Relazione Assignments Web Semantico

Studio Libreria RDFLib e Definizione di una Ontologia RDF/OWL per Wireless Sensor Network e Multi-Agent System

A.A. 2020/2021

Filippo Paganelli

0000926989

filippo.paganelli3@studio.unibo.it

Indice

1 Introduzione **3**

2 Features RDFLib **6**
2.1 Tecnologie Principali 7
2.1.1 RDF/XML 7
2.1.2 N3 - Turtle 8
2.1.3 JSON-LD 8
2.1.4 SPARQL 8
2.1.5 Knowledge Graph 9

3 Struttura Libreria **10**
3.1 RDF Terms 10
3.2 RDF Nodes 10
3.3 RDF Graph 10
3.4 RDF Namespaces 11
3.5 RDF Store 12
3.5.1 IOMemory 12

4 Analisi Dominio WSN-MAS **13**
4.1 Analisi Ontologie Riusabili 13
4.2 Binding Namespace RDFLib 14
4.3 Enumerazione Termini 15
4.3.1 Multi-Agent System 15
4.3.2 Wireless Sensor Network 15
4.4 Definizione Tassonomia e Proprietà 16

5 Definizione RDFS **18**

6 Definizione RDF **21**

7 Definizione OWL **24**
7.1 Definizione Istanze 27

8 Serializzazione e Validazione **29**

9 Criticità RDFLib **32**
9.1 Github Issues/PR 32

Riferimenti bibliografici **34**

1 Introduzione

Il primo assignment consiste nello studio di una tecnologia riguardante il Web Semantico e nella descrizione delle sue features, tecnologie utilizzate e criticità. Come tecnologia di studio è stata scelta la libreria RFDLib, una libreria open-source sviluppata in Python per la manipolazione di RDF, mantenuta su Github [1] e disponibile tramite Python Package Index (PyPI) [2]. La comunity che mantiene la libreria RDFLib è attiva anche su altri progetti sempre inerenti al Web Semantico, ad esempio:

sparqlwrapper - Wrapper SPARQL in Python per eseguire query remote.

pyLODE - Tool Python per lo sviluppo di ontologie OWL basato su LODE.

rdflib-jsonld - Plugin per RDFLib, implementazione di JSON-LD.

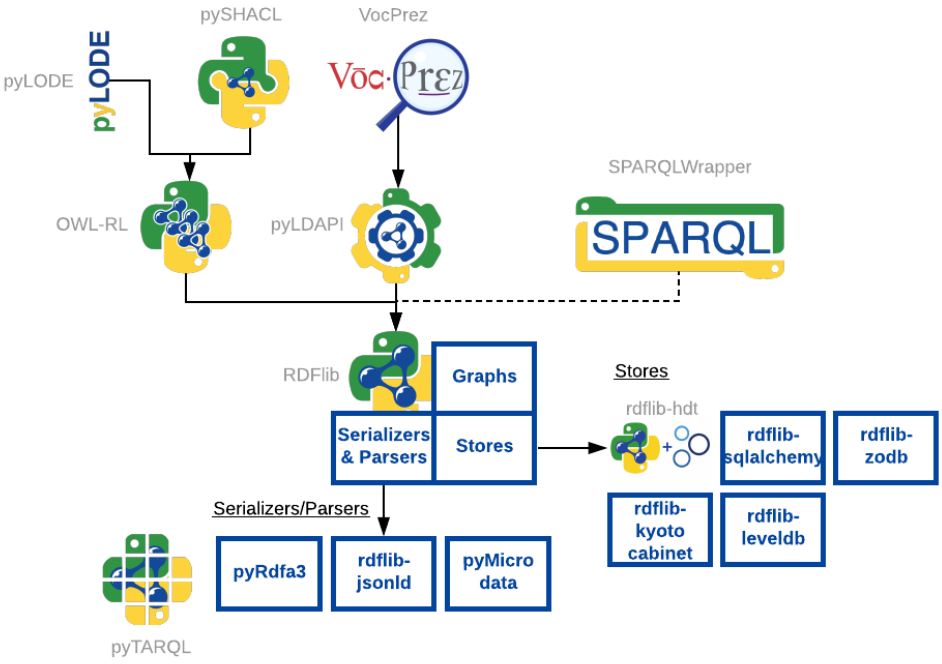


Figura 1: Progetti open-source manenuti dalla comunity RDFLib

[1] https://github.com/RDFLib/rdflib
[2] https://pypi.org/project/rdflib/

Il secondo e terzo assignment consistono rispettivamente nel modellare la conoscenza di un qualche specifico dominio a scelta (o sottoparte di esso) utilizzando RDF e nell'estensione di quest'ultimo utilizzando OWL, con l'obiettivo di collegare quanto più possibile i dati rappresentati a vocabolari esistenti e condivisi.
Il dominio scelto per la modellazione della conoscenza è quello rappresentante i progetti già realizzati per i corsi *Sistemi Distribuiti* e *Smart City e Tecnologie Mobile*: *Forest Fire Detection using Wireless Sensor Network and Multi-Agent System*. Il progetto è stato realizzato con un approccio top-down, cioè modellando prima il sistema multi-agente e successivamente la rete fisica di sensori/attuatori in modo da permettere il deploy effettivo dell'intero sistema.

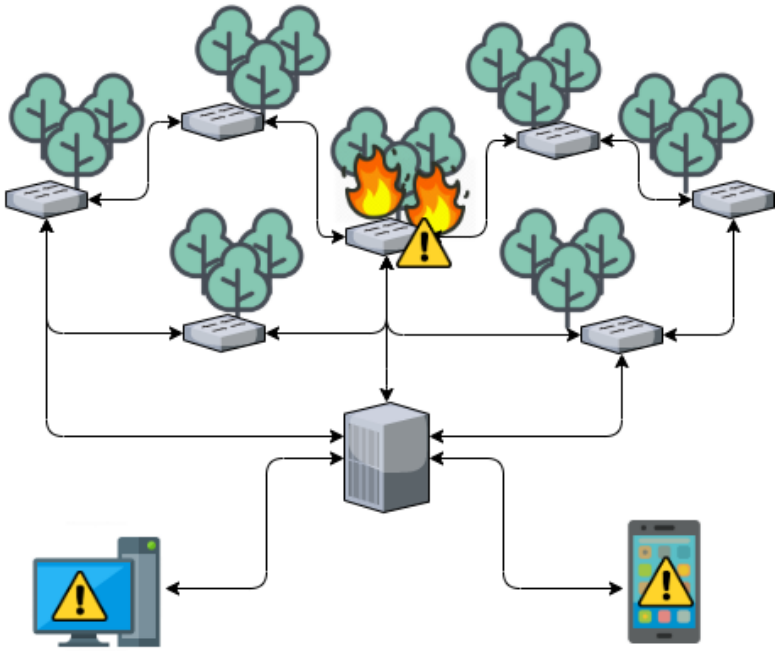


Figura 2: Esempio di scenario applicativo comune di una WSN per il rilevamento di incendi nelle foreste.

Si descrive nei seguenti capitoli il processo di sviluppo adottato per la modellazione della conoscenza del dominio, seguendo a grandi linee quello indicato da Noy e McGuinness in "*Ontology Development 101: A Guide to Create Your First Ontology*"[2]:

- Determine scope - Analisi Dominio

- Consider Reuse - Analisi Ontologie Riusabili
- Enumerate Terms - Enumerazione Termini
- Define Taxonomy - Definizione Tassonomia e Proprietà
- Define Properties - Definizione Tassonomia e Proprietà
- Define Facets - Definizione Tassonomia e Proprietà
- Define Instances - Definizione Schema RDFS, Definizione RDF, Definizione OWL
- Check for Anomalies - Serializzazione e Validazione

Come gran parte dei moderni processi di sviluppo si segue un modello iterativo e non a cascata: ad esempio la serializzazione e la validazione degli elaborati realizzati avviene durante tutte le fasi di sviluppo RDFS, RDF e OWL tramite appositi tool forniti dal W3C [3] presentati durante il corso.

[3] `https://www.w3.org/RDF/Validator/`

2 Features RDFLib

Ad oggi la release corrente della libreria RDFLib è la *5.0.0* [4] ed è classificata dal W3C nelle seguenti due categorie [5] di software inerente il Web Semantico:

Triple Store - Tools che possono essere usati come RDF Database. Altri esempi di tools appartenenti a questa categoria sono RDFox, Oracle Spatial, Apache Jena e GraphDB.

Programming Environment - Tools che foniscono un ambiente funzionante per lo sviluppo di applicazioni riguardanti il Web Semantico nella forma di librerie o moduli. Alcuni di questi, come la libreria RDFLib in Python, può essere usata direttamente in uno specifico linguaggio di programmazione. Altri esempi di tools appartenenti a questa categoria sono Apache Jena, Mobi, Sesame e Redland RDF.

Le principali funzionalità fornite dalla libreria RDFLib sono:

- parsers e serializers per RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, Trig and JSON-LD
- una interfaccia Graph che supporta diverse implementazioni di Store (single graph, conjunctive graph o dataset graphs)
- implementazioni store per memorizzazioni persistenti o in-memory basate su BerkeleyDB
- supporto a query e update SPARQL 1.1

Tra i tool inerenti il Web Semantico che sono stati sviluppati utilizzando la libreria RDFLib è possibile trovare Sparta, RDFAlchemy, SuRF, ORDF, e Fuxi. Per analizzare la libreria e comprendere il suo funzionamento sono stati sviluppati alcuni esempi di utilizzo base della libreria RDFLib per la manipolazione di RDF, gestione grafi e gestione store. Tutto il codice sviluppato per il progetto è disponibile nel repository pubblico Gitlab [6], ulteriori esempi sull'utilizzo della libreria sono invece disponibili nel repository ufficiale RDFLib [7].

[4] `https://github.com/RDFLib/rdflib/releases/tag/5.0.0`
[5] `https://www.w3.org/2001/sw/wiki/RDFLib`
[6] `https://gitlab.com/paganelli.f/ws-project-paganelli-2021`
[7] `https://github.com/RDFLib/rdflib/tree/master/examples`

2.1 Tecnologie Principali

Si descrivono brevemente in questa sezione le tecnologie utilizzate/implementate dalla libreria RDFLib per fornire le funzionalità precedentemente elencate.

2.1.1 RDF/XML

XML (eXtensible Markup Language) è un metalinguaggio universale per la definizione di markup. Fornisce un framework per lo scambio/manipolazione di dati tra applicazioni ma non permette di associare un significato a quei dati. Per esempio, non esiste alcun significato associato alla struttura dei tag: ogni applicazione deve interpretare il nesting.
Con RDF (Resource Description Framework) si intende invece un framework ideato appositamente per aggiungere semantica ai dati scambiati tra applicazioni. Il modello dei dati RDF si basa su tre principi chiave [8]:

- Qualunque cosa è identificata da un Uniform Resource Identifier (URI). Un URI può essere un URL o un altro tipo di identificatore univoco. RDF permette infatti di estendere il sistema di connessione del web utilizzando identificatori anche per la relazione tra oggetti
- The least power: utilizzare il linguaggio meno espressivo per definire qualunque cosa.
- Qualunque cosa può dire qualunque cosa su qualunque cosa. L'unità base per rappresentare un'informazione in RDF è lo statement. Uno statement è una tripla del tipo $\langle subject, predicate, object \rangle$, dove il subject è una risorsa, il predicate è una proprietà e l'object è un valore (e quindi anche un URI che punta ad un'altra risorsa).

Questi principi chiave di RDF permettono il riuso dei dati, l'integrazione dei dati provenienti da sorgenti differenti evitando l'implementazione di appositi middleware custom e la decentralizzazione dei dati.

[8] `https://it.wikipedia.org/wiki/Resource_Description_Framework`

2.1.2 N3 - Turtle

Notation-3 (o N3) è uno dei formati, insieme a N-triples, Turtle e XML, utilizzati per la rappresentazione di un grafo RDF.
Turtle è un'alternativa a RDF/XML, la sintassi originariamente unica e standardizzata per RDF. Al contrario di RDF/XML, Turtle non è basato su XML e in generale è riconosciuto come maggiormente leggibile e più semplice da modificare manualmente. Turtle ("Terse RDF Triple Language") è un formato ideato per esprimere dati di tipo RDF con una sintassi adatta a SPARQL. Secondo le convenzioni RDF, le informazioni sono rappresentate per mezzo di "triple", ciascuna delle quali consiste di un soggetto, un predicato e un oggetto. Ognuno di questi elementi è espresso come URI.

2.1.3 JSON-LD

Con JSON-LD (JavaScript Object Notation for Linked Data) si intende un metodo per la codifica di linked data utilizzando il formato JSON. È progettato per fornire un contesto ai dati memorizzati tramite mappings aggiuntivi tra il formato JSON e il modello RDF: mette in relazione oggetti in formato JSON con concetti contenuti in una ontologia.
Questo formato è molto diffuso tra i più conosciuti Knowledge Graph esistenti, ad esempio Google Knowledge Graph, e permette l'utilizzo di tecnologie NoSQL per la memorizzazione delle triple RDF.

2.1.4 SPARQL

RDF è un framework per la rappresentazione di dati nel Web e la sua specifica definisce una sintassi ed una semantica per il linguaggio di interrogazione SPARQL, utile ad esprimere query strutturate su sorgenti eterogenee di dati in formato RDF. SPARQL (SPARQL Protocol and RDF Query Language) è uno degli elementi chiave del Web Semantico poiché permette l'estrazione di informazioni dalla base di conoscenza distribuita sul web (Knowledge Graph).

2.1.5 Knowledge Graph

Un Knowledge Graph rappresenta una collezione di descrizioni di entità (oggetti, eventi e concetti) connesse tra loro. Fornisce un framework per integrare, unificare, analizzare e condividere dati combinando caratteristiche di diversi paradigmi di data management:

Database - I dati possono essere interrogati tramite query strutturate.

Graph - I dati possono essere analizzati come ogni altra struttura dati a rete.

Knowledge Base - Semantica necessaria per interpretare i dati e inferire nuova conoscenza.

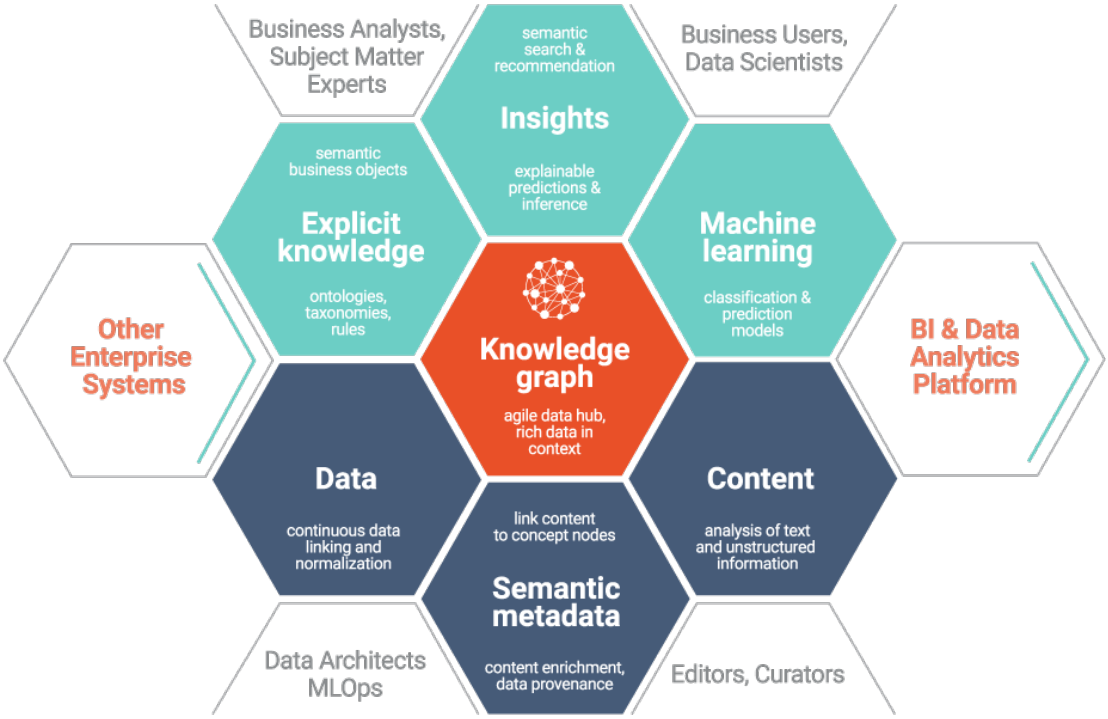


Figura 3: Tecnologie principali abilitanti il Knowledge Graph

Non tutti gli RDF Graph sono Knowledge Graphs. Una rappresentazione a grafo dei dati è utile ma può essere non necessaria per catturare la conoscenza semantica dei dati. Ad esempio l'insieme dei dati statistici rappresentati in RDF non è un Knowledge Graph.

3 Struttura Libreria

In questa sezione di descrive la struttura della libreria RDFLib, i suoi componenti e la loro implementazione confrontando gli standard W3C.

3.1 RDF Terms

Con RDF Terms si intendono le tipologie di oggetti che possono comparire all'interno di una tripla RDF. Ognuna delle seguenti principali tipologie di RDF Terms è sottoclasse della classe *Identifier*:

Blank Node - (BNode) Nodo in un RDF Graph che rappresenta una risorsa, detta risorsa anonima, per il quale non è presente un URI o un literal. Può essere usato solamente come subject o object in una tripla RDF mentre in altre sintassi come N3 può essere usato anche come predicato.

URI Reference - (URIRefs) Stringa Unicode che non contiene caratteri di controllo e produce una sequenza di caratteri rappresentante un URI valido.

Literal - Rappresenta il valore degli attributi RDF e possono avere un data-type o un tag language. Un literal in un RDF Graph contiene uno o più componenti named.

3.2 RDF Nodes

I nodi del grafo sono un sottoinsieme dei termini che lo store memorizza e consistono nei subject e object delle triple del grafo.

3.3 RDF Graph

Un modello RDF è rappresentabile da un grafo orientato sui cui nodi ci sono risorse o tipi primitivi e i cui archi rappresentano le proprietà. Un grafo RDF è rappresentato fisicamente mediante una serializzazione. Le principali serializzazioni adottabili per un grafo RDF sono: (*i*) RDF/XML, (*ii*) N-Triples e (*iii*) Notation3. La struttura che si crea forma una grafo orientato a etichette dove gli archi rappresentano le relazioni tra le risorse, queste ultime rappresentate dai nodi.

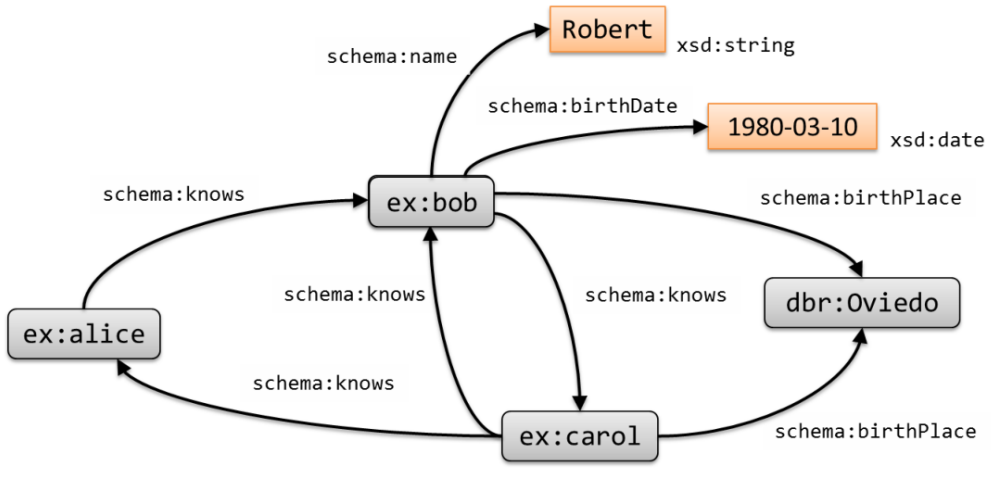


Figura 4: Esempio di grafo RDF

Un RDF Graph può essere creato indicando un *Identifier* che lo identifica per nome: se non indicato, di default viene assegnato un BNode per identificarlo. La libreria RDFLib definisce le seguenti tipologie di grafo:

Graph - Insieme di triple RDF, modellate con tuple a 3 terms nella forma $\langle subject, predicate, object \rangle$.

Quoted Graph - Ad ogni grafo è associato un identificatore fornito dal parser N3 e permette la definizione di formule N3. Estende la definizione base '*Graph*' con il concetto di nodo '*Formula*'.

Conjunctive Graph - Aggregazione (unnamed) di tutti i named graphs contenuti in uno store.

Dataset - Estensione del grafo di tipo Conjunctive Graph che non può essere identificato da un BNode. Fornisce metodi differenti per manipolare il grafo.

L'insieme dei nodi di un grafo è dato dai *subject* e *object* contenuti nelle triple del grafo.

3.4 RDF Namespaces

Con XML namespace si intende una collezione di nomi, identificati da URI e utilizzati nei documenti XML come nomi per gli attributi e tipi per gli elementi. I seguenti namespace comuni sono disponibili all'interno della libreria: RDF,

RDFS, OWL, XSD, FOAF, SKOS, DOAP, DC, DCTERMS e VOID, ma è possibile anche definire un proprio namespace tramite l'apposito *NamespaceManager* [9] ed effettuare il binding al documento RDF.

3.5 RDF Store

Gli RDF Store sono chiamati anche Triple Store o Quad Store e sono ideati per memorizzare triple RDF e altre informazioni sul contesto e sul grafo [10]. Un RDF Store dovrebbe fornire sia interfacce standard per la gestione delle connessioni a DBMS come Oracle, MySQL, Berkeley DB e Postgres che interfacce standard per la manipolazione, gestione, creazione o rimozione (operazioni CRUD) di triple RDF.

Uno store può essere:

Formula-aware - Store in grado di distinguere statements asserted da statements quoted.

Context-aware - Store in grado di memorizzare statements insieme al contesto, permettendo la partizione del modello RDF che rappresenta in named subgraphs.

3.5.1 IOMemory

L'implementazione RDF Store di default fornita dalla libreria consiste in uno integer-key-optimized context-aware in-memory store [11] basata su Berlekey DB. Tale store fa uso di indici a tre dict (subjects, objects e predicates) per mantenere le triple. Le informazioni sul contesto sono invece tracciate tramite un dict separato dove la chiave è data dalla tripla mentre il valore è dato da un dict nella forma $\{context : quoted\}$. Altre implementazioni di store per l'integrazione con diverse tipologie di DBMS sono disponibili tramite plugins [12]: fra questi è possibile trovare SPARQL [13], come indicato nelle principali features della libreria RDFLib.

[9] `https://rdflib.readthedocs.io/en/stable/namespaces_and_bindings.html`
[10] `https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/`
[11] `https://rdflib.readthedocs.io/en/stable/apidocs/rdflib.plugins.html`
[12] `https://rdflib.readthedocs.io/en/stable/plugin_stores.html`
[13] `https://rdflib.readthedocs.io/en/stable/apidocs/rdflib.plugins.stores.html`

4 Analisi Dominio WSN-MAS

La definizione di una ontologia può essere paragonata alla definizione di un insieme di dati e della loro struttura in modo che possano essere utilizzati da altri programmi. Una ontologia rappresenta il modello di uno specifico dominio, realizzato appositamente per un certo scopo. Nel caso specifico del progetto il dominio rappresenta un sistema application-specific per il rilevamento di incendi nella foreste utilizzando tecnologie come Wireless Sensor Network e Multi-Agent System.

4.1 Analisi Ontologie Riusabili

Con la diffusione del Semantic Web le ontologie disponibili in rete sono sempre più numerose. Raramente è necessario partire da zero nella definizione di una nuova ontologia: quasi sempre è disponibile una ontologia di terze parti che fornisce un buon punto di partenza.
Le ontologie esistenti individuate come base di sviluppo sono:

SSN (Semantic Sensor Network Ontology) [14] - Ontologia che descrive sensori, attuatori, osservazioni e concetti relativi.

SOSA (Sensor, Observation, Sample, and Actuator Ontology) [15] - Ontologia basata sull'ontologia SSN sviluppata dal W3C Semantic Sensor Networks Incubator Group (SSN-XG) con considerazioni dal W3C/OGC Spatial Data on the Web Working Group.

QUDT [16] - Ontologia che descrive specifiche riguardanti unità di misura, tipo di quantità, dimensioni e tipi di dato.

GEO [17] - Ontologia per rappresentare latitudine, longitudine e altre informazioni relative a oggetti spatially-located.

[14] `https://www.w3.org/ns/ssn/`
[15] `https://www.w3.org/ns/sosa/`
[16] `https://www.qudt.org/`
[17] `https://www.w3.org/2003/01/geo/`

4.2 Binding Namespace RDFLib

Ogni istanza di grafo RDFLib contiene al suo interno una istanza di *Namespace Manager* che si occupa di gestire i namespace, mantenendo il mapping con i relativi prefissi. I prefissi vengono utilizzati principalmente durante la serializzazione dei documenti RDF o nell'esecuzione delle query SPARQL.
Tramite il metodo *bind* fornito dal namespace manager è possibile definire ulteriori mapping tra namespace e prefissi custom rispetto a quelli di default.

```
SOSA: Namespace = Namespace('http://www.w3.org/ns/sosa/')
wsd_ffd_rdf.bind('sosa', SOSA)

SSN: Namespace = Namespace('http://www.w3.org/ns/ssn/')
wsd_ffd_rdf.bind('ssn', SSN)

SSN_SYSTEM: Namespace =
    Namespace('http://www.w3.org/ns/ssn/systems/')
wsd_ffd_rdf.bind('ssn-system', SSN_SYSTEM)

GEO: Namespace =
    Namespace('http://www.w3.org/2003/01/geo/wgs84_pos#')
wsd_ffd_rdf.bind('geo', GEO)

QUDT: Namespace = Namespace('http://qudt.org/1.1/schema/qudt#')
wsd_ffd_rdf.bind('qudt-1-1', QUDT)

QUDT_UNIT: Namespace =
    Namespace('http://qudt.org/1.1/vocab/unit#')
wsd_ffd_rdf.bind('qudt-unit-1-1', QUDT_UNIT)

SCHEMA: Namespace = Namespace('http://schema.org/')
wsd_ffd_rdf.bind('schema', SCHEMA)
```

Listing 1: Binding namespaces utilizzati

In XML i namespace sono utilizzati solamente con lo scopo di disambiguare gli elementi del documento. In RDF i namespace sono utilizzati invece come mec-

canismo di espansione: ovvero altri documenti RDF che definiscono determinate risorse che possono essere importate permettendo il riuso.

4.3 Enumerazione Termini

Il primo passo utile per la definizione di una ontologia è la scrittura di una lista di tutti i termini rilevanti del dominio che ci si aspetta di modellare con la ontologia. Sono elencati di seguito, suddivisi in tre categorie, gran parte dei principali termini rilevanti individuati nell'analisi del dominio.

4.3.1 Multi-Agent System

- Platform
- Agent
- JIDS
- Strategy
- DataStreamer
- SensorStreamer
- FFDSensorStreamer
- SensorAgent
- FFDSensorAgent
- SensorStrategy
- ActuatorStrategy
- SpreadingStrategy
- DBManagerAgent
- DBManagerStrategy
- FFDDBManagerAgent
- TriggerAgent
- TriggerStrategy
- FFDTriggerAgent
- StatisticsAgent
- FrontEndAgent

4.3.2 Wireless Sensor Network

- Device
- Sensor
- DHT11
- MQ-2
- KY-026
- Actuator
- SFM-27
- Node/Platform
- Gateway/Coordinator
- Router
- EdgeDevice/EndDevice
- Topology
- Environment

4.4 Definizione Tassonomia e Proprietà

Con tassonomia si intende la disciplina che si occupa della classificazione gerarchica di elementi. Dopo aver identificato i termini rilevanti è necessario organizzarli in una ben precisa tassonomia gerarchica che permetta il rispetto della semantica dei costrutti primitivi forniti dalle tecnologie adottate come *rdfs:subClassOf* e *owl:subClassOf*.
È normale definire anche delle proprietà durante l'organizzazione gerarchica dei termini, ovvero delle informazioni aggiuntive riguardanti il dominio e la classificazione come vincoli e dipendenze.

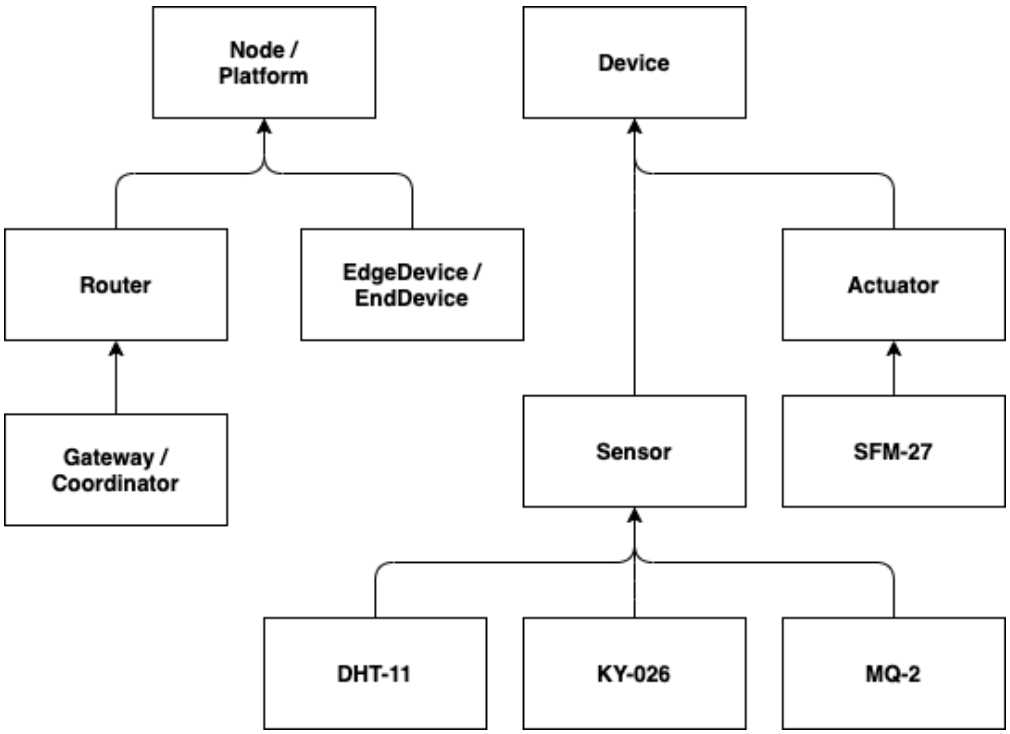


Figura 5: Tassonomie Nodi, Sensori e Attuatori WSN.

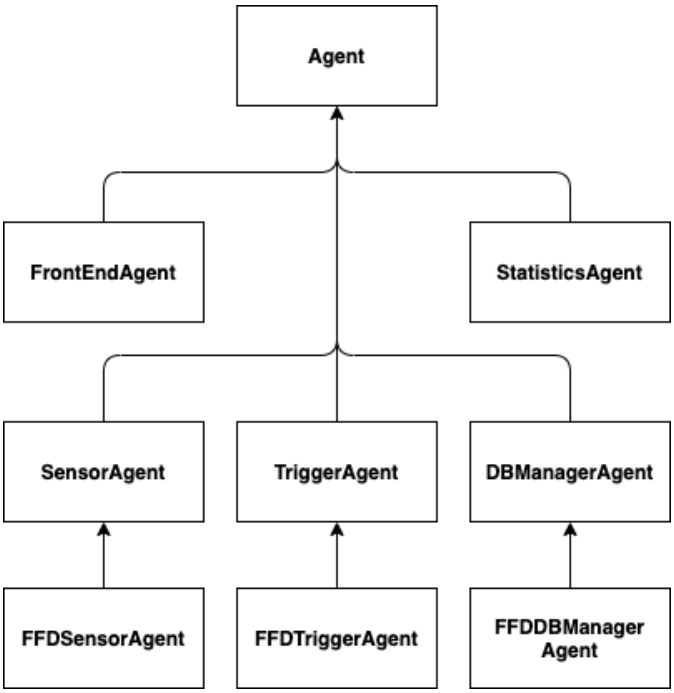


Figura 6: Tassonomia Agenti MAS.

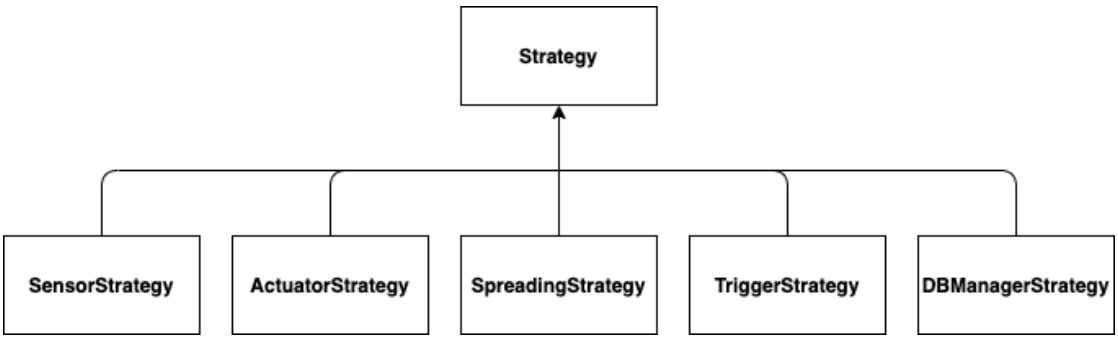


Figura 7: Tassonomia Strategie MAS.

5 Definizione RDFS

Lo schema RDF permette la definizione del vocabolario utilizzato nel data model RDF. In particolare consente di descrivere quali proprietà sono applicate a quali tipologie di oggetti, quali valori esse possono assumere e le relazioni che intercorrono tra gli oggetti. RDF è *domain-independent*, ovvero non è fatta alcuna assunzione su un particolare dominio che deve essere utilizzato: per questo motivo è compito dell'utilizzatore definire la propria terminologia facendo uso di RDFS.
In RDFLib è possibile definire sia lo schema RDFS che le istanze RDF tramite l'inserimento di triple in un grafo aventi la forma standard (*subject*, *predicate*, *object*).

```
wsd_ffd_rdf: Graph = Graph()
base_ontology: str = 'http://wsn-ffd.org/'
dht11 = URIRef(value='DHT11', base=base_ontology)
wsd_ffd_rdf.add((dht11, RDF.type, RDFS.Class))
wsd_ffd_rdf.add((dht11, RDFS.comment,
    Literal('DHT11 - humidity and temperature sensor')))
wsd_ffd_rdf.add((dht11, RDFS.seeAlso,
    Literal('https://www.adafruit.com/product/386')))
wsd_ffd_rdf.add((dht11, RDFS.subClassOf, SSN.Sensor))
```

Listing 2: Definizione RDFS Schema - DHT11 Sensor Class

```
temp_accuracy = URIRef(value='TempAccuracy', base=base_ontology)
wsd_ffd_rdf.add((temp_accuracy, RDF.type, RDF.Property))
wsd_ffd_rdf.add((temp_accuracy, RDFS.comment,
    Literal('Temperature accuracy')))
wsd_ffd_rdf.add((temp_accuracy, RDFS.domain, dht11))
wsd_ffd_rdf.add((temp_accuracy, RDFS.range, RDFS.Literal))
wsd_ffd_rdf.add((temp_accuracy, RDFS.subPropertyOf,
    SSN_SYSTEM.Accuracy))
```

Listing 3: Definizione RDFS Schema - DHT11 Sensor Property

Come primo step nella realizzazione della ontologia si definisce lo schema RDFS per alcuni dei termini principali individuati nella fase di enumerazione dei termini.

Questi includono sensori/attuatori, nodo base (edge device) per quanto riguarda le Wireless Sensor Network, agente base e sensor-agent per il Multi-Agent System.

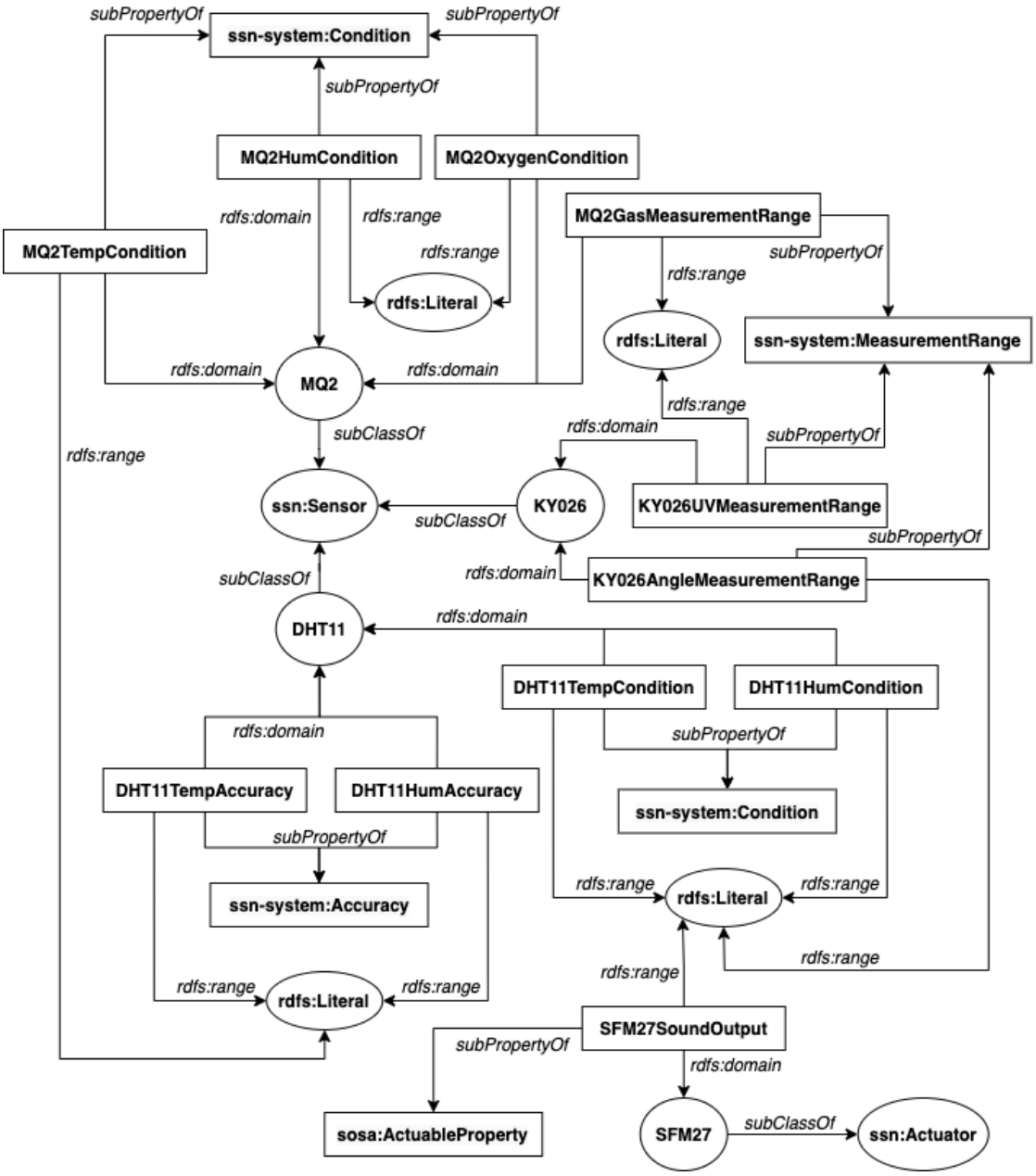


Figura 8: Schema RDFS - Sensori e attuatori installati su un nodo della rete.

Lo scopo della definizione di soltanto alcuni termini base del dominio è la successiva estensione tramite OWL, oggetto del terzo assignment.

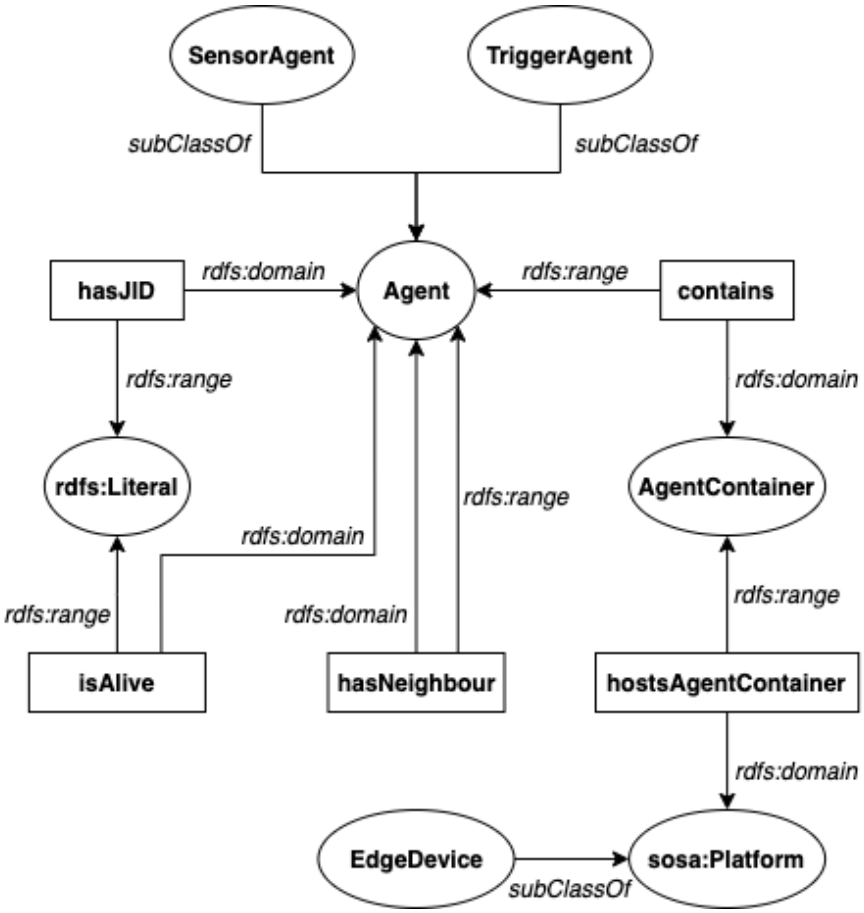


Figura 9: Schema RDFS - Agent/SensorAgent/TriggerAgent.

La proprietà *hostsAgentContainer*, con cui è stata estesa la classe *Platform* dell'ontologia SOSA, rappresenta il punto di connessione tra i due domini Wireless Sensor Network e Multi-Agent System. Essa infatti definisce la relazione tra la piattaforma hardware della rete di sensori, cioè il nodo, e l'insieme/container di agenti, modellato tramite la classe *AgentContainer*, che utilizza tale piattaforma come ambiente di esecuzione.

6 Definizione RDF

Le ontologie vengono utilizzate per organizzare un insieme di istanze, le quali sono tipicamente molto più numerose rispetto al numero delle classi e proprietà definite nell'ontologia.
Una ontologia, per essere definita tale deve rispettare alcune proprietà, tra cui:

Linguaggio formale - Il formalismo è una condizione necessaria per permettere alle macchine l'elaborazione delle informazioni.

Assunzioni su un dominio - Definisce/modella uno specifico dominio concettualizzando aspetti reali.

Condivisa - La comunità ritiene adeguata la modellazione del dominio fatta, cioè diversi utilizzatori hanno la stessa visione del dominio modellato.

Riassumendo, i linguaggi ontologici consentono agli utenti di scrivere concettualizzazioni esplicite e formali dei modelli di dominio.

```
edge_device_instance = URIRef(value='ED0000001',
                              base=base_ontology)

wsd_ffd_rdf.add((edge_device_instance, RDF.type, edge_device))
wsd_ffd_rdf.add((edge_device_instance, GEO.lat,
                                   Literal(55.701)))
wsd_ffd_rdf.add((edge_device_instance, GEO.long,
                                   Literal(12.552)))
wsd_ffd_rdf.add((edge_device_instance, SOSA.hosts,
                 Bag(wsd_ffd_rdf, BNode(),
                     [dht11_instance, mq2_instance,
                      ky026_instance, sfm27_instance
                     ]).uri
                 ))
wsd_ffd_rdf.add((edge_device_instance, is_routed_by,
                 Bag(wsd_ffd_rdf, BNode(),
                         [router_instance]).uri))
```

Listing 4: Esempio di definizione di istanza nodo EdgeDevice

Dall'esempio è possibile notare tre proprietà rilevanti che identificano un nodo:

Posizione geografica - Tramite le proprietà *lat* e *long* importate dal namespace GEO viene definita la posizione geografica della specifica istanza di nodo EdgeDevice.

Sensori/attuatori ospitati - Anche in questo caso si sfrutta una proprietà importata da un namespace esterno (SOSA). La proprietà *hosts* permette di indicare uno o più sensori/attuatori che risiedono fisicamente sul nodo.

Topologia WSN - La proprietà *isRoutedBy* è una proprietà custom definita nel namespace del progetto per indicare uno o più nodi di tipo Router in grado di instradare le informazioni verso uno specifico nodo. In questo modo si definisco le connessioni tra i nodi ed è quindi possibile indurre la topologia della WSN da queste proprietà.

Sia nel caso dei sensori/attuatori ospitati che per la topologia WSN viene utilizzato l'elemento container *Bag* fornito da RDF per indicare una collezione di risorse come unica risorsa nello statement. Nella tripla la risorsa container assume il ruolo di object.

```
ky026_instance = URIRef(value='KY0260000001',
                                base=base_ontology)
wsd_ffd_rdf.add((ky026_instance, RDF.type, ky026))
wsd_ffd_rdf.add((ky026_instance,
                 SOSA.isHostedBy,
                 edge_device_instance
                ))

ky026_power_range = BNode()
wsd_ffd_rdf.add((ky026_power_range, SCHEMA.minValue,
                                        Literal(3)))
wsd_ffd_rdf.add((ky026_power_range, SCHEMA.maxValue,
                                        Literal(5)))
wsd_ffd_rdf.add((ky026_power_range, SCHEMA.unitCode,
                                        QUDT_UNIT.V))
wsd_ffd_rdf.add((ky026_instance,
```

```
                    SSN_SYSTEM.OperatingPowerRange,
                    ky026_power_range
                   ))

ky026_uv_meas = BNode()
wsd_ffd_rdf.add((ky026_uv_meas, SCHEMA.minValue, Literal(760)))
wsd_ffd_rdf.add((ky026_uv_meas, SCHEMA.maxValue, Literal(1100)))
wsd_ffd_rdf.add((ky026_uv_meas, SCHEMA.unitCode,
                                    QUDT_UNIT.NanoM))
wsd_ffd_rdf.add((ky026_instance, uv_measurement, ky026_uv_meas))

ky026_angle_op_range = BNode()
wsd_ffd_rdf.add((ky026_angle_op_range, SCHEMA.value,
                                           Literal(60)))
wsd_ffd_rdf.add((ky026_angle_op_range, SCHEMA.unitCode,
                                           QUDT_UNIT.DEG))
wsd_ffd_rdf.add((ky026_instance,
                    angle_operating_range,
                    ky026_angle_op_range
                   ))
```

Listing 5: Esempio di definizione di istanza sensore KY-026

Per la definizione di sensori/attuatori è necessaria l'inclusione del namespace QUDT per esprimere correttamente tutte le grandezze fisiche che caratterizzano i rilevamenti effettuati dai sensori o le azioni eseguite dagli attuatori.

7 Definizione OWL

Un importante vantaggio nell'utilizzo di OWL rispetto RDFS è la possibilità di individuare inconsistenze nella ontologia stessa o in un insieme di istanze definite per popolare tale ontologia. Un esempio tipico di inconsistenza è l'incompatibilità dei *domain* e *range* definiti per le proprietà.
Con OWL (Ontology Web Language) si intende un semantic markup language standard W3C per la rappresentazione della conoscenza tramite la definizione e la condivisione di ontologie sul web, fornendo una maggiore espressività rispetto a RDF/RDFS. Questi ultimi permettono certamente la rappresentazione di qualche conoscenza ontologica ma soffrono la mancanza di alcune features importanti, ad esempio:

- Definizione di range delle proprietà che si applicano solamente ad alcune classi.
- Disgiunzione tra le classi. In RDFS è possibile definire solo relazioni del tipo *subClassOf*.
- Definizione di nuove classi tramite la combinazione di altre classi utilizzando operatori come unione, intersezione e complemento.
- Definizione di vincoli di cardinalità delle proprietà.

```
OWL: Namespace = Namespace('http://www.w3.org/2002/07/owl#')
wsd_ffd_owl.bind('owl', OWL)

ontology = URIRef(value='wsn-mas-ontology', base=base_ontology)
wsd_ffd_owl.add((ontology, RDF.type, OWL.Ontology))
wsd_ffd_owl.add((ontology, OWL.versionInfo, Literal('1.0')))
wsd_ffd_owl.add((ontology, OWL.imports,
                        URIRef('https://wsn-ffd-rdf.org/')))
```

Listing 6: Import namespace OWL e namespace RDF definito nel secondo assignment per estensione

Il precedente listato rappresenta la definizione dell'elemento *owl:Ontology* tramite la libreria RDFLib. Tale elemento è spesso presente all'inizio dell'ontologia e contiene un insieme di asserzioni come commenti e inclusioni di namespace. Rispetto

a RDF, l'import di un namespace indica che tutto il suo contenuto è parte della ontologia corrente. In questo caso l'elemento *owl:imports* include il documento RDF definito nella parte precedente del progetto con lo scopo di estenderlo.

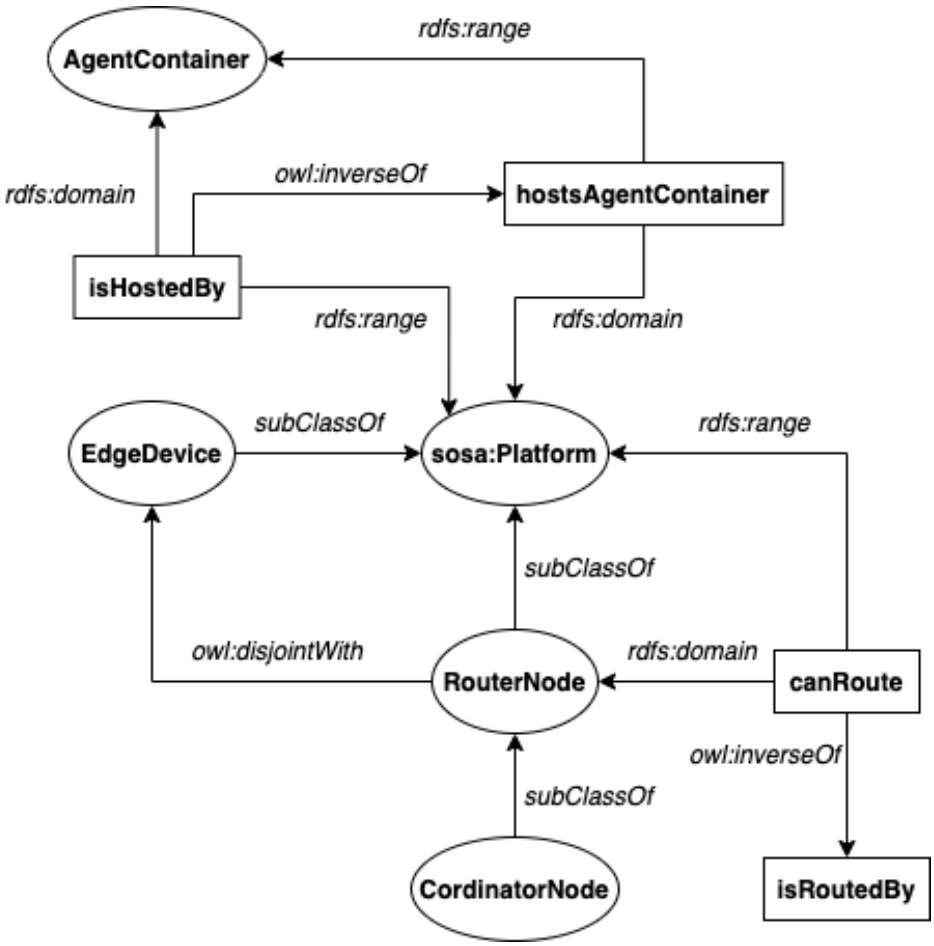


Figura 10: Estensione OWL - RouterNode/Coordinator

Tramite il linguaggio OWL è stato possibile definire associazioni/vincoli tra le classi e le proprietà che compongono il sistema. Alcuni dei costrutti utilizzati sono:

Restriction - Utilizzato per modellare i vincoli del dominio.

onProperty - Utilizzato per associare i vincoli alle specifiche proprietà.

inverseOf - Utilizzato per esplicitare proprietà che sono una l'inverso dell'altra.

disjointWith - Utilizzato per esplicitare classi disgiunte fra loro.

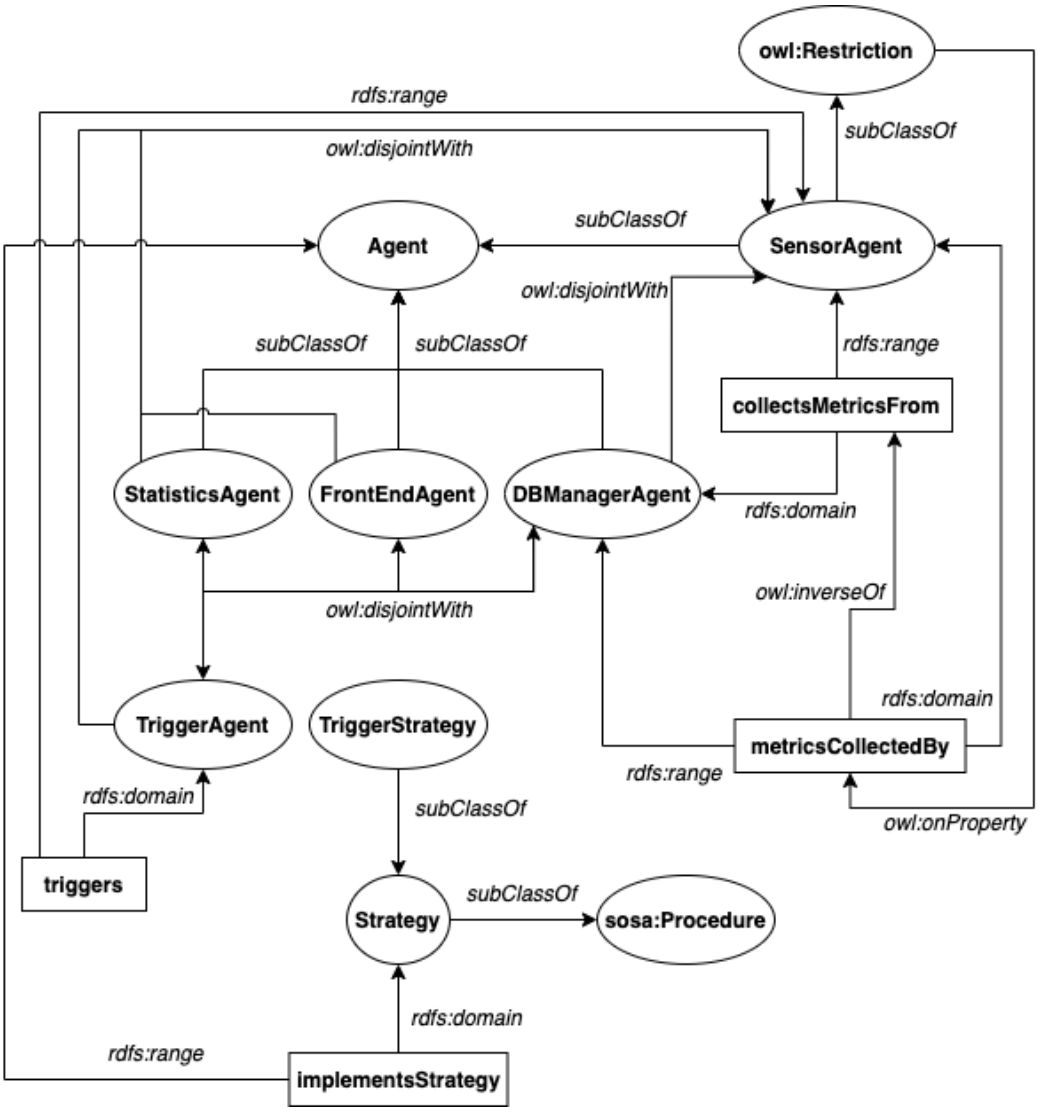


Figura 11: Estensione OWL - DBManagerAgent/StatisticsAgent/FrontEndAgent/Strategy

7.1 Definizione Istanze

Si definisce ora come è stato implementato un esempio di intero sistema "realmente" dispiegato in un ambiente tramite la modellazione delle istanze che lo compongono e le relazione che intercorrono tra esse. Il sistema descritto si suddivide nei due sottodomini Wireless Sensor Network e Multi-Agent System, indicati separatamente per comodità.
La WSN modellata è composta da un nodo coordinator, tre nodi router e sei nodi edge per un totale di dieci nodi. Le proprietà *canRoute* e *isRoutedBy* permettono di definire la topologia della rete:

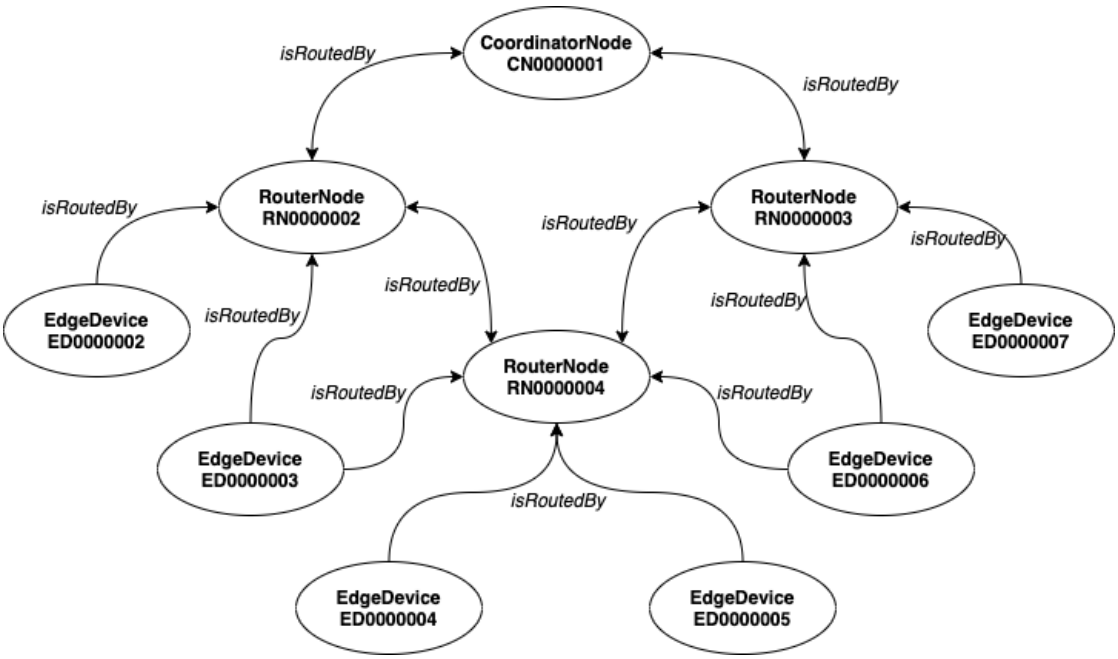


Figura 12: Topologia della WSN definita tramite OWL

Per ogni nodo edge della rete è definito un relativo *AgentContainer* che rappresenta l'ambiente di esecuzione degli agenti che compongono il MAS. Il seguente schema modella il container ospitato dal nodo edge identificato con *ED0000002*:

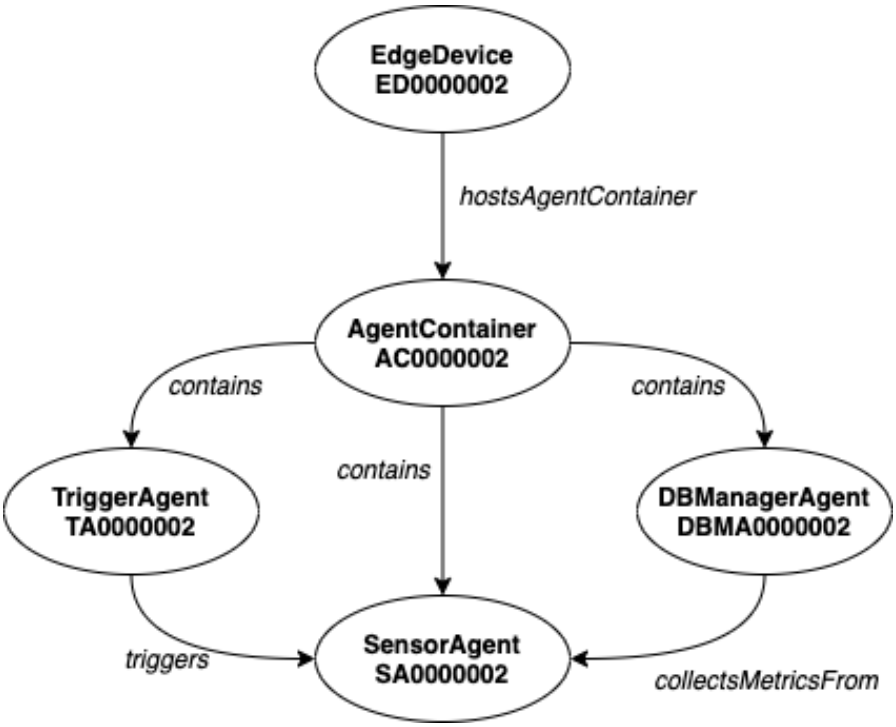


Figura 13: Struttura MAS - agenti in esecuzione su un nodo edge della rete

8 Serializzazione e Validazione

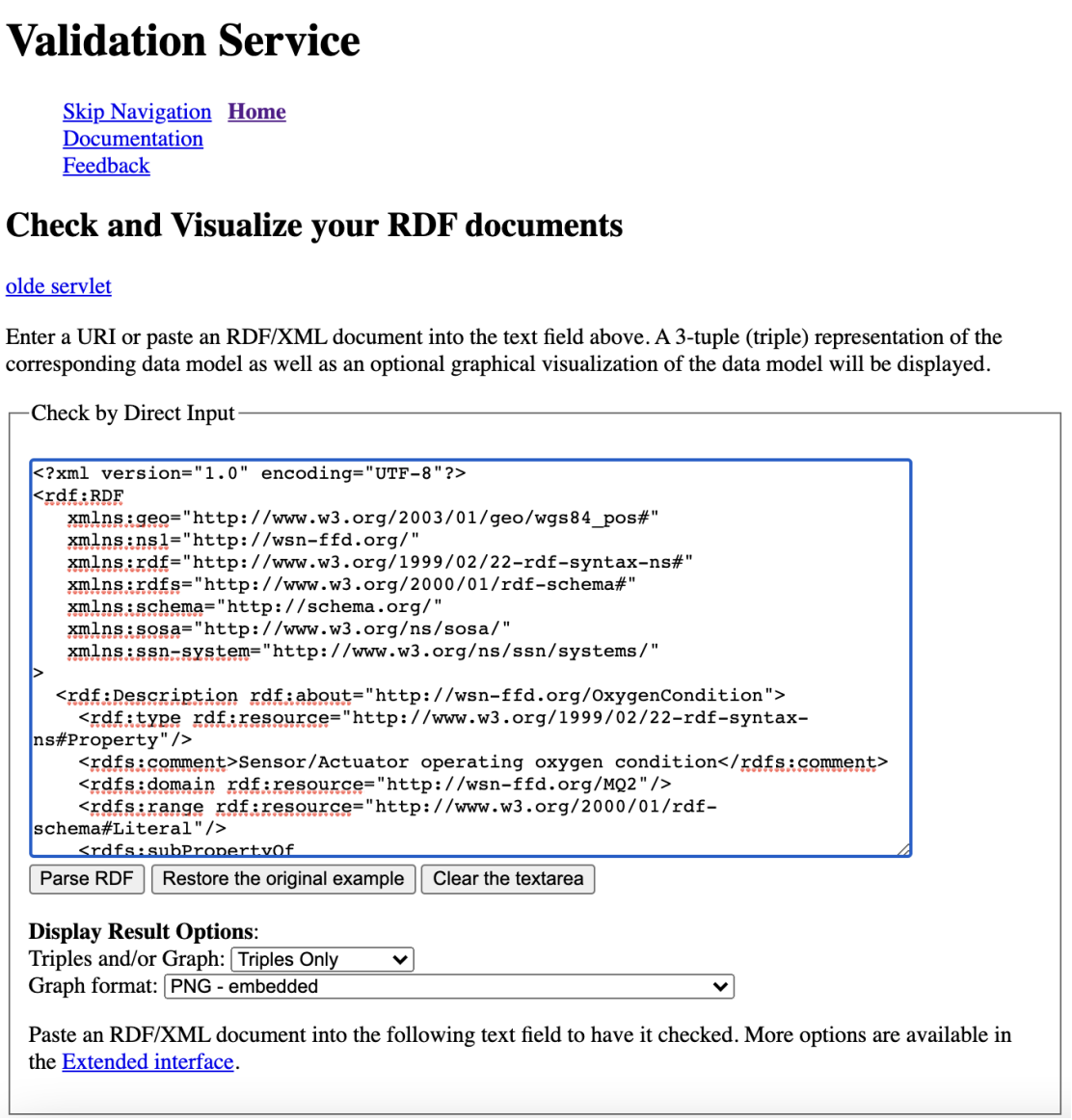
Una delle importanti features della libreria RDFLib è la possibilità di serializzare il grafo RDF definito in diversi formati e diversi encoding. Il formato di default di serializzazione è *turtle* ma è possibile indicare uno dei seguenti formati built-in o aggiungerne altri tramite plugins: *xml*, *n3*, *turtle*, *nt*, *pretty-xml*, *trix*, *trig* e *nquads*.
Il grafo RDF definito per il progetto è stato serializzato nei formati (*i*) *xml*, (*ii*) *pretty-xml* e (*iii*) *turtle*, documenti rinominati rispettivamente come (*i*) *wsd-ffd-xml.rdf*, (*ii*) *wsd-ffd-pretty-xml.rdf* e (*iii*) *wsd-ffd-turtle.ttl*.

```
wsd_ffd_rdf: Graph = Graph()

wsd_ffd_rdf.serialize(
    destination=os.getcwd()+'/wsd-ffd-xml.rdf',
    format='xml'
)

wsd_ffd_rdf.serialize(
    destination=os.getcwd()+'/wsd-ffd-pretty-xml.rdf',
    format='pretty-xml'
)

wsd_ffd_rdf.serialize(
    destination=os.getcwd()+'/wsd-ffd-turtle.ttl',
    format='turtle'
)
```

Listing 7: Esempio di serializzazione grafo RDF in XML, Pretty-XML e Turtle con RDFLib

La serializzazione delle ontologie realizzate per il terzo assignment avviene utilizzando la stessa API della libreria includendo il documento RDF realizzato precedentemente per il secondo assignment. Anche in questo caso i documenti sono stati serializzati utilizzando i tre formati *xml*, *pretty-xml* e *turtle*.

Ognuno dei tre documenti creati è stato testato/validato tramite il tool ufficiale W3C [18] con successo:



Validation Service

Skip Navigation **Home**
Documentation
Feedback

Check and Visualize your RDF documents

olde servlet

Enter a URI or paste an RDF/XML document into the text field above. A 3-tuple (triple) representation of the corresponding data model as well as an optional graphical visualization of the data model will be displayed.

Check by Direct Input

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
   xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
   xmlns:ns1="http://wsn-ffd.org/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:schema="http://schema.org/"
   xmlns:sosa="http://www.w3.org/ns/sosa/"
   xmlns:ssn-system="http://www.w3.org/ns/ssn/systems/"
>
  <rdf:Description rdf:about="http://wsn-ffd.org/OxygenCondition">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
    <rdfs:comment>Sensor/Actuator operating oxygen condition</rdfs:comment>
    <rdfs:domain rdf:resource="http://wsn-ffd.org/MQ2"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Literal"/>
    <rdfs:subPropertyOf
```

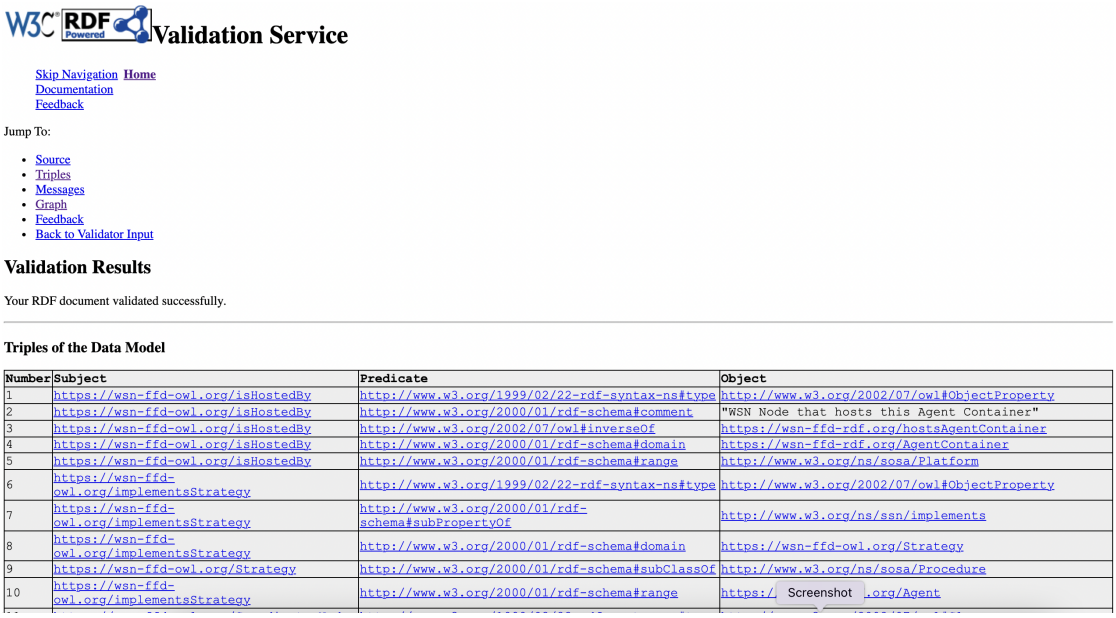
Parse RDF | Restore the original example | Clear the textarea

Display Result Options:
Triples and/or Graph: Triples Only
Graph format: PNG - embedded

Paste an RDF/XML document into the following text field to have it checked. More options are available in the Extended interface.

Figura 14: Screenshot W3C Validator Tool - Input documento RDF.

[18] https://www.w3.org/RDF/Validator/



W3C RDF Validation Service

Skip Navigation **Home**
Documentation
Feedback

Jump To:

- Source
- Triples
- Messages
- Graph
- Feedback
- Back to Validator Input

Validation Results

Your RDF document validated successfully.

Triples of the Data Model

Number	Subject	Predicate	Object
1	https://wsn-ffd-owl.org/isHostedBy	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
2	https://wsn-ffd-owl.org/isHostedBy	http://www.w3.org/2000/01/rdf-schema#comment	"WSN Node that hosts this Agent Container"
3	https://wsn-ffd-owl.org/isHostedBy	http://www.w3.org/2002/07/owl#inverseOf	https://wsn-ffd-rdf.org/hostsAgentContainer
4	https://wsn-ffd-owl.org/isHostedBy	http://www.w3.org/2000/01/rdf-schema#domain	https://wsn-ffd-rdf.org/AgentContainer
5	https://wsn-ffd-owl.org/isHostedBy	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/ns/sosa/Platform
6	https://wsn-ffd-owl.org/implementsStrategy	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
7	https://wsn-ffd-owl.org/implementsStrategy	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	http://www.w3.org/ns/ssn/implements
8	https://wsn-ffd-owl.org/implementsStrategy	http://www.w3.org/2000/01/rdf-schema#domain	https://wsn-ffd-owl.org/Strategy
9	https://wsn-ffd-owl.org/Strategy	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/ns/sosa/Procedure
10	https://wsn-ffd-owl.org/implementsStrategy	http://www.w3.org/2000/01/rdf-schema#range	https:/ Screenshot .org/Agent

Figura 15: Screenshot W3C Validator Tool - Output risultato validazione.

9 Criticità RDFLib

La release corrente stabile della libreria RDFLib è la 5.0.0, rilasciata in Aprile 2020 con supporto a Python 2 e 3.4 . Dalla roadmap ufficile la release della nuova versione 6.0.0 è schedulata per fine 2021 dove il supporto a Python passerà alle versioni maggiori o uguali alla 3.6 .
La libreria è testata tramite unit test sviluppati utilizzando il framework di testing Python Nose [19] ed il processo di sviluppo prevede pipeline Jenkins [20] per la continuous integration e la continuous delivery della libreria, pubblicata sul Python Package Index (PyPI).

9.1 Github Issues/PR

Ad oggi sono più di 200 le issue ancora aperte e le issues chiuse superano le 600 mentre le pull request ancora aperte sono circa 27 e quelle chiuse sopra le 500 ma la maggior parte di queste, sia issues che pull requests, riguardano i seguenti aspetti/criticità comuni:

- Problemi di compatibilità tra le dipendenze della libreria. Non per tutte le dipendenze di installazione e di sviluppo, definite nel file setup.py [21], è definita una specifica versione: questo comporta una risoluzione automatica delle dipendenze da parte di PyPI che potrebbe portare a conflitti/breaking changes [22].

- Scarse performance di alcuni algoritmi riguardanti principalmente l'esecuzione di query SPARQL e parsing [23].

- Conflitti con alcune specifiche dello standard W3C.

[19] `https://nose.readthedocs.io/en/latest/testing.html`
[20] `https://www.jenkins.io/`
[21] `https://github.com/RDFLib/rdflib/blob/master/setup.py`
[22] `https://github.com/RDFLib/rdflib/issues/1204`
[23] `https://github.com/RDFLib/rdflib/issues/1297`

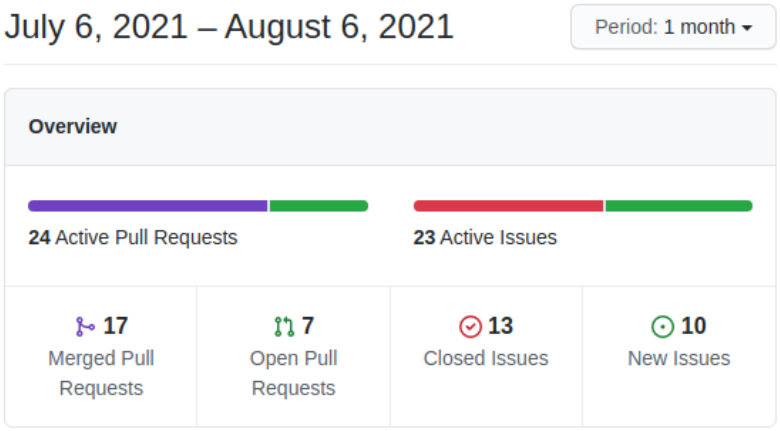


Figura 16: Attività della comunity nel periodo dal 6 Luglio al 6 Agosto 2021 - Chiusura/Apertura Issue e Pull Request.

Riferimenti bibliografici

[1] Grigorious Antoniou, Frank van Harmelen,
A Semantic Web Primer - second edition,
The MIT Press

[2] Natalya F. Noy, Deborah L. McGuinness,
Ontology Development 101: A Guide to Creating Your First Ontology,
Stanford University

[3] Grigorious Antoniou, Frank van Harmelen,
A Semantic Web Primer - second edition,
The MIT Press

[4] Toby Segaran, Colin Evans, Jamie Taylor,
Programming the Semantic Web,
O'Reilly

[5] Jesús Barrasa,
RDF Triple Stores vs. Labeled Property Graphs: What's the Difference?,
Neo4J

[6] Bryce Merkl Sasaki,
Graph Databases for Beginners: Other Graph Technologies,
Neo4J