà relative con i rispettivi domain e range.

```
  <rdf:Description rdf:about="http://somewhere#Algoritmo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdf:type rdf:resource="http://somewhere#Concept"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#usa">
    <rdf:type rdf:resource="http://somewhere#usesConcept"/>
    <rdfs:domain rdf:resource="http://somewhere#Articolo Scientifico"/>
    <rdfs:range rdf:resource="http://somewhere#Algoritmo"/>
  </rdf:Description>
    <rdf:Description rdf:about="http://somewhere#definisceAlgoritmo">
    <rdf:type rdf:resource="http://somewhere#defines"/>
    <rdfs:domain rdf:resource="http://somewhere#Articolo Scientifico"/>
    <rdfs:range rdf:resource="http://somewhere#Algoritmo"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#autore">
    <rdfs:domain rdf:resource="http://somewhere#Articolo Scientifico"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#Articolo Scientifico">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdf:type rdf:resource="http://somewhere#Description"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://somewhere#presenteIn">
    <rdf:type rdf:resource="http://somewhere#isConceptUsedIn"/>
    <rdfs:domain rdf:resource="http://somewhere#Algoritmo"/>
    <rdfs:range rdf:resource="http://somewhere#Articolo Scientifico"/>
  </rdf:Description>
```

Qui sono stati definiti i concetti del livello intensionale Algoritmo e Articolo Scientifico.

```
<rdf:Description rdf:about="http://somewhere#Worst-case analysis of a new heuristic for the travelling salesman problem">
  <rdf:type rdf:resource="http://somewhere#Articolo Scientifico"/>
  <uri:usa rdf:resource="http://somewhere#Eulerian path"/>
  <uri:usa rdf:resource="http://somewhere#minimum spanning tree"/>
  <uri:definisceAlgoritmo rdf:resource="http://somewhere#Christofides algorithm"/>
</rdf:Description>
<rdf:Description rdf:about="http://somewhere#Solutio problematis ad geometriam situs pertinentis">
  <uri:definisceAlgoritmo rdf:resource="http://somewhere#Eulerian path"/>
</rdf:Description>
<rdf:Description rdf:about="http://somewhere#State of the art Algorithms for the Minimum Spanning Trees">
  <uri:definisceAlgoritmo rdf:resource="http://somewhere#minimum spanning tree"/>
</rdf:Description>
<rdf:Description rdf:about="http://somewhere#minimum spanning tree">
  <rdf:type rdf:resource="http://somewhere#Algoritmo"/>
</rdf:Description>
<rdf:Description rdf:about="http://somewhere#Christofides algorithm">
  <rdf:type rdf:resource="http://somewhere#Algoritmo"/>
  <uri:autore>Christofides</uri:autore>
</rdf:Description>
<rdf:Description rdf:about="http://somewhere#Eulerian path">
  <rdf:type rdf:resource="http://somewhere#Algoritmo"/>
  <uri:autore>Euler</uri:autore>
</rdf:Description>
</rdf:RDF>
```

In quest'ultima porzione di codice sono presentate alcune istanze delle classi Algoritmo e Articolo Scientifico.

Vediamo ora una query sull'esempio appena illustrato.
Vogliamo verificare tutte gli elementi che hanno come metaconcetto la classe Concept e sono relazionati con elementi che hanno come metaclasse Description attraverso una qualche proprietà.

```
1 PREFIX  uriRDF: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX  uri:  <http://somewhere#>
3
4 SELECT DISTINCT  ?x
5 WHERE
6   { ?x  uriRDF:type  uri:Concept .
7     ?y  uriRDF:type  uri:Description ;
8         ?p           ?x .
9   }
```

che fornisce come risultato

```
http://somewhere#Christofides algorithm    http://somewhere#definisceAlgoritmo
    http://somewhere#minimum spanning tree    http://somewhere#usa
    http://somewhere#Eulerian path    http://somewhere#usa
```

infatti il risultato della query fornisce algoritmi (quindi istanze di Algoritmo a sua volta istanza di Concept) , che sono connessi ad un'istanza di Articolo Scientifico(a sua volta istanza di Description) tramite o la proprietà "definisceAlgoritmo" o la proprietà "usa".

Appendice A

In questa appendice presentiamo commentato il codice Java per la realizzazione dei grafi RDF e per le query SPARQL.

A.1 codice Animale- Cibo

```
package seminarioIS;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;
import com.hp.hpl.jena.sparql.util.IndentedWriter;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.query.ResultSet;

public class AnimaliCibo {

	public static final String uri = "http://somewhere#";
	public static final String uriFN = "http://www.w3.org/2001/vcard-rdf/3.0#";
	public static final String uriRDF =    "http://www.w3.org/1999/02/22-rdf-
syntax-ns#";
	public static final String uriRDFS =  RDFS.getURI();
	public static final String NL = System.getProperty("line.separator") ;

//metodo main. Viene creato un modello vuoto tramite il metodo createModel() e
//stampato a video il relativo codice RDF.
//Per poter gestire le eventuali inferenze tramite le proprietà RDFS il modello viene
//convertito  in un RDFSModel.
//Infine vengono eseguite alcune query SPARQL che stampano a video il codice
//SPARQL ed il risultato della query

	public static void main(String[] args) {
		Model model = createModel() ;
		model.write(System.out);
		Model x = ModelFactory.createRDFSModel(model);
		System.out.println("\n\n***Query 1***\n\n");
		queryProprietarioMangiaSuoAnimale(model);
		System.out.println("\n\n***Query 2***\n\n");
		queryProprietarioMangiaComeSuoAnimale(x);
		System.out.println("\n\n***Query 3***\n\n");
		queryChiMangia(x);

	}
```

```
        //Il metodo createModel() costruisce il grafo RDF dell'esempio.

        public static Model createModel()
{
                //Viene settato il modello e vengono creati i dati
                Model m = ModelFactory.createDefaultModel() ;
                m.setNsPrefix( "uri", uri );

    //Definizione delle Resources del livello intensionale

    Resource cavallo = m.createResource(uri + "Cavallo") ;
    Resource mucca = m.createResource(uri + "Mucca") ;
    Resource erba = m.createResource(uri + "Erba") ;
    Resource insalata = m.createResource(uri + "Insalata") ;
    Resource animale = m.createResource(uri + "Animale") ;

    //Vengono create le Property del livello intensionale con I relativi range e
       domain

    Property mangia = m.createProperty(uri + "mangia") ;
    mucca.addProperty(RDF.type,RDFS.Class) ;
    cavallo.addProperty(RDF.type,RDFS.Class) ;
    erba.addProperty(RDF.type,RDFS.Class) ;
    insalata.addProperty(RDF.type,RDFS.Class) ;
    animale.addProperty(mangia, erba) ;
    mucca.addProperty(RDFS.subClassOf,animale);
    cavallo.addProperty(RDFS.subClassOf,animale);

    Resource persona = m.createResource(uri + "Persona") ;
    Property possiede  = m.createProperty( uri + "possiede" ) ;
    possiede.addProperty(RDFS.domain , persona);
    persona.addProperty(RDF.type,RDFS.Class);

    //Vengono definiti gli elementi del livello estensionale

    Resource person1 = m.createResource(uri + "JohnSmith") ;
    person1.addProperty(RDF.type , persona)
                    .addProperty(VCARD.FN , "John Smith");
    Resource person2 = m.createResource(uri + "UncleTom") ;
    person2.addProperty(RDF.type , persona)
                    .addProperty(VCARD.FN , "Uncle Tom");
    Resource horse1 = m.createResource(uri+ "Varenne") ;
    Resource cow1 = m.createResource(uri+ "Carolina") ;
    horse1.addProperty(RDF.type , cavallo);
    person1.addProperty(possiede,horse1) ;
    Property nomeAnimale = m.createProperty(uri + "nome") ;
    horse1.addProperty(nomeAnimale, "Varenne") ;
    cow1.addProperty(nomeAnimale, "Carolina") ;
    cow1.addProperty(RDF.type , mucca);
    person2.addProperty(possiede,cow1) ;
```

```
//      Tutte le definizioni relative alle Risorse del metalivello

        Resource carne = m.createResource(uri + "Carne") ;
        Resource verdura = m.createResource(uri + "Verdura") ;
        Resource cibo = m.createResource(uri + "Cibo") ;

        Property pasto = m.createProperty(uri + "pasto") ;
        pasto.addProperty(RDFS.domain, persona) ;
        pasto.addProperty(RDFS.range, cibo) ;

        carne.addProperty(RDF.type,RDFS.Class);
        carne.addProperty(RDFS.subClassOf ,cibo );
        verdura.addProperty(RDF.type,RDFS.Class);
        verdura.addProperty(RDFS.subClassOf ,cibo );
        cibo.addProperty(RDF.type,RDFS.Class);

        //connessione tra il metalivello ed il livello sottostante

        cavallo.addProperty(RDF.type , carne) ;
        mucca.addProperty(RDF.type , carne) ;
        erba.addProperty(RDF.type , verdura) ;
        insalata.addProperty(RDF.type , verdura) ;


        person1.addProperty(pasto, carne) ;
        person2.addProperty(pasto, verdura) ;

        return m ;
    }

          public static void queryProprietarioMangiaSuoAnimale(Model model)     {

              String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> PREFIX uriRDFS: <"+uriRDFS+">" ;
              String queryString = prolog + NL +
        "SELECT   ?nome ?nomeAnimale WHERE { ?x uri:"+
          model.getProperty("possiede")+"      ?animale ; " +
          "uriFN:"+model.getProperty("FN")+" ?nome; uri:"+
                            model.getProperty("pasto")+" ?cibo. ?animale uriRDF:"+
                    model.getProperty("type")+"?type; uri:"+
                    model.getProperty("nome")+"?nomeAnimale. ?type uriRDF:" +
                            model.getProperty("type")+" ?cibo. }" ;
              Query query = QueryFactory.create(queryString) ;


        query.serialize(new IndentedWriter(System.out,true)) ;
        System.out.println() ;
        QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


        System.out.println("ProprietarioMangia: ") ;
```

```
        try {

           ResultSet rs = qexec.execSelect() ;

           // L'ordine dei risultati è indefinito

           for ( ; rs.hasNext() ; )
           {
              QuerySolution rb = rs.nextSolution() ;


              RDFNode x = rb.get("nome") ;

              // Controllo il tipo del risultato
              if ( x.isLiteral() )
              {
                 Literal titleStr = (Literal)x  ;
                 System.out.print("    "+titleStr) ;
              }
              else if ( x.isResource() )
              {
                Resource r = (Resource)x ;
                System.out.print("    "+r.toString()) ;
              }
              else
                 System.out.print("  "+x) ;


              RDFNode x2 = rb.get("nomeAnimale") ;

              // Controllo il tipo del risultato
              if ( x2.isLiteral() )
              {
                 Literal titleStr2 = (Literal)x2  ;
                 System.out.println("    "+titleStr2) ;
              }
              else if ( x2.isResource() )
              {
                Resource r = (Resource)x2 ;
                System.out.println("    "+r.toString()) ;
              }
              else
                 System.out.println("  "+x2) ;

           }
        }
        finally
        {
           // gli oggetti di tipo QueryExecution devono essere chiusi per liberare risorse
//di sistema
           qexec.close() ;
```

```
    }
      }

      public static void queryProprietarioMangiaComeSuoAnimale(Model model)
      {
        String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> PREFIX uriRDFS: <"+uriRDFS+">" ;
        String queryString = prolog + NL +
    "SELECT   ?nome ?nomeAnimale ?cibo WHERE { ?x uri:"+
      model.getProperty("possiede")+"      ?animale ; " +
      "uriFN:"+model.getProperty("FN")+" ?nome; uri:"+
                    model.getProperty("pasto")+" ?cibo.?animale uri:"+
              model.getProperty("nome")+"?nomeAnimale; uriRDF:"+
              model.getProperty("type")+" ?tipoAnimale. ?tipoAnimale  uri:"+
        model.getProperty("mangia")+"?ciboAnimale. ?ciboAnimale uriRDF:" +
                    model.getProperty("type")+" ?cibo.  }" ;
        Query query = QueryFactory.create(queryString) ;


    query.serialize(new IndentedWriter(System.out,true)) ;
    System.out.println() ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


    System.out.println("ProprietarioMangiaComeSuoAnimale: ") ;
    try {

      ResultSet rs = qexec.execSelect() ;


      for ( ; rs.hasNext() ; )
      {
        QuerySolution rb = rs.nextSolution() ;


        RDFNode x = rb.get("nome") ;


        if ( x.isLiteral() )
        {
          Literal titleStr = (Literal)x  ;
          System.out.print("    "+titleStr) ;
        }
        else if ( x.isResource() )
        {
          Resource r = (Resource)x ;
          System.out.print("    "+r.toString()) ;
        }
        else
          System.out.print("  "+x) ;
```

```
            RDFNode x2 = rb.get("nomeAnimale") ;


            if ( x2.isLiteral() )
            {
               Literal titleStr2 = (Literal)x2  ;
               System.out.print("    "+titleStr2) ;
            }
            else if ( x2.isResource() )
            {
              Resource r = (Resource)x2 ;
              System.out.print("    "+r.toString()) ;
            }
            else
               System.out.print("   "+x2) ;


         RDFNode x3 = rb.get("cibo") ;


         if ( x3.isLiteral() )
         {
            Literal titleStr2 = (Literal)x3  ;
            System.out.println("    "+titleStr2) ;
         }
         else if ( x3.isResource() )
         {
           Resource r = (Resource)x3 ;
           System.out.println("    "+r.toString()) ;
         }
         else
            System.out.println("   "+x3) ;
      }
     }

      finally
      {

         qexec.close() ;
      }
         }


         public static void queryChiMangia(Model model)   {
            String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> " ;
            String queryString = prolog + NL +
       "SELECT   ?nome ?cibo WHERE { {?x uriFN:"+
         model.getProperty("FN")+" ?nome; uri:"+
                              model.getProperty("pasto")+" ?cibo.} " +
```

```
" UNION{?x uri:"+
   model.getProperty("nome")+" ?nome; uriRDF:"+
   model.getProperty("type")+" ?type. ?type uri:"+
                        model.getProperty("mangia")+" ?cibo. } }" ;
      Query query = QueryFactory.create(queryString) ;


query.serialize(new IndentedWriter(System.out,true)) ;
System.out.println() ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


System.out.println("Esseri che mangiano: ") ;
try {

   ResultSet rs = qexec.execSelect() ;


   for ( ; rs.hasNext() ; )
   {
      QuerySolution rb = rs.nextSolution() ;


      RDFNode x = rb.get("nome") ;


      if ( x.isLiteral() )
      {
         Literal titleStr = (Literal)x  ;
         System.out.print("    "+titleStr) ;
      }
      else if ( x.isResource() )
      {
        Resource r = (Resource)x ;
        System.out.print("    "+r.toString()) ;
      }
      else
         System.out.print("   "+x) ;


      RDFNode x2 = rb.get("cibo") ;


   if ( x2.isLiteral() )
   {
      Literal titleStr2 = (Literal)x2  ;
      System.out.println("    "+titleStr2) ;
   }
   else if ( x2.isResource() )
   {
     Resource r = (Resource)x2 ;
```

```
            System.out.println("    "+r.toString()) ;
          }
          else
             System.out.println("   "+x2) ;
       }
      }

       finally
       {

          qexec.close() ;
       }
          }
}
```

A.2 codice Animali- Cibo realizzato con la reificazione

```
package seminarioIS;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.*;
import com.hp.hpl.jena.sparql.util.IndentedWriter;
import com.hp.hpl.jena.query.*;



public class AnimaliCiboReificato {

        public static final String uri = "http://somewhere#";
        public static final String uriFN = "http://www.w3.org/2001/vcard-rdf/3.0#";
        public static final String uriRDF =    "http://www.w3.org/1999/02/22-rdf-
syntax-ns#";
        public static final String uriRDFS =  RDFS.getURI();
        public static final String NL = System.getProperty("line.separator") ;

        public static void main(String[] args) {
                Model model = createModel() ;
                model.write(System.out);
                Model x = ModelFactory.createRDFSModel(model);
                System.out.println("\n\n***Query 1***\n\n");
                queryProprietarioMangiaSuoAnimale(x);
                System.out.println("\n\n***Query 2***\n\n");
                queryProprietarioMangiaComeSuoAnimale(x);
                System.out.println("\n\n***Query 3***\n\n");
```

```
            queryChiMangia(x);

        }

        public static Model createModel()
{
            //Setting the model and creating the data.

            Model m = ModelFactory.createDefaultModel() ;
    m.setNsPrefix( "uri", uri );

    //All definitions of the Resources of intensional level

    Resource cavallo = m.createResource(uri + "Cavallo") ;
    Resource mucca = m.createResource(uri + "Mucca") ;
    Resource erba = m.createResource(uri + "Erba") ;
    Resource insalata = m.createResource(uri + "Insalata") ;
    Resource animale = m.createResource(uri + "Animale") ;

    //All Property of intensional level with domain and range

    Property mangia = m.createProperty(uri + "mangia") ;
    animale.addProperty(mangia, erba) ;

    mucca.addProperty(RDF.type,RDFS.Class) ;
    cavallo.addProperty(RDF.type,RDFS.Class) ;
    erba.addProperty(RDF.type,RDFS.Class) ;
    insalata.addProperty(RDF.type,RDFS.Class) ;
    mucca.addProperty(RDFS.subClassOf,animale);
    cavallo.addProperty(RDFS.subClassOf,animale);


    Resource persona = m.createResource(uri + "Persona") ;
    Property possiede  = m.createProperty( uri + "possiede" ) ;
    possiede.addProperty(RDFS.domain , persona);
    persona.addProperty(RDF.type,RDFS.Class);

    //All elements of extensional level

    Resource person1 = m.createResource(uri + "JohnSmith") ;
    person1.addProperty(RDF.type , persona)
                        .addProperty(VCARD.FN , "John Smith");
    Resource person2 = m.createResource(uri + "UncleTom") ;
    person2.addProperty(RDF.type , persona)
                        .addProperty(VCARD.FN , "Uncle Tom");
    Resource horse1 = m.createResource(uri+ "Varenne") ;
    Resource cow1 = m.createResource(uri+ "Carolina") ;
    horse1.addProperty(RDF.type , cavallo);
    person1.addProperty(possiede,horse1) ;
    Property nomeAnimale = m.createProperty(uri + "nome") ;
    horse1.addProperty(nomeAnimale, "Varenne") ;
```

```
        cow1.addProperty(nomeAnimale, "Carolina") ;
        cow1.addProperty(RDF.type , mucca);
        person2.addProperty(possiede,cow1) ;

//      All definitions of the Resources of metalevel
//              In this case we are using reification

        Resource reifCibo = m.createResource() ;
        reifCibo.addProperty(RDF.type, RDF.Statement) ;
        reifCibo.addProperty(RDF.subject, m.createResource(uri + "Cibo"));
        reifCibo.addProperty(RDF.predicate, RDF.type);
        reifCibo.addProperty(RDF.object, RDFS.Class);

        Resource reifCarne = m.createResource() ;
        reifCarne.addProperty(RDF.type, RDF.Statement) ;
        reifCarne.addProperty(RDF.subject, m.createResource(uri + "Carne"));
        reifCarne.addProperty(RDF.predicate, RDF.type);
        reifCarne.addProperty(RDF.object, RDFS.Class);

        Resource reifVerdura = m.createResource() ;
        reifVerdura.addProperty(RDF.type, RDF.Statement) ;
        reifVerdura.addProperty(RDF.subject, m.createResource(uri + "Verdura"));
        reifVerdura.addProperty(RDF.predicate, RDF.type);
        reifVerdura.addProperty(RDF.object, RDFS.Class);

        Resource reifVerdura2 = m.createResource() ;
        reifVerdura2.addProperty(RDF.type, RDF.Statement) ;
        reifVerdura2.addProperty(RDF.subject, uri + "Verdura");
        reifVerdura2.addProperty(RDF.predicate, RDFS.subClassOf);
        reifVerdura2.addProperty(RDF.object,
             reifCibo.getProperty(RDF.subject).getObject());

        Property pasto = m.createProperty(uri + "pasto") ;
        pasto.addProperty(RDFS.domain, persona) ;
        pasto.addProperty(RDFS.range, reifCibo.getProperty(RDF.subject).getObject()) ;

        //connection between intensional and metamodel level

        cavallo.addProperty(RDF.type , reifCarne.getProperty(RDF.subject).getObject())
;
        mucca.addProperty(RDF.type , reifCarne.getProperty(RDF.subject).getObject()) ;
        erba.addProperty(RDF.type , reifVerdura.getProperty(RDF.subject).getObject()) ;
        insalata.addProperty(RDF.type ,
reifVerdura.getProperty(RDF.subject).getObject()) ;

        //connection intra level
        person1.addProperty(pasto, reifCarne.getProperty(RDF.subject).getObject()) ;
        person2.addProperty(pasto, reifVerdura.getProperty(RDF.subject).getObject()) ;

        return m ;
    }
```

```
        public static void queryProprietarioMangiaSuoAnimale(Model model)     {
            String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> PREFIX uriRDFS: <"+uriRDFS+">" ;
            String queryString = prolog + NL +
       "SELECT   ?nome ?nomeAnimale WHERE { ?x uri:"+
          model.getProperty("possiede")+"      ?animale ; " +
          "uriFN:"+model.getProperty("FN")+" ?nome; uri:"+
                            model.getProperty("pasto")+" ?cibo. ?animale uriRDF:"+
                    model.getProperty("type")+"?type; uri:"+
                    model.getProperty("nome")+"?nomeAnimale. ?type uriRDF:" +
                            model.getProperty("type")+" ?cibo. }" ;
            Query query = QueryFactory.create(queryString) ;


       query.serialize(new IndentedWriter(System.out,true)) ;
       System.out.println() ;
       QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


       System.out.println("ProprietarioMangia: ") ;
       try {

          ResultSet rs = qexec.execSelect() ;


          for ( ; rs.hasNext() ; )
          {
             QuerySolution rb = rs.nextSolution() ;


             RDFNode x = rb.get("nome") ;


             if ( x.isLiteral() )
             {
                Literal titleStr = (Literal)x  ;
                System.out.print("    "+titleStr) ;
             }
             else if ( x.isResource() )
             {
               Resource r = (Resource)x ;
               System.out.print("    "+r.toString()) ;
             }
             else
                System.out.print("  "+x) ;


             RDFNode x2 = rb.get("nomeAnimale") ;
```

```
            if ( x2.isLiteral() )
            {
               Literal titleStr2 = (Literal)x2  ;
               System.out.println("    "+titleStr2) ;
            }
            else if ( x2.isResource() )
            {
              Resource r = (Resource)x2 ;
              System.out.println("    "+r.toString()) ;
            }
            else
               System.out.println("   "+x2) ;

         }
      }
      finally
      {

         qexec.close() ;
      }
         }

         public static void queryProprietarioMangiaComeSuoAnimale(Model model)
         {
            String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> PREFIX uriRDFS: <"+uriRDFS+">" ;
            String queryString = prolog + NL +
      "SELECT   ?nome ?nomeAnimale ?cibo WHERE { ?x uri:"+
         model.getProperty("possiede")+"      ?animale ; " +
         "uriFN:"+model.getProperty("FN")+" ?nome; uri:"+
                           model.getProperty("pasto")+" ?cibo.?animale uri:"+
                  model.getProperty("nome")+"?nomeAnimale; uriRDF:"+
                  model.getProperty("type")+" ?tipoAnimale. ?tipoAnimale  uri:"+
                  model.getProperty("mangia")+"?ciboAnimale. ?ciboAnimale uriRDF:"
+
                           model.getProperty("type")+" ?cibo.  }" ;
            Query query = QueryFactory.create(queryString) ;


      query.serialize(new IndentedWriter(System.out,true)) ;
      System.out.println() ;
      QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


      System.out.println("ProprietarioMangiaComeSuoAnimale: ") ;
      try {

         ResultSet rs = qexec.execSelect() ;


         for ( ; rs.hasNext() ; )
```

```
{
   QuerySolution rb = rs.nextSolution() ;


   RDFNode x = rb.get("nome") ;


   if ( x.isLiteral() )
   {
      Literal titleStr = (Literal)x  ;
      System.out.print("    "+titleStr) ;
   }
   else if ( x.isResource() )
   {
     Resource r = (Resource)x ;
     System.out.print("    "+r.toString()) ;
   }
   else
      System.out.print("  "+x) ;


   RDFNode x2 = rb.get("nomeAnimale") ;


   if ( x2.isLiteral() )
   {
      Literal titleStr2 = (Literal)x2  ;
      System.out.print("    "+titleStr2) ;
   }
   else if ( x2.isResource() )
   {
     Resource r = (Resource)x2 ;
     System.out.print("    "+r.toString()) ;
   }
   else
      System.out.print("   "+x2) ;


RDFNode x3 = rb.get("cibo") ;

// Check the type of the result value
if ( x3.isLiteral() )
{
   Literal titleStr2 = (Literal)x3  ;
   System.out.println("    "+titleStr2) ;
}
else if ( x3.isResource() )
{
  Resource r = (Resource)x3 ;
  System.out.println("    "+r.toString()) ;
}
```

```
        else
           System.out.println("   "+x3) ;


     }
    }

     finally
     {
        // QueryExecution objects should be closed to free any system resources
        qexec.close() ;
     }
        }


        public static void queryChiMangia(Model model)   {
           String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+">
PREFIX uriRDF: <"+uriRDF+"> " ;
           String queryString = prolog + NL +
     "SELECT   ?nome ?cibo WHERE { {?x uriFN:"+
        model.getProperty("FN")+" ?nome; uri:"+
                        model.getProperty("pasto")+" ?cibo.} " +
     " UNION{?x uri:"+
        model.getProperty("nome")+" ?nome; uriRDF:"+
        model.getProperty("type")+" ?type. ?type uri:"+
                        model.getProperty("mangia")+" ?cibo. } }" ;
           Query query = QueryFactory.create(queryString) ;


     query.serialize(new IndentedWriter(System.out,true)) ;
     System.out.println() ;
     QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


     System.out.println("Esseri che mangiano: ") ;
     try {

        ResultSet rs = qexec.execSelect() ;


        for ( ; rs.hasNext() ; )
        {
           QuerySolution rb = rs.nextSolution() ;


           RDFNode x = rb.get("nome") ;


           if ( x.isLiteral() )
           {
              Literal titleStr = (Literal)x  ;
```

```
                System.out.print("    "+titleStr) ;
            }
            else if ( x.isResource() )
            {
              Resource r = (Resource)x ;
              System.out.print("    "+r.toString()) ;
            }
            else
                System.out.print("  "+x) ;


            RDFNode x2 = rb.get("cibo") ;

          // Check the type of the result value
          if ( x2.isLiteral() )
          {
              Literal titleStr2 = (Literal)x2  ;
              System.out.println("    "+titleStr2) ;
          }
          else if ( x2.isResource() )
          {
            Resource r = (Resource)x2 ;
            System.out.println("    "+r.toString()) ;
          }
          else
              System.out.println("   "+x2) ;
       }
      }

       finally
       {

          qexec.close() ;
       }
          }
}
```

A.3 codice Tipi di Entità

```
package seminarioIS;

import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.reasoner.rulesys.*;
import com.hp.hpl.jena.vocabulary.*;
import com.hp.hpl.jena.sparql.util.IndentedWriter;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.query.ResultSet;

public class LibraryEntity {

        public static final String uri = "http://somewhere#";
        public static final String uriFN = "http://www.w3.org/2001/vcard-rdf/3.0#";
        public static final String uriRDF =    "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
        public static final String uriRDFS = RDFS.getURI();
        public static final String NL = System.getProperty("line.separator") ;

        public static Model createModel()
  {
                //Setting the model and creating the data.
                Model m = ModelFactory.createDefaultModel() ;
    m.setNsPrefix( "uri", uri );

    //All definitions of the Resources of meta level
    Resource entity = m.createResource(uri + "Entity") ;
    Resource quality = m.createResource(uri + "Quality") ;
    Resource event = m.createResource(uri + "Event") ;
    Resource object = m.createResource(uri + "Object") ;
    Resource abstractConcept = m.createResource(uri + "Abstract") ;
    quality.addProperty(RDFS.subClassOf ,entity );
    event.addProperty(RDFS.subClassOf ,entity );
    object.addProperty(RDFS.subClassOf ,entity );
    abstractConcept.addProperty(RDFS.subClassOf ,entity );
    entity.addProperty(RDF.type,RDFS.Class) ;
    quality.addProperty(RDF.type,RDFS.Class) ;
    event.addProperty(RDF.type,RDFS.Class) ;
    object.addProperty(RDF.type,RDFS.Class) ;
    abstractConcept.addProperty(RDF.type,RDFS.Class) ;

    //All definitions of the Resources of intensional level
```

```
        Resource libro = m.createResource(uri + "Libro") ;
        Resource autore = m.createResource(uri + "Autore") ;
        Resource letturaPubblica = m.createResource(uri + "LetturaPubblica") ;
        libro.addProperty(RDF.type, object)
                        .addProperty(RDF.type,RDFS.Class) ;
        autore.addProperty(RDF.type, object)
                        .addProperty(RDF.type,RDFS.Class) ;
        letturaPubblica.addProperty(RDF.type, event)
                        .addProperty(RDF.type,RDFS.Class) ;
        Property autoreLibro = m.createProperty(uri + "autoreLibro");
        autoreLibro.addProperty(RDFS.domain,libro)
                        .addProperty(RDFS.range,autore);
        Property titolo = m.createProperty(uri + "titolo");
        titolo.addProperty(RDFS.domain,libro)
                        .addProperty(RDF.type,abstractConcept);
        Property letturaLibro = m.createProperty(uri + "letturaLibro");
        letturaLibro.addProperty(RDFS.domain,letturaPubblica) ;
        letturaLibro.addProperty(RDFS.range,libro);
        Property genere = m.createProperty(uri + "genere");
        genere.addProperty(RDFS.domain, libro)
                    .addProperty(RDF.type,abstractConcept);

//      All definitions of the Resources of extensional level

        Resource autore1 = m.createResource(uri + "PDick") ;
        Resource libro1 = m.createResource(uri + "Ubik") ;
        autore1.addProperty(VCARD.FN, "Philip Kindred Dick")
                        .addProperty(RDF.type,autore);
                libro1.addProperty(autoreLibro,autore1)
                        .addProperty(genere, uri + "Fantascienza")
                        .addProperty(RDF.type,libro);
        libro1.addProperty(titolo,"Ubik");
        Resource libro2 = m.createResource(uri + "Paycheck") ;
        libro2.addProperty(autoreLibro,autore1)
                        .addProperty(genere,uri + "Fantascienza")
                        .addProperty(RDF.type,libro);
        libro1.addProperty(titolo,"Ubik");
        Resource letturaUbik = m.createResource(uri + "lettura Ubik") ;
        letturaUbik.addProperty(DC.date,"2008-07-25");
        letturaUbik.addProperty(letturaLibro, libro1)
                        .addProperty(RDF.type,letturaPubblica);

        return m ;
    }

          public static void queryAllEntity(Model model)      {
              String prolog = "PREFIX uri: <"+uri+"> PREFIX uriFN: <"+uriFN+"> PREFIX
uriRDF: <"+uriRDF+"> PREFIX uriRDFS: <"+uriRDFS+">" ;
              String queryString = prolog + NL +
        "SELECT  DISTINCT ?x  ?type WHERE {?x uriRDF:"+
```

```
    model.getProperty("type")+"  ?type. ?type uriRDFS:"+
                        model.getProperty("subClassOf")+" uri:"+
    model.getResource("Entity")+"." +
                        "}" ;
        Query query = QueryFactory.create(queryString) ;

query.serialize(new IndentedWriter(System.out,true)) ;
System.out.println() ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


System.out.println("Tutte le entità: ") ;
try {

    ResultSet rs = qexec.execSelect() ;


    for ( ; rs.hasNext() ; )
    {
        QuerySolution rb = rs.nextSolution() ;


        RDFNode x = rb.get("x") ;


        if ( x.isLiteral() )
        {
            Literal titleStr = (Literal)x  ;
            System.out.print("    "+titleStr) ;
        }
        else if ( x.isResource() )
        {
           Resource r = (Resource)x ;
           System.out.print("    "+r.toString()) ;
        }
        else
            System.out.print("  "+x) ;


        RDFNode x2 = rb.get("type") ;

    // Check the type of the result value
    if ( x2.isLiteral() )
    {
        Literal titleStr2 = (Literal)x2  ;
        System.out.println("    "+titleStr2) ;
    }
    else if ( x2.isResource() )
    {
       Resource r = (Resource)x2 ;
       System.out.println("    "+r.toString()) ;
```

```
        }
        else
            System.out.println("    "+x2) ;
    }
   }

    finally
    {

        qexec.close() ;
    }
        }

        public static void main(String[] args) {

                Model model = createModel() ;
                model.write(System.out);

                String rules = "[rule1: (?a rdf:type ?b) (?b rdf:type ?c) -> (?a rdf:type ?c)]";
            GenericRuleReasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
            reasoner.setDerivationLogging(true);
            InfModel inf = ModelFactory.createInfModel(reasoner, model);

                Model x = ModelFactory.createRDFSModel(model);
                System.out.println("\n\n***Query 1***\n\n");
                queryAllEntity(inf);
        }
}
```

A.4 Codice Description

```
package seminarioIS;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.reasoner.rulesys.GenericRuleReasoner;
import com.hp.hpl.jena.reasoner.rulesys.Rule;
import com.hp.hpl.jena.vocabulary.*;
import com.hp.hpl.jena.sparql.util.IndentedWriter;
import com.hp.hpl.jena.query.*;
import com.hp.hpl.jena.query.ResultSet;
```

```
public class ConceptDescription {

        public static final String uri = "http://somewhere#";
        public static final String uriFN = "http://www.w3.org/2001/vcard-rdf/3.0#";
        public static final String uriRDF =    "http://www.w3.org/1999/02/22-rdf-syntax-
ns#";
        public static final String uriRDFS = RDFS.getURI();
        public static final String NL = System.getProperty("line.separator") ;

        public static Model createModel()
   {

                //Setting the model and creating the data.
                Model m = ModelFactory.createDefaultModel() ;
      m.setNsPrefix( "uri", uri );

      //All definitions of the Resources of meta level
      Resource concept = m.createResource(uri + "Concept");
      Resource description = m.createResource(uri + "Description");
      concept.addProperty(RDF.type,RDFS.Class) ;
      description.addProperty(RDF.type,RDFS.Class) ;
      Property defines = m.createProperty(uri + "defines");
      defines.addProperty(RDFS.range,concept);
      defines.addProperty(RDFS.domain,description);
      Property usesConcept = m.createProperty(uri + "usesConcept");
      usesConcept.addProperty(RDFS.range,concept);
      usesConcept.addProperty(RDFS.domain,description);
      Property isConceptUsedIn = m.createProperty(uri + "isConceptUsedIn");
      isConceptUsedIn.addProperty(RDFS.range,description);
      isConceptUsedIn.addProperty(RDFS.domain,concept);
      Property isDefinedIn = m.createProperty(uri + "isDefinedIn");
      isDefinedIn.addProperty(RDFS.range,description);
      isDefinedIn.addProperty(RDFS.domain,concept);

      //All definitions of the Resources of intensional level
      Resource articolo = m.createResource(uri + "Articolo Scientifico");
      Resource algoritmo = m.createResource(uri + "Algoritmo");
      articolo.addProperty(RDF.type,description)
                        .addProperty(RDF.type,RDFS.Class) ;
      algoritmo.addProperty(RDF.type,concept)
                        .addProperty(RDF.type,RDFS.Class) ;
      Property usa = m.createProperty(uri + "usa");
      usa.addProperty(RDFS.range,algoritmo);
      usa.addProperty(RDFS.domain,articolo);
      usa.addProperty(RDF.type,usesConcept);
      Property definisceAlgoritmo = m.createProperty(uri + "definisceAlgoritmo");
      definisceAlgoritmo.addProperty(RDFS.range,algoritmo);
      definisceAlgoritmo.addProperty(RDFS.domain,articolo);
      definisceAlgoritmo.addProperty(RDF.type,defines);
      Property presenteIn = m.createProperty(uri + "presenteIn");
      presenteIn.addProperty(RDFS.range,articolo);
```

```
        presenteIn.addProperty(RDFS.domain,algoritmo);
        presenteIn.addProperty(RDF.type,isConceptUsedIn);
        Property autore = m.createProperty(uri + "autore");
        autore.addProperty(RDFS.domain,articolo);

        //All definitions of the Resources of extensional level
        Resource articolo1 = m.createResource(uri + "Worst-case analysis of a new
heuristic for the travelling salesman problem");
        Resource articolo2 = m.createResource(uri + "State of the art Algorithms for the
Minimum Spanning Trees");
        Resource articolo3 = m.createResource(uri + "Solutio problematis ad geometriam
situs pertinentis");
        Resource alg1 = m.createResource(uri + "Christofides algorithm");
        Resource alg2 = m.createResource(uri + "minimum spanning tree");
        Resource alg3 = m.createResource(uri + "Eulerian path");
        articolo1.addProperty(definisceAlgoritmo,alg1);
        articolo1.addProperty(usa,alg2);
        articolo1.addProperty(usa,alg3);
        articolo1.addProperty(RDF.type,articolo);
        articolo2.addProperty(definisceAlgoritmo,alg2);
        articolo3.addProperty(definisceAlgoritmo,alg3);
        alg1.addProperty(autore, "Christofides");
        alg3.addProperty(autore, "Euler");
        alg1.addProperty(RDF.type,algoritmo);
        alg2.addProperty(RDF.type,algoritmo);
        alg3.addProperty(RDF.type,algoritmo);

            return m;
    }

            public static void queryAllConceptInDescription(Model model)    {
                String prolog = "PREFIX uri: <"+uri+">  PREFIX uriRDF: <"+uriRDF+">" ;
                String queryString = prolog + NL +
        "SELECT  DISTINCT ?x ?p WHERE {?x uriRDF:"+
            model.getProperty("type")+"  uri:"+
            model.getResource("Concept")+". ?y uriRDF:" +
            model.getProperty("type")+"  uri:"+
            model.getResource("Description")+            " . ?y ?p ?x.}" ;
                Query query = QueryFactory.create(queryString) ;

        query.serialize(new IndentedWriter(System.out,true)) ;
        System.out.println() ;
        QueryExecution qexec = QueryExecutionFactory.create(query, model) ;


        System.out.println("Tutti i Concetti relativi a Descrizioni: ") ;
        try {

            ResultSet rs = qexec.execSelect() ;
```

```
    for ( ; rs.hasNext() ; )
    {
       QuerySolution rb = rs.nextSolution() ;


       RDFNode x = rb.get("x") ;


       if ( x.isLiteral() )
       {
          Literal titleStr = (Literal)x  ;
          System.out.print("    "+titleStr) ;
       }
       else if ( x.isResource() )
       {
         Resource r = (Resource)x ;
         System.out.print("    "+r.toString()) ;
       }
       else
          System.out.print("  "+x) ;

       RDFNode x2 = rb.get("p") ;

    // Check the type of the result value
    if ( x2.isLiteral() )
    {
       Literal titleStr2 = (Literal)x2  ;
       System.out.println("    "+titleStr2) ;
    }
    else if ( x2.isResource() )
    {
      Resource r = (Resource)x2 ;
      System.out.println("    "+r.toString()) ;
    }
    else
       System.out.println("   "+x2) ;
 }
}

 finally
 {

    qexec.close() ;
 }
    }

    public static void main(String[] args) {
            Model model = createModel() ;
            model.write(System.out);

            String rules = "[rule1: (?a rdf:type ?b) (?b rdf:type ?c) -> (?a rdf:type ?c)]";
```

```
            GenericRuleReasoner reasoner = new
GenericRuleReasoner(Rule.parseRules(rules));
            reasoner.setDerivationLogging(true);
            InfModel inf = ModelFactory.createInfModel(reasoner, model);

                  Model x = ModelFactory.createRDFSModel(model);
                  System.out.println("\n\n***Query 1***\n\n");
                  queryAllConceptInDescription(inf);

      }
}
```

Bibliografia

[1] W3C Recommendation, RDF: Primer, Concepts, Syntax, Semantics, Vocabulary

[2] W3C Recommendation, SPARQL Query Language for RDF

[3] Jena Documentation, http://jena.sourceforge.net/documentation.html

[4] Presutti, Gangemi e al., A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies, 2008

[5] G. De Giacomo, dispense del corso “Seminari di ingegneria del software”