**ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA**

**Seconda Facoltà di Ingegneria con sede a Cesena**

**CORSO DI LAUREA IN INGEGNERIA INFORMATICA**

# IMPLEMENTAZIONE C# DELLE QUERY SPARQL PER UNA PIATTAFORMA DI CONDIVISIONE SEMANTICA DELLE INFORMAZIONI

Elaborata nel corso di : Calcolatori Elettronici

CANDIDATO:
Randy Ammar

RELATORE:
Prof. Luca Roffia

CORRELATORI:
Ing. Francesco Morandi
Ing. Alfredo D'Elia

**ANNO ACCADEMICO 2011/12**
**SESSIONE I**

# PAROLE CHIAVE

Smart Environments

Smart M3

Web Semantico

Ontologie

OWL

RDF

SPARQL

# Indice

# Introduzione

Lo Smart Environment è un ecosistema di oggetti in grado di auto-organizzarsi, fornire servizi e manipolare dati complessi, all'interno del quale persone e oggetti interagiscono e si relazionano tra loro continuamente. Uno Smart Environment può quindi essere definito come la combinazione di un ambiente fisico, un'infrastruttura per la gestione dei dati (chiamato Smart Space), un insieme di elementi eterogenei in grado di raccogliere dati dall'ambiente stesso e una particolare soluzione di connettività per trasmettere questi dati allo Smart Space. La chiave di volta che regge uno Smart Environment è la connessione tra il mondo fisico e il mondo delle informazioni, al fine di rendere disponibili le informazioni del mondo fisico attraverso dispositivi intelligenti.
In questo contesto si colloca il progetto europeo SOFIA (Smart Objects For Intelligent Applications) finalizzato a rendere disponibili le informazioni del mondo fisico su dispositivi intelligenti (Smart Devices). L'obiettivo è quello di creare una piattaforma innovativa che permetta lo sviluppo di nuove interfacce e nuove modalità di interazione tra sistema e utenti, in modo da massimizzare i benefici introdotti dagli Smart Environments. La piattaforma potrà essere sfruttata da dispositivi di qualsiasi casa produttrice e sarà capace di adattarsi alle caratteristiche hardware più variegate.
Smart-M3 è il nome del software open source sviluppato all'interno del progetto SOFIA il cui scopo è appunto quello di consentire il collegamento tra mondi reali e mondi virtuali. La base di questo sistema è quella di permettere sia ai dispositivi che ai software di mettere a disposizione di altri dispositivi e di altri software i propri dati e le proprie informazioni. Tutto questo tramite un semplice intermediario per condividere informazioni: la SIB (Semantic Information Broker).
La piattaforma realizzata, pone le sue basi sui concetti del Semantic Web, in particolare per quanto riguarda il modello di rappresentazione delle informazioni. Il Semantic Web fornisce infatti un'infrastruttura comune che consente ai dati di essere condivisi e riutilizzati fra applicazioni e associarli ad informazioni e metadati in modo da renderne possibile l'interrogazione e l'interpretazione automatizzata attraverso logiche inferenziali di tipo semantico.
Un concetto chiave del Semantic Web sono le ontologie, ovvero vocabolari di termini e proprietà rappresentate secondo gli standard del Semantic Web per fornire un modello di descrizione delle informazioni rilevanti in uno Smart Environment. Negli Smart Environment da me studiati la comprensibilità delle informazioni si basa su modelli comuni di dati definiti su un'ontologia e resi disponibili dalla SIB, in cui i dati sono memorizzati secondo il modello RDF anch'esso definito nell'ambito del Semantic Web.
La tesi da me effettuata si è soffermata in un primo momento sullo studio del modello RDF (Resource Description Framework) ed in particolare del linguaggio di query SPARQL. Dopo aver compreso i principi alla base del modello di rappresentazione delle informazioni e del suo linguaggio di query ho contribuito ad estendere le librerie C# (parte integrante delle API fornite da Smart-M3) per fornire la funzionalità di eseguire query SPARQL e per gestirne i risultati.
Il presente lavoro è strutturato in due parti. La prima parte illustra l'infrastruttura per la condivisione di informazioni di tipo "Semantic Web" ovvero SMART-M3, e i principi architetturali del Semantic Web che sono stati indispensabili per capire meglio il ruolo che rivestono le sue tecnologie. Nel seguito vengono descritte le principali tecnologie utilizzate del Semantic Web, quali RDF e OWL, insieme alle modalità di interrogazione e quindi il linguaggio SPARQL.
La seconda parte presenta l'implementazione dell'estensione della libreria realizzata, i test delle funzionalità e le considerazioni finali.

# Capitolo 1

# Smart-M3

## 1.1 Introduzione

Smart-M3 [1] è il nome di un progetto software open-source sviluppato all'interno del progetto SOFIA che fornisce un'infrastruttura per la condivisione di informazioni di tipo "Semantic Web" tra entità software e dispositivi hardware. La condivisione avviene mediante un Semantic Information Brokers (SIB) che garantisce l'indipendenza del sistema dai dispositivi e dai fornitori. Il suffisso M3 deriva da "Multi vendor, Multi device, Multi domain" e intende combinare le idee di sistemi distribuiti, rete e web semantico. Smart-M3 è una soluzione che permette a dispositivi come Smartphone, televisori o laptop di interagire tra loro al fine di trarre vantaggio dalle possibilità che gli ambienti di utilizzo offrono. L'obiettivo finale è quello di avere ambienti intelligenti permettendo il collegamento di mondi reali e virtuali (digitali).
L'idea chiave è che dispositivi e software possano pubblicare le loro informazioni per altri dispositivi e software in maniera semplice. Per rendere compatibili le informazioni pubblicate dai diversi dispositivi si fa riferimento ad un modello comune di ontologia, con il quale è anche possibile dare una descrizione più accurata delle informazioni.

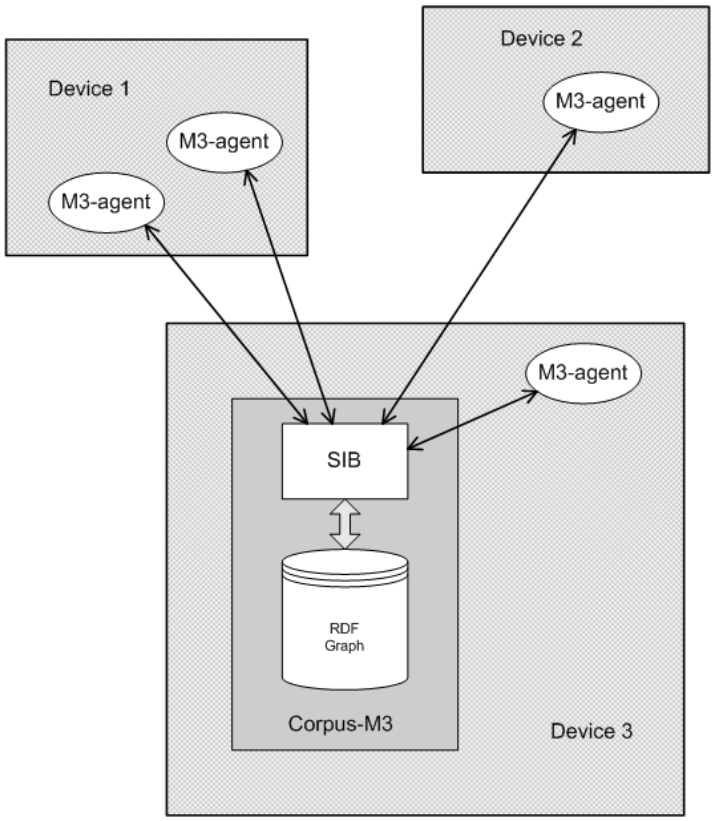



**Figura 1: Esempio di struttura Smart-M3**

Un'altra particolarità è la separazione dei vari attori del sistema: dispositivi, dominio ed utilizzatori delle informazioni sono indipendenti tra loro, dando così la possibilità di uno sviluppo anche separato dei vari componenti.
In tal modo si realizza un ambiente intelligente garantendo la massima interoperabilità tra i dispositivi che possono essere immessi nell'ambiente stesso. Vale a dire che le informazioni devono essere rappresentate in modo comprensibile a tutti i calcolatori, una rappresentazione che incorpori anche la semantica dei dati stessi. Inoltre un dispositivo per accedere al sistema dovrà soltanto conoscere il protocollo di comunicazione che qualsiasi dispositivo può implementare, se dotato di un minimo di connettività e di capacita computazionale.
Secondo le nozioni dei sistemi distribuiti, Smart-M3 è un sistema con architettura a "Spazio di Dati Condiviso". Una serie di programmi detti Agenti o Knowledge Processor-KP (notazione propria di Smart-M3), realizzano la capacità computazionale del sistema stesso. Questi per scambiare informazioni tra di loro o verso il mondo esterno utilizzano una sorta di repository, lo spazio di dati condiviso che contiene tutte le informazioni di interesse: ogni agente può accedervi e inserire informazione oppure ricevere una notifica in caso una certa informazione si presenta all'interno di essa secondo un meccanismo di publish-subscribe [2].
Lo spazio di dati condiviso in linguaggio Smart-M3 prende il nome di SIB (Semantic Information Broker). I dati all'interno della SIB sono memorizzati in RDF [3] che è il linguaggio ideale per rappresentare ogni tipo di informazione in modo semantico.
La tecnologia alla base di Smart-M3 è in continuo sviluppo e si basa sul concetto di spazio di tuple [4], utilizzato per esempio in progetti come LINDA [5], o nel progetto dell'università di Bologna TuCSoN [6], al quale si aggiunge il concetto di ontologia utilizzato per arricchire di semantica le informazioni.

## 1.2 Tecnologia

Come già precedentemente anticipato, Smart-M3 vuole risolvere i problemi dell'interazione tra componenti di diversa natura. Solitamente, in ambienti ben definiti, l'interazione tra componenti diversi viene gestita attraverso l'utilizzo di un determinato linguaggio o di un particolare protocollo, attraverso i quali avviene lo scambio delle informazioni, consentendo quindi l'interazione tra i vari dispositivi.
Questo tipo di approccio è valido, ma pone limiti quando un elemento esterno all'ambiente, che non conosce il linguaggio specifico o il formato dei dati, compromettendo l'utilizzo di determinate funzionalità.
Smart-M3 crea un sistema di pubblicazione e fruizione delle informazioni comprensibile a tutti i dispositivi software che ne vogliano fare uso. La distinzione tra produttori e consumatori di informazioni non è più vincolante, i contenuti pubblicati sono a disposizione di tutti i consumatori in maniera trasparente ai produttori. Con queste premesse i dati possono essere manipolati in ogni momento, modo e contesto da dispositivi di qualunque marca e sviluppati in qualsiasi tecnologia. Questo approccio apre le porte allo sviluppo di nuove tipologie di servizi di diversa natura.

## 1.3 Architettura e funzionamento

Come anticipato i due componenti principali di Smart-M3 sono la SIB (Semantic Information Broker) e i KP (Knowledge Processor).
La SIB è dove vengono memorizzate le triple RDF. I KP invece accedono a queste informazioni per svolgere un compito specifico: è da notare che la filosofia di Smart-M3 impone che ogni KP sia il più semplice possibile, per svolgere un compito complesso ci vogliono tanti KP semplici e organizzati tra loro. L'organizzazione tra i KP è dovuta al fatto che essi concordano sull'ontologia delle informazioni e conoscono quali parti di queste informazioni sono di loro interesse. Infine il

mezzo di comunicazione tra questi componenti è dato dal protocollo SSAP (Smart Space Access Protocol) che definisce i messaggi scambiati per ognuna delle operazioni che un KP può compiere nei confronti della SIB.

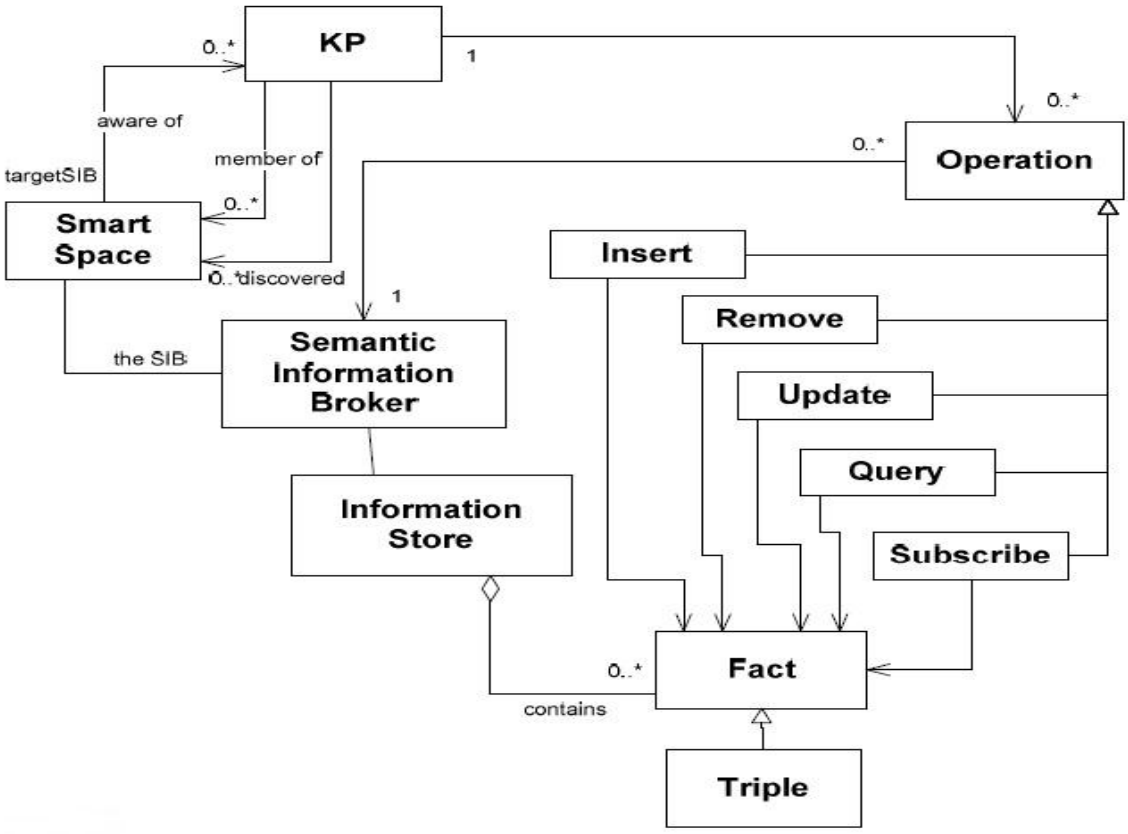



**Figura 1.2: Modello di Dominio dello Smart-M3**

Le operazioni primitive sono [7]:

- **Join**: con la quale un KP accede alla SIB;
- **Leave**: con la quale un KP lascia la SIB;
- **Insert**: equivale all'operazione generica "publish" con la quale il KP pubblica (atomicamente) delle informazioni nella SIB sotto forma di triple;
- **Remove**: con la quale vengono rimosse atomicamente delle triple;
- **Update**: con la quale si modificano automicamente delle triple (può essere vista come una combinazione di remove e insert ma fatta in modo atomico).
- **Query**: con la quale un KP richiede delle informazioni contenute all'interno della SIB. Esistono tre tipi di query:
    - **Query Wilbur**, a partire da un nodo nel grafo è definito un percorso formato dai predicati delle triple si ottengono tutti i nodi collegati al nodo di partenza da quel percorso. Questo tipo di query non è però più supportato nella versione corrente.
    - **Query RDF**, con la quale si specifica un template di tripla e si ottengono tutte quelle che rispecchiano quel criterio, E' previsto l'uso dell'Uri "any" che viene utilizzato in modo simile al carattere "*" nei sistemi operativi, ed esprime il concetto di "qualsiasi cosa";
    - **Query SPARQL** permettono di fare interrogazioni a dei grafi RDF per ricevere informazioni o nodi specifici;

- **Subscribe**: un KP si può registrare (sottoscrivere) a una certa collezione di triple, quando una di queste viene creata o modificata riceve una notifica con il suo nuovo valore in un'ottica Event Based;
- **Unsubscribe**: per annullare la sottoscrizione.

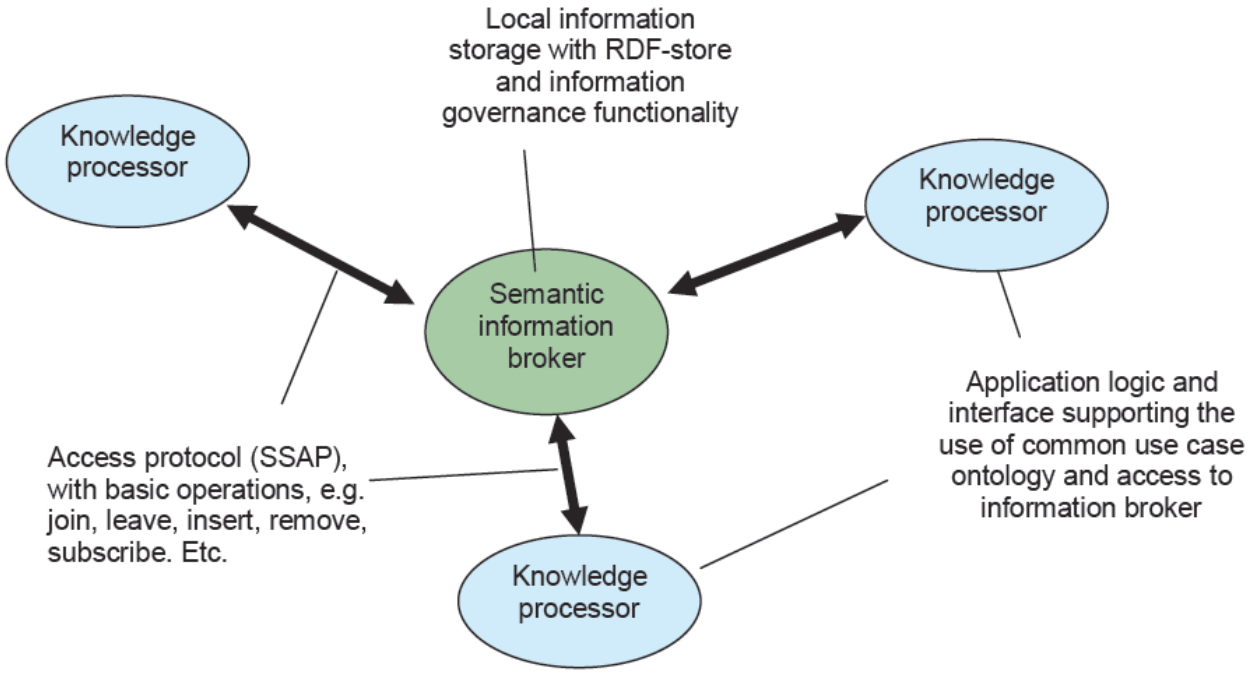



**Figura 1.3: Visuale del Livello Informativo di Smart-M3 [8]**

Dato il protocollo si possono suddividere i KP in tre classi in base alle operazioni che compiono:

- **Producer**: quando un KP produce nuova informazione e quindi esegue prevalentemente delle insert (o update);
- **Consumer**: quando il KP utilizza le informazioni nella SIB per mostrarle all'utente oppure per azionare degli attuatori e quindi esegue prevalentemente delle subscribe;
- **Aggregator**: quando invece il KP, a partire da informazioni già presenti nella SIB, ne crea di nuove eseguendo dei ragionamenti, e quindi esegue sia insert che subscribe.

Ogni KP in fase di inizializzazione può eseguire delle query per venire a conoscenza ad esempio dello stato dell'ambiente. E' necessario che ogni KP faccia parte di una sola di queste classi e che si concentri solamente sul suo scopo in modo da essere il più semplice possibile.
Se rispettato, questo requisito porta un'applicazione ad essere modulare, permettendo l'aggiunta o la rimozione di feature dinamicamente senza creare problemi al resto dell'applicazione.
Per capire questa potenzialità occorre immaginare una situazione in cui in una stanza esiste un sensore di temperatura e un piccolo display che visualizza in tempo reale la temperatura.
In questo sistema il sensore di temperatura conterrà un KP di tipo producer che invia regolarmente alla SIB le informazioni sulla temperatura, il display invece conterrà un KP consumer che ogni volta che il sensore manda un nuovo dato si limiterà a visualizzarlo. Il sistema descritto è semplice e modulare, infatti, nel caso si volesse aggiungere una nuova funzionalità, per esempio l'invio di un SMS di allarme (nel caso la temperatura scendesse sotto una certa soglia), basterà aggiungere un terzo KP consumer sottoscritto a sua volta alle informazioni sulla temperatura che in caso di superamento della soglia inoltrerà il messaggio di allarme grazie ad hardware apposito.

## 1.4 Protocollo SSAP

Si vuole ora analizzare il protocollo di comunicazione attraverso il quale è possibile interagire con la SIB. La comunicazione tra il Semantic Information Broker (SIB) e i diversi software si basa su un protocollo chiamato Smart Space Access Protocol (SSAP) basato a sua volta su messaggi XML, che non prevede fasi di negoziazione complesse tra le uniche due parti coinvolte (la SIB e i singoli KP). SSAP permette di collegarsi o abbandonare il sistema (join e leave) e gestire cinque tipi di transizioni (insert, remove, update, subscription e query).
Transazioni come la join, la leave, l'insert, l'update e la remove non necessitano di particolari spiegazioni: la prima permette ai KP di unirsi al sistema per poter usufruire dei servizi offerti, la seconda permette di comunicare la volontà di volersi congedare, le ultime tre permettono, rispettivamente, di inserire, modificare e rimuovere delle triple RDF all'interno del dispositivo di storage.
Per recuperare informazioni il protocollo SSAP mette a disposizione la transazione query. Il sistema supporta tre tipi di approcci: wilbur query language (WQL), query RDF-M3 e query SPARQL.
Il Wilbur query language mette a disposizione diversi tipi di query. Attraverso di essi si possono ricavare triple effettuando una navigazione all'interno del grafo RDF, specificando un nodo di partenza e un percorso lungo il quale navigare il grafo RDF.
La query RDF-M3 permette di recuperare informazioni sfruttando la struttura degli statement RDF. Attraverso l'utilizzo dell'URI "any" (http://www.nokia.com/NRC/M3/sib#any) è possibile recuperare tutte le triple RDF che fanno matching con gli statements specificati nella query. E' anche possibile specificare più statement RDF in una query e in questo caso il risultato ottenuto sarà l'unione dei singoli risultati.
La query SPARQL permette di fare interrogazioni a dei grafi RDF per ricevere informazioni o nodi specifici. Consentendo alle applicazioni di lavorare sui risultati delle query SPARQL invece che con le dichiarazioni RDF.
Altra transazione importante è la subscription: permette a un KP di chiedere che gli venga notificato il cambiamento del valore di un dato di particolare interesse. Il meccanismo attraverso il quale si può specificare l'informazione interessata è simile a quello utilizzato dalle query RDF-M3: specificando per esempio gli URI di due delle tre componenti (soggetto, predicato o oggetto), e utilizzando l'URI "any" per la terza, è possibile riceve notifiche sugli aggiornamenti del valore della terza componente.
Le librerie che implementato il protocollo SSAP, ovvero che consentono ai KP di comunicare con la SIB, sono al momento disponibili nei seguenti linguaggi [9]: C#, Java, Python, Php, C e Prolog.

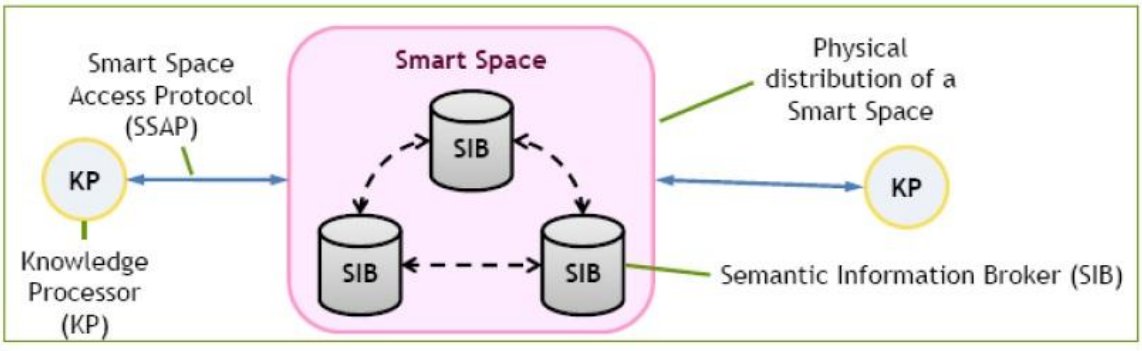



**Figura 1.4: Comunicazione in Smart-M**

# Capitolo 2

# Il Web Semantico

## 2.1 Introduzione

Il Web è frequentemente utilizzato da una larga varietà di persone per utilizzi e finalità diverse ed è anche il maggior contenitore di conoscenza. Nel Web spesso l'informazione ricercata dall'utente è diffusa fra più fonti informative, e sarebbe molto utile avere la possibilità che anche le macchine potessero autonomamente estrarre e dedurre conoscenza. Le tecnologie del Semantic Web puntano a questo obiettivo. Infatti, il Semantic Web rappresenta la volontà di estendere e dare valore aggiunto al Web di oggi attraverso la creazione di dati elaborabili e comprensibili direttamente dalle macchine attraverso l'uso della logica e dei linguaggi di rappresentazione della conoscenza.

## 2.2 Il Web Semantico

Il Web Semantico è definito dalla parola semantica. Semantica è quella parte della linguistica che studia il significato delle parole, degli insiemi delle parole, delle frasi e dei testi. La semantica associa ad una parola chiave un significato utile attraverso la costituzione delle relazioni. Il Web Semantico è semplicemente un Web di Dati descritti e linkati in maniera tale da stabilire un contesto o delle semantiche che aderiscono ad una definita grammatica.
Il Web Semantico fornisce una piattaforma comune che consente ai dati di essere condivisi e riutilizzati fra applicazioni, imprese e comunità.
Si tratta di una collezione di tecnologie e standard che forniscono un'ambiente alle macchine per comprendere la semantica delle informazioni sul Web.
E' stato introdotto come termine ufficialmente nel maggio del 2001 nell'articolo "The Semantic Web" di Berners-Lee pubblicato su Scientific American [10]: "Il Web Semantico è una estensione del Web corrente in cui all'informazione è dato un significato ben definito, migliorando la possibilità ai computer e alle persone di lavorare in cooperazione."
Sono tre i punti chiave di questa definizione: il Semantic Web è un'estensione del Web attuale; lo scopo del Semantic Web è la cooperazione tra computer e persone, in modo che le macchine possano essere maggiormente di supporto agli utenti nell'esecuzione e nell'automazione di operazioni; la realizzazione del Semantic Web è possibile solo dando un significato ben definito all'informazione, in modo che le macchine possano raggiungere funzionalità avanzate di ragionamento e capacità di rispondere a domande complesse.

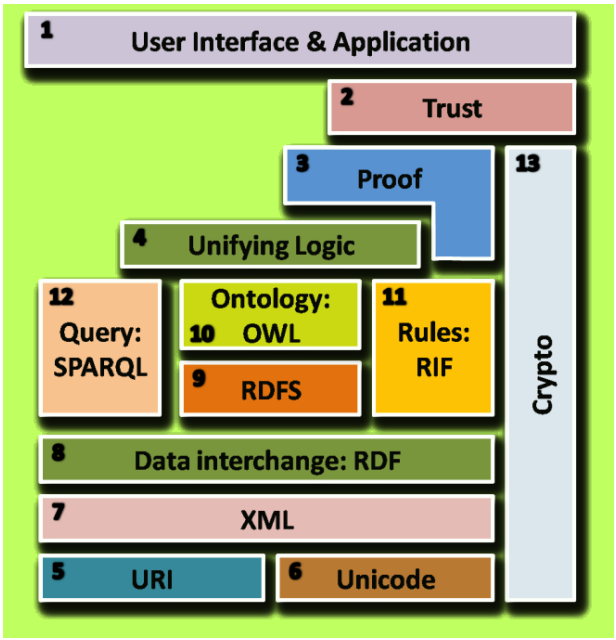



**Figura 2.1 – La pila del Web Semantico**

Tale standardizzazione negli anni ha subito delle revisioni aggiornandosi rispetto alla crescente standardizzazione dei linguaggi di programmazione e delle contestuali innovazioni tecnologiche.
Questo tipo di stratificazione (mostrata in Figura 2.1) ha due funzioni principali: prevenire ad un layer di livello superiore di implementare le funzionalità fornite da un layer sottostante e consentire ad un'applicazione, che conosce solo un layer di livello più basso, di interpretare almeno porzioni di definizioni da un layer di livello superiore.

## 2.3 Linked Data

E' importante avere un'enorme quantità di dati sul Web disponibili in un formato standard, raggiungibili e gestibili dagli strumenti del Web Semantico così che il Web di Dati diventi una realtà. Inoltre, il Web Semantico non solo ha bisogno di accedere ai dati, ma anche le relazioni tra i dati devono essere rese disponibili per creare un Web di Dati. Questa collezione di set di dati interconnessi sul Web può anche essere indicato come Linked Data.
Uno degli obiettivi del Web è di costruire una comunità globale dove chiunque può condividere informazioni o conoscenza. Per questo il Web fa uso di un singolo sistema globale di identificazione che si chiama URI (Uniform Resource Identifier). Gli URI identificano qualsiasi termine o concetto rilevante per codificare le informazioni.
Linked Data evidenzia quattro regole proposte da Tim Berners-Lee per avere un meccanismo per condividere , esporre e connettere i dati sul Web utilizzando identificatori e relazioni:

- Usare gli URI come nomi per le risorse
- Usare URI HTTP in modo che un client (macchina o utente) possa cercare questi nomi
- Quando qualcuno cerca un URI, alcune informazioni utili devono essere fornite
- Includere collegamenti ad altri URI in modo che un client possa scoprire più risorse

Uno dei progetti che si è focalizzato sull'idea e l'implementazione del Web di Dati negli ultimi anni è il Linking Open Data Community Project [11], sponsorizzato dal W3C Semantic Web Education and Outreach Group. Il suo obiettivo è di estendere il Web con dati comuni pubblicando diversi insiemi di dati in RDF e impostando link RDF tra elementi di dati provenienti da diversi fonti.

Sfruttando un browser per il Web Semantico, i link RDF ci permettono di navigare l'informazione proveniente da diverse sorgenti di dati. I link RDF possono essere seguiti anche dai crawler dei motori di ricerca del Web Semantico per fornire poi ricerche e funzionalità di query sofisticate sui dati ricevuti. I risultati delle query sono dati strutturati e possono essere riutilizzati da altre applicazioni. In Figura 2.3 viene mostrato lo stato attuale del Linked Open Data [12]

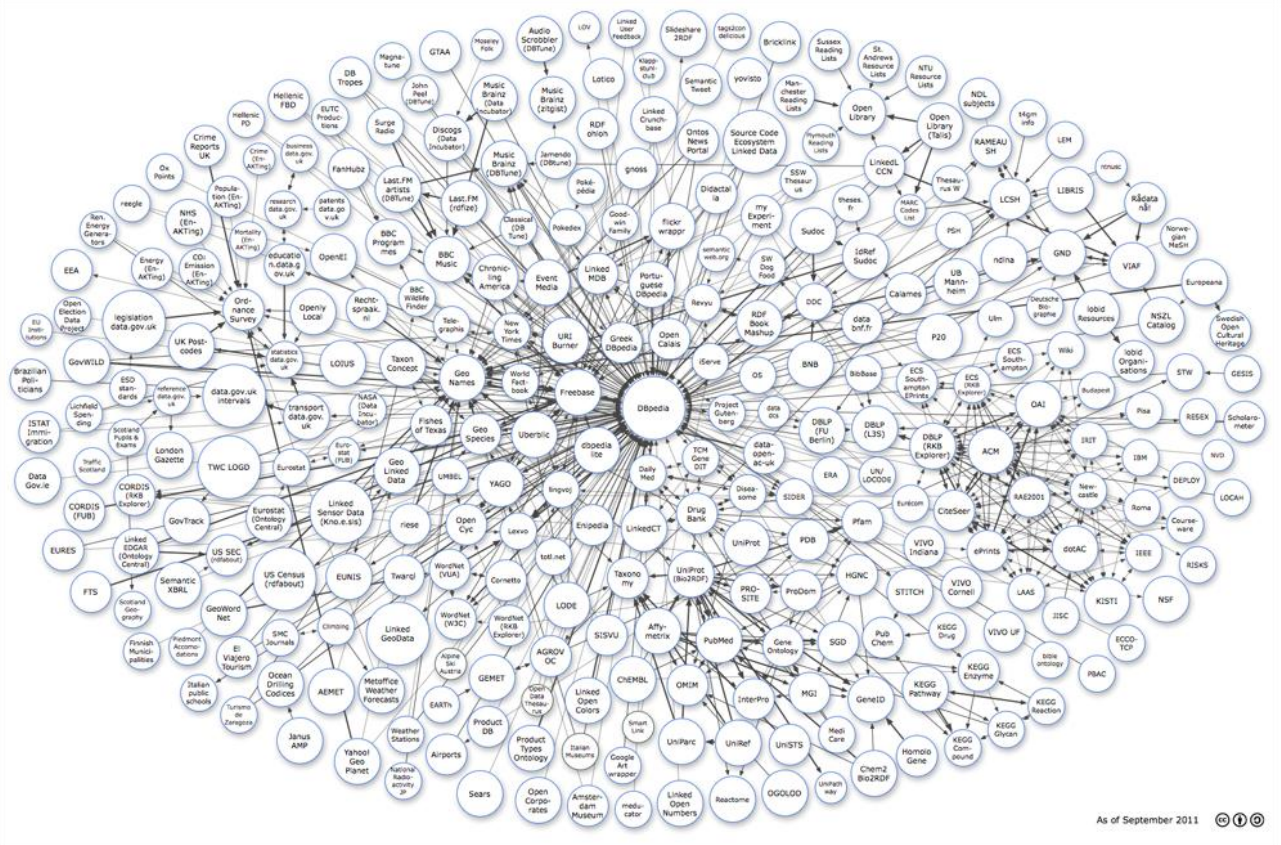



**Figura 2.3 - Linking Open Data cloud diagram**

## 2.4 RDF

Il set di linguaggi secondo W3C su cui costruire il Web Semantico è RDF (Resource Description Framework), un set di linguaggi dichiarativi basati su sintassi XML.
E' stato creato nel 1999 dalla W3C come standard per la codifica dei metadata. L'idea principale di RDF era di definire un nuova piattaforma per la visualizzazione, la manipolazione e l'associazione di collezioni di informazioni distribuite, in modo da garantire l'interoperabilità tra applicazioni che si scambiano informazioni. Questa visione è stata inizialmente descritta nell'articolo di Tim Berners-Lee, Semantic Web Road map [13]. Nel 2004, RDF è diventato una raccomandazione W3C e nei tempi d'oggi è considerato lo standard RDF, rispondendo ancora agli obiettivi di interconnettere l'informazione e garantire l'interoperabilità. RDF è organizzato in grafi composti da strutture base semplici e da un unico sistema globale di identificazione fornito dagli URI. I grafi non hanno radice e nessuna singola risorsa ha un significato intrinseco rispetto a qualsiasi altra. Questo semplifica molto la combinazione dei grafi senza avere il vincolo della radice. RDF è un modello di dati per le risorse e le relazioni fra loro e fornisce una semplice semantica dei dati i quali possono essere rappresentati in sintassi XML.
RDF riutilizza l'approccio del Web per identificare le risorse tramite gli URI e permette a chiunque di rappresentare in modo esplicito la relazione tra queste risorse. Tali dichiarazioni possono

provenire da qualsiasi altra sorgente sul Web e possono essere mischiate con altre dichiarazioni con l'obiettivo di un'integrazione globale dei dati. Usando e riusando gli URI chiunque può dire qualsiasi cosa su qualsiasi argomento, chiunque può aggiungere informazioni sulle cose dette e così via.
Inoltre sfruttando RDFS si possono definire classi e proprietà specifiche su un dominio per descrivere queste risorse e organizzarli in gerarchie. Questi schemi possono anche essere pubblicati e scambiati in formato RDF. RDF fornisce una raccomandazione per pubblicare e collegare i dati. RDFS fornisce una raccomandazione per condividere la semantica dei loro schemi. RDF e RDFS sono stati riutilizzati in diverse altre attività del W3C come standard di fatto per i metadati e come fornitore del layer fondamentale per produrre informazioni interpretabili per le macchine.


**Figura 2.4 – I grafici RDF sulla base dello stak della raccomandazione W3C**

RDF è stato inizialmente pubblicato come un layer base degli standard del Web Semantico e si è evoluto verso un modello general-purpose di rappresentazione dei dati in grafi diventando, insieme a XML, uno dei due modi per strutturare i dati in tutti gli standard Web conosciuti (Figura 2.4). RDF offre un modello di dati interoperabile, flessibile, estensibile e schematico.

RDF usa un modello astratto composto dai seguenti componenti chiave:

- dichiarazione o tripla (statement)
- soggetto e oggetto (subject and object)
- predicato (predicate)

L'informazione viene suddivisa in dichiarazioni o meglio statement che hanno una semantica definita in modo chiaro, strutturata in modo tale che i programmi applicativi possono operare con le informazioni espresse.
Le dichiarazioni sono formate da una tripla ordinata Soggetto-Predicato-Oggetto, dove il soggetto è una risorsa, il predicato è una proprietà e l'oggetto è un valore (Figura 2.5). Quest'ultimo può essere

a sua volta una risorsa o un letterale. La collezione di queste dichiarazioni (triple) è chiamato grafo RDF (soggetto = nodo di partenza; oggetto = nodo d'arrivo).
Le triple sono uno strumento potente per l'integrazione dell'informazione. Le triple sono solo collezioni di URI (e letterali) e ogni URI (o letterale) ha intrinsecamente uno scopo globale. L'uso di nomi globali è di fondamentale importanza perché significa che le triple possono essere unite senza la traduzione dei nomi. Poichè ogni dichiarazione costituente un grafo può essere utilizzata senza traduzione, interi grafi possono essere trasportati e combinati senza nessuna traduzione, rendendoli validi in ogni contesto e in ogni sistema.

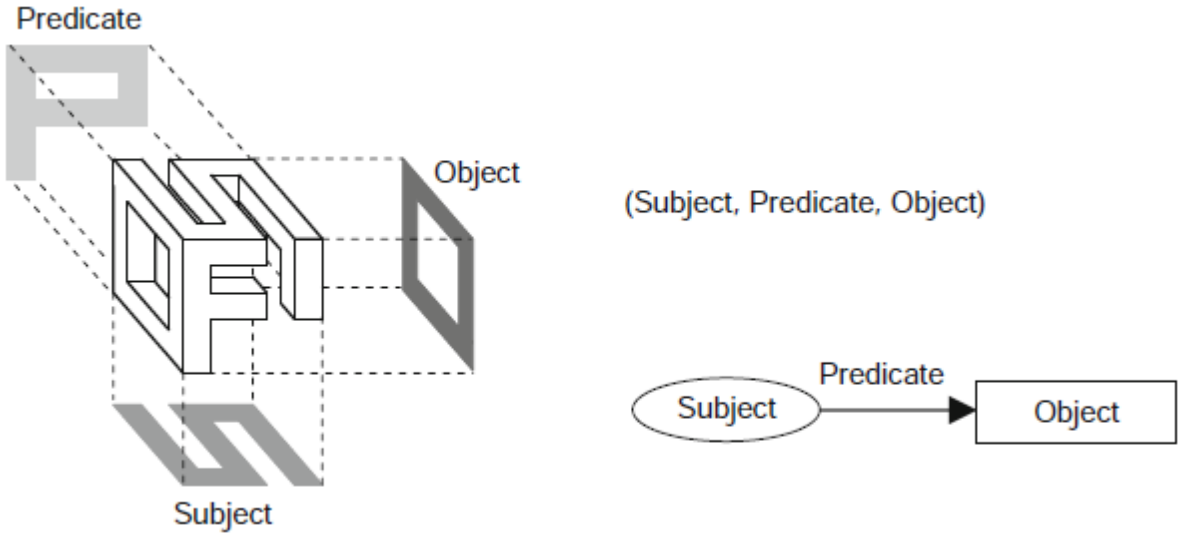



**Figura 2.5 – la tripla RDF, l'atomo della conoscenza per il Web Semantico**

I nodi di un grafo RDF sono i soggetti e gli oggetti delle dichiarazioni che compongono il grafo. Esistono due tipi di nodi: risorse e letterali. I letterali rappresentano valori concreti di dati come numeri o stringhe e non possono essere soggetti di dichiarazioni, bensì solo oggetti. Le risorse rappresentano tutto il resto e possono essere sia soggetti che oggetti.
I letterali RDF sono semplici dati di testo e possono essere usati come valori di proprietà riguardo un soggetto. I predicati, chiamati anche proprietà, rappresentano le relazioni fra soggetti e oggetti, quindi fra risorse. I predicati stessi sono risorse e le dichiarazioni RDF possono essere fatte sui predicati nello stesso modo come per le risorse. Anche il nome dei predicati deve essere globale e identificato da un URI. In questo modo si garantisce la distinzione fra predicati e la possibilità di aggiungere delle informazioni aggiuntive su di loro.
Non tutti i nodi hanno un URI come identificatore nello spazio globale dei nomi e queste risorse sono chiamate nodi vuoti o blank nodes [14]. Questo perché si possono trovare situazioni in cui non si conosce l'URI di una risorsa a cui si desidera fare riferimento o dove non vi è alcun identificatore disponibile.

Il modello astratto RDF deve seguire delle regole per far si che:

- le dichiarazioni RDF siano comprensibili per la macchina;
- sia garantita l'aggregazione dell'informazione distribuita quindi esprimere il significato di un documento Web in un modo processabile per le applicazioni.

Queste regole sono:

- la conoscenza (o l'informazione) è espressa come una lista di dichiarazioni. Ogni dichiarazione prende la forma di Soggetto-Predicato-Oggetto e questo ordine non dovrebbe mai essere cambiato;
- il nome di una risorsa deve essere globale e deve essere identificato da un URI (Uniform Resource Identifier). Medesima regola vale anche nel nome del predicato.
- posso parlare di qualsiasi risorsa di mia volontà e se scelgo di usare un URI esistente per identificare la risorsa è vero che:
    - la risorsa di cui sto parlando e la risorsa già identificata da questo URI esistente sono la medesima.
    - tutto ciò che è stato detto su questa risorsa viene considerata una ulteriore conoscenza su questa risorsa.

### 2.4.1 RDF/XML

Un possibile tipo di serializzazione di RDF, pubblicato dal W3C, è RDF/XML. Lo scopo principale di RDF/XML è di essere processabile dalle macchine usando uno standard di fatto per la formattazione dei documenti Web come XML. Questo permette ai documenti RDF di essere facilmente scambiati fra sistemi e applicazioni molto differenti. RDF/XML è ingombrante da leggere perché non è destinato ad un uso umano. XML forza grafi arbitrari RDF ad essere rappresentati in alberi rendendo RDF/XML uno strumento molto verboso ostacolando l'espressività. Purtroppo non esiste una rappresentazione canonica per RDF/XML è questo implica che si verificano delle differenze usando degli strumenti diversi per la serializzazione dello stesso grafo.
Ogni documento RDF/XML inizia con un elemento radice <rdf:RDF> che contiene la dichiarazione dei namespace che sono definiti esattamente come in XML. Per indicare la risorsa si usa l'elemento rdf:Description con l'attributo rdf:ID o rdf:about; se l'oggetto del predicato è una risorsa, allora il nodo conterrà a sua volta un elemento rdf:Description con l'attributo rdf:about che contiene l'URI della risorsa; se invece l'oggetto è un letterale, allora il nodo del predicato conterrà il valore semplicemente come testo. Se il letterale dovesse essere tipizzato, allora il nodo predicato conterrà l'attributo rdf:datatype. I riferimenti tramite rdf:ID o rdf:about permettono la specificazione dei grafi che sono nascosti negli alberi XML.
Quindi i soggetti delle dichiarazioni sono costruiti usando l'attributo rdf:about di un tag rdf:Description. Gli oggetti delle dichiarazioni si presentano negli attributi rdf:resource dei predicati. I letterali si presentano come contenuto del testo di un predicato. Quindi gli URI si possono utilizzare come tipi di dato per letterali creando dei tipi personalizzati.

### 2.4.2 N-Triples

N-Triples è una serializzazione molto semplice ma dettagliata [15]. Ogni linea di output in formato N-Triple contiene un soggetto, predicato e oggetto seguito da un punto. Fatta eccezione per i blank nodes e i letterali, i soggetti, i predicati e gli oggetti sono espressi come URI assolute racchiuse tra parentesi angolari. La serializzazione N-Triples consente nuove linee tra le risorse e il punto rappresenta il termine di una tripla, altrimenti sarebbe stato difficile scrivere una tripla in una singola linea.

### 2.4.3 Notation 3 RDF (N3)

La rappresentazione più semplice degli enunciati è scriverla per intero, dove ogni tripla è rappresentata come: URI del soggetto, URI del predicato, URI dell'oggetto o rispettivo letterale.
Questa rappresentazione si chiama Notation3 o N3 ed è una forma più compatta di serializzazione sviluppata da Tim Berners-Lee [16]. N3 non è una serializzazione XML del modello RDF ed è stato progettato per rendere più leggibile l'RDF. È quindi molto più leggibile del formatto XML/RDF. Gli N-Triples sono concettualmente molto semplici ma si possono constatare molte ripetizioni. L'informazione ridondante richiede tempo supplementare per essere trasmessa e analizzata. Quando si lavora con grandi quantità di dati questo rappresenta un inconveniente. Aggiungendo alcune strutture, N3 concentra molto la ripetizione nel formato N-Triple.
In sintesi, la notazione N3:

- fornisce, come XML, un meccanismo di namespace per generare brevi Qualified Name (qnames) per i nodi dei grafi;
- permette di definire un prefisso URI e identificare gli URI delle entità relativi ad una serie di prefissi dichiarati all'inizio del documento;
- riduce le ripetizioni anche nel caso in cui dobbiamo fare delle dichiarazioni multiple sullo stesso soggetto semplicemente usando il punto e virgola dopo la prima dichiarazione;
- offre anche dei shortcut che ci permettono di esprimere un gruppo di dichiarazioni che condividono lo stesso blank node senza dover specificare un nome interno per il nodo stesso.

### 2.4.4 TURTLE

Turtle (Terse RDF Triple Language) [17] è un formato testuale compatto per serializzare un grafo RDF, meno vincolato di N-Triples e ancora più semplice di N3. Turtle non ha la restrizione della singola linea per dichiarazione come N-Triples, e in più fornisce varie abbreviazioni per gli URI. A Turtle non vengono applicati determinati vincoli imposti a RDF/XML dall'uso di XML e namespace che gli proibiscono di codificare tutti i grafi RDF.
Le caratteristiche del linguaggio Turtle sono:

- ogni dichiarazione deve finire con un punto;
- il soggetto deve essere rappresentato da un URI ;
- il predicato deve essere rappresentato da un URI ;
- l'oggetto può essere un URI o un letterale;
- un URI deve essere racchiuso con le parentesi angolari che sono usate per delineare un dato URI.

I letterali vengono scritti usando i doppi apici quando non contengo interruzioni di righe come per esempio "letterale semplice" oppure """letterale lungo""" quando contengono interruzioni di linee.
Un dato letterale può avere come suffisso un URI per la lingua o per il tipo di dato, ma non è permesso averli entrambi. Se è stato dato un suffisso per la lingua, il suffisso è formato dal carattere @ insieme al tag della lingua.

# Capitolo 3

# Ontologie per il Web Semantico

## 3.1 Introduzione

L'interoperabilità semantica è la capacità dei sistemi informatici di trasmettere dati con un significato non ambiguo e condiviso. Realizzare l'interoperabilità semantica tra diversi sistemi computazionali è molto impegnativo e soggetto ad errori in un'ambiente distribuito ed eterogeneo come il Web. L'eterogeneità delle informazioni si verifica a tre livelli: la sintassi, la struttura e la semantica. L'eterogeneità sintattica è relativa all'uso di differenti formati di dati. Per risolvere questo problema, formati standard come XML, RDF, RDFS e OWL vengono utilizzati per descrivere dati in modo uniforme rendendo più semplice il trattamento automatico delle informazioni condivise. La standardizzazione ha un ruolo importante riguardo l'eterogeneità sintattica, però non risolve l'eterogeneità strutturale in cui l'informazione viene strutturata anche in ambienti omogenei sintatticamente. Nonostante siano state sviluppate soluzioni per risolvere il problema dell'eterogeneità strutturale, resta da risolvere il problema dell'eterogeneità semantica. L'eterogeneità semantica si verifica quando due contesti non condividono la stessa interpretazione dell'informazione. Sfruttare le ontologie come componente chiave per risolvere il problema dell'eterogeneità semantica è funzionale all'interoperabilità semantica tra diversi applicazioni e servizi web.

## 3.2 Ontologia

L'ontologia fornisce un vocabolario semantico per definire il significato delle cose. Studia ciò che esiste in un dominio di interesse e codifica la conoscenza di questo dominio in una forma processabile per la macchina per renderlo a disposizione ai sistemi informativi.
Un'ontologia si occupa di un dominio specifico e definisce i termini usati per descrivere e rappresentare un'area di conoscenza. Le ontologie sono usate da applicazioni, basi di dati e utenti che necessitano di condividere l'informazione del dominio, e contiene i termini e le relazioni tra di loro. I termini sono spesso chiamati classi oppure concetti, e queste parole sono interscambiabili. La relazione tra queste classi possono essere espressi usando una struttura gerarchica: la super-classe rappresenta concetti di alto livello e le sotto-classi quelli più fini. I concetti più fini hanno tutti gli attributi e caratteristiche che i concetti di più alto livello hanno. Oltre al rapporto fra le classi esiste un altro livello di relazione: le proprietà. Queste proprietà descrivono varie caratteristiche e attributi dei concetti e possono essere usate per associare insieme diverse classi. Le relazioni fra le classi non sono solo super-classe o sotto-classe ma anche relazioni espresse in termini di proprietà. Definendo i termini e le relazioni chiaramente, l'ontologia codifica la semantica del dominio in maniera che sia comprensibile e interpretata da una machina rendendo possibile l'elaborazione a larga scala.
L'ontologia fornisce delle definizioni comuni e condivise su certi concetti chiave di un dominio, ci propone i termini che possiamo usare per creare documenti RDF nel dominio, un modo di riusare la conoscenza e rende espliciti i presupposti del dominio. Insieme ai linguaggi di descrizione dell'ontologia (come RDFS e OWL), ci fornisce una maniera di codificare la conoscenza

## 3.3 RDFS (RDF Schema)

RDF Schema [18] è una raccomandazione di W3C e rappresenta un'estensione semantica di RDF che comprende i costrutti per creare un vocabolario e descrivere classi, sottoclassi e proprietà di risorse RDF.
Il linguaggio di descrizione del vocabolario RDF è formato da classi e proprietà, simile al linguaggio di programmazione Java. RDF si differenzia da questi sistemi perché, invece di definire una classe in termini delle proprietà che le sue istanze possono avere, descrive le proprietà in termini delle classi delle risorse a cui si applicano.
Tutti i termini RDFS sono indentificati da degli URI predefiniti che condividono la stringa principale http://www.w3.org/2000/01/rdf-schema#. Questa stringa URI è associata con il namespace con prefisso rdfs: ed è particolarmente usata nel formato RDF/XML con prefisso rdfs.
I termini RDFS possono essere suddivisi nei seguenti gruppi:

- classi in cui sono inclusi i termini RDFS che possono essere usati per definire classi e include i termini come: rdfs:Class, rdfs:Resource, rdfs:Literal, rdfs:Datatype.
- proprietà in cui sono inclusi i termini RDFS che possono essere usati per definire proprietà e include i termini come: rdfs:subClassOf, rdfs:subPropertyOf, rdfs:range, rdfs:domain, rdfs:comment, rdfs:label,.
- utilità in cui sono inclusi i termini RDFS che sono usati per vari fini e include i termini come: rdfs:isDefinedBy e rdfs:seeAlso.

## 3.4 OWL

Nel novembre del 2001 W3C ha creato il Web Ontology Working Group. OWL è diventato un W3C Recommendation ufficiale il 10 Febbraio del 2004 (OWL 1) [19].
La standardizzazione di OWL ha portato allo sviluppo di molte ontologie in OWL in molti campi Un esempio è la comunità delle scienze naturali dove è diventato uno standard di fatto per lo sviluppo delle ontologie e lo scambio dei dati.
Web Ontology Language (OWL) è il linguaggio più popolare per la creazione delle ontologie. OWL ha esattamente lo stesso scopo di RDF Schema, ovvero definire ontologie che comprendono classi, proprietà e le loro relazioni per un dominio specifico applicativo. Questo linguaggio estende le funzionalità offerte da RDF Schema e fornisce la capacità di esprimere relazioni più complesse e ricche. Per questo motivo viene usato spesso OWL per lo sviluppo di ontologie. Poichè OWL è basato su RDF Schema , tutti i termini contenuti nel vocabolario RDFS possono essere utilizzati nella creazione di documenti OWL.
Il vasto numero di campi in cui OWL è stato applicato ha rilevato inoltre delle carenze dal punto di vista dell'utente. Gli ingegneri e i progettisti degli strumenti OWL hanno identificato delle limitazioni importanti nell'espressività e la praticità di OWL. Pertanto il 27 Ottobre del 2009, con l'uscita dello standard OWL2, il lavoro di sviluppo del W3C OWL Working Group ha portato dei miglioramenti e aggiunto nuove caratteristiche al OWL 1, con le seguenti specifiche di base:
OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax [20]
OWL 2 Web Ontology Language Mapping to RDF Graphs [21]
OWL 2 Web Ontology Language Direct Semantics [22]
OWL 2 Web Ontology Language RDF-Based Semantics [23]
OWL 2 Web Ontology Language Conformance [24]
OWL 2 Web Ontology Language Profiles [25]

In tutti i casi pratici le ontologie OWL 1 sono valide ontologie OWL 2 con inferenze identiche.

La Figura 3 mostra il linguaggio OWL 2 rappresentando i blocchi principali che lo costituiscono e le loro relazioni. In cima ci sono varie sintassi concrete che possono essere utilizzate per serializzare e scambiare le ontologie, nel centro invece c'è la nozione astratta di un'ontologia che può essere rappresentata come una struttura astratta o come un grafo RDF.

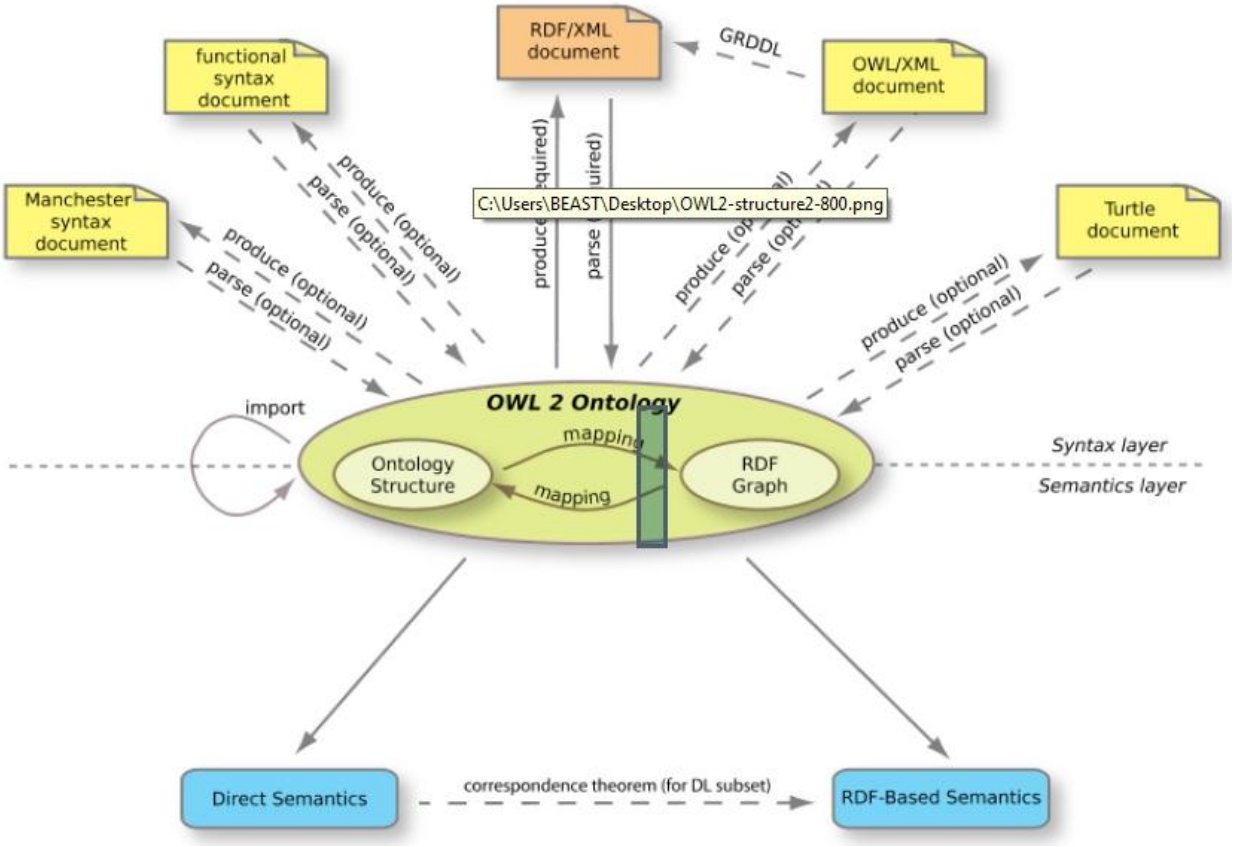



**Figura 3 – La struttura del OWL 2**

OWL è una collezione di termini per definire classi e proprietà per uno specifico dominio applicativo. Questi termini predefiniti hanno tutti come stringa iniziale il seguente URI http://www.w3.org/2002/07/owl#.

### 3.4.1 Le nozioni base di OWL

Le nozioni base di OWL sono [26]:

- Gli assiomi sono una dichiarazione base di un'ontologia e ciascuno rappresenta un pezzo basilare della conoscenza. Ogni assioma dovrà coinvolgere qualche classe, proprietà e degli individuali
- Le entità sono elementi atomici degli assiomi in OWL; l'entità individuale viene chiamata oggetto, un'entità della classe viene chiamata categoria e un'entità di proprietà viene chiamata relazione
- Tramite le espressioni è possibile combinare diverse entità sia di classe che di proprietà per crearne delle nuove. Le espressioni rappresentano l'espressività più avanzata che OWL ha rispetto a RDFS
- IRI (Internationalized Resource Identifiers) sono come gli URI ma possono utilizzare l'intera gamma dei caratteri Unicode (mentre, gli URI sono limitati al sottoinsieme di caratteri ASCII)

### 3.4.2 Le sintassi in OWL

OWL offre diverse sintassi per la condivisione, modifica e la persistenza delle ontologie:

- Functional-Style syntax è progettata per tradurre le specifiche strutturali di diverse sintassi ed è la più utilizzata
- RDF/XML syntax è l'unica sintassi il cui supporto è obbligatorio per strumenti OWL e per tale motivo viene usata dalle ontologie più conosciute
- Manchester syntax fornisce una rappresentazione testuale delle ontologie OWL ed è usata per modificare le espressioni di classe
- OWL/XML è il formato per rappresentare le ontologie OWL, è conforme ad un XML Schema pertantoè possibile utilizzare gli strumenti XML per l'elaborazione e l'esecuzione di task per le query

# Capitolo 4

# SPARQL

## 4.1 Introduzione

SPARQL (Simple Protocol And RDF Query Language) è il linguaggio di interrogazione per il recupero dei dati espressi in RDF nonchè un linguaggio di query RDF e un protocollo di accesso ai dati per il Web Semantico. Rappresenta l'ultimo tassello per l'edificazione del Semantic Web da W3C e il 15 Gennaio del 2008 è stato standardizzato dallo SPARQL Working Group del W3C.
SPARQL consiste in tre specifiche:

- SPARQL Query Language specification [27] che costituisce il nucleo
- SPARQL Query Results Format specification [28] che descrive un formato XML per serializzare il risultato di una query SPARQL
- SPARQL Protocol for RDF specification [29] che utilizza WSDL 2.0 per definire protocolli semplici HTTP e SOAP per fare le query in remoto su basi di dati RDF

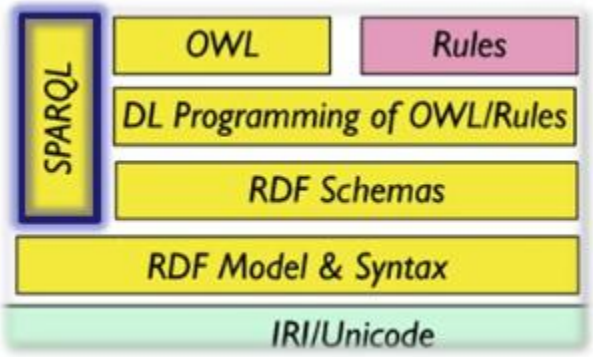



**Figura 4: Architettura Semantic Web**

Lo SPARQL Protocol for RDF permette a un generico client di interrogare uno o più endpoint SPARQL inviando una richiesta espressa nel linguaggio di interrogazione e ricevendo come risposta il risultato in formato XML. Il protocollo SPARQL è basato su WSDL 2.0 (Web Services Description Language) [30] e la sua specifica descrive sia l'interfaccia astratta, sia i legami di questa verso gli standard attualmente utilizzati sul Web.
Lo SPARQL Query Language è un linguaggio di interrogazione pensato per il Web, infatti oltre a prevedere la clausola SELECT come in SQL, offre anche altri costrutti. E' possibile infatti prevedere che :

- non si conosca a priori lo schema dei dati. Per risolvere questo problema SPARQL ha la clausola DESCRIBE, che permette di ottenere una descrizione della risorsa cercata;
- si presenti la necessità di sapere se un certo enunciato o un certo pattern di dati sia presente nella sorgente dati. Per questo motivo SPARQL propone la clausola ASK.

In aggiunta SPARQL consiste in un linguaggio di query (ovvero un formato XML) dove i risultati delle query verranno restituiti, e un protocollo per sottoporre una query ad un servizio di elaborazione di query in remoto.
Finora abbiamo considerato RDF come un modello e/o formato che è possibile utilizzare per creare contenuti strutturati da leggere per la macchina. Come abbiamo visto, con l'utilizzo di RDF Schema e il linguaggio OWL si possono creare ontologie. Siccome i documenti RDF creati condividono

queste ontologie comuni è più semplice per le macchine trarre conclusioni basate sui contenuti di questi, e generare ulteriori dichiarazioni RDF (questo procedimento viene chiamato reasoning). Come risultato si avranno più contenuti espressi in formatto RDF. A questo punto una grande quantità di documenti RDF viene pubblicata in Internet e comincia a prendere forma un Web leggibile per le macchine. In seguito c'è la necessita di individuare informazioni specifiche nel Web di Dati. Una possibile soluzione può essere quella di costruire un nuovo motore di ricerca che lavorerà sul Web Semantico. Seppure questo motore di ricerca avrà prestazioni migliori di quelli tradizionali non userà pienamente il potenziale del Web Semantico. Il risultato sarà una collezione di pagine che potrebbero contenere la risposta ma non direttamente. Per arrivare ad una soluzione migliore avremo bisogno di un linguaggio query che possiamo utilizzare su questi dati del Web e, semplicemente sottoponendo una query, saremo capaci di ricevere direttamente la risposta. E' in questo contesto che si colloca SPARQL.
I vantaggi principali di avere un linguaggio di query come SPARQL sono:

- fare le query a dei grafici RDF per ricevere informazioni specifiche;
- fare le query ad un server remoto RDF e ricevere indietro i risultati in streaming;
- eseguire delle query regolarmente in automatico verso insiemi di dati RDF per generare resoconti;
- consentire lo sviluppo di applicazioni ad un livello più alto; le applicazioni possono lavorare con risultati delle query SPARQL, non direttamente con le dichiarazioni RDF.

Nonostante tale vantaggi, rimane un linguaggio incompleto in confronto ad altri linguaggi di interrogazione, ma ciò è dovuto al fatto che è ancora in fase di sviluppo. Una query SPARQL si basa sul graph matching [31] e si compone di quattro parti:

- RDF dataset che permette di scegliere il grafo su cui eseguire la query;
- il graph pattern che applica il graph matching e crea la soluzione parziale della query;
- i modificatori di soluzione che applicati alla soluzione parziale manipolano il risultato;
- la forma del risultato in cui si può scegliere l'output della query.

## 4.2 RDF Data Store

Un RDF Data Store è lo strumento di memorizzazione dati RDF rendendoli poi disponibili per un utilizzo futuro. Quindi un RDF Data Store (RDF Database o Triple Store) è un sistema particolare di basi di dati pensato per la memorizzazione e il recupero delle dichiarazioni RDF. Diversamente, i sistemi di gestione delle basi di dati relazionali (DBMS) sono costruiti per scopi generali. Siccome ogni progetto ha le sue caratteristiche, tutte funzionalità devono essere più generali possibile, e quindi le prestazioni non possono avere la priorità assoluta. Mentre, se dedichiamo un sistema di basi di dati solo per la memorizzazione delle dichiarazioni RDF, conoscendo a priori il tipo di modello di dati, che dovrà contenere dichiarazioni nella forma soggetto-predicato-oggetto, possiamo mirare ad ottenere prestazioni migliori. E' per questo motivo che nascono i Triple Store. Alcuni esempi di RDF Data Store sono Redland [32], ARC [33], 4store [34] e Virtuoso [35] implementati in C JENA [36], Joseki [37] e Sesame [38] implementati in Java.

## 4.3 SPARQL endpoint

Uno SPARQL endpoint rappresenta l'interfaccia a cui gli utenti possono accedere per fare delle query ad un RDF Data Store usando il linguaggio SPARQL. La sua funzione è accettare delle query e riportare un risultato. Per gli utenti un endpoint può essere un'applicazione stand-alone oppure Web-based. Diversamente dalle applicazioni, gli endpoint prendono la forma di un insieme di API che possono essere utilizzate dall'agente chiamante.

Uno SPARQL endpoint può essere configurato per riportare risultati in diversi formati, ad esempio, se usato dagli utenti in modo iterativo presenta il risultato in forma di una tabella HTML che può essere costruita applicando XSLT [39] sui risultati XML riportati dall'endpoint. Se, invece vengono acceduti dalle applicazioni, i risultati sono solitamente serializzati in un formato processabile per le macchine (per esempio XML).

## 4.4 Le path expression

SPARQL adotta la sintassi Turtle (un'estensione di N-Triples), alternativa al tradizionale RDF/XML. Per esprimere le interrogazioni, SPARQL introduce il concetto di path expression:
la path expression è l'insieme delle triple necessarie a rispondere all'interrogazione, in cui sostituiamo uno o più identificativi (risorse o letterali) con una variabile, espressa da una parola arbitraria preceduta dal simbolo "?".
Inoltre, la path expression rappresenta il pattern dei dati che vogliamo recuperare e quindi i risultati dell'interrogazione saranno tutte e sole le triple RDF che soddisfano la path expression sostituendo alle variabili le risorse o i letterali corrispondenti.
Le query SPARQL si basano in particolare sul triple pattern [40] che ricalca la configurazione a triple delle asserzioni RDF, fornendo un modello flessibile per la ricerca di corrispondenze. Infatti, soggetto, predicato e oggetto possono essere delle
variabili come nell'esempio sottostante che rappresentano una tripla che costituisce una path expression:

```
?titolo cd:autore ?autore
```

Al posto del soggetto e dell'oggetto questo triple pattern prevede due variabili, contrassegnate con "?". Le variabili fungono in un certo senso da incognite dell'interrogazione, cd:autore funge invece da costante: le triple RDF che trovano riscontro nel modello assoceranno i propri termini alle variabili corrispondenti.
Le forme di query di SPARQL sono:

- SELECT query
- CONSTRUCT query
- ASK query
- DESCRIBE query

Tra queste, la query SELECT è la forma di query più utilizzata. Molte forme di query SPARQL contengono un insieme di tripple pattern chiamati graph pattern. I tripple pattern sono come le triple RDF ad eccezione che ciascuno degli elementi della tripla (soggetto, predicato e oggetto) può essere una variabile, come mostrato nell'esempio seguente:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
<http://danbri.org/foaf.rdf#danbri> foaf:name ?name.
```

In questo esempio il soggetto di questo triple pattern è l'URI di Dan Brickley, il predicato è foaf:name e il componente oggetto della triple pattern è una variabile, identificata dal carattere ? all'inizio della stringa name.

Un esempio invece di graph pattern (ovvero una collezione di triple pattern) è il seguente:

```
{
        ?who foaf:name ?name.
        ?who foaf:interest ?interest.
        ?who foaf:knows ?others.
}
```

Per capire come un graph pattern viene usato per selezionare le risorse da un dato grafo RDF dobbiamo ricordare un punto chiave sul graph pattern: se una variabile compare in diversi triple pattern all'interno del graph pattern, il suo valore in tutti i triple pattern in cui compare deve essere lo stesso. In altre parole ogni risorsa restituita deve essere in grado di sostituire tutte le occorrenze della variabile.
Per chiarire meglio questo concetto, ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
?titolo cd:anno ?anno
}
```

Nella prima riga viene dichiarato il namespace utilizzato e a differenza della sintassi N3, la parola chiave PREFIX è senza il simbolo '@' ed alla fine della dichiarazione non c'è il punto. Se si volesse dichiarare un namespace di default, si potrebbe usare la parola chiave **BASE** al posto di PREFIX.
Nelle righe successive ci sono altre parole chiave del linguaggio SPARQL:

- **SELECT** definisce le variabili di ricerca da prendere in considerazione nel risultato;
- **FROM** specifica il set di dati su cui opererà la query. E' inoltre possibile utilizzare le clausole FROM **NAMED** e la parola chiave **GRAPH** per specificare più set di dati;
- **WHERE** definisce il criterio di selezione specificando tra parentesi graffe uno o più "triple patterns" separati da punto.

La query precedente ha catturato esclusivamente le triple dotate di tutti e tre i termini richiesti (titolo, autore, anno). È possibile riformulare la query in modo più flessibile, con la possibilità di inserire triple in cui vi sia l'assenza di alcuni termini come mostra l'esempio seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
?titolo cd:autore ?autore.
OPTIONAL { ?titolo cd:anno ?anno }
}
```

Nell'esempio precedente, il secondo pattern è dichiarato opzionale: l'informazione è inclusa nel risultato solo se disponibile, altrimenti le variabili appariranno prive di valore. Le risorse prive della proprietà 'anno' sono mostrate ugualmente e le celle dei valori mancanti sono lasciate vuote. Un altro modo che ci assicura una certa flessibilità nel reperimento dei dati è il seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
{ ?titolo cd:autore ?autore. }
UNION { ?titolo cd:anno ?anno }
 }
```

La parola chiave **UNION** esprime un OR logico: la query non si limita pertanto alle triple che soddisfano entrambi i triple patterns, ma sia quelle che soddisfano solo il primo, sia quelle che soddisfano solo il secondo. È possibile mettere restrizioni sui valori da associare alle variabili:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
?titolo cd:autore ?autore.
FILTER ( ?anno > 2000 )
 }
```

La restrizione è effettuata tramite l'operatore di confronto '>'e il filtro esclude i termini che non soddisfano la condizione definita tra le parentesi tonde. Gli operatori utilizzabili all'interno di una clausola FILTER sono costituiti da connettivi logici (AND e OR, rappresentati da '&&' e '||'), operazioni di comparazione (come '>', '<', '=', '!=', ecc.), espressioni regolari ed una serie di operatori unari specifici di SPARQL.
Un esempio di operatore specifico di SPARQL è 'regex' [41], il quale permette di usare espressioni regolari per il matching dei letterali. Nel seguente esempio il filtro seleziona, senza differenza fra maiuscole o minuscole, solo gli autori che iniziano per "au":

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM <http://cd.com/listacd.ttl>
WHERE {
?titolo cd:autore ?autore.
FILTER regex (?autore, "^au", "i")
}
```

E' possibile escludere dal risultato i valori duplicati mediante la parola chiave **DISTINCT**:

```
SELECT DISTINCT ?titolo ?autore
```

Altri costrutti supportati da SPARQL per la manipolazione del risultato sono:

```
ORDER BY DESC ( ?autore )
LIMIT 10
OFFSET 10
```

- **ORDER BY** imposta l'ordine dei risultati della query in base ad una certa variabile;
- **LIMIT** impone restrizioni al numero dei risultati, limitandoli, secondo quanto indicato;
- **OFFSET** permette di "saltare" un certo numero di risultati.

Le query **CONSTRUCT** consentono di restituire il risultato dell'interrogazione sotto forma di grafo RDF, sulla base di un template [42]. Questo è costituito prendendo ogni soluzione della query nella sequenza della soluzione, sostituendo le variabili nel template del grafo e combinando le triple in un unico grafo RDF tramite l'unione.
L'esempio seguente mostra come viene utilizzato:

```
PREFIX cd: <http://example.org/cd/>
CONSTRUCT { ?titolo ?autore }
FROM <http://cd.com/listacd.ttl>
WHERE {
?titolo cd:autore ?autore.
}
```

Le applicazioni possono usare la forma **ASK** per testare se un pattern di query possiede una soluzione. Non viene restituita alcuna informazione sulla possibile soluzione, ma viene solo testato se esiste oppure no una soluzione.

```
PREFIX cd: <http://example.org/cd/>
ASK { ?x cd:autore "Elisa" }
```

La query **DESCRIBE** viene utilizzata nei casi in cui non si conosce molto riguardo un grafo di dati. In questi casi chiediamo di descrivere le risorse che vogliamo conoscere. Dopo aver ricevuto la query, il processore SPARQL creerà e restituirà un grafo RDF; il contenuto del grafo è deciso dal query processor, non dalla query stessa:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE { ?x foaf:mbox <mailto:prova@prova.org> }
```

In questo esempio l'unica cosa che conosciamo è l'indirizzo e-mail. Per questa motivo chiediamo al processore SPARQL di descriverci la risorsa, l'indirizzo e-mail della quale è dato da <prova@prova.org>. Il risultato della query è un altro grafo RDF le cui dichiarazioni sono determinate dal query processor.

## 4.5 Output di una query in formato XML

Il risultato di una query SPARQL può essere serializzato utilizzando il linguaggio XML. Nel seguito viene riportato un esempio.

Eseguendo la seguente query:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM <http://cd.com/listaCd.owl>
WHERE {
?titolo cd:autore ?autore
}
```

Il risultato in formato XML è:

```
<?xml version="1.0"?>
<sparql xmlns=http://www.w3.org/2005/sparql-results#>
<head> <variable name="titolo" /><variable name="autore" /></head>
<results ordered="false" distinct="false">
<result>
<binding name="titolo"><literal>Heart</literal></binding>
<binding name="autore"><literal>Elisa</literal></binding>
</result>
<result>
<binding name="titolo"><literal>Ivy</literal></binding>
<binding name="autore"><literal>Elisa</literal></binding>
</result>
<result>
<binding name="titolo"><literal>Pearl Days</literal></binding>
<binding name="autore"><literal>Elisa</literal></binding>
</result>
</results>
</sparql>
```

La sintassi è leggibile ed intuitiva da gestire mediante fogli di stile. L'elemento radice è "sparql", l'attributo "xmlns" definisce il namespace di riferimento.
Nella sottosezione "head" sono elencate le variabili da prendere in considerazione nel risultato, indicate come valori dell'attributo "name" degli elementi vuoti "variable".
La seconda sottosezione, "results", contiene una sequenza di elementi "result" che esprimono, per ciascun risultato della query, le variabili cercate, indicate come valore dell'attributo "name" dell'elemento "binding", e i rispettivi valori.
I valori booleani degli attributi "ordered" e "distinct" dell'elemento "results" indicano se vengono presi in considerazione o meno gli eventuali costrutti ORDER BY o SELECT DISTINCT della query.

## 4.6 Piattaforme SPARQL

Sono stati sviluppati diverse piattaforme open source che permettono di effettuare il parsing di strutture RDF, l'esecuzione di query ed altre particolari funzioni.
Attualmente le più diffusi piattaforme sono Redland, Twinkle ,Jena, Rap, Sesame, ARC2 e si differenziano principalmente per il linguaggio di programmazione utilizzato (PHP o JSP).

# Capitolo 5

# Applicazione

## 5.1 Introduzione

Uno dei principali obiettivi di Smart-M3 è di offrire la massima interoperabilità ai sistemi che lo usano. All'interno del gruppo di ricerca ARCES dell'Università di Bologna sono già state implementate le query di tipo WQL e RDF-M3 in differenti linguaggi di programmazione.

L'obiettivo della mia tesi è stato quello implementare una particolare funzionalità alle librerie SSAP in linguaggio C#; in particolare estendere la libreria aggiungendo la possibilità di effettuare query SPARQL.
Lo scopo finale sarà di dimostrare l'ampio grado di interoperabilità garantito da Smart-M3 mostrando come si possono usare tecnologie diverse e apparentemente incompatibili per ricavare informazioni che, a loro volta, possono essere usate da qualsiasi altra tecnologia.
Per prima cosa è importante sottolineare che per poter sviluppare un'applicazione C# è necessario aver installato sul proprio pc Visual Studio, scaricabile dal sito della Microsoft in versione di prova, nel mio caso ho utilizzato Visual Studio 2010 l'ultima versione disponibile. L'applicazione è stata interamente realizzata partendo dalle librerie disponibili sul sito di sourceforge in linguaggio C# e nel prossimo paragrafo verrà illustrato come è stata sviluppata.

## 5.2 Implementazione

Innanzi tutto ho scaricato dal sito http://sourceforge.net/projects/m3-csharp-kpi/ la libreria KPI in C#. In particolare l'oggetto principale per l'utilizzo di Smart-M3 è un'istanza della classe KPICore. Al momento della creazione vanno specificate le coordinate in rete della SIB, indirizzo IP e Porta in genere 10010, e il nome dello spazio intelligente, che salvo diversa configurazione sarà "X".

```
KPICore.KPICore core = new KPICore.KPICore("192.168.0.1",10010,"X");
```

e andrà creato l'oggetto parser in questo modo:

```
SSAP_XMLTools parser = new SSAP_XMLTools(
                                        "00000000-0000-0000-0000-DEAD0000BEEF",
                                        "X", http://www.nokia.com/NRC/M3/sib#any
                                         );
```

**SSAP_Message**
Come spiegato nel paragrafo 2.4 Smart Space Access Protocol (SSAP) è un protocollo per la comunicazione tra il Semantic Information Broker (SIB) e i diversi software, basato su messaggi XML. Non prevede fasi di negoziazione complesse tra le uniche due parti coinvolte (la SIB e i singoli KP). SSAP permette di collegarsi o abbandonare il sistema (join e leave) e gestire cinque tipi di transizioni (insert, remove, update, subscription e query).

Un esempio di messaggio SSAP (ovvero del metodo che andrà a costruirlo) è il seguente:

```
private string ssap_message(string transaction_type, string body){
string ssap = "<SSAP_message>"
"<node_id>{" + nodeID + "}</node_id>"
"<space_id>" + SMART_SPACE_NAME + "</space_id>"
"<transaction_type>" + transaction_type + "</transaction_type>"
"<message_type>REQUEST</message_type>"
"<transaction_id>" + ++transaction_id + "</transaction_id>"
body
"</SSAP_message>";
return ssap;
}
```

dove "**body**" è la parte riguardante la richiesta che si vuole effettuare ad es. join, insert, rdf_query, sparql_query,ecc...

Nel mio caso l'interrogazione che devo realizzare è una sparql_query e le informazioni da unire alla body del ssap_message sono:

```
private string ssap_sparql_query(string query){
return "<parameter name = \"type\">sparql</parameter>"
"<parameter name = \"query\">"
query
"</parameter>";
}
```

**Join , Leave e Insert**
Le triple vengono rappresentate da 4 stringhe, quindi gli element in ordine sono soggetto, predicato, oggetto, tipo del soggetto (ovvero Uri o literal).

```
string responce = core.join();
bool conf = parser.isJoinConfirmed(responce);
```

```
string responce = core.leave();
bool conf = parser.isLeaveConfirmed(responce);
```

Si può osservare che prima di eseguire delle operazioni sulla SIB è necessario eseguire la join e successivamente la leave.

Facendo un'insert o un'update, il metodo può richiedere i 4 campi della tripla separatamente nel caso in cui se ne voglia aggiungere solo una, in caso di più triple bisogna passare in ingresso un "grafo" (ossia una collezione di triple) che viene inserito atomicamente, questo permette un considerevole risparmio sul tempo di esecuzione poichè si risparmia l'overhead dato dall'instaurazione della comunicazione con la SIB.

```
ArrayList graph = new ArrayList();
graph.Add(parser.newTriple("sogg1","pred1","ogg1","oggType"));
…
String responce = core.insertGraph(graph);
bool conf =  parser.isInsertConfirmed(responce);
```

**Query RDF**

Le query RDF restituiscono una stringa e per definire i termini di ricerca bisogna passare in ingresso al metodo un "template" di tripla, ovvero usando l'URI speciale "any" ci permette di descrivere una categoria ampia di triple che la SIB restituirà successivamente. Invece il metodo per la getQuery RDF restituisce un ArrayList ovvero un vettore di triple.

```
string responce = core.queryRDF("any", "predicato", "oggetto",  "oggType");
ArrayList result = parser.getQueryTriple(responce);
foreach (string[] item in result){
Console.WriteLine("Subject:" + item[0] + "Predicate:" + item[1] +
Object:" + item[2] + ObjType: " + item[3]);
}
```

**Query Wilbur**

Le query Wilbur restituiscono una collezione di nodi del grafo. Un nodo si può definire come una coppia di stringhe, nel caso in cui il nodo sia un'istanza allora la prima stringa sarà il suo Uri e la seconda sarà "Uri", nel caso in cui sia un valore letterale la prima sarà il valore stesso e la seconda "literal". L'obiettivo di una query Wilbur è di trovare tutti i nodi nel grafo che a partire da un nodo di partenza sono collegati ad esso tramite un percorso specificato. Inizialmente va definito il nodo di partenza. Il nodo consiste in un vettore di stringhe lungo due e deve essere creato "a mano":

```
string[] startNode = parser.newNode("Person", "uri");
```

Dopo aver definito il nodo di partenza, va definito il percorso che va dal nodo di partenza fino ai nodi che vogliamo ottenere alla fine della query. Il percorso viene definito secondo la sintassi descritta nel documento, il manuale di Piglet [43], un'infrastruttura software all'interno della SIB che gestisce il grafo delle informazioni. Il percorso si esprime come un "pattern" e deve rispettare questa struttura:

```
Pattern ::= '[' operator { ',' pattern }* ']' | atom
```

Gli operatori di interesse in questo contesto sono "seq" per esprimere una sequenza di elementi, e "inv" che rappresenta il concetto di inversione dell'elemento al quale si riferisce.
Le proprietà, i predicati delle triple presenti nel grafo, sono degli atom. Il percorso è una sequenza di proprietà alcune delle quali possono essere percorse in senso inverso, da queste premesse si può costruire una stringa che esprime un percorso qualsiasi all'interno del grafo.

La stringa sarà del tipo:

```
string path = "['seq' , 'prop1' , ['inv' , 'prop2'] ,['inv' , 'prop3'] , 'prop4']";
```

Dove prop1 e prop4 sono proprietà da percorrersi nel senso giusto, mentre prop2 e prop3 nel senso inverso. Per eseguire la query Wilbur si usano i seguenti metodi:

```
string responce = core.queryWQL_values(startNode, path);
ArrayList nodes = parser.getValuesQueryNodeList(responce);
foreach (string[] item in nodes){
Console.WriteLine("  Node   : (" + item[0] + " , " + item[1] + " )");
}
```

Il vantaggio principale delle query Wilbur è verificare, ad esempio, a che classe appartiene un'istanza o operazioni simili.
Il progetto Piglet è stato abbandonato e quindi anche le query WQL non vengono più supportate dalle nuove versioni di SIB. E' anche per questo motivo che si è deciso di introdurre all'interno della SIB il supporto a SPARQL.

**Subscribe RDF**
Come nelle query bisogna passare in ingresso i template delle triple alle quali ci si vuole sottoscrivere. Inoltre si dovrà definire una classe che implementi l'interfaccia "iKPIC_subscribeHandler" che offrirà in particolare il metodo "kpic_SIBEventHandler" che verrà richiamato nel caso in cui la SIB invii una notifica al KP sulla modifica di una o più triple.

```
KPIC_subscribeHandler handler = new KPIC_subscribeHandler();
```

Il metodo subscribe creerà un nuovo Thread che rimarrà in ascolto in attesa di eventi SIB, quindi al momento opportuno lancerà il metodo della classe appena creata in modo concorrente.

```
string responce = core.subscribeRDF ("any", "predicato", "oggetto", "oggType",
iKPIC_subscribeHandler handler);
string subscriptionID = parser.getSubscriptionID(responce);
```

**Query SPARQL**
Questa è la parte della libreria che ho implementato , Il metodo per le querySPARQL restituisce una stringa. Mentre il metodo per la GetSPARQLResults restituisce uno SPARQLResults ovvero una classe che è stata illustrata in seguito.

```
public class SPARQLResults {
        public List<string> variables;
        public List<SPARQLResult> results;
public class SPARQLResult { public List<SPARQLBinding> bindings; }
public class SPARQLBinding{
        public string name;
        public string value;
        public SPARQLValueType type;
}
public enum SPARQLValueType { URI, LITERAL,BNODE };
}
```

```
string responce = core.querySPARQL (stringQuery);
KPICore.SPARQLResults result = core.GetSPARQLResults(responce);
```

Se la query inoltrata contiene alcuni particolari caratteri chiamati XMLEntities questi vengono sostituiti opportunamente di modo che il documento XML risultante sia ben scritto e quindi non provochi errori nella fase di parsing da parte della SIB. La query SPARQL, all'interno di un messaggio SSAP, è infatti testo all'interno di un file XML e pertanto le seguenti sostituzioni sono necessarie.

```
private string preProcessSPARQLQuery(string query){
string t1 = query.Replace("&", "&amp;");
string t2 = t1.Replace("<", "&lt;");
string t3 = t2.Replace(">", "&gt;");
string t4 = t3.Replace("\"", "&quot;");
string t5 = t4.Replace("'", "&apos;");
return t5;
 }
```

La funzione querySPARQL invia la richiesta (parametro di ingresso) al parser della SIB sotto forma di messaggio SSAP, restituisce la risposta della interrogazione inoltrata ovvero un'informazione di tipo XML in una stringa.
Mentre la funzione GetSPRQLResults è un algoritmo che esamina il parametro di ingresso ossia una informazione XML che dopo aver effettuato i dovuti controlli, cattura solo le informazioni contenute nelle tag "head" e "results".
In particolare per quanto riguarda il tag head vengono controllati i tag figli "variable" prendendo il contenuto del parametro "name", e "link" prendendo il contenuto del parametro "href".
Questi rappresentano le variabili e vengono assegnati al campo variables della classe SPARQLResults.

```
KPICore.SPARQLResults results = new KPICore.SPARQLResults(); results.variables = new
List<string>();
XmlNodeList parameters = root.GetElementsByTagName("head");
foreach (XmlElement p in parameters){
        XmlNodeList var = p.GetElementsByTagName("variable");
        foreach (XmlElement v in var)
        results.variables.Add(v.GetAttribute("name"));
        XmlNodeList link = p.GetElementsByTagName("link");
        foreach (XmlElement l in link)
        results.variables.Add(l.GetAttribute("href"));
}
```

Invece, riguardo ai tag "result" che rappresentano i risultati della richiesta effettuata viene esaminato il tag figlio "binding" prendendo l'informazione del parametro "name" e il contenuto del tag di quest'ultimo che rappresenta il tipo della varabile ovvero uri, literal, bnode o unbound.

```
XmlNodeList res = root.GetElementsByTagName(“result”);
if (res == null) return results;
if (res.Count == 0) return results;
foreach (XmlElement rt in res)
XmlNodeList bind = rt.GetElementsByTagName(“binding”);
foreach (XmlElement b in bind)
string nome = b.GetAttribute(“name”);
string tipo = b.FirstChild.Name;
```

A questo punto viene confrontato il tipo della variabile e se è unbound viene ignorato, altrimenti ci sono i vari casi e per ogni caso viene assegnato il tipo al campo SPARQLValueType un enumeratore di 3 elementi ossia URI, LITERAL e BNODE della classe SPARQLBinding.

Quindi per ogni result viene creato un oggetto SPARQLBinding in cui vengono assegnati i valori:

- Name, il valore del attributo di binding name
- Value, il contenuto del tag figlio di binding
- Tipo, il name del tag figlio di binding

```
KPICore.SPARQLResults.SPARQLBinding binding = new
KPICore.SPARQLResults.SPARQLBinding();
if (tipo == “unbound”) continue;
binding.name = nome;
binding.value = b.GetElementsByTagName(tipo)[0].FirstChild.Value;
switch (tipo) {
                case “uri”:
                   binding.type = KPICore.SPARQLResults.SPARQLValueType.URI;
                   break;
               case “bnode”:
                   binding.type = KPICore.SPARQLResults.SPARQLValueType.BNODE;
                   break;
              case “literal”:
                   binding.type = KPICore.SPARQLResults.SPARQLValueType.LITERAL;
                   break;
}
```

### 5.2.1 Interfaccia

La Form principale permette all'utente di inserire i dati per accedere alla SIB, ovvero indirizzo IP, nome e porta che si possono inserire da tastiera o eventualmente utilizzare quelli memorizzati.
Nella parte di codice relativa, è stato pensato di implementare un controllo per l'inserimento dell'indirizzo IP, verificando che il valore immesso rispetti la formattazione ed i limiti imposti da un indirizzo IP. Il campo Port permette l'inserimento di soli valori numerici. Dato che questi valori potrebbero rimanere gli stessi, in quanto la SIB all'interno di un ambiente potrebbe avere parametri statici, sono stati salvati dei valori di riferimento sfruttando le proprietà della combobox, ovvero un particolare editor messo a disposizione all'interno della piattaforma Visual Studio per il salvataggio delle preferenze.
L'interfaccia offre la possibilità di cancellare il contenuto della SIB oppure quella della cache, caricare file owl sulla cache o caricare ontologie sulla SIB e viceversa. Infine viene data anche la possibilità di aggiornare i namespace sulla SIB. Con la checkbox "show triples", consente anche di mostrare tutte le triple presenti sulla cache caricate precedentemente, che una volta caricata vengono mostrate tutti i namespace presenti nell'antologia sulla datagridview "NAMESPACES" e viene data la possibilità di scegliere fra le proprietà, classi, individui o resources da visualizzare nella sua apposita datagridview.

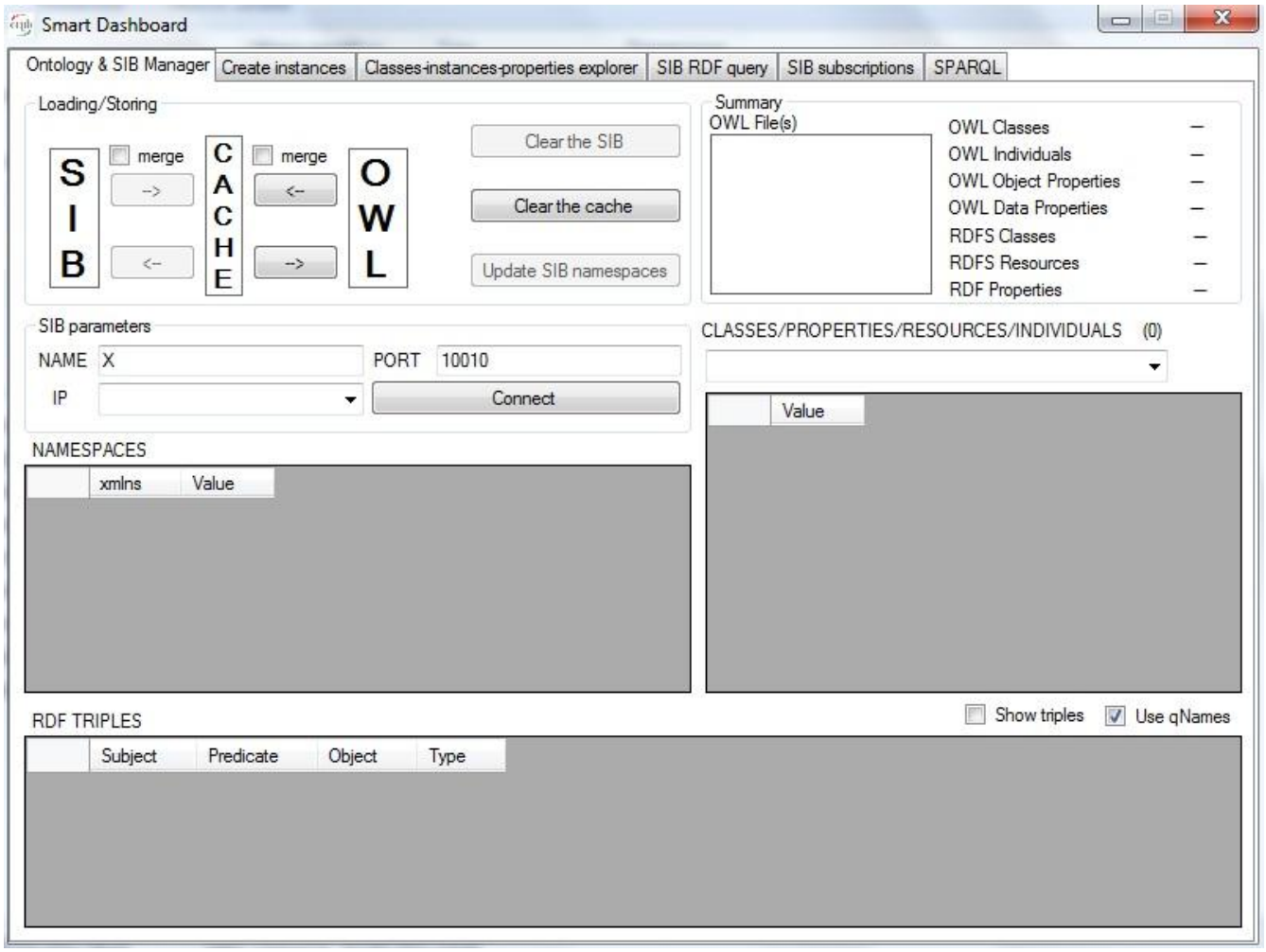



**Figura 5.1: Form Principale dell'applicazione**

L’interfaccia consente inoltre di svolgere ulteriori operazioni come la creazione di istanze, la visualizzazione tramite una struttura ad albero delle istanze presenti nella SIB, l’esecuzione di query RDF e la gestione delle sottoscrizioni.
Per la gestione delle query SPARQL è stata inserita una nuova schermata in cui sono presenti due campi di testo uno per scrivere la query SPARQL che si vuole eseguire, l’altro per mostrare la risposta in formato XML. Nel primo vengono visualizzati automaticamente anche i namespaces disponibili nell’ontologia presente sulla SIB per facilitare la creazione della query. Sempre allo scopo di avere una maggiore leggibilità, nel campo del risultato viene mostrata la risposta alla query ma filtrata sulle informazioni che interessano, ovvero dei tag “head” e “result”. Infine è stata predisposta una tabella (tramite una struttura dati di tipo datagridview) nella quale viene mostrato il risultato della query effettuata , resa in modo più leggibile e visionabile dall’interessato rispetto a un file xml.

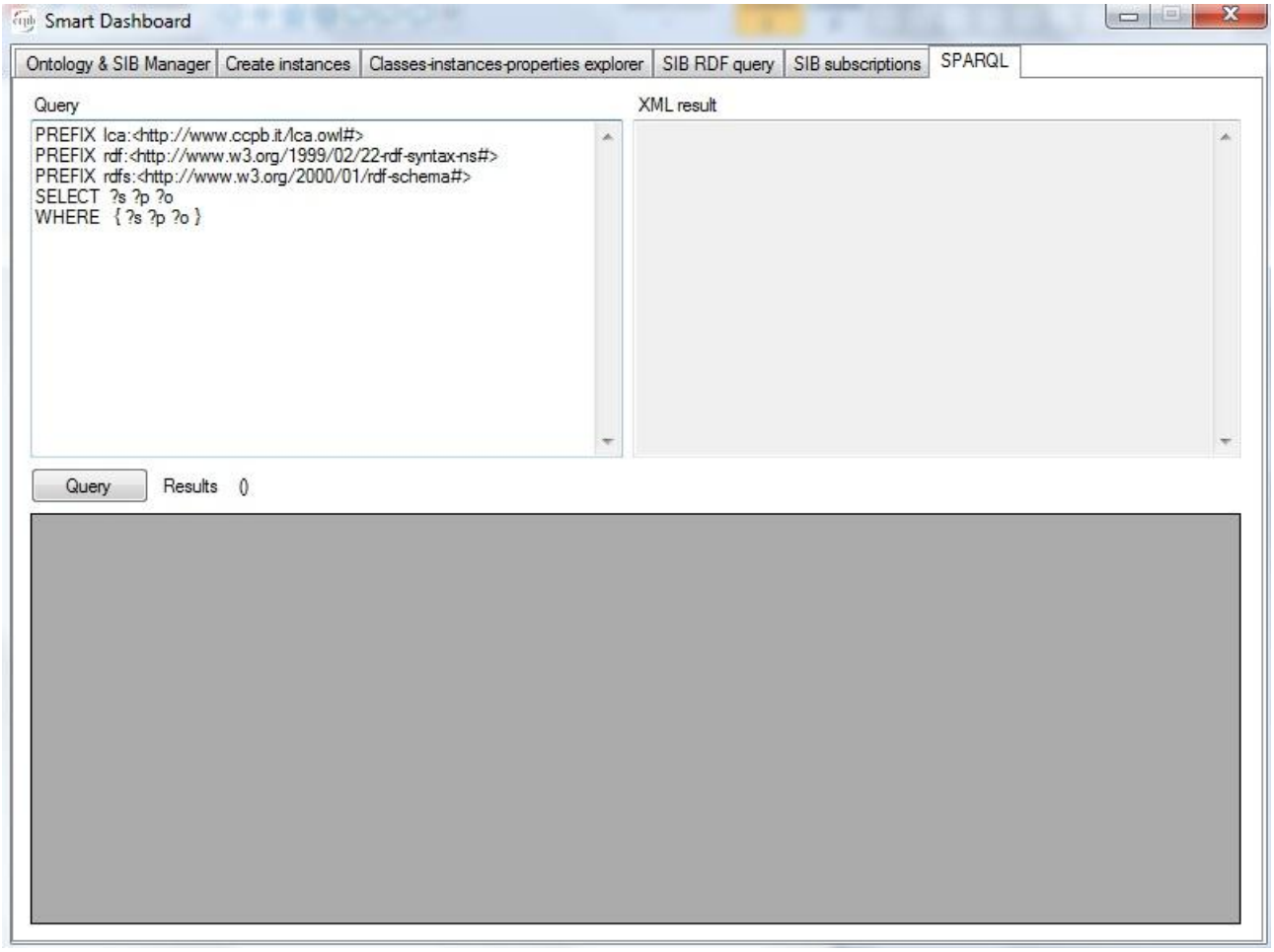



**Figura 5.2: Form SPARQL dell’applicazione**

## 5.3 Testing delle funzionalità

Per effettuare il testing ho inizialmente utilizzato un Server contenente la SIB, che è stata installata su una piattaforma virtuale. Successivamente è stata usata la SIB pubblica del gruppo Arces, raggiungibile all’indirizzo “mml.arces.unibo.it”. Per come è stata concepita la SIB deve essere messa in esecuzione su di una macchina con sistema operativo Linux.

Per effettuare i test di validazione del lavoro svolto, ho creato un'ontologia con Protègè [44] una piattaforma open-source che può esportare le ontologie in vari formati: RDF(S), XML Schema e OWL.
La piattaforma Protégé consente di utilizzare due modalità per creare le ontologie:

- Il **Protégé-Frames editor**, consente di costruire e popolare le ontologie che sono basate su "frame", secondo il protocollo OKBC (Open Knowledge Base Connectivity) [45];
- Il **Protégé-OWL editor**, consente di costruire ontologie per il Semantic Web, in particolare secondo il linguaggio OWL. Un'ontologia OWL può includere descrizioni di classi, di proprietà e le loro istanze.

E' stato creata una nuova ontologia OWL a cui è stato assegnato il nome di "human".

Un'ontologia è costituita da:

- Classi (concetti generali del dominio di interesse.)
- Relazioni tra queste classi
- Proprietà assegnate a ciascun concetto: descrivono vari tipi di attributi o proprietà
- Restrizioni sulle proprietà impongono il tipo di dato sul valore che la proprietà può assumere.
- Istanze: a partire dalle classi dell'ontologia è possibile definire delle istanze, che rappresentano specifici oggetti del mondo reale e ereditano attributi e relazioni dalle classi

L'ontologia sviluppata è rappresentata graficamente nelle figure 5.3, 5.4, 5.5 e 5.6.
Sono rappresentate la tassonomia delle Classi (Figura 5.3), le proprietà fra istanze di classi (ObjectProperty, Figure 5.4 – 5.5) e le proprietà fra istanze e tipi di dato (DataType Property, Figura 5.6).

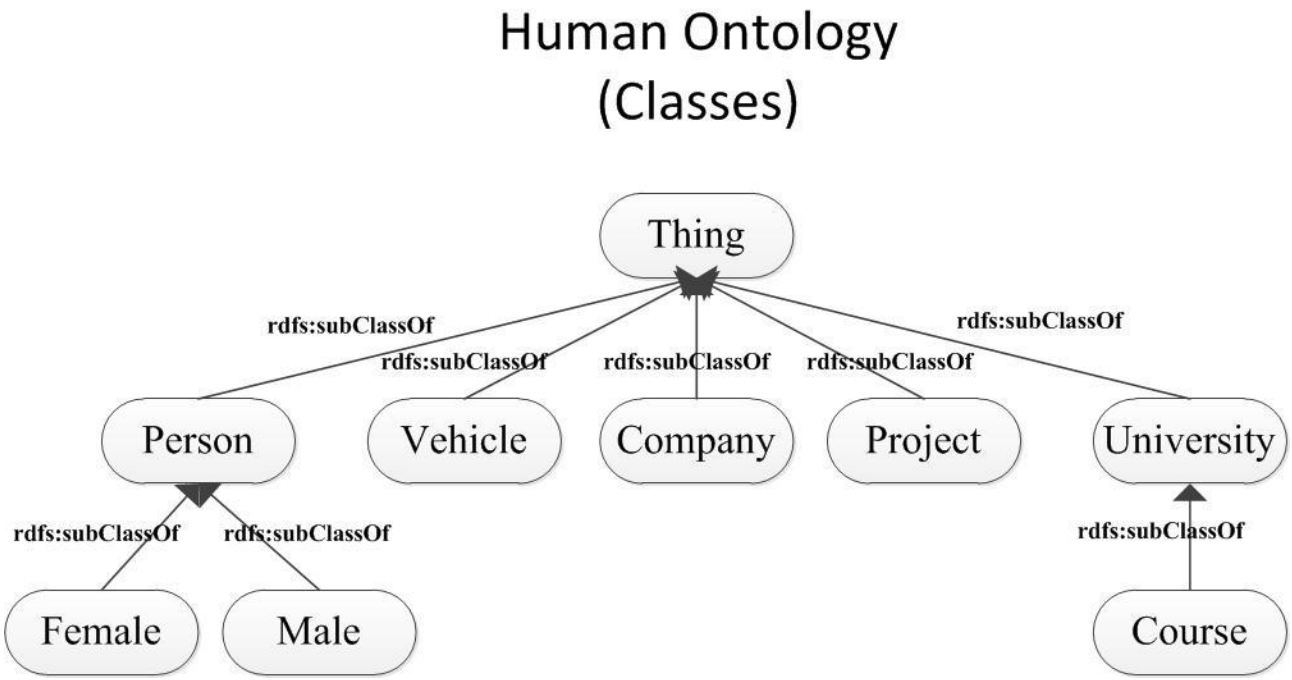



**Figura 5.3: Classi della Human Ontology**

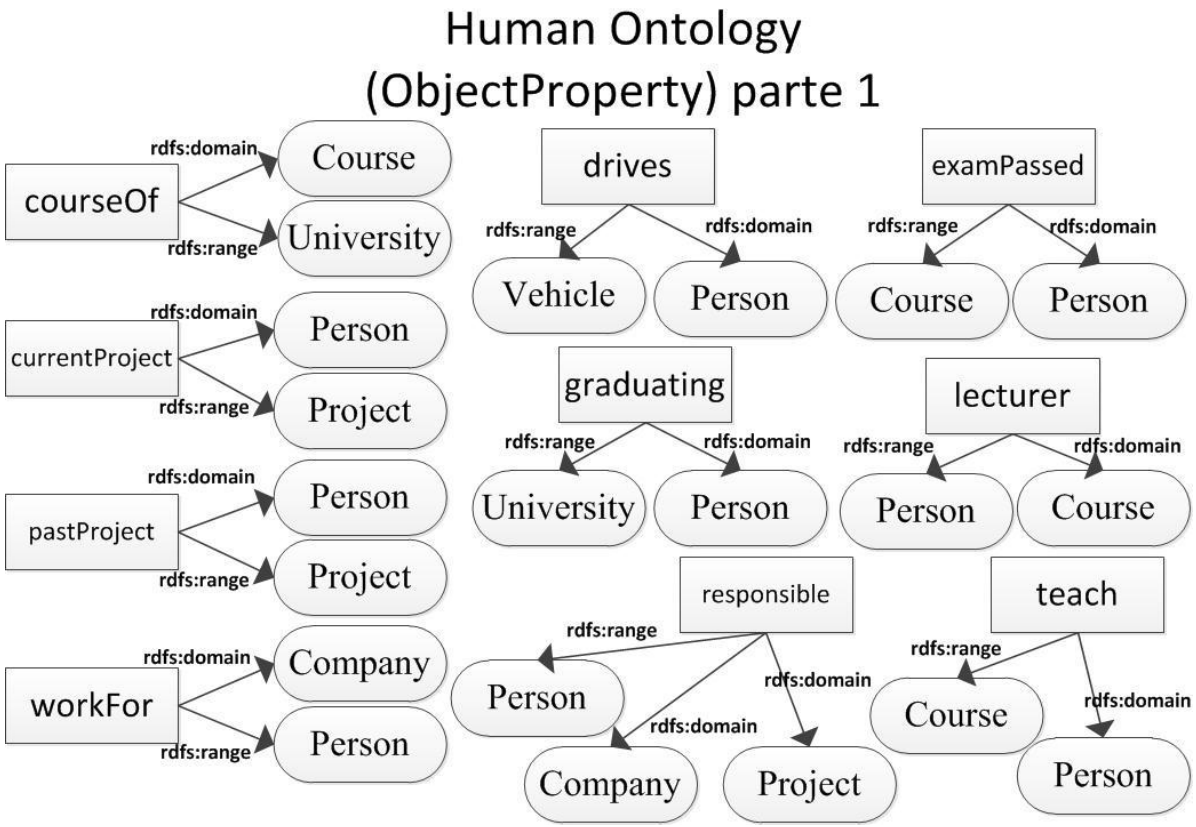



**Figura 5.4: ObjectProperty della Human Ontology (parte1)**

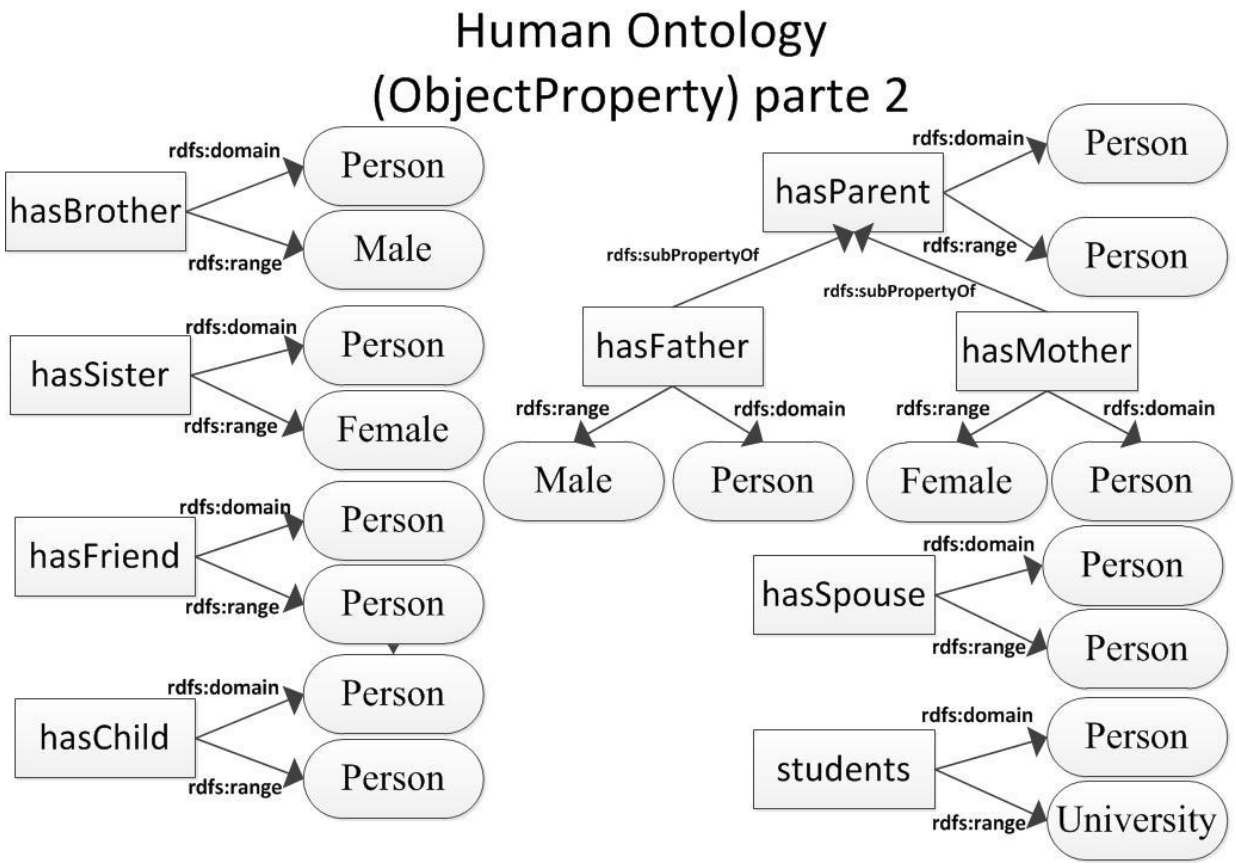


**Figura 5.5: ObjectProperty della Human Ontology (parte2)**

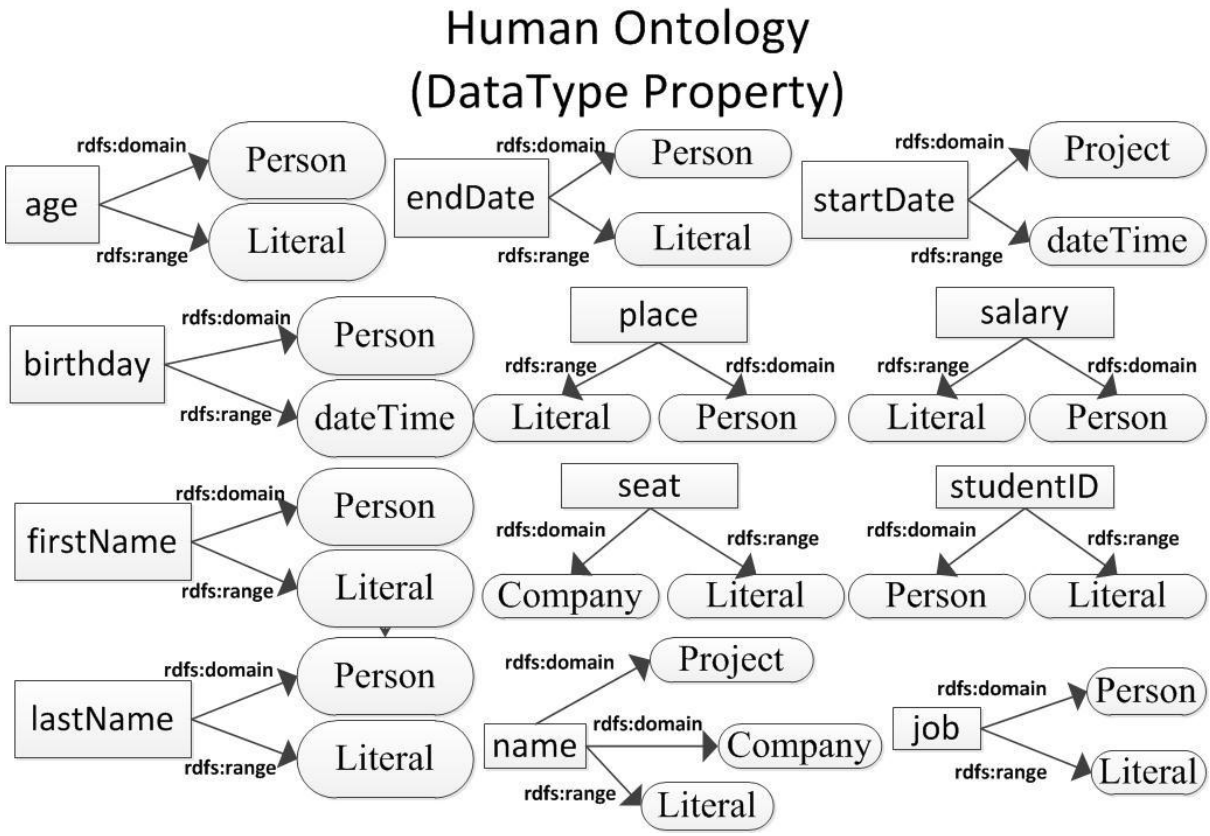



**Figura 5.6: DataType Property della Human Ontology**

I test effettuati su questa ontologia con l'applicazione creata sono illustrati in dettaglio in seguito:

**Quali sono gli URI di tutte le OWL Class?**

```
SELECT ?s WHERE{?s rdf:type owl:Class}
```

**Quali sono gli URI delle Object Property presenti?**

```
SELECT ?s WHERE{?s rdf:type owl:ObjectProperty}
```

**Quali sono gli URI delle Data Property presenti?**

```
SELECT ?s WHERE{?s rdf:type owl:DatatypeProperty}
```

**Quali sono gli URI di tutti gli individui dell'ontologia?**

```
SELECT ?s WHERE{?s rdf:type owl:NamedIndividual}
```

**Quali sono gli URI delle persone sposate e l'URI del relativo coniuge?**

```
SELECT ?s ?o
WHERE{?s :hasSpouse ?o}
```

**Quali sono gli URI delle sottoclassi presenti nell'antologia e l'URI della relativa super-classe?**

```
SELECT ?s ?o
WHERE{?s rdfs:subClassOf ?o}
```

**Quali sono gli URI degli uomini che sono sposati, insieme agli URI della moglie?**

```
SELECT ?s ?o
WHERE{:Male rdfs:subClassOf contact:Person.
?s :hasSpouse ?o
}
```

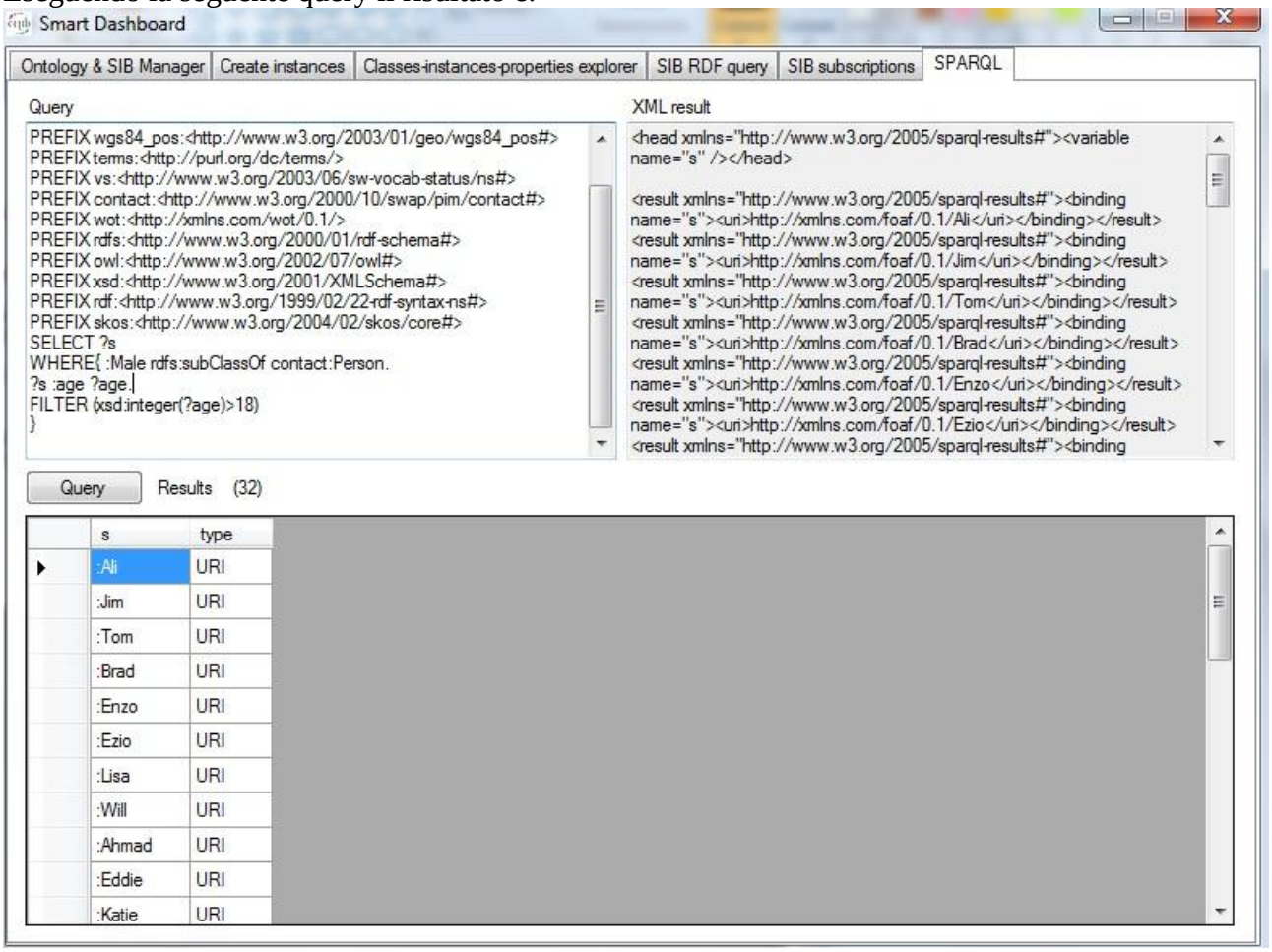
**Quali sono gli URI delle persone che hanno almeno un parente?**

```
SELECT  DISTINCT ?o
WHERE {{?o :hasChild ?s} UNION {?o :hasParent ?s}}
```

**Quali sono gli URI degli uomini maggiorenni?**

```
SELECT ?s
WHERE{ :Male rdfs:subClassOf contact:Person.
?s :age ?age.
FILTER (xsd:integer(?age)>18)
}
```

Eseguendo la seguente query il risultato è:



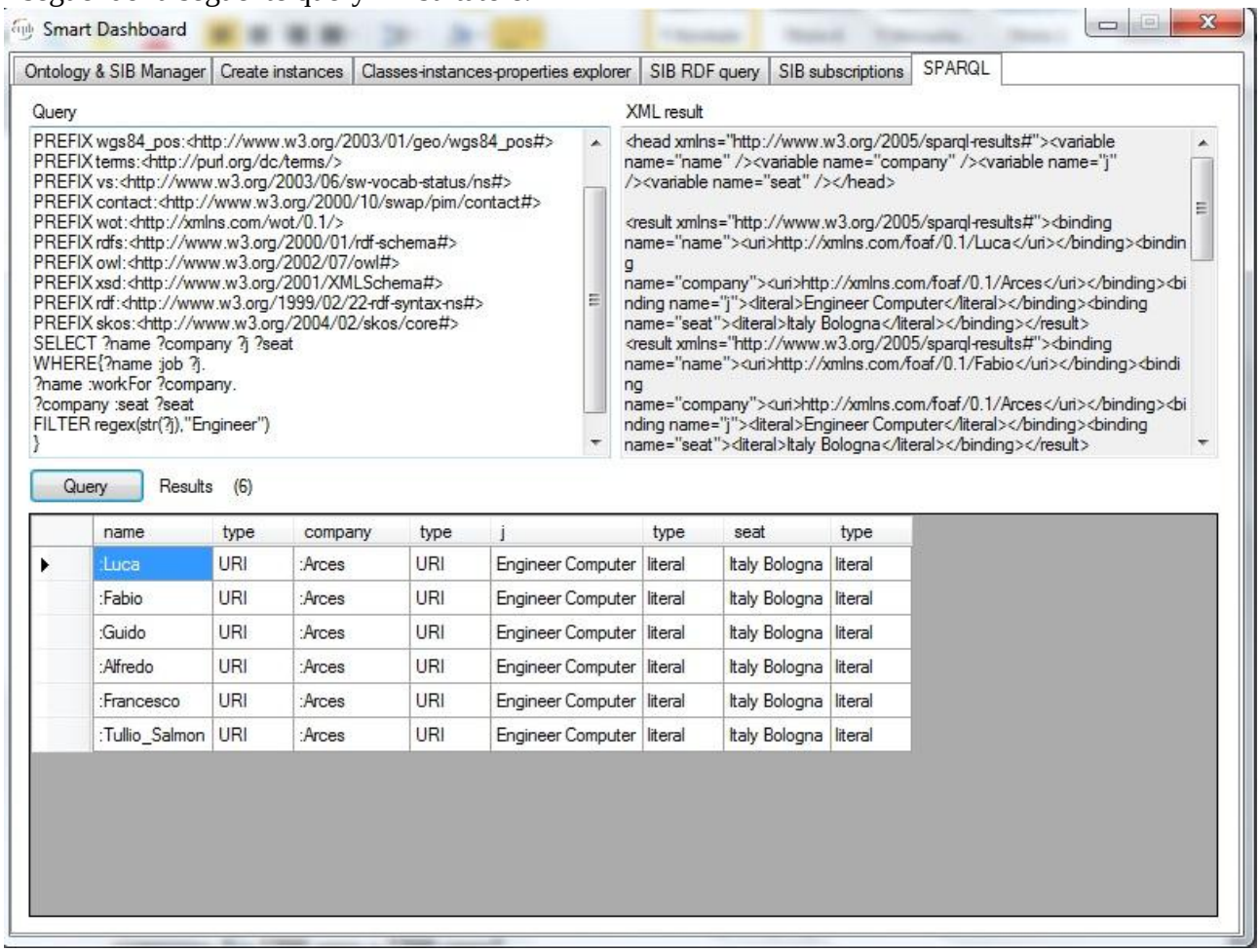
**Il cognome, età, giorno del compleanno e l'URI del sesso di tutte le persone, dove età e giorno del compleanno sono opzionali**

```
SELECT ?n ?age ?b ?sex
WHERE{?sex rdfs:subClassOf contact:Person.
?s :lastName ?n
OPTIONAL {?s :age ?age}
OPTIONAL {?s :birthday ?b}
}
```

**Quali sono gli URI dei Nomi degli ingegneri, insieme all'URI dell'azienda per cui lavorano e la loro sede?**

```
SELECT ?name ?company ?j ?seat
WHERE{?name :job ?j.
?name :workFor ?company.
?company :seat ?seat
FILTER regex(str(?j),"Engineer")
}
```

Eseguendo la seguente query il risultato è:



**Quali sono gli URI degli amici di "Randy" che hanno 25 anni e la loro età?**

```
SELECT ?o ?age
WHERE{:Randy :hasFriend ?o.
?o :age ?age
FILTER (xsd:integer(?age) = 25)
}
```

**Quali sono gli esami passati da "Lorenzo"?**

```
SELECT ?s ?o
WHERE{?s :examPassed ?o
FILTER regex(str(?o), "Lorenzo")
}
```

**Quali sono gli URI dei progetti terminati dopo il 01-01-2000, insieme all'URI del responsabile e la data di fine progetto?**

```
SELECT ?s ?r ?o
WHERE{?s :endDate ?o.
?s :responsible ?r
FILTER (?o > "2000-01-01T00:00:00Z^^xsd:dateTime")
}
```

**Quali sono gli URI dell'Aziende, sede insieme all'URI degli impiegati e loro stipendio compreso fra 1300 euro e 2300 euro?**

```
SELECT ?nome ?s ?p ?sede
WHERE{?nome :salary ?s.
?nome :workFor ?p.
?p :seat ?sede
FILTER (xsd:integer(?s) > 1300 && xsd:integer(?s) < 2300 )
}
```

**Quali sono gli URI dei Nomi dei lavoratori , località di residenza e salario, insieme all'URI dell'azienda, sede e il progetto su cui lavorano?**

```
SELECT ?nome ?place ?s ?work ?seat ?project
WHERE{?nome :salary ?s.
?nome :place ?place.
?nome :workFor ?work.
?work :seat ?seat.
?nome :currentProject ?project
}
```

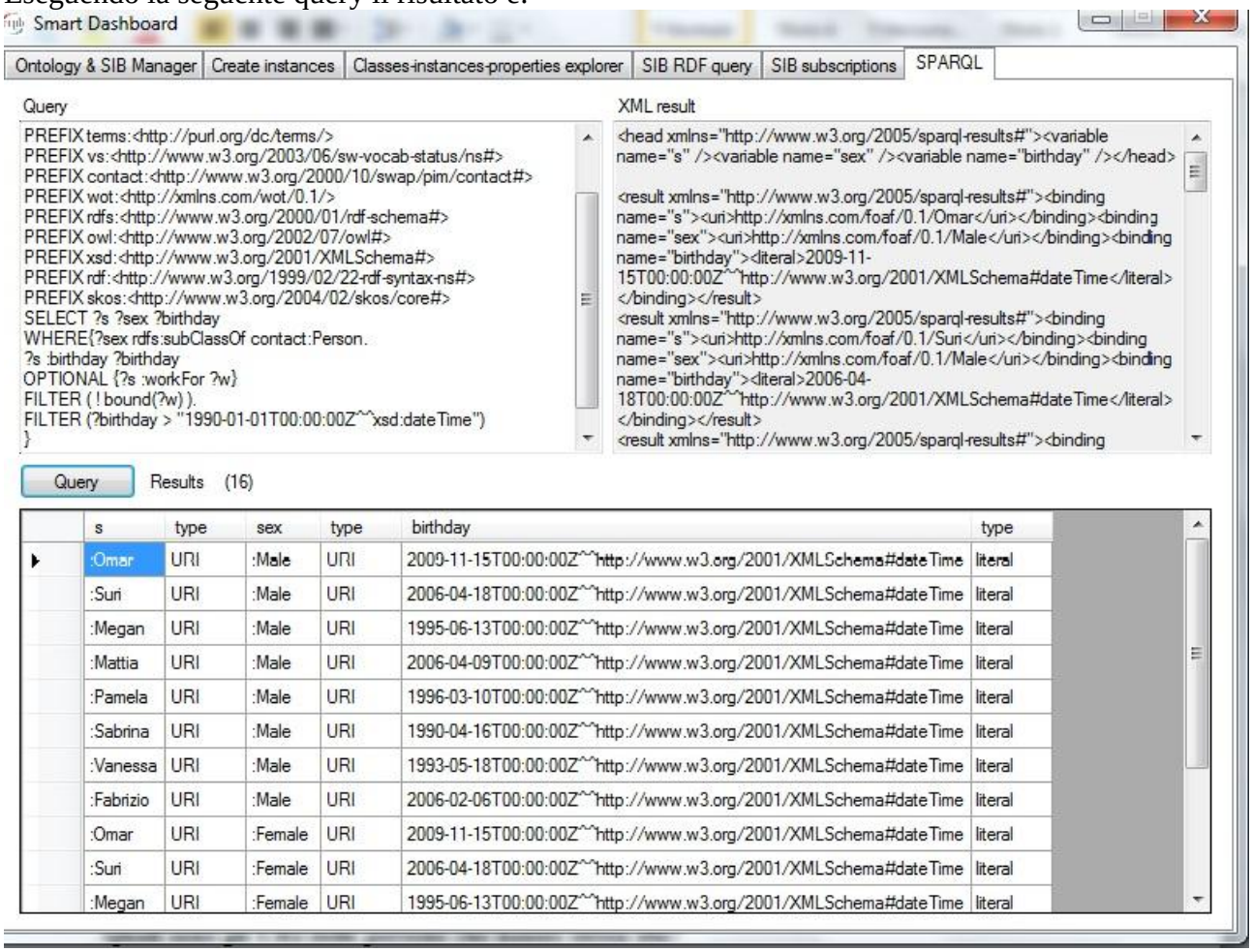
**Quali sono gli URI dell'azienda e sede, insieme all'URI del responsabile, località di residenza, salario e l'URI dei mezzi che è in grado di guidare?**

```
SELECT ?company ?seat ?responsible ?place ?salary ?drives
WHERE{?company rdf:type :Company.
?company :responsible ?responsible.
?company :seat ?seat.
?responsible :place ?place.
?responsible :salary ?salary.
?responsible :drives ?drives
}
```

**Quali sono gli URI delle persone che non lavorano, l'URI del loro sesso e il giorno del compleanno per cui sono nati dopo il 1990-01-01?**

```
SELECT ?s ?sex ?birthday
WHERE{?sex rdfs:subClassOf contact:Person.
?s :birthday ?birthday
OPTIONAL {?s :workFor ?w}
FILTER ( ! bound(?w) ).
FILTER (?birthday > "1990-01-01T00:00:00Z^^xsd:dateTime")
}
```

Eseguendo la seguente query il risultato è:




| s | type | sex | type | birthday | type |
|---|---|---|---|---|---|
| :Omar | URI | :Male | URI | 2009-11-15T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Suri | URI | :Male | URI | 2006-04-18T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Megan | URI | :Male | URI | 1995-06-13T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Mattia | URI | :Male | URI | 2006-04-09T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Pamela | URI | :Male | URI | 1996-03-10T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Sabrina | URI | :Male | URI | 1990-04-16T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Vanessa | URI | :Male | URI | 1993-05-18T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Fabrizio | URI | :Male | URI | 2006-02-06T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Omar | URI | :Female | URI | 2009-11-15T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Suri | URI | :Female | URI | 2006-04-18T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |
| :Megan | URI | :Female | URI | 1995-06-13T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime | literal |

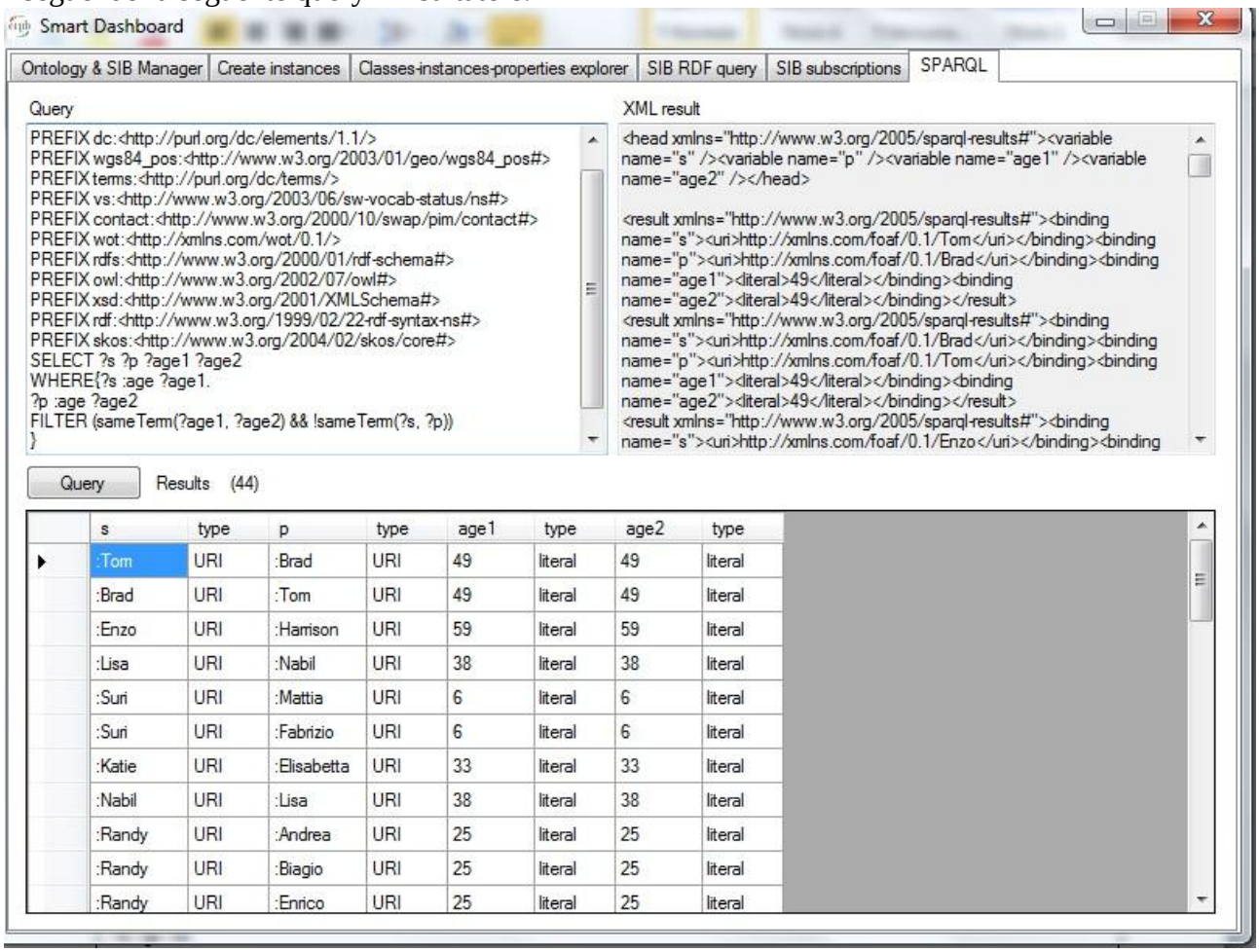
**Quali sono gli URI delle persone che parlano la lingua inglese?**

```
SELECT ?s ?o
WHERE{?s :languages ?o
FILTER regex(str(?o), "@en")
}
oppure
SELECT ?s ?o
WHERE{?s :languages ?o
FILTER (lang(?o) = "en")
}
```

**Quali sono gli URI delle persone che hanno stessa età?**

```
SELECT ?s ?p ?age1 ?age2
WHERE{?s :age ?age1.
?p :age ?age2
FILTER (sameTerm(?age1, ?age2) && !sameTerm(?s, ?p))
}
```

Eseguendo la seguente query il risultato è:




| s | type | p | type | age1 | type | age2 | type |
|---|---|---|---|---|---|---|---|
| :Tom | URI | :Brad | URI | 49 | literal | 49 | literal |
| :Brad | URI | :Tom | URI | 49 | literal | 49 | literal |
| :Enzo | URI | :Harrison | URI | 59 | literal | 59 | literal |
| :Lisa | URI | :Nabil | URI | 38 | literal | 38 | literal |
| :Suri | URI | :Mattia | URI | 6 | literal | 6 | literal |
| :Suri | URI | :Fabrizio | URI | 6 | literal | 6 | literal |
| :Katie | URI | :Elisabetta | URI | 33 | literal | 33 | literal |
| :Nabil | URI | :Lisa | URI | 38 | literal | 38 | literal |
| :Randy | URI | :Andrea | URI | 25 | literal | 25 | literal |
| :Randy | URI | :Biagio | URI | 25 | literal | 25 | literal |
| :Randy | URI | :Enrico | URI | 25 | literal | 25 | literal |

**Quali sono gli URI delle persone che hanno un IRI nell'oggetto?**

```
SELECT ?s
WHERE {?s rdfs:subClassOf contact:Person.
?s ?p ?o
FILTER isIRI(?o)
}
```

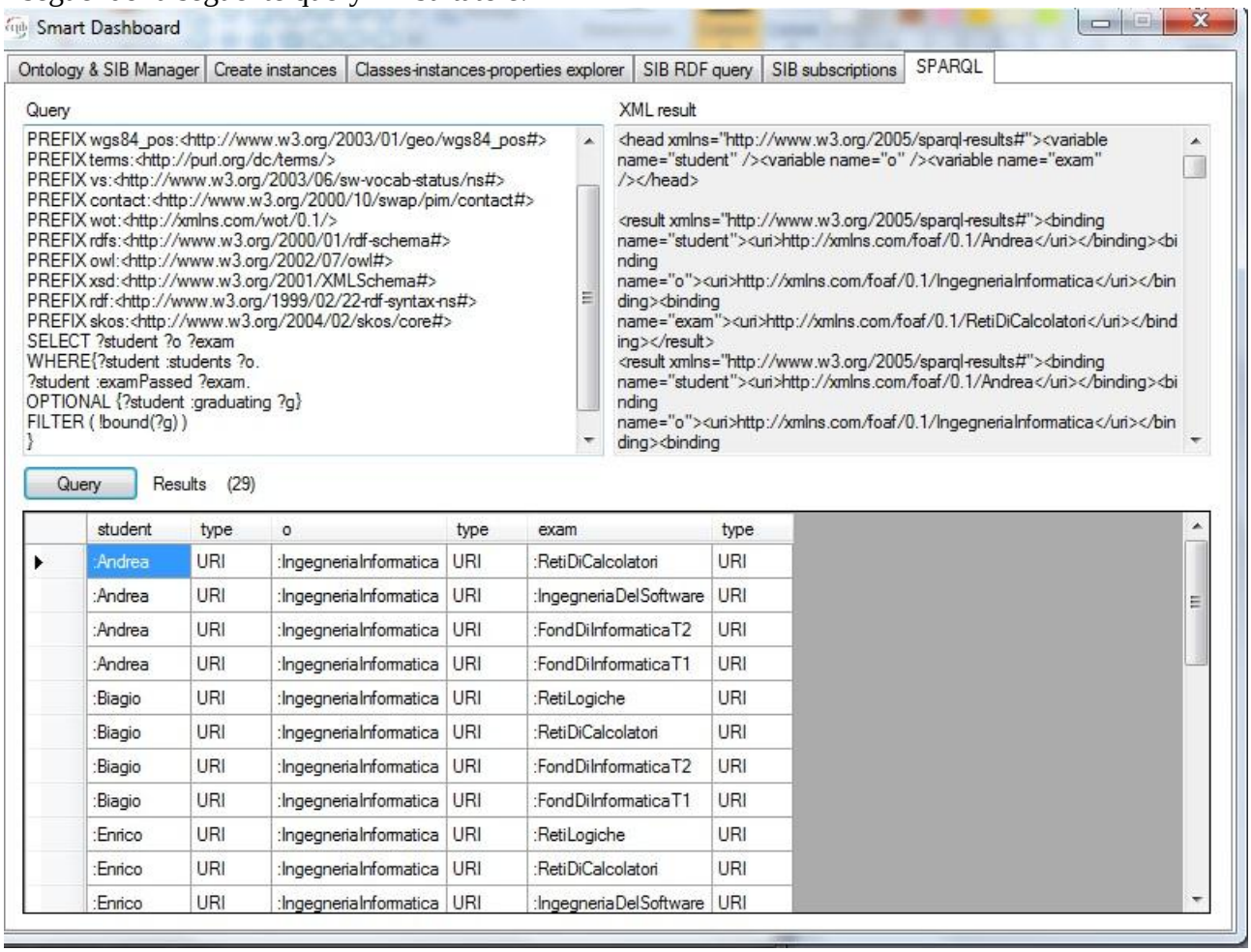
**Quali sono le triple che hanno un literale nell'oggetto?**

```
SELECT ?s ?p ?o
WHERE {
?s ?p ?o
FILTER isLiteral(?o)
}
```

**Quali sono gli URI degli studenti, l'URI della facoltà e gli URI degli esami passati per cui non hanno ancora finito la carriera universitaria?**

```
SELECT ?student ?o ?exam
WHERE{?student :students ?o.
?student :examPassed ?exam.
OPTIONAL {?student :graduating ?g}
FILTER ( !bound(?g) )
}
```

Eseguendo la seguente query il risultato è:




| student | type | o | type | exam | type |
|---|---|---|---|---|---|
| :Andrea | URI | :IngegneriaInformatica | URI | :RetiDiCalcolatori | URI |
| :Andrea | URI | :IngegneriaInformatica | URI | :IngegneriaDelSoftware | URI |
| :Andrea | URI | :IngegneriaInformatica | URI | :FondDiInformaticaT2 | URI |
| :Andrea | URI | :IngegneriaInformatica | URI | :FondDiInformaticaT1 | URI |
| :Biagio | URI | :IngegneriaInformatica | URI | :RetiLogiche | URI |
| :Biagio | URI | :IngegneriaInformatica | URI | :RetiDiCalcolatori | URI |
| :Biagio | URI | :IngegneriaInformatica | URI | :FondDiInformaticaT2 | URI |
| :Biagio | URI | :IngegneriaInformatica | URI | :FondDiInformaticaT1 | URI |
| :Enrico | URI | :IngegneriaInformatica | URI | :RetiLogiche | URI |
| :Enrico | URI | :IngegneriaInformatica | URI | :RetiDiCalcolatori | URI |
| :Enrico | URI | :IngegneriaInformatica | URI | :IngegneriaDelSoftware | URI |

**Quali sono gli URI dei maschi e l'URI delle femmine che sono amici?**

```
SELECT ?s ?friend
WHERE{:Male rdfs:subClassOf contact:Person.
?s :hasFriend ?friend.
?friend rdf:type :Female
}
```

**Costruisci una tripla con Datatype Property pari a name per cui il Datatype Property era pari a firstName**

```
CONSTRUCT { ?x :name ?name }
WHERE  { ?x :firstName ?name }
```

**E' presente una tripla in cui il firstName è Randy?**

```
ASK {?s :firstName "Randy"}
```

L'insieme dei test sopra riportati ha consentito di validare l'implementazione effettuata e verificare il corretto supporto a SPARQL da parte della SIB.

# Capitolo 6

# Conclusioni

La mia tesi, in conclusione, ha portato a comprendere ed estendere le potenzialità di Smart-M3 e del Web Semantico.
Dopo lo studio dei formalismi semantici, del linguaggio di interrogazione SPARQL e del protocollo SSAP , ho implementato in linguaggio C# la funzionalità di effettuare delle query SPARQL ed ho progettato una struttura dati atta a gestirne i risultati. Le nuove funzionalità potranno essere utilizzate per poter navigare il grafo RDF contenuto nella SIB di uno Smart Environment con tutte le funzionalità fornite da SPARQL 1.0 .Queste funzionalità sono state testate e sono già disponibili pubblicamente sul sito di sourceforge.
Il progetto di questa tesi è stato molto interessante perché mi ha portato alla conoscenza di un argomento importante ed affascinante quale è il Semantic Web, e le sue applicazioni relativamente agli ambienti intelligenti. Inoltre le competenze acquisite in questo periodo sono state tante grazie alla supervisione ed alla guida del relatore, il Prof. Luca Roffia, ai suggerimenti operativi del gruppo di ricerca ARCES dell'Università di Bologna, ed in particolare l'Ing. Francesco Morandi e l'Ing. Alfredo D'Elia, oltre che grazie ai diversi approfondimenti personali attraverso approfondite letture sulla sintassi e sulla filosofia delle logiche della semantica che mi hanno portato ad una maggiore comprensione.
Il lavoro proposto potrebbe essere esteso considerando la versione 1.1 del linguaggio SPARQL quindi aggiungendo nuove potenzialità.
Le novità introdotte dall'ultima estensione sono:

- le funzioni aggregate: COUNT, MAX, MIN, SUM, AVG. Si applicano a un gruppo di soluzioni che devono essere specificati attraverso la sintassi della clausola GROUP BY;
- i pattern EXIST e NOT EXIST per testare se all'interno di un graph pattern e in espressioni FILTER trova il match dei dati o meno;
- le sottoquery permettono di innestare delle query all'interno della clausola WHERE rendendo più potente l'espressività delle interrogazioni;
- la parola chiave AS che non deve essere già stata dichiarata precedentemente,
  permette di scegliere le variabili dei pattern di confronto da includere nei risultati e introduce nuove variabili ed espressioni che forniscono il valore da attribuire alle variabili. Le espressioni permettono di combinare variabili per generare un nuovo valore.

# Bibliografia

**[1]** Smart-M3, http://en.wikipedia.org/wiki/Smart-M3
**[2]** Publish-Subscribe, http://it.wikipedia.org/wiki/Publish/subscribe
**[3]** RDF, http://www.w3.org/TR/rdf-primer/
**[4]** Spazio di tuple , http://en.wikipedia.org/wiki/Tuple_space
**[5]** Walter Cazzola, Linda
**[6]** Andrea Vallorani Oliviero Riganelli, Luce+TuCSon
**[7]** L.Roffia, A.D'Elia, F.Vergari, D.Manaroli, S.Bartolini, G.Zamagni, T.Salmon, Cinotti J.Honkola, A Smart-M3 lab course: approach and design style to support student projects
**[8]** J. Soininen, P. Liuha, A. Lappetelainen, J. Honkola, K. Framling, R.Raisamo, Device Interoperability: Emergence of the smart environment ecosystems 2010
**[9]** Progetto-Smart-M3, http://sourceforge.net/projects/smart-m3/
**[10]** The Semantic Web-Tim Berners-Lee, http://www.scientificamerican.com/article.cfm?id=the-semantic-web
**[11]** Linking Open Data Community Project, http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
**[12]** Richard Cyganiak, Anja Jentzsch Cloud Diagram, http://richard.cyganiak.de/2007/10/lod/
**[13]** Semantic Web Road map Tim Berners-Lee, http://www.w3.org/DesignIssues/Semantic.html
**[14]** Blank Node, http://en.wikipedia.org/wiki/Blank_node
**[15]** N-Triples, http://www.w3.org/2001/sw/RDFCore/ntriples/
**[16]** Notation 3, http://www.w3.org/DesignIssues/Notation3
**[17]** Turtle, http://www.w3.org/TeamSubmission/turtle/
**[18]** RDF Schema, http://www.w3.org/TR/rdf-schema/
**[19]** OWL, http://www.w3.org/2004/OWL/
**[20]** OWL Style Syntax, http://www.w3.org/TR/owl2-syntax/
**[21]** OWL Mapping to RDF, http://www.w3.org/TR/owl2-mapping-to-rdf/
**[22]** OWL Direct Semantics, http://www.w3.org/TR/owl2-semantics/
**[23]** OWL RDF-Based Semantics, http://www.w3.org/TR/owl2-rdf-based-semantics/
**[24]** OWL Conformance, http://www.w3.org/TR/owl2-test/
**[25]** OWL Profiles, http://www.w3.org/TR/owl2-profiles/
**[26]** Liyang Yu, A Developer's Guide to the Semantic Web 2011
**[27]** SPARQL Query Language, http://www.w3.org/TR/rdf-sparql-query/
**[28]** SPARQL Query Results Format, http://www.w3.org/TR/rdf-sparql-XMLres/
**[29]** SPARQL Protocol for RDF, http://www.w3.org/TR/rdf-sparql-protocol/
**[30]** WSDL 2.0, http://www.w3.org/TR/wsdl20/
**[31]** Graph Matching, http://www.w3.org/TR/rdf-sparql-query/#sparqlBGPExtend
**[32]** Redland, http://librdf.org
**[33]** ARC, http://arc.semsol.org
**[34]** 4store, http://www.4store.org/
**[35]** Virtuoso, http://virtuoso.openlinksw.com
**[36]** JENA, http://jena.apache.org/
**[37]** Joseki, http://www.joseki.org
**[38]** Sesame, http://www.openrdf.org
**[39]** XSLT, http://www.w3.org/TR/xslt
**[40]** Triple pattern, http://www.w3.org/TR/rdf-sparql-query/#sparqlTriplePatterns
**[41]** Regex, http://www.w3.org/TR/rdf-sparql-query/#funcex-regex
**[42]** Template, http://www.w3.org/TR/rdf-sparql-query/#construct
**[43]** O.Lassila, Semantic Web Programming using Piglet 2008
**[44]** Protègè, http://protege.stanford.edu/

**[45]** OKBC, http://www.ai.sri.com/~okbc/