



Università degli Studi di Milano Bicocca  
Scuola di Scienze  
Dipartimento di Informatica, Sistemistica e Comunicazione  
Corso di laurea in Informatica

# Sviluppo di un Algoritmo per la Generazione e la Verifica Automatica di Vincoli su Grafi RDF

**Relatore:** Prof. Andrea Maurino  
**Co-relatore:** Ing. Blerina Spahiu, PhD

**Relazione della prova finale di:**  
*Christian Bernasconi*  
*Matricola 816423*

Anno Accademico 2018–2019



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Background Tecnico</b>	<b>5</b>
2.1	Semantic Web . . . . .	5
2.1.1	RDF . . . . .	6
2.1.2	RDFS e OWL . . . . .	7
2.1.3	SPARQL e Triplestore . . . . .	8
2.2	Linked Open Data . . . . .	9
2.3	Qualità dei Dati . . . . .	10
2.4	SHACL . . . . .	12
<b>3</b>	<b>Stato dell'Arte</b>	<b>13</b>
3.1	Lavori Correlati . . . . .	13
3.2	Caratteristiche a Confronto . . . . .	15
<b>4</b>	<b>Approccio</b>	<b>19</b>
4.1	Abstat . . . . .	19
4.2	ShaclGenerator . . . . .	21
4.2.1	Implementazione . . . . .	21
4.2.2	Rapporto tra Shapes e AKPs . . . . .	23
4.3	Validazione Vincoli . . . . .	24
4.3.1	Approccio 1: Shaclex . . . . .	25
4.3.2	Approccio 2: API Estrazione Triple . . . . .	28
4.4	Integrazione Abstat . . . . .	32

4.4.1	Architettura Abstat . . . . .	32
4.4.2	Fasi di Integrazione . . . . .	33
<b>5</b>	<b>Esperimenti</b>	<b>37</b>
5.1	Casi di studio DBpedia . . . . .	37
5.1.1	Violazione Certa . . . . .	38
5.1.2	Violazione Probabile . . . . .	40
5.1.3	Violazione Cardinalità Inversa . . . . .	42
5.1.4	Discussione dei Risultati . . . . .	44
5.2	DBpedia 2015-2016: Analisi Generale . . . . .	45
5.2.1	Proposta Metodologia di Analisi . . . . .	45
5.2.2	Analisi Summaries . . . . .	46
5.2.3	Esempi Analisi Approfondita . . . . .	47
<b>6</b>	<b>Conclusioni</b>	<b>51</b>
<b>7</b>	<b>Sviluppi Futuri</b>	<b>53</b>
<b>Elenco delle figure</b>		<b>55</b>
<b>Elenco delle tabelle</b>		<b>57</b>
<b>Bibliografia</b>		<b>59</b>

# Capitolo 1

## Introduzione

L’evoluzione che negli anni ha attraversato il Web ha portato alla trasformazione del "Web of Documents" in quello che viene chiamato "Web of Data". Da qui prende forma il concetto di Semantic Web, ossia un Web di dati strutturati che si è sviluppato attraverso l’introduzione di nuove tecnologie e nuovi standard per la rappresentazione delle informazioni [4]. Sulla base di tali standard nascono i Linked Open Data<sup>1</sup>, progetto in continua crescita che prevede la pubblicazione nel cloud di datasets a licenza libera collegati tra loro. La crescita del cloud<sup>2</sup> nel corso del tempo ha portato ad una copertura sempre più ampia dei vari domini di interesse, con un numero di datasets interconnessi che è passato dai 12 del 2007 ai 1239 del 2019.

Dato il continuo incremento del numero dei datasets disponibili nel cloud, un aspetto molto importante in questo contesto è quello della qualità dei dati [25, 29, 9]. Con l’incessante crescita del numero di dati aperti messi a disposizione diventa fondamentale garantire all’utente fruitore una consistenza e un’attendibilità delle informazioni il più elevate possibile. Prendendo in esempio il caso del dataset di DBpedia<sup>3</sup>, il quale verrà ripreso nel corso di questa tesi, si può osservare come nei passaggi da una versione a quella successiva ci sia un aumento della quantità di informazioni che, parallelamente alla correzione di dati inconsistenti delle versioni precedenti, porta con sè nuovi potenziali errori [22].

---

<sup>1</sup><http://linkeddata.org/>

<sup>2</sup>[lod-cloud.net](http://lod-cloud.net)

<sup>3</sup><https://dbpedia.org>

Per far fronte al problema della qualità dei Linked Data, il W3C<sup>4</sup> propone come standard per la definizione di vincoli il linguaggio SHACL<sup>5</sup>. Tramite questo linguaggio è possibile definire delle regole di validazione che permettono di individuare gli errori di qualità presenti nel dataset. Per poter applicare i vincoli al dataset è necessario utilizzare un validatore SHACL, grazie al quale viene generato un report di validazione contenente tutti i dettagli delle violazioni riscontrate.

Per comprendere meglio l'importanza della qualità dei dati, soprattutto per quanto riguarda i Linked Open Data, è bene presentare un esempio concreto di scenario di utilizzo. Supponiamo che uno sviluppatore debba implementare un'applicazione che tratti libri e che decida di utilizzare i dati forniti da DBpedia. Supponiamo inoltre che un requisito dell'applicazione sia quello di consentire all'utente la ricerca di un libro tramite codice ISBN, il quale in una situazione ideale dovrebbe identificare univocamente ogni libro. Per permettere un corretto funzionamento dell'applicazione è necessario che nel dataset sia garantita l'univocità dell'associazione del codice ad un solo libro. Nel caso in cui il dataset presenti carenze di qualità riguardo tale informazione, però, potrebbe essere che lo stesso ISBN sia associato a diversi libri è ciò porterebbe ad un comportamento indesiderato dell'applicazione.

Dall'esempio precedente si può capire quanto sia importante la necessità di poter effettuare controlli sulla qualità dei dati aperti. Mettendosi nei panni di un esperto del dominio di interesse del dataset che deve effettuare delle valutazioni sulla qualità, le domande che ci si può porre per rendere più semplice il lavoro sono le seguenti: dato un dataset, esistono strumenti di supporto per individuare in modo automatico eventuali errori che ne inficiano la qualità? Che competenze tecniche richiedono? Sono utilizzabili in qualsiasi campo di applicazione?

A tale scopo vengono proposti nel corso degli anni svariati approcci accompagnati dallo sviluppo di tools di supporto per la valutazione della qualità di un dataset [29]. I vari approcci hanno diverse caratteristiche e affrontano il tema della qualità concentrandosi su più aspetti.

---

<sup>4</sup><https://www.w3.org/>

<sup>5</sup><https://www.w3.org/TR/shacl/>

Allo stato dell'arte troviamo approcci automatici come ad esempio LinkQA<sup>6</sup>, approcci semi-automatici come RDFUnit<sup>7</sup>, altri come Luzzu<sup>8</sup> che fanno della personalizzazione dei vincoli il loro punto di forza, altri ancora che come TripleCheckMate<sup>9</sup> si concentrano su specifiche dimensioni della qualità [14, 17, 8, 28]. In questa tesi verrà invece proposto un nuovo approccio completamente automatico, scalabile, che non richieda alcuna competenza tecnica all'utilizzatore e che sia utilizzabile per qualsiasi dominio di interesse.

L'approccio proposto prevede la generazione automatica di vincoli SHACL tramite lo ShaclGenerator<sup>10</sup>, i quali potranno poi essere utilizzati per validare il dataset ed effettuare test di regressione tra le sue versioni [26]. L'utente potrà inoltre consultare i dettagli dei vincoli violati per essere guidato in un'eventuale correzione delle informazioni errate. Il tutto è stato integrato nel tool di Abstat<sup>11</sup>, strumento che effettuando una profilazione del dataset fornisce all'utente un supporto per la sua analisi.

Nel capitolo 2 vengono fornite le nozioni tecniche di base necessarie per comprendere i capitoli successivi. Il capitolo 3 presenta lo stato dell'arte sugli approcci per la valutazione della qualità, illustrando le loro caratteristiche principali e confrontandole con quelle della soluzione proposta nella tesi. Tale soluzione sarà poi discussa nel dettaglio nel capitolo 4, nel quale vengono mostrati i passaggi che da un primo approccio fallimentare hanno portato alla soluzione finale. Nel capitolo 5, invece, troviamo dei test sulla qualità di diverse versioni DBpedia effettuati tramite lo strumento realizzato e una proposta di metodologia di analisi. Le conclusioni sul lavoro svolto e sui risultati ottenuti vengono discusse al capitolo 6. Infine, nel capitolo 7 vengono proposte delle nuove idee per possibili sviluppi futuri.

---

<sup>6</sup><https://github.com/LATC/24-7-platform/tree/master/latc-platform/linkqa>

<sup>7</sup><https://github.com/AKSW/RDFUnit>

<sup>8</sup><https://github.com/EIS-Bonn/Luzzu>

<sup>9</sup><https://github.com/AKSW/TripleCheckMate>

<sup>10</sup><https://github.com/christianbernasconi96/shacl>

<sup>11</sup>[https://bitbucket.org/disco\\_unimib/abstat](https://bitbucket.org/disco_unimib/abstat)



# **Capitolo 2**

## **Background Tecnico**

### **2.1 Semantic Web**

Il Semantic Web rappresenta l’evoluzione del World Wide Web, ossia il passaggio da un Web di documenti connessi da collegamenti ipertestuali a un Web di dati connessi da relazioni semantiche [4, 5]. Questa trasformazione avviene attraverso l’arricchimento dei documenti tramite informazioni e metadati che vanno a definirne il contesto semantico. Seguendo dei precisi standard è possibile definire gerarchie tra concetti e risolvere eventuali ambiguità come ad esempio quelle dovute ad omonimie. Ciò permette di rappresentare le informazioni in un formato che ne favorisce l’interpretazione e ne agevola l’elaborazione automatica, in quanto tramite i metadati è possibile definire relazioni tra i documenti molto più precise e significative rispetto a quelle ottenibili mediante un semplice link.

Come mostrato in figura 2.1, il Semantic Web è realizzato con un’architettura a strati. Nelle sezioni successive ne verranno illustrate brevemente le parti di interesse per questa tesi.

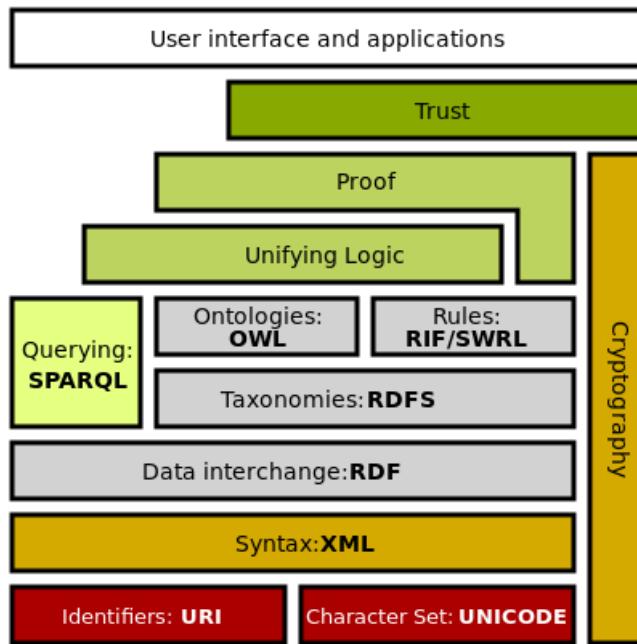


Figura 2.1 *Semantic Web Stack*

### 2.1.1 RDF

RDF<sup>1</sup> (Resource Description Framework) è uno standard proposto dal W3C nel 2014 che consente alle applicazioni di gestire e scambiarsi dati nel contesto del Semantic Web [10, 16]. Qualsiasi oggetto di qualsiasi tipo descritto tramite RDF è definito risorsa e viene univocamente identificato tramite un URI. Il costrutto base del modello di dati RDF è detto statement, ovvero una tripla  $\langle subject, predicate, object \rangle$  che rappresenta un'informazione e permette di esprimere relazioni tra le risorse. In una tripla, *subject* identifica la risorsa descritta, *predicate* rappresenta una proprietà, mentre *object* corrisponde al valore associato alla proprietà del *subject*.

La rappresentazione delle informazioni tramite questo modello permette di raccogliere le triple definendo un grafo RDF<sup>2</sup>, identificato a sua volta da un URI, in cui i soggetti e gli oggetti costituiscono i nodi e le proprietà corrispondono agli archi orientati. Un grafo RDF può essere serializzato attraverso vari formati, tra cui RDF/XML, N-Triple, N3, Turtle.

<sup>1</sup><https://www.w3.org/RDF/>

<sup>2</sup>sinonimi utilizzati nel corso della tesi: dataset, grafo di conoscenza

In particolare nel lavoro di questa tesi verranno utilizzati i formati N-Triple e Turtle.

Tra le tipologie di valori che può assumere un nodo del grafo troviamo URI, literal e blank node. Viene definito literal ciò che è rappresentato da una stringa di testo. Un literal può essere *plain* se è una stringa associata ad una lingua tramite tag (es: "Milan"@en), oppure *typed* se è una stringa associata all'URI di un datatype (es: "33"^^xsd:integer). Viene definito blank node un nodo non identificato da un URI, ossia un nodo non direttamente referenziabile dichiarato all'occorrenza in una tripla.

I valori che possono assumere gli elementi di una tripla sono i seguenti:

- **Soggetto:** URI HTTP / blank node
- **Predicato:** URI HTTP
- **Oggetto:** URI HTTP / literal / blank node

Nel caso in cui l'oggetto di una tripla sia un URI si parla di *object type triple*, mentre nel caso in cui l'oggetto sia un literal si parla di *data type triple*.

### 2.1.2 RDFS e OWL

RDFS<sup>3</sup> (RDF Schema) e OWL<sup>4</sup> (Web Ontology Language) sono linguaggi utilizzati per definire vocabolari RDF e ontologie [6, 3]. Un'ontologia è una collezione di assiomi il cui scopo è quello di definire delle regole, classificare termini generici (es: concetti, tipi di dati, proprietà) e specificare relazioni tra essi [7]. In particolare, un'ontologia può essere definita come una rappresentazione formale, esplicita (priva di ambiguità) e compatta (priva di ridondanze) di un'astrazione del dominio di interesse. Un vocabolario RDF, invece, descrive classi e relazioni in modo meno restrittivo e dettagliato.

In sintesi, RDFS e OWL si possono riassumere come:

- **RDFS:** estensione di RDF che permette attraverso nuovi termini di definire un vocabolario RDF [6]. Le classi base introdotte sono *Class* e *Property*. Tra i predici base,

---

<sup>3</sup><https://www.w3.org/TR/rdf-schema/>

<sup>4</sup><https://www.w3.org/OWL/>

invece, troviamo *subClassOf*, *subPropertyOf*, *domain* e *range*.

- **OWL**: estensione di RDF molto più espressiva e articolata rispetto a RDFS, utilizzata per definire ontologie [3]. OWL permette di descrivere dettagliatamente relazioni semantiche e caratteristiche di classi e proprietà. Tramite OWL è possibile esprimere svariati vincoli a cui le triple di un dataset devono sottostare, come ad esempio *hasValue* per imporre su una proprietà restrizioni sui valori, oppure *DatatypeProperty* per indicare che la proprietà deve associare al soggetto un literal.

Un ulteriore aspetto importante delle relazioni contenute in un’ontologia è quello della definizione di gerarchie tra i vari termini attraverso la proprietà di *subtype*. Ciò permette di costruire un albero gerarchico in cui la radice rappresenta la classe più generica da cui derivano tutti gli altri sottotipi.

### 2.1.3 SPARQL e Triplestore

SPARQL (SPARQL Protocol and RDF Query Language) è un linguaggio<sup>5</sup> nonché protocollo<sup>6</sup> per l’interrogazione di dati RDF, proposto come standard dal W3C nel 2008 [10]. Esso consente di estrarre informazioni da grafi di conoscenza tramite l’esecuzione di SPARQL queries verso un endpoint. Uno SPARQL endpoint è un server HTTP (identificato da un URL) sul quale sono memorizzate le triple.

Per poter eseguire le SPARQL queries su un dataset è necessario che i dati RDF siano contenuti in un Triplestore. Un Triplestore è un database atto alla memorizzazione e indicizzazione di triple RDF. Per il lavoro svolto in questa tesi il Triplestore che è stato utilizzato è quello di Virtuoso<sup>7</sup>, uno dei più diffusi allo stato attuale.

---

<sup>5</sup><https://www.w3.org/TR/rdf-sparql-query/>

<sup>6</sup><https://www.w3.org/TR/rdf-sparql-protocol/>

<sup>7</sup><http://vos.openlinksw.com/owiki/wiki/VOS>

## 2.2 Linked Open Data

Il concetto Linked Open Data<sup>8</sup> (LOD), nato dall'unione di Linked Data e Open Data, si basa fortemente sui principi del Semantic Web [10].

Con il termine Linked Data si intende una modalità di pubblicazione di dati strutturati collegati tra loro seguendo standard web aperti (es: HTTP, URI). Grazie a RDF è possibile definire in modo completo e dettagliato qualsiasi oggetto del mondo reale come concetto astratto. I collegamenti tra i dati vengono espressi in RDF mediante le relazioni. Attraverso questi meccanismi i dati vengono resi interrogabili semanticamente e di conseguenza ne viene agevolato il loro utilizzo da parte delle macchine.

I principi su cui fanno affidamento i Linked Data sono i seguenti:

1. identificare ogni oggetto con un URI
2. utilizzare HTTP URI per rendere reperibili alle persone i dati
3. fornire informazioni utili attraverso standard come RDF e SPARQL quando un URI viene dereferenziato
4. includere nella pubblicazione di dati link ad altri URI per favorire la ricerca di informazioni correlate

Per quanto riguarda il termine Open Data, esso fa riferimento a dati accessibili, riutilizzabili e ridistribuiti da chiunque, con unico eventuale vincolo quello di citare le fonti o di mantenere la base di dati sempre aperta.

I Linked Open Data rappresentano quindi un insieme di datasets aperti interconnessi seguendo le regole dei Linked Data, con l'obiettivo di fornire una base di conoscenza completa in continua evoluzione. Basti pensare che la situazione di partenza del LOD Cloud<sup>9</sup> nel 2007 contava un totale di 12 datasets, mentre ora questo numero è cresciuto fino ai 1239 del 2019. In figura 2.2 si può osservare la situazione attuale del LOD Cloud. Nell'immagine

---

<sup>8</sup><http://linkeddata.org/>

<sup>9</sup><https://lod-cloud.net>

viene anche messo in evidenza il dataset di DBpedia<sup>10</sup>, il quale sarà oggetto di test nel corso della tesi.

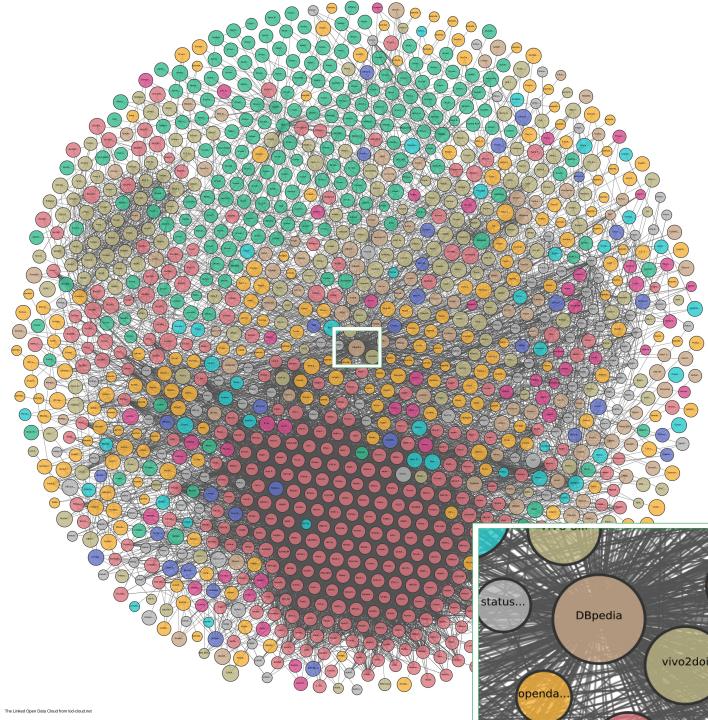


Figura 2.2 *Linked Open Data Cloud 2019 - Focus su DBpedia*

## 2.3 Qualità dei Dati

Quello della qualità dei dati è un concetto astratto che presenta varie sfaccettature e che si basa sulla valutazione di molteplici aspetti di un dataset [29, 2]. Esistono varie definizioni di qualità di dati, tra cui la più popolare è quella di "*'fitness for use'*[15]". In generale, il termine "qualità" va inteso come l'adeguatezza di determinati dati rispetto all'utilizzo a cui sono destinati. La qualità dei dati risulta quindi essere soggettiva rispetto alle esigenze secondo cui gli stessi verranno impiegati.

I fattori che influenzano la qualità dei dati sono svariati e ne forniscono più dimensioni [1]. Le dimensioni della qualità si possono principalmente dividere in *accessibility*,

---

<sup>10</sup><https://dbpedia.org>

*intrinsic, contextual e representational* [29]. Tra queste approfondiremo il gruppo delle dimensioni di tipo *intrinsic*, ossia quelle che sono indipendenti dal contesto e che si concentrano su correttezza (sintattica e semantica), compattezza e completezza delle informazioni. In particolare è utile render noti i concetti di *consistency* e di *semantic accuracy*, in quanto avranno un ruolo importante nella parte sperimentale di questa tesi.

La dimensione di *consistency* si basa sullo schema di riferimento di un dataset. Un dataset si dice *consistente* se e solo se è privo di contraddizioni logiche rispetto alla sua rappresentazione e ai relativi meccanismi di inferenza [29]. Considerando per esempio alcuni linguaggi come OWL, vengono definite delle semantiche ben precise che permettono di stabilire delle regole da rispettare per garantire consistenza in uno specifico contesto. In altri termini, portando questa definizione nell'ambito dei LOD, un dataset presenterà un'inconsistenza ognqualvolta dovessero essere riscontrate delle contraddizioni rispetto alle regole dettate dall'ontologia di riferimento. Un esempio di inconsistenza può essere la presenza nel dataset di un soggetto a cui sono state associate due date di nascita sebbene l'ontologia ne imponga come limite massimo una, generando di conseguenza una contraddizione.

La dimensione di *semantic accuracy*, al contrario, non fa riferimento allo schema di un dataset ma è più incentrata sui valori dei dati stessi. Con *accuratezza semantica* di un dataset si intende il grado con cui i dati rispecchiano i fatti del mondo reale [29]. La valutazione di errori semantici è quindi più vicina ad un'interpretazione umana dei fatti. L'individuazione di potenziali errori può essere effettuata tramite metodi statistici e algoritmi di machine learning. Per mostrare la differenza rispetto alla consistenza ci serviamo del seguente esempio. Supponiamo che la proprietà che rappresenta il colore di un soggetto sia una stringa vincolata ad assumere i valori "Rosso"/"Verde"/"Blu", ma che per una particolare istanza abbia come valore "Azzurro". In questo caso saremmo in presenza di un'inconsistenza in quanto viene violata una regola imposta al dataset, ma nonostante ciò il dato può considerarsi semanticamente accurato in quanto rappresenta realmente un colore.

## 2.4 SHACL

SHACL<sup>11</sup> (Shapes Constraint Language) è un linguaggio che permette di definire vincoli di validazione di un grafo RDF [13]. Esso, tra i vari linguaggi di validazione, è proposto dal W3C come standard nel 2017. Con SHACL i vincoli vengono espressi in RDF tramite delle shapes, ossia dei nodi contenenti le regole che a cui devono sottostare le triple di un dataset.

Per la definizione dei vincoli, SHACL mette a disposizione le due componenti SHACL Core<sup>12</sup> e SHACL SPARQL<sup>13</sup>. SHACL Core è un vocabolario RDF che permette di definire i vincoli base più comuni delle shapes. Tra essi troviamo ad esempio vincoli su classe o datatype dell’oggetto di una tripla, sulle cardinalità di una relazione, sulla lingua e molti altri ancora. Per quanto riguarda la componente SHACL SPARQL, essa è un’estensione del SHACL Core che prevede l’impiego di SPARQL queries all’interno di una shape. Sfruttando tale meccanismo è possibile definire delle nuove regole di validazione riutilizzabili e che non potrebbero essere realizzate con i soli vincoli base. Queste due componenti rendono SHACL un linguaggio estremamente versatile in quanto, se combinate tra loro, permettono di definire vincoli adatti a coprire ogni tipo di esigenza.

Ai fini della validazione di un grafo RDF attraverso le shapes è necessario fornire ad un validatore SHACL un data graph (dataset da validare) ed uno shape graph (dataset contenente i vincoli). L’output prodotto dal validatore sarà un report<sup>14</sup> in cui saranno riportate tutte le violazioni riscontrate. Il report verrà considerato conforme soltanto nel caso in cui durante la validazione non dovessero venire riscontrate violazioni, altrimenti verranno mostrati all’utente tutti gli errori relativi ad ogni singola shape.

---

<sup>11</sup><https://www.w3.org/TR/shacl/>

<sup>12</sup><https://www.w3.org/TR/shacl/#core-components>

<sup>13</sup><https://www.w3.org/TR/shacl/#sparql-constraints>

<sup>14</sup><https://www.w3.org/TR/shacl/#validation-report>

# Capitolo 3

## Stato dell'Arte

Il lavoro svolto per questa tesi ha l’obiettivo di proporre un approccio per l’individuazione di particolari cause di violazione della qualità di un dataset. Allo stato dell’arte si possono trovare svariati approcci e tools utili al fine di identificare e correggere eventuali errori che vanno a compromettere la qualità di un grafo RDF. In questo capitolo verranno illustrati alcuni dei possibili approcci in relazione alle funzionalità offerte e alle metodologie utilizzate.

### 3.1 Lavori Correlati

*Luzzu*<sup>1</sup> è un tool che offre all’utilizzatore la possibilità di verificare la qualità di un dataset sulla base delle metriche scelte [8]. Le metriche sono definibili e personalizzabili dall’utente tramite l’apposito linguaggio LQML<sup>2</sup>. Alle metriche può essere inoltre attribuito un peso per rendere la valutazione più precisa rispetto alle esigenze dell’utilizzatore. Il processo si svolge prendendo in input un dataset (fisico/endpoint) e una lista di metriche pesate, e producendo in output un report dei problemi e dei metadati contenenti informazioni sulla qualità.

*Flemming*, similmente a *Luzzu*, si propone di offrire la possibilità di effettuare una valutazione sulla base di metriche pesate scelte dall’utente [11]. In questo caso, però, non è possibile definire nuove metriche personalizzate. Il processo di valutazione della qualità

---

<sup>1</sup><https://github.com/EIS-Bonn/Luzzu>

<sup>2</sup>Luzzu Quality Metric Language, non richiede particolari competenze

di un dataset produce in output un report in forma di testo non strutturato, quindi non interrogabile e non direttamente utilizzabile dalle macchine.

**LinkQA**<sup>3</sup> è un tool che permette di verificare la qualità di un dataset e i relativi cambiamenti tramite tecniche di network analysis [14]. Oltre alle metriche messe a disposizione dal tool, è possibile specificarne di nuove purchè relative a misure topologiche. Dopo che l'utente seleziona le metriche e il dataset su cui validarle, il processo produce in output un report in formato HTML.

**Sieve**<sup>4</sup> fa parte di LDIF<sup>5</sup>, framework per applicazioni che trattano Linked Data [18]. Questo tool valuta la qualità sulla base dei metadati relativi alla provenienza dei grafi. Le metriche vengono selezionate dall'utente, il quale può definirne di nuove in formato XML. Inoltre è possibile predisporre un processo di pulizia del dataset configurabile dall'utente. Le informazioni prodotte come output vengono salvate come quadruple nella forma *<subject, predicate, object, graph>*.

**RDFUnit**<sup>6</sup> si concentra sulla verifica di vincoli di integrità [17]. La valutazione della qualità avviene applicando i vincoli ad un dataset eseguendo SPARQL queries tramite endpoint. L'utente potrà definire dei vincoli personalizzati in aggiunta a quelli built-in seguendo un apposito template SPARQL. L'output del processo di validazione è costituito da un report di validazione e dai risultati, entrambi salvati come Linked Data e mostrati in HTML.

**LiQuate** ha come obiettivo quello di identificare potenziali problemi di completezza, ridondanza e inconsistenza di un dataset [24]. Gli algoritmi impiegati si basano sui modelli probabilistici delle reti bayesiane e su sistemi rule-based. L'output fornito all'utente suggerisce con delle probabilità le eventuali situazioni di ambiguità e di incompletezza relative ai dati o alle relazioni tra le risorse.

---

<sup>3</sup><https://github.com/LATC/24-7-platform/tree/master/latc-platform/linkqa>

<sup>4</sup><http://wifo5-03.informatik.uni-mannheim.de/bizer/sieve/>

<sup>5</sup>Linked Data Integration Framework, <http://ldif.wbsg.de/>

<sup>6</sup><https://github.com/AKSW/RDFUnit>

***TripleCheckMate***<sup>7</sup> è un tool che si basa su reasoners e algoritmi di machine learning [28]. La risorsa selezionata sarà valutata in base alle metriche e a quanto appreso sulla base di conoscenza tramite il machine learning. Il processo di valutazione fornisce all'utente le informazioni per effettuare la verifica della risorsa selezionata, in particolar modo per valutarne la correttezza semantica.

***KBMetrics*** propone un approccio che consente di analizzare le caratteristiche, effettuare comparazioni su indici di qualità e verificare "l'adeguatezza al contesto"<sup>8</sup> di un dataset [23]. L'utente a seconda dell'utilizzo sceglierà metriche, specificherà i requisiti del contesto e selezionerà un eventuale dataset come modello da confrontare con quello di interesse. In base alle metriche selezionate verranno effettuate valutazioni tramite SPARQL query oppure assistite dall'utente. I risultati prodotti in output saranno interrogabili tramite SPARQL.

***Loupe***<sup>9</sup> è un tool che, dato un dataset, permette di esplorare i patterns delle triple (con relative statistiche e frequenze) per poter individuare potenziali problemi di qualità [19]. La UI messa a disposizione da Loupe permette di esplorare classi, proprietà, patterns di triple e ontologie. L'individuazione dei problemi di qualità necessita di una valutazione da parte dell'utente accompagnata da un eventuale supporto di altri tools [20].

## 3.2 Caratteristiche a Confronto

Gli approcci descritti precedentemente permettono di coprire molte dimensioni della qualità dei dati affrontando il problema attraverso diverse metodologie. In questa sezione verranno messi a confronto gli aspetti principali che differenziano tali altri approcci da quello che verrà proposto nel capitolo 4.

**Scalabilità.** La scalabilità è la capacità di un sistema di gestire la crescita del cari-

---

<sup>7</sup><https://github.com/AKSW/TripleCheckMate>

<sup>8</sup>"fitness for use"

<sup>9</sup><http://loupe.linkeddata.es/loupe/>

co di lavoro. Nel caso dei tools di valutazione della qualità dei dati, essa corrisponde alla capacità di gestire dataset di grandi dimensioni. La soluzione proposta in questa tesi può considerarsi scalabile in quanto, come si potrà osservare nella parte sperimentale al capitolo 5, permette di effettuare senza evidenziare problemi validazioni di dataset del calibro di DBpedia. In base a quanto raccolto dai documenti degli approcci proposti, risultano scalabili quelli di Luzzu, LinkQA, Sieve, RDFUnit, TripleCheckMate, KBMetrics e Loupe. Tra quelli non scalabili troviamo invece Flemming e LiQuate.

**Personalizzazione.** La personalizzazione rappresenta la possibilità da parte di un utente di configurare il sistema per adattarlo alle proprie esigenze. Tra i tools considerati, quelli che presentano un scarsa personalizzazione sono LinkQA e LiQuate, in quanto l’utente non può aggiungere datasets. Riguardo alla configurazione di metriche, quelli più personalizzabili sono Luzzu, Sieve e RDFUnit, a cui seguono poi i restanti tools. Nel caso della nostra soluzione, essa consente l’aggiunta di datasets e ontologie da parte dell’utente. Dal punto di vista delle metriche, però, non è personalizzabile dato che la valutazione della qualità è focalizzata sui vincoli di cardinalità. In merito a ciò va specificato che il lavoro è stato svolto in ottica futura prevedendo di consentire una discreta personalizzazione dei vincoli.

**Automazione.** Con automazione si intende la capacità di un sistema di eseguire determinati processi riducendo l’intervento umano. Tra i tools proposti, l’unico ad essere completamente automatico è LinkQA in quanto non richiede alcuno intervento da parte dell’utente. Tutti gli altri tools sono invece semi-automatici. A seconda dei casi l’utente può essere coinvolto nella definizione di metriche, nell’attribuzione di pesi per definire il contesto o nella valutazione dei possibili errori emersi dalle verifiche del tool. L’approccio proposto in questa tesi permette di tradurre in modo automatico nello standard SHACL i vincoli da applicare al dataset. Una volta selezionato il vincolo da verificare, l’unica azione lasciata all’utente sarà quella di effettuare un’eventuale correzione del dataset.

**Competenze richieste.** Tra i tools che permettono la definizione di metriche custom quello che richiede minori competenze tecniche per la formulazione di vincoli è Luzzu, il

quale mette a disposizione il linguaggio LQML (in alternativa alla possibilità per i più esperti di utilizzare Java per definire vincoli più articolati). Per quanto riguarda Sieve sono invece richieste conoscenze di base XML. Segue poi RDFUnit che richiede conoscenze SPARQL per costruire nuovi pattern sulla base dei templates forniti. Infine troviamo LinkQA che richiede competenze Java e LiQuate che richiede la presenza di un esperto che formuli vincoli sulla base di modelli probabilistici. Per quanto riguarda il nostro approccio si può dire che non richieda alcuna particolare conoscenza tecnica tra quelle sopracitate. Ciò è possibile in quanto le SPARQL queries con cui viene effettuata la validazione dei vincoli vengono generate automaticamente.

Altre differenze (meno rilevanti per lo scopo di questo lavoro) tra questi approcci ed ulteriori alternative vengono illustrate e confrontate in [29]<sup>10</sup>.

---

<sup>10</sup>Caratteristiche a confronto in Table 7 e Table 8



# **Capitolo 4**

## **Approccio**

In questo capitolo verrà illustrato l'intero processo di validazione e l'approccio che, passando per un primo tentativo fallimentare, ha portato alla sua realizzazione. L'obiettivo finale è quello di realizzare uno strumento che prendendo in input la profilazione di un dataset qualsiasi possa tradurre in SHACL e poi verificare vincoli sulla qualità dei dati in maniera completamente automatica. Il tutto sarà poi integrato con il tool di profilazione di Abstat.

Nelle sezioni successive per prima cosa verrà presentato Abstat e i concetti su cui si basa in 4.1, dopodichè in 4.2 e 4.3 verranno mostrati rispettivamente i processi di generazione delle shapes SHACL e di validazione dei vincoli. Infine in 4.4 verranno spiegate le fasi con cui è avvenuta l'integrazione del tutto con Abstat.

### **4.1 Abstat**

Il LOD cloud, con la sua continua crescita, mette a disposizione un'enorme quantità di informazioni per ogni dominio di interesse. L'utente che intende fruire di tali dati, però, può trovarsi disorientato nel capire come siano organizzate le informazioni e di conseguenza come reperire quelle a lui necessarie. Tutto ciò porta ad un dispendio di tempo rilevante nella fase conoscitiva del dataset. A questo scopo nascono tools di supporto come Abstat, estremamente utili per assistere l'utente nella comprensione di grandi datasets.

Abstat<sup>1</sup> (Ontology-Driven Linked Data Summarization with ABstraction and STATistics) è un framework in continuo sviluppo nato da un gruppo di ricercatori di questo Ateneo [21, 27]. Esso ha come scopo quello di fornire all’utente un supporto all’esplorazione e alla comprensione di datasets di grandi dimensioni attraverso la loro profilazione in un summary.

Un *summary* viene calcolato da Abstat a partire da un dataset e dall’ontologia di riferimento e permette all’utente di visualizzarne le caratteristiche in modo semplice e compatto. Attraverso l’esplorazione di un summary l’utente può conoscere in modo rapido le proprietà utilizzate, i tipi delle risorse e le relazioni che intercorrono tra di esse, il tutto corredato da statistiche come la frequenza di un pattern, le relative istanze e i descrittori di cardinalità. La generazione e la consultazione di summary sono disponibili tramite applicazione web<sup>2</sup>.

Il concetto cardine attorno al quale ruota Abstat e che lo distingue dagli altri tools (es: Loupe [19]) è quello della *minimalità*. Un summary di Abstat, infatti, è costituito da una collezione di patterns minimali chiamati *Abstract Knowledge Patterns* (AKPs). Un *AKP* è un pattern nella forma  $\langle \text{subjectType}, \text{predicate}, \text{objectType} \rangle$  che rappresenta le triple  $\langle \text{subject}, \text{predicate}, \text{object} \rangle$  del dataset, tali che *subjectType* sia un tipo minimale<sup>3</sup> di *subject* e che *objectType* sia un tipo minimale di *object*. Allo stesso modo in cui le triple possono essere catalogate come *data type triples* e *object type triples*, gli AKPs si dividono in *Datatype AKPs* e *Object AKPs* a seconda del fatto che *objectType* rappresenti una classe o un literal. Nel caso in cui una risorsa non sia tipizzata nel dataset, essa verrà catalogata da Abstat con il tipo più generico dell’ontologia, cioè *owl:Thing* se si tratta di un concetto oppure *rdfs:Literal* in caso di literal. Sintetizzare un dataset tenendo conto della minimalizzazione permette di avere un summary molto compatto senza perdita di informazioni [27].

Per fornire un quadro completo delle caratteristiche di un dataset, Abstat mostra

---

<sup>1</sup>[bitbucket.org/disco\\_unimib/abstat](https://bitbucket.org/disco_unimib/abstat)

<sup>2</sup><http://backend.abstat.disco.unimib.it/>

<sup>3</sup>il tipo più specifico tra i tipi della risorsa; stabilito tramite le relazioni di *subtype* nell’ontologia di riferimento

per ogni AKP  $\langle C, P, D \rangle$  le seguenti statistiche:

- **occorrenze** di  $C, P$  e  $D$  nel dataset
- **frequenza** del pattern  $\langle C, P, D \rangle$  nel dataset
- **numero di istanze**, ossia quante istanze fanno riferimento al pattern includendo quelle in cui  $C, P$  e  $D$  possono essere dedotte tramite inferenza
- **max/min/avg subjs-obj**, ossia le cardinalità massime/minime/medie per cui diversi soggetti di tipo  $C$  sono associati allo stesso oggetto di tipo  $D$  tramite il predicato  $P$ . Nel corso della tesi ci riferiremo a queste misure come *cardinalità inversa*.
- **max/min/avg subj-objs**, ossia le cardinalità massime/minime/medie per cui distinte entità soggetto di tipo  $C$  sono associate a diversi oggetti di tipo  $D$  tramite il predicato  $P$ . Nel corso della tesi ci riferiremo a queste misure come *cardinalità diretta*.

## 4.2 ShaclGenerator

Lo sviluppo dello ShaclGenerator è stato il punto di partenza di questo lavoro. L’obiettivo iniziale era quello di creare un algoritmo che potesse generare in modo completamente automatico delle shapes SHACL a partire dal summary di un dataset prodotto da Abstat [26]. In particolare i vincoli sulla base dei quali vengono prodotte le shapes sono quelli delle cardinalità massime dirette e inverse.

In questa sezione verranno illustrati i dettagli dell’algoritmo in 4.2.1 e il rapporto tra le shapes prodotte e le informazioni del summary in 4.2.2.

### 4.2.1 Implementazione

Il linguaggio con cui è stato sviluppato lo ShaclGenerator è Java, stesso linguaggio con cui è implementato il framework di Abstat. Data l’assenza di esigenze particolari che potessero far virare l’implementazione su un altro linguaggio, questa scelta implementativa è stata fatta per permettere un’immediata integrazione del generatore di shapes con le componenti dell’architettura di Abstat.

Nella fase iniziale dello sviluppo sono state predisposte due modalità di generazio-

ne delle shapes che prevedevano diversi tipi di raggruppamento dei vincoli. Data una serie di AKPs  $\langle \text{subjectType}, X, Y \rangle$ , nella prima versione era prevista un’aggregazione per *subjectType*. Questo tipo di raggruppamento prevedeva che ogni vincolo relativo ad un AKP con soggetto *subjectType* venisse raccolto in un’unica shape per ogni *predicate X* e per ogni *objectType Y*. Questo tipo di rappresentazione delle shapes, sebbene più compatto, si è poi rivelato inadeguato rispetto alle modalità di validazione previste. A questo scopo è stata sviluppata una seconda modalità di generazione, la quale ha poi sostituito definitivamente la prima, che prevedesse la generazione di una singola shape SHACL per ogni AKP estratto dal summary.

Ora ci concentreremo sulla seconda modalità dell’algoritmo di generazione delle shapes, il quale è implementato con il metodo *generateShacl()* esposto dalla classe *ShaclGenerator*.

### **generateShacl()**

Per poter eseguire l’algoritmo di generazione delle shapes è necessario disporre dei files contenenti le informazioni relative ad un summary<sup>4</sup>. Tra essi, i files utilizzati per formulare i vincoli SHACL di un AKP sono quelli contenenti:

- lista degli *Object AKPs*
- lista dei *Datatype AKPs*
- cardinalità previste per ogni AKP

Una volta resi disponibili tali files, lo ShaclGenerator si costruirà una mappa in cui catalogherà gli AKPs in base al tipo e tutto sarà pronto per eseguire l’algoritmo. L’esecuzione dell’algoritmo consiste nella lettura riga per riga del file contenente le cardinalità previste. Ogni riga di tale file corrisponde ad un AKP seguito dai valori delle cardinalità da imporre come limite massimo. Per ogni AKP iterato viene effettuato un controllo per verificarne il tipo, ovvero se si tratti di un Datatype AKP o di un Object AKP. Una volta estratte le cardinalità previste e verificato il tipo del pattern viene costruita la shape sulla base di queste informazioni. Le shapes vengono definite in formato turtle. La scelta del formato è semplice-

---

<sup>4</sup><https://github.com/christianbernasconi96/shacl/tree/master/pattern>

mente dovuta a motivi di leggibilità, in quanto il validatore per il quale vengono prodotte le shapes supporta questo ed altri formati. Ogni shape viene infine salvata in un file con estensione *.ttl*, al quale viene dato un nome riconducibile univocamente al singolo AKP da cui è stata generata.

#### 4.2.2 Rapporto tra Shapes e AKPs

Come si è potuto capire da quanto detto in precedenza, gli insiemi di shapes e AKPs sono legati da una corrispondenza biunivoca. Dalla spiegazione dell’algoritmo è emerso come i fattori di un AKP che determinano la formulazione dei vincoli di una shape siano il tipo del pattern e le cardinalità previste. A tal proposito ora scenderemo ancor più nel dettaglio, analizzando attraverso un esempio come le informazioni di un AKP vengono utilizzate per produrre una shape.

Prendiamo in analisi il caso dell’AKP *<dbo:Person, dbo:residence, dbo:City>*, il cui tipo è Object AKP e le cui cardinalità massime previste sono 2 per la diretta e 12 per l’inversa. Ora procediamo ad una spiegazione per punti della shape in figura 4.1 generata a partire da questo pattern.

```


dbo:Person a sh:NodeShape;
    sh:targetClass dbo:Person ; #SubjectType
    sh:property [
        sh:message "Vincolo cardinalita diretta (subj-objs) violato";
        sh:path dbo:residence , #Property
        sh:qualifiedValueShape [ sh:class dbo:City ; ]; #ObjectType
        sh:qualifiedMinCount 1 ;
        sh:qualifiedMaxCount 2 , #Max subj-objs
        sh:severity sh:Warning ,
        sh:property [
            sh:path [ sh:inversePath dbo:residence ];
            sh:message "Vincolo cardinalita inversa (subjs-obj) violato";
            sh:class dbo:Person ;
            sh:minCount 1 ;
            sh:maxCount 12 ; #Max subjs-obj
            sh:severity sh:Warning ;
        ];
    ];


```

Figura 4.1 Esempio shape: *<dbo:Person, dbo:residence, dbo:City>*.

- **dichiarazione shape:** come mostra la parte evidenziate in grigio, la dichiarazione di una shape avviene attraverso la definizione di un nodo di tipo *sh:NodeShape*
- **individuazione triple:** la parte della shape che serve per individuare nel dataset tutte le triple che fanno riferimento all'AKP è evidenziata in giallo. Il tipo del soggetto è identificato con la proprietà *sh:targetClass*, il predicato è specificato con *sh:path* e il tipo dell'oggetto attraverso *sh:qualifiedValueShape*. Quest'ultima proprietà serve ad isolare la partizione di interesse in base al tipo dell'oggetto. In altri termini, essa è necessaria per non vincolare tutte le triple con soggetto *dbo:Person* e predicato *dbo:residence* ad avere un oggetto di tipo *dbo:City*. Un'altra osservazione su questa proprietà è relativa al blank node associato. Come si può osservare, nel caso in esempio è specificato il tipo dell'oggetto attraverso *sh:class* in quanto si tratta di un Object AKP. Nel caso di Datatype AKP, invece, la proprietà sarebbe stata quella di *sh:datatype*
- **cardinalità diretta:** la proprietà che stabilisce la cardinalità massima diretta è *sh:qualifiedMaxCount* ed è evidenziata in verde
- **cardinalità inversa:** per definire un vincolo di cardinalità inversa è necessario combinare in un nodo interno le proprietà evidenziate in azzurro. Con *sh:inversePath* si percorre la relazione del pattern in modo inverso, con *sh:class* si specifica il tipo del soggetto e con *sh:maxCount* si impone il limite massimo di cardinalità inversa
- **livello violazione:** la parte evidenziata in fucsia permette di assegnare ai vincoli un diverso livello di violazione. L'assegnazione del livello avviene tramite la proprietà *sh:severity* che nell'esempio è valorizzata con *sh:Warning*. Nel caso in cui nell'ontologia sia specificato un vincolo di cardinalità per il predicato dell'AKP in questione, invece, il livello di violazione assegnato sarà quello di *sh:Violation*.

## 4.3 Validazione Vincoli

Una volta generati i vincoli di cardinalità per ogni AKP occorre uno strumento per la verifica della loro validità nel dataset. In questa sezione verranno illustrate l'idea iniziale (accompagnata dai motivi per cui è stata momentaneamente congelata) in 4.3.1 e la soluzione finale alternativa in 4.3.2.

### 4.3.1 Approccio 1: Shaclex

Per poter individuare le triple di un dataset che violano i vincoli di qualità descritti tramite shapes occorre un validatore SHACL. Tra le alternative di validatori disponibili la scelta è ricaduta su Shaclex, in quanto sembrava essere il più stabile e completo [12]. Questo validatore, infatti, propone tra le features sviluppate e quelle in via di sviluppo tutto ciò che occorre per le nostre esigenze di validazione.

Shaclex<sup>5</sup> è una libreria realizzata in Scala che implementa la validazione di shapes SHACL e ShEx<sup>6</sup>, invocabile quindi tramite jar per programmi Java/Scala o utilizzabile da riga di comando. Tra le varie funzionalità offerte dalla libreria quella di nostro interesse è ovviamente quella della validazione SHACL. È inoltre disponibile una demo online che permette di testare tutte le funzionalità offerte dal validatore<sup>7</sup>.

L'input del processo di validazione è costituito dal dataset da validare e da uno schema (contenente una o più shapes), mentre l'output corrisponde ad un report di validazione. Nel report vengono riportate tutte le violazioni riscontrate durante il processo di validazione e le relative istanze coinvolte.

Shaclex offre svariate opzioni relative ai formati di input ed output (N-Triples, Turtle, RDF/XML, RDF/JSON, N-Quads, ecc.) e diverse modalità di validazione (via url, endpoint, file). Allo stato attuale, però, è emerso come alcune funzionalità necessarie per svolgere il lavoro di cui tratta questa tesi non siano ancora state del tutto sviluppate o non siano completamente stabili. I problemi che sono sorti nel tentativo di utilizzare questa libreria per il nostro scopo sono i seguenti:

- **minimalità**: come abbiamo visto in precedenza, una shape viene generata a partire da un pattern minimale e dalle sue informazioni estratte da Abstat. Di conseguenza, dato un AKP  $\langle C, P, D \rangle$ , i vincoli contenuti nella sua shape vanno applicati solo

---

<sup>5</sup><https://github.com/labra/shaclex>

<sup>6</sup>Shape Expressions, altro linguaggio per definire vincoli; a differenza di SHACL non è considerato uno standard

<sup>7</sup><http://rdfshape.weso.es/>

ed esclusivamente alle triple in cui  $C$  è il tipo minimale del soggetto e  $D$  è il tipo minimale dell’oggetto. Nell’effettuare la validazione, però, Shaclex va ad applicare i vincoli della shape a tutte le triple in cui tra i tipi del soggetto e dell’oggetto sono presenti rispettivamente  $C$  e  $D$  a prescindere dalla presenza di altri sottotipi. Va sottolineato che questo comportamento del validatore è di per sé corretto, ma nel nostro caso potrebbe portare a degli effetti indesiderati. Consideriamo ad esempio le shapes dei pattern  $\langle dbo:Artist, dbo:birthPlace, dbo:Settlement \rangle$  e  $\langle dbo:MusicalArtist, dbo:birthPlace, dbo:Settlement \rangle$ , in cui  $dbo:MusicalArtist$  è sottotipo di  $dbo:Artist$ . Consideriamo inoltre tutte le triple  $\langle subject, predicate, object \rangle$  in cui nel dataset *subject* sia contemporaneamente tipizzato come  $dbo:Artist$  e  $dbo:MusicalArtist$ , *predicate* sia  $dbo:birthPlace$  e *object* tipizzato come  $dbo:Settlement$ . Andando ad effettuare una validazione otterremmo l’applicazione di entrambe le shapes su tutte le triple di questo tipo e ciò porterebbe a produrre dei risultati incoerenti rispetto a quanto estratto da Abstat. In casi come questo potrebbe accadere che un AKP venga erroneamente segnalato come non valido a causa della violazione di vincoli relativi ad una shape di un pattern più generico.

- **owl:Thing / rdfs:Literal:** abbiamo già visto che in assenza di tipizzazione di una risorsa nel dataset Abstat attribuisce i tipi *owl:Thing* o *rdfs:Literal* a seconda del tipo di risorsa. Il problema che sorge è che, non essendo presente una dichiarazione di tipo nel dataset, risulta complicato per il validatore individuare le triple coinvolte nel pattern. La difficoltà è dovuta al fatto che non è possibile specificare una classe target a cui applicare i vincoli. Ciò potrebbe essere realizzabile implementando dei casi specifici in Shaclex, oppure combinando SHACL Core e SHACL SPARQL per individuare questi casi particolari, ma allo stato attuale quest’ultimo non è ancora stato implementato.
- **endpoint:** nel caso di dataset di dimensioni rilevanti è fondamentale effettuare una validazione fornendo come input l’endpoint del dataset anzichè il file contenente le triple. Per la validazione da file, Shaclex carica in memoria le triple per poi verificare su di esse i vincoli. Di conseguenza datasets di grandi dimensioni come DBpedia richiederebbero una quantità di memoria e una capacità di calcolo troppo elevate per poter portare a termine il processo di validazione. Per questo è necessario che il dataset venga validato da endpoint per poter sfruttare i meccanismi del triple store, ma

sfortunatamente anche tale funzionalità è ancora in fase di sviluppo.

- **bugs:** tramite vari test sono stati riscontrati dei bugs che potrebbero portare a comportamenti indesiderati nella validazione di alcuni vincoli. In particolare è stato rilevato un funzionamento anomalo nei casi di validazione di cardinalità inversa su valori di tipo *xmls:date*. In questi casi il validatore conteggiando in modo errato le cardinalità produce in output un report di validazione non veritiero.

Nonostante ciò, il lavoro svolto che precede la fase di validazione è comunque già predisposto ad una futura integrazione con il validatore Shaclex. Nel momento in cui il validatore soddisferà in modo stabile i precedenti punti basterà dare in input le shapes e il dataset via endpoint per ottenere un report di validazione corretto.

In figura 4.2 è rappresentato il workflow che era stato pianificato per questo approccio.

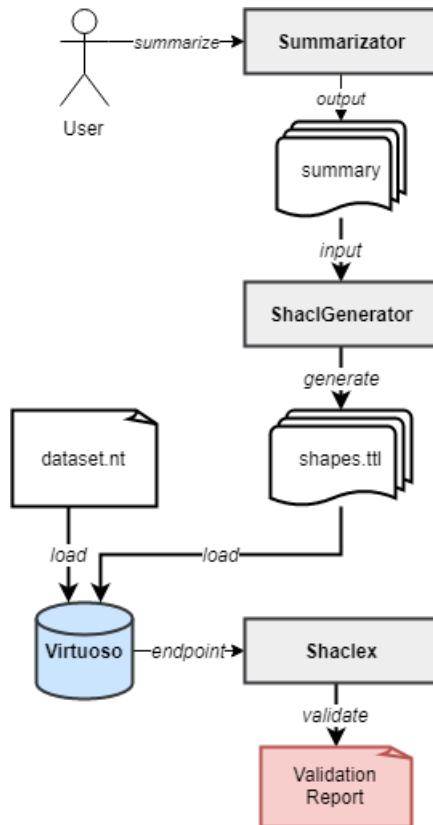


Figura 4.2 Approccio iniziale basato sul validatore Shaclex

### 4.3.2 Approccio 2: API Estrazione Triple

A fronte dei problemi sorti con il validatore Shaclex è stato necessario intraprendere una strada alternativa per la validazione. Per l'individuazione degli outliers presenti nel dataset si è quindi deciso di implementare delle apposite API che vadano ad effettuare la verifica dei vincoli di cardinalità. Tali API, sulla base delle combinazioni di valori dei parametri passati, costruiscono delle SPARQL queries, le eseguono sul dataset tramite endpoint e restituiscono in formato JSON i risultati ottenuti dalla loro esecuzione.

Differentemente da quanto sarebbe avvenuto utilizzando il validatore Shaclex, questo approccio non prevede la generazione di un report di validazione completo del dataset. Mentre prima con l'utilizzo del validatore era prevista una validazione effettuata a priori sull'intero dataset, ora invece con il nuovo approccio viene effettuata un'estrazione on demand degli outliers relativi al singolo AKP specificato.

Al fine di rendere più leggere le esecuzioni delle SPARQL queries e le risposte delle API al client si è pensato di dividere il processo di estrazione in due fasi. La prima fase è gestita con l'API che risponde al nome di *groupedExtractor*, la quale permette di individuare le entità coinvolte nelle violazioni senza necessariamente mostrare i valori incriminati. La seconda fase, realizzata attraverso l'API *singleExtractor*, permette di entrare nel dettaglio delle singole entità estratte nella prima fase individuando tutte le triple coinvolte nella violazione. Il vantaggio che si ha adottando questa strategia a due fasi è che si evita l'esecuzione di queries innestate complesse o di interrogazioni eseguite a catena che occorrerebbero per estrarre il tutto in un'unica chiamata. Ciò porta come ulteriore beneficio anche un alleggerimento dei JSON restituiti al chiamante, rendendoli così più gestibili lato client. Inoltre, considerato che l'estrazione avviene on demand al momento della selezione di un AKP, l'utente può beneficiare di risposte più immediate che migliorano l'esperienza di utilizzo del tool.

## groupedExtractor

L'API *groupedExtractor* viene invocata a partire da un AKP e rappresenta il primo step per l'estrazione degli outliers. In questo passaggio, come suggerito dal nome, l'estrazione delle triple avviene attraverso raggruppamenti. L'API, raccolte le informazioni relative all'AKP, al dataset e all'ontologia, in base ai parametri formula ed esegue una SPARQL query e trasforma i risultati in JSON. Nello specifico, i parametri sulla base dei quali viene costruita la query sono i seguenti:

- **subj**: corrisponde al *subjectType* dell'AKP; è utilizzato nella query per selezionare le triple con soggetto di tipo *subjectType*
- **pred**: corrisponde al *predicate* dell'AKP; è utilizzato nella query per selezionare la proprietà del soggetto
- **obj**: corrisponde all'*objectType* dell'AKP; è utilizzato nella query per selezionare le triple con oggetto di tipo *objectType*
- **dataset**: corrisponde al nome del dataset da cui è stato estratto l'AKP; è utilizzato nella query per individuare il grafo contenente le triple
- **ontology**: corrisponde al nome dell'ontologia su cui si basa il dataset da cui è stato estratto l'AKP; è utilizzato nella query per individuare il grafo dell'ontologia al fine di applicare i filtri sui sottotipi necessari a garantire minimalità
- **akpType**: corrisponde al tipo di AKP, ovvero *Datatype AKP* oppure *Object AKP*; a seconda del suo valore la query filtrerà in modo diverso l'oggetto della tripla
- **predictedCardinality**: corrisponde alla cardinalità prevista per l'AKP; è utilizzato nella query per effettuare il controllo che permette di isolare gli outliers
- **cardinalityType**: corrisponde al tipo di cardinalità che si desidera verificare, ovvero *MaxSubjObjs* (massima cardinalità diretta) o *MaxSubjsObj* (massima cardinalità inversa); è utilizzato per stabilire se nella query sia necessario un raggruppamento per soggetto (cardinalità diretta) o per oggetto (cardinalità inversa)
- **limit**: corrisponde al numero di risultati che si vogliono mostrare; è utilizzato nella query nella clausola *limit*; parametro opzionale
- **offset**: corrisponde all'offset che si vuole dare ai risultati estratti dalla query; è utilizzato nella query nella clausola *offset*; serve per consentire lato client la gestione dei risultati attraverso una paginazione; parametro opzionale

- **sort**: permette di ordinare i risultati dando rilevanza ai peggiori, ovvero quelli che sforano maggiormente il limite della cardinalità prevista; è utilizzato per stabilire se aggiungere alla query un ordinamento decrescente rispetto al conteggio della cardinalità; parametro opzionale
- **showTriples**: permette di aumentare il grado di dettaglio estraendo oltre alle cardinalità anche i valori coinvolti e raggruppandoli in un array; è utilizzato per stabilire se nella query è richiesta la funzione di *GROUP\_CONCAT* per concatenare i valori conteggiati nella cardinalità; parametro opzionale<sup>8</sup>

L’oggetto JSON contenente i risultati della query si differenzia a seconda del tipo di cardinalità richiesta. Nel caso in cui venga richiesta la verifica della cardinalità diretta, allora il JSON conterrà per ogni soggetto estratto il numero di oggetti distinti ad esso associati tramite la proprietà specificata. Viceversa, per la cardinalità inversa il JSON conterrà per ogni oggetto il numero di soggetti distinti a cui è associato tramite la specifica proprietà. Se il parametro *showTriples* è valorizzato a *true*, inoltre, viene aggiunto ad ogni oggetto del JSON un array contenente tutti i valori coinvolti nel conteggio della cardinalità.

### **singleExtractor**

L’API *singleExtractor* viene invocata a partire dai risultati estratti da *groupedExtractor* e costituisce la seconda fase della validazione. In base ai parametri passati all’API con le informazioni sulla risorsa coinvolta, sull’AKP, sul dataset e sull’ontologia, viene costruita una SPARQL query che estrae dal dataset tutte le singole triple coinvolte nella violazione del vincolo di cardinalità. Il risultato dell’esecuzione viene poi convertito in JSON prima di essere restituito al chiamante. Nello specifico, i parametri utilizzati per la costruzione della query sono i seguenti:

- **subj**: corrisponde all’URI della risorsa soggetto in caso di cardinalità diretta oppure all’URI del *subjectType* dell’AKP in caso di cardinalità inversa; è utilizzato nella query per individuare la risorsa soggetto nel caso di cardinalità diretta oppure per filtrare il tipo del soggetto nel caso di cardinalità inversa

---

<sup>8</sup>Si noti che tale parametro è stato utilizzato per lo più in fase di test, mentre nel caso di grandi dataset con elevate cardinalità è sconsigliato il suo utilizzo. Questo perché in presenza di cardinalità elevate vorrebbe dire estrarre per ogni entità tutti i valori aggregati senza poter imporre limitazioni sul loro numero

- **pred**: corrisponde al *predicate* dell'AKP; è utilizzato nella query per selezionare la proprietà del soggetto
- **obj**: corrisponde all'URI della risorsa oggetto in caso di cardinalità inversa oppure all'URI dell'*objectType* dell'AKP in caso di cardinalità diretta; è utilizzato nella query per individuare la risorsa oggetto nel caso di cardinalità inversa oppure per filtrare il tipo dell'oggetto nel caso di cardinalità diretta
- **dataset**: corrisponde al nome del dataset da cui è stato estratto l'AKP; è utilizzato nella query per individuare il grafo contenente le triple
- **ontology**: corrisponde al nome dell'ontologia su cui si basa il dataset da cui è stato estratto l'AKP; è utilizzato nella query per individuare il grafo dell'ontologia al fine di applicare i filtri sui sottotipi necessari a garantire minimalità
- **akpType**: corrisponde al tipo di AKP, ovvero *Datatype AKP* oppure *Object AKP*; a seconda del suo valore la query filtrerà in modo diverso l'oggetto della tripla
- **cardinalityType**: corrisponde al tipo di cardinalità che si desidera verificare, ovvero *MaxSubjObjs* (massima cardinalità diretta) o *MaxSubjsObj* (massima cardinalità inversa); è utilizzato per stabilire il ruolo di *subj* e *obj* nella query
- **limit**: corrisponde al numero di risultati che si vogliono mostrare; è utilizzato nella query nella clausola *limit*; parametro opzionale
- **offset**: corrisponde all'offset che si vuole dare ai risultati estratti dalla query; è utilizzato nella query nella clausola *offset*; serve per consentire al client la gestione dei risultati attraverso una paginazione; parametro opzionale

Il JSON restituito al chiamante sarà composto dalla collezione di triple coinvolte nella violazione del vincolo di cardinalità dell'istanza dell'AKP selezionata. In questo modo l'utente avrà a disposizione tutte le informazioni necessarie per effettuare la valutazione di eventuali errori.

In figura 4.3 è rappresentato il workflow di questo approccio.

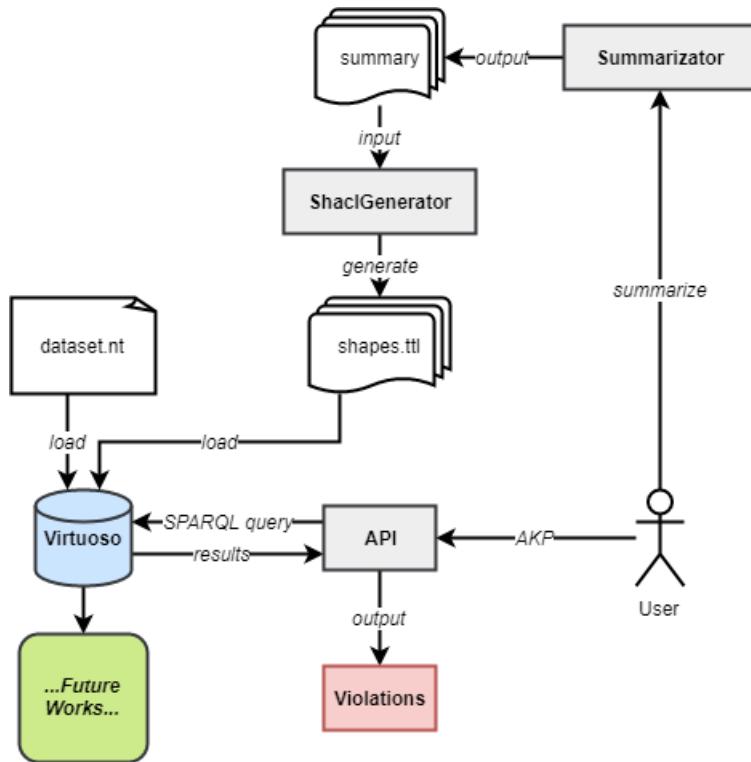


Figura 4.3 Soluzione finale basata sulle API

## 4.4 Integrazione Abstat

Una volta ultimato lo ShaclGenerator e il meccanismo di validazione tramite API lo sviluppo si è spostato sull'integrazione del tutto con Abstat. L'integrazione si divide in più steps, ognuno dei quali va ad interessare le diverse componenti del framework. Prima di illustrare le fasi di integrazione, però, è bene fornire una breve spiegazione dei moduli che costituiscono l'architettura di Abstat.

### 4.4.1 Architettura Abstat

Come si può osservare in figura 4.4, Abstat si divide in 5 moduli [21]:

- **Builder:** modulo che contiene gli algoritmi necessari per effettuare un summary. Esso prende in input un dataset e un'ontologia ed esegue il processo di generazione del summary sulla base di una configurazione specificata dall'utente
- **Storer:** modulo che si occupa del salvataggio dei dati grezzi prodotti dal Builder durante la generazione di un summary

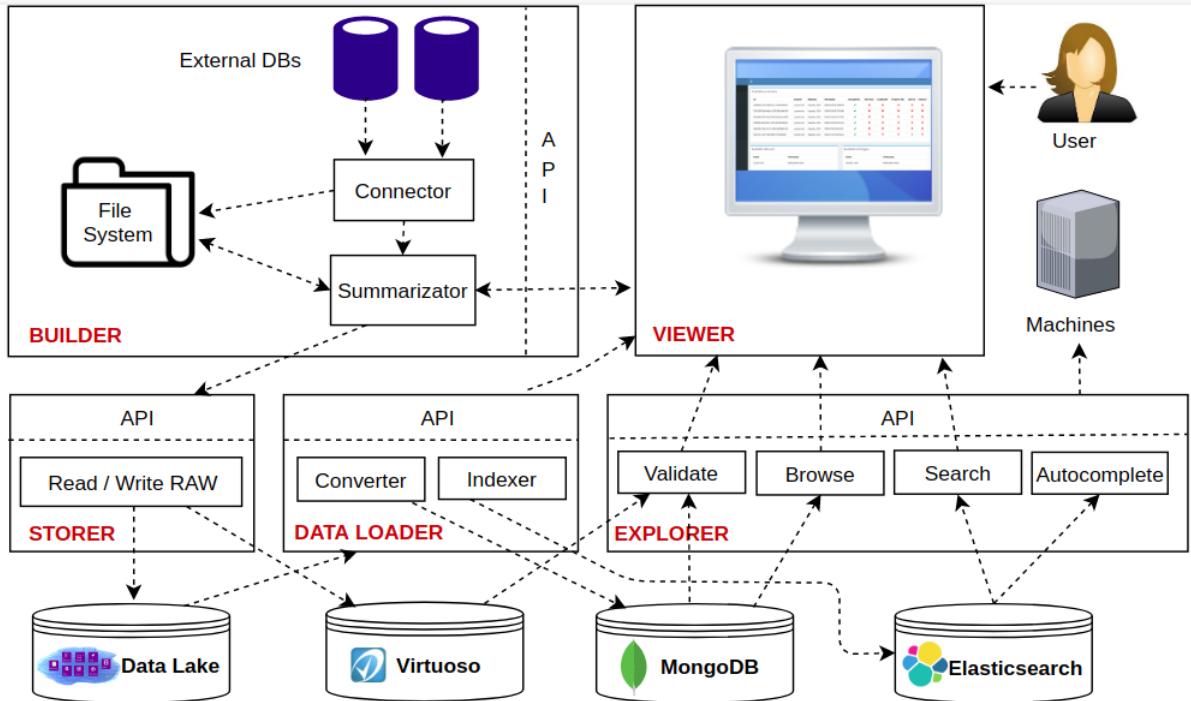


Figura 4.4 Moduli architettura Abstat

- **Loader:** modulo che converte i dati grezzi di un summary per renderli disponibili all'Explorer e li indicizza per favorire la ricerca
- **Explorer:** modulo che gestisce l'esplorazione dei dati prodotti da un summary
- **Viewer:** modulo che costituisce l'interfaccia grafica e che permette all'utente di fruire delle funzionalità offerte dalle altre componenti

I moduli che sono stati interessati da modifiche per l'integrazione sono nell'ordine: Builder, Loader, Explorer e Viewer.

#### 4.4.2 Fasi di Integrazione

L'integrazione con Abstat del lavoro svolto si è compiuta attraverso quattro fasi principali. Facendo sempre riferimento alla figura 4.4, verranno illustrate le modifiche effettuate alle componenti dei moduli di Abstat nel corso di ciascuna fase.

La **prima fase** interessa il modulo del Builder, in particolare nella componente del Summarizer. Come detto precedentemente, il Builder accetta in input una configurazione

da parte dell’utente che viene utilizzata per stabilire cosa calcolare in un summary (es: cardinalità, concetti minimali, proprietà minimali, ecc). Per poter permettere all’utente di predisporre il summary ad una validazione è stato necessario estendere le possibilità di configurazione aggiungendo l’opzione "Cardinality validation (SHACL)". Nel momento in cui il Builder riceve una richiesta di summarization, se tale opzione è selezionata, viene innescata la seguente serie di passaggi:

1. Viene avviato il processo di caricamento<sup>9</sup> del dataset nel Triple Store di Virtuoso
2. Viene avviato il processo di caricamento dell’ontologia nel Triple Store di Virtuoso
3. Vengono lette le cardinalità previste per ogni AKP; nel caso in cui l’ontologia non specifichi nessun vincolo di cardinalità, allo stato attuale la previsione corrisponde a un valore pari al doppio della cardinalità media
4. Viene avviato lo ShaclGenerator fornendo come input i file appena prodotti dalla summarization
5. Viene avviato il processo di caricamento delle shapes nel Triple Store di Virtuoso

La **seconda fase** è quella che coinvolge il modulo del Data Loader. Questa fase consiste nell’integrazione del Converter, il quale si occupa di leggere i dati grezzi prodotti dalla summarization, convertirli in JSON e caricarli in MongoDB<sup>10</sup>. Ai dati prodotti nel summary si aggiungono quelli relativi alle cardinalità previste del punto 3 della prima fase, per cui è stato necessario integrare il caricamento in MongoDB per poter registrare le informazioni sulle previsioni. Nello specifico, per ogni AKP viene effettuato un confronto tra le cardinalità massime effettive e le cardinalità massime previste. Nel caso in cui la cardinalità dell’AKP superi il limite imposto dalla previsione, allora il pattern verrà marcato come non valido attraverso un apposito attributo.

La **terza fase** coinvolge il modulo dell’Explorer ed è quella relativa alla validazione, ossia all’ estrazione delle triple non valide tramite le apposite API. A tale scopo questo modulo è stato integrato con la componente Validate che si occupa di gestire le chiamate alle

---

<sup>9</sup>[https://bitbucket.org/disco\\_unimib/abstat/src/master/scripts/virtuoso.sh](https://bitbucket.org/disco_unimib/abstat/src/master/scripts/virtuoso.sh)  
al fine di rendere possibile il trattamento di dataset con triple malformate è stato modificato il normale processo di caricamento previsto dalle funzioni standard di Virtuoso

<sup>10</sup><https://www.mongodb.com>

API per l'estrazione.

La **quarta fase**, invece, coinvolge il modulo del Viewer. Le modifiche che sono state apportate riguardano le interfacce grafiche per l'interazione dell'utente con Builder, Data Loader ed Explorer. Per quanto riguarda il Builder è stata aggiunta la possibilità di effettuare al Summarizator una richiesta di validazione del dataset. Per il Data Loader sono state aggiunte le informazioni che indicano se con il summary è stata effettuata anche una validazione. Infine per l'Explorer è stato aggiunto alle informazioni sugli AKPs un campo che indica se la cardinalità del pattern supera i limiti previsti. Per ogni AKP marcato come non valido, inoltre, è stata aggiunta la possibilità di verificare nel dettaglio le triple che violano il vincolo di cardinalità.



# **Capitolo 5**

## **Esperimenti**

In questo capitolo verranno presentati gli esperimenti effettuati per mostrare l'utilità e l'efficacia del tool sviluppato. Per prima cosa alla sezione 5.1 troveremo degli esempi di indagine sulla qualità basata sull'individuazione di outliers tramite le API sviluppate. Dopodichè alla sezione 5.2 verrà mostrata una possibile metodologia di analisi e confronto di due versione di un dataset.

### **5.1 Casi di studio DBpedia**

I casi di studio proposti sono stati scelti in modo tale da poter intuire la presenza di violazioni pur non essendo esperti di dominio. Tra i casi studiati si possono distinguere le violazioni certe da quelle “probabili”. Le violazioni certe sono quelle certificate da informazioni sulle cardinalità contenute nell’ontologia di riferimento. Le violazioni probabili, invece, sono quelle causate da una cardinalità effettiva che supera la cardinalità prevista da Abstat, ma che necessitano di conferma da parte di un esperto di dominio. Per ogni test, tra le triple estratte ne sono state prese a campione alcune tra quelle che sforavano maggiormente le cardinalità previste e sono state messe a confronto tra le versioni 2014 e attuale<sup>1</sup> del dataset di DBpedia<sup>2</sup>.

Si noti che le queries sulla versione attuale di DBpedia sono state effettuate tramite

---

<sup>1</sup>test condotti in maggio 2019

<sup>2</sup>la scelta del confronto di due versioni non contigue del dataset è stata fatta per poter individuare più facilmente degli esempi esaustivi sulle differenze riscontrabili

l'apposito endpoint<sup>3</sup>, mentre le interrogazioni alla versione 2014 sono state effettuate tramite un'installazione locale di Virtuoso.

### 5.1.1 Violazione Certa

DBpedia fornisce tramite la sua ontologia informazioni sulle proprietà che possono essere molto utili per migliorare la qualità dei dati. Dato un pattern  $\langle \text{subjectType}, \text{predicate}, \text{objectType} \rangle$ , è possibile verificare se tra i tipi di *predicate* sia presente ***owl:FunctionalProperty***<sup>4</sup>. Questa proprietà fa riferimento alla cardinalità diretta ed implica che per ogni soggetto di tipo *subjectType* può essere associato tramite *predicate* un unico oggetto di tipo *objectType*. In altre parole ciò significa che la cardinalità massima diretta deve essere pari a 1. Proprietà con cardinalità massima diretta pari a 1 sono per esempio *dbo:weight*, *dbo:birthYear*, *dbo:length*, *dbo:height*, ecc.

Considerato che questo vincolo di cardinalità viene dettato dall'ontologia di riferimento del dataset, la dimensione della qualità coinvolta negli esempi proposti è quella di *consistency*.

#### Esempio 1

Pattern: <dbo:Software, dbo:latestReleaseDate, xsd:date>

Cardinalità diretta 2014 - attuale: 5 - 5

SPARQL Query:

```
SELECT ?subj dbo:latestReleaseDate AS ?prop COUNT(DISTINCT(?obj)) AS ?nobjs  
WHERE {  
?subj a dbo:Software.  
?subj dbo:latestReleaseDate ?obj.  
#Minimalità subject
```

---

<sup>3</sup><https://dbpedia.org/sparql>

<sup>4</sup><https://www.w3.org/TR/owl-ref/#FunctionalProperty-def>

```
FILTER NOT EXISTS { ?subj a ?tsubj. ?tsubj rdfs:subClassOf+ dbo:Software. }.
#Datatype object
FILTER(datatype(?obj) = xsd:date).
}
GROUP BY ?subj
HAVING (COUNT(DISTINCT(?obj)) > 1)
ORDER BY DESC(?nobjs)
```

Osservazioni:

nella versione attuale l'entità che sfida maggiormente il vincolo di cardinalità diretta è <http://dbpedia.org/resource/IcedTea> con una cardinalità pari a 5. Andando a verificare lo stesso vincolo sulla versione 2014 si può notare che la medesima entità non compare tra i risultati peggiori in quanto presenta una cardinalità minore, seppur non valida, pari a 2. Di conseguenza questo caso si può catalogare come peggioramento della qualità di quella specifica risorsa nel passaggio di versione.

Un esempio opposto è quello della risorsa [http://dbpedia.org/resource/Serif\\_products](http://dbpedia.org/resource/Serif_products), la quale passa da una cardinalità pari a 2 nella versione 2014 a non violare più il vincolo nella versione attuale. In questo caso c'è stata quindi una correzione che ne ha migliorato la qualità.

## Esempio 2

Pattern: <dbo:Road, dbo:length, xsd:double>

Cardinalità diretta 2014 - attuale: 15 - 69

SPARQL Query:

```
SELECT ?subj dbo:length AS ?prop COUNT(DISTINCT(?obj)) AS ?nobjs
WHERE {
    ?subj a dbo:Road.
    ?subj dbo:length ?obj.
#Minimalità subject
FILTER NOT EXISTS { ?subj a ?tsubj. ?tsubj rdfs:subClassOf+ dbo:Road. }.
#Datatype object
```

```
FILTER(datatype(?obj) = xsd:double).
}
GROUP BY ?subj
HAVING (COUNT(DISTINCT(?obj)) > 1)
ORDER BY DESC(?nobjs)
```

Osservazioni:

Nella versione attuale l'entità con la maggiore cardinalità diretta è [http://dbpedia.org/resource/State\\_highways\\_serving\\_Virginia\\_state\\_institutions](http://dbpedia.org/resource/State_highways_serving_Virginia_state_institutions), con una cardinalità pari a 69. Tale risorsa non esisteva nella versione del 2014 e ha portato la cardinalità diretta massima per questo pattern a salire da 15 a 69 nel passaggio di versione.

L'entità peggiore della versione 2014 è

[http://dbpedia.org/resource/List\\_of\\_secondary\\_highways\\_in\\_Algoma\\_District](http://dbpedia.org/resource/List_of_secondary_highways_in_Algoma_District), la cui cardinalità di 15 però è rimasta invariata.

Andando ad analizzare i dettagli delle due risorse si può notare come in entrambi i casi le risorse vengano utilizzate per rappresentare un insieme di *dbo:Road*, per cui è ipotizzabile che ci sia stato un utilizzo improprio di tale concetto.

### **5.1.2 Violazione Probabile**

I casi proposti in questo paragrafo fanno riferimento a patterns in cui l'ontologia non pone vincoli di cardinalità sul predicato, ma con cardinalità effettiva maggiore rispetto alla previsione di Abstat. La cardinalità prevista, di conseguenza, è stata posta pari al doppio della cardinalità media delle triple rappresentate dal pattern.

In questi casi di studio, data l'assenza di proprietà che impongono delle restrizioni sulle cardinalità, non si rientra più nella dimensione della qualità di *consistency*. Considerando che i limiti delle cardinalità vengono ottenuti tramite una previsione fatta sulla base di dati statistici, la dimensione in cui ricade la verifica degli esempi proposti è quella di *semantic accuracy*.

## Esempio 1

Pattern: <dbo:Book, dbo:numberOfPages, xsd:positiveInteger>

Cardinalità diretta 2014 - attuale: 13 - 21

SPARQL Query:

```
SELECT ?subj dbo:numberOfPages AS ?prop COUNT(DISTINCT(?obj)) AS ?nobjs
WHERE {
    ?subj a dbo:Book.
    ?subj dbo:numberOfPages ?obj.
    #Minimalità subject
    FILTER NOT EXISTS { ?subj a ?tsubj. ?tsubj rdfs:subClassOf+ dbo:Book. }.
    #Datatype object
    FILTER(datatype(?obj) = xsd:positiveInteger).
}
GROUP BY ?subj
HAVING (COUNT(DISTINCT(?obj)) > 2)
ORDER BY DESC(?nobjs)
```

Osservazioni:

In questo esempio i casi peggiori delle due versioni del dataset fanno entrambi riferimento alla stessa risorsa <[http://dbpedia.org/resource/Cthulhu\\_Mythos\\_anthology](http://dbpedia.org/resource/Cthulhu_Mythos_anthology)> , con una cardinalità che peggiora passando da 13 a 21.

Tra gli altri casi estratti a campione non risultano significativi miglioramenti/peggioramenti e restano per la maggior parte invariati.

## Esempio 2

Pattern: <dbo:Country, dbo:capital, dbo:City>

Cardinalità diretta 2014 - attuale: 6 - 6

SPARQL Query:

```
SELECT ?subj dbo:capital AS ?prop COUNT(DISTINCT(?obj)) AS ?nobjs
WHERE {
```

```
?subj a dbo:Country.  
?subj dbo:capital ?obj.  
?obj a dbo:City.  
#Minimalità subject  
FILTER NOT EXISTS { ?subj a ?tsubj. ?tsubj rdfs:subClassOf+ dbo:Country. }.  
#Minimalità object  
FILTER NOT EXISTS { ?obj a ?tobj. ?tobj rdfs:subClassOf+ dbo:City. }.  
}  
GROUP BY ?subj  
HAVING (COUNT(DISTINCT(?obj)) > 2)  
ORDER BY DESC(?nobjs)
```

Osservazioni:

Nonostante la cardinalità diretta massima del pattern a cui si riferisce questo esempio sia pari a 6 per entrambe le versioni, le risorse da cui viene estratta non sono le stesse e permettono di osservare delle modifiche significative avvenute nel passaggio di versione.

Il caso peggiore della versione attuale è quello della risorsa [http://dbpedia.org/resource/Roman\\_Empire](http://dbpedia.org/resource/Roman_Empire). Nella versione 2014 però tale risorsa non era ancora presente nel dataset.

Il caso peggiore della versione 2014, invece, è quello che fa riferimento alla risorsa [http://dbpedia.org/resource/Samma\\_Dynasty](http://dbpedia.org/resource/Samma_Dynasty). Andando a verificare come si sia evoluta questa entità nel passaggio di versione si può osservare come la sua cardinalità diretta sia migliorata significativamente passando da 6 a 1.

### 5.1.3 Violazione Cardinalità Inversa

Gli esempi precedenti fanno tutti riferimento a violazioni della cardinalità diretta. Per quanto riguarda la violazione dei vincoli di cardinalità inversa il discorso è analogo, quindi ci sarà la possibilità di riscontrare violazioni certe e violazioni probabili. Come in precedenza, per verificare il predicato di un pattern abbia vincoli di cardinalità inversa basta andare a controllare che tra i suoi tipi ci sia *owl:InverseFunctionalProperty*<sup>5</sup>. Data una tripla

---

<sup>5</sup><https://www.w3.org/TR/owl-ref/#InverseFunctionalProperty-def>

*<subjectType, predicate, objectType>*, la presenza di questo tipo implica che un oggetto di tipo *objectType* può essere valore della proprietà *predicate* per un solo soggetto di tipo *subjectType*. In altri termini, la cardinalità massima inversa deve essere pari a 1.

### **Esempio**

Pattern: <dbo:Book, dbo:isbn, rdfs:Literal>

Cardinalità diretta 2014 - attuale: 2103 - 5

SPARQL Query:

```
SELECT dbo:isbn AS ?prop ?obj COUNT(DISTINCT(?subj)) AS ?nsubjs
WHERE {
    ?subj a dbo:Book.
    ?subj dbo:isbn ?obj.
    #Minimalità subject
    FILTER NOT EXISTS { ?subj a ?tsubj. ?tsubj rdfs:subClassOf+ dbo:Book.}.
    #Datatype object
    FILTER(isLiteral(?obj)).
}
GROUP BY ?obj
HAVING (COUNT(DISTINCT(?subj)) > 2)
ORDER BY DESC(?nsubjs)
```

Osservazioni:

Il caso di questo esempio è uno dei più significativi per mostrare un miglioramento generale della situazione rispetto alla versione del 2014.

Considerando i casi peggiori del 2014 si può facilmente osservare come facciano tutti riferimento a situazioni particolari in cui in realtà manca l'informazione. Nel caso peggiore si ha che 2103 libri diversi hanno come codice ISBN la stringa “NA” a voler indicare l'assenza di informazione. A questo seguono i casi con valore “N/A”, “n/a”, “ISBN”, ecc.

La cosa interessante di questo esempio è che tutti questi casi sono stati corretti e non sono più presenti nella versione attuale, dove il caso peggiore ha cardinalità 5 e fa riferimento a un ISBN reale. In questo caso c’è stata quindi una pulizia generale di questa porzione del dataset.

### 5.1.4 Discussione dei Risultati

Dall’osservazione delle triple che non hanno superato la validazione sono emerse due cause principali di violazione:

1. Entità che contengono informazioni errate oltre all’informazione corretta
2. Entità che vengono utilizzate erroneamente come concetti, ma che rappresentano una famiglia di entità. *Conseguenza: presenza di informazioni (potenzialmente corrette) non riconducibili alle singole entità.*

Per poter risolvere il problema è necessario l’intervento di un esperto di dominio che, una volta individuate le triple sospette, può effettuare interventi sul dataset. Considerando la migliore delle ipotesi, ossia quella in cui l’esperto di dominio abbia conoscenze e informazioni tali da poter stabilire quali siano le correzioni da effettuare, si presentano due scenari di azione. Nel **caso 1** (esempio: `<dbo:Book, dbo:isbn, rdfs:Literal>`) si tratta semplicemente di correggere/eliminare le triple contenenti l’informazione errata. Nel **caso 2** (esempio: `<dbo:Road, dbo:length, xsd:double>`), invece, la modifica necessaria è più articolata e si potrebbe dividere in due passaggi. Il primo passaggio è quello di separare le entità creandone delle apposite ridistribuendo le rispettive proprietà. Il secondo passaggio è quello di capire se sia anche necessario creare un nuovo concetto superiore a cui associare le nuove entità per poterle raggruppare. Questo scenario però è realizzabile soltanto nel migliore dei casi. Nel momento in cui un dataset dovesse richiedere un data enrichment in cui verrebbero aggiunte triple provenienti da fonti esterne, infatti, non sarebbe più garantito che l’esperto di dominio sia in grado di effettuare valutazioni sul da farsi per correggere eventuali nuove violazioni.

In conclusione, sulla base dei risultati dei casi proposti, è osservabile come nei passaggi di versione non è detto che la qualità dei dati venga migliorata. Se per alcune entità vengono effettuate delle correzioni che portano a dei miglioramenti effettivi, per altre avviene l’esatto contrario con l’aggiunta di nuove informazioni che portano la qualità dei dati per una specifica risorsa a peggiorare. Molti, invece, sono i casi in cui la situazione è rimasta mediamente invariata.

## 5.2 DBpedia 2015-2016: Analisi Generale

Mentre i casi di studio illustrati precedentemente rappresentano degli esempi di indagine nel dettaglio delle triple di alcune specifiche entità, in questa sezione verrà mostrato come poter valutare il dataset in termini più concreti. Partendo dai dataset di DBpedia nelle versioni *2015-10* e *2016-10*, per prima cosa verrà mostrata una possibile metodologia di analisi per la valutazione di diversi aspetti della correttezza del dataset. Successivamente verranno fatte delle considerazioni sulla base dei summaries prodotti da Abstat per le due versioni del dataset. Infine attraverso degli esempi concreti verrà mostrato il calcolo di alcuni possibili indici di correttezza. Tutte le considerazioni sulle violazioni che verranno fatte in seguito si basano sui risultati del lavoro svolto e per questo si riferiscono alle inconsistenze causate da errori di cardinalità.

### 5.2.1 Proposta Metodologia di Analisi

Considerando come punto di partenza i summaries di due versioni di un dataset e sfruttando le API sviluppate per la validazione è possibile proporre una metodologia per valutare degli indici di correttezza. Per poter effettuare una valutazione accurata occorre scendere nei dettagli di tutte le entità di ogni singolo AKP violato, verificarne le triple coinvolte ed effettuare un controllo incrociato tra i due datasets per identificare nuovi errori o eventuali correzioni. La predisposizione di un’analisi di questo tipo, però, è piuttosto articolata ed è prevista negli sviluppi futuri. Per questo motivo ora verrà solamente illustrata ad alto livello una possibile strategia di analisi.

L’idea che sta alla base di questa metodologia consiste nell’analizzare uno ad uno gli AKPs coinvolti nelle functional properties. Per ognuno di essi verrebbe verificata la presenza di violazioni tramite le API e sarebbero poi calcolate delle statistiche (es: numero di entità coinvolte, % triple corrette, ecc.) sulla base degli errori riscontrati. Nel caso in cui un pattern dovesse essere presente anche nella versione precedente del dataset, ne verrebbero calcolate le medesime statistiche e verrebbero effettuati dei controlli incrociati per individuare le differenze (es: errori corretti, errori nuovi, errori ereditati). Infine, una volta conclusa l’analisi di ogni singolo AKP, i risultati intermedi potrebbero essere aggregati e

utilizzati per attribuire degli scores (es: % triple valide, tasso di correzione, ecc.) sul grado di correttezza dell’intero dataset. Segue la rappresentazione in pseudocodice del procedimento illustrato.

---

**Algoritmo 1:** Analisi correttezza

---

**Require:** dataset\_v1, dataset\_v2

```
for all akp ∈ dataset_v2.summary | akp.predicate ∈ FunctionalProperties do
    violations_v2 ⇐ triplesExtractorAPI(dataset_v2,akp)
    akp_stats ⇐ computeAkpStats(violations_v2,akp)
    akps_stats_v2.add(akp_stats)
    if akp ∈ dataset_v1.summary then
        violations_v1 ⇐ triplesExtractorAPI(dataset_v1,akp)
        akp_stats ⇐ computeAkpStats(violations_v1,akp)
        akps_stats_v1.add(akp_stats)
        crossed_check ⇐ crossDatasetErrorsCheck(violations_v1,violations_v2)
        akps_changes.add(crossed_check)
    end if
end for
computeDatasetScores(akps_stats_v1,akps_stats_v2,akps_changes)
```

---

Degli esempi concreti del calcolo di alcuni indici di correttezza saranno mostrati in 5.2.3 attraverso dei casi di studio tratti dalle due versioni di DBpedia.

### 5.2.2 Analisi Summaries

Tramite le statistiche degli AKPs estratte durante la generazione di un summary è possibile fare delle prime considerazioni ad alto livello sull’utilizzo delle functional properties nei due datasets.

vers.	# invalid AKPs	# AKPs	card. media invalid AKPs	# Object AKPs	# Datatype AKPs
2015	349	1247	1.534	0	1247
2016	304	884	1.419	0	884

Tabella 5.1 Summaries delle versioni 2015 e 2016 di DBpedia - confronto AKPs functional properties

La prima informazione che possiamo ricavare dalla tabella 5.1 riguarda il numero di AKPs che presentano violazioni. Nel passaggio di versione si può notare una diminuzione generale del numero di patterns i cui predicati fanno parte delle functional properties: nella versione 2015 ne abbiamo 349/1247 non validi, mentre nella 2016 passano a 304/884. Riguardo questi numeri, è bene ricordare che per considerare un AKP non valido è sufficiente la presenza di una singola violazione a prescindere dalla frequenza di utilizzo. A tal proposito, dato che l'informazione sulla validità non può tener conto del numero di triple coinvolte, questi dati servono soltanto a dare un'indicazione di quanti AKPs all'interno del dataset necessitino di un'indagine per l'individuazione di errori.

Un'altra informazione ricavabile dai summaries può essere quella relativa alla cardinalità media degli AKPs non validi. Il dato mostrato in tabella è un'approssimazione ottenuta tenendo conto delle triple coinvolte in ogni pattern. Esso corrisponde alla media ponderata delle cardinalità medie, in cui il peso di ogni AKP è dato dalla frequenza di utilizzo. Osservando il valore che assume nelle due versioni si può notare un leggero abbassamento. Più il valore si avvicina a 1, più le functional properties sono utilizzate correttamente.

Un'ultima cosa osservabile analizzando i due summaries è che curiosamente non esiste nessun Object AKP riferito ad una functional property. Nel tentativo di separare l'analisi dei Datatype AKPs da quella degli Object AKPs, infatti, è stata riscontrata la totale assenza di patterns del secondo tipo per i predicati funzionali.

### 5.2.3 Esempi Analisi Approfondita

Essendo basate esclusivamente sulle statistiche di un summary, le precedenti considerazioni si limitano a dipingere la situazione da un punto di vista puramente informativo. Chiaramente

per poter effettuare una valutazione più accurata occorrerebbe condurre un’analisi basata su quanto proposto in 5.2.1. Ora ci limiteremo a simulare questo processo tramite degli esempi, mostrando come ricavare da essi degli indici descrittivi sulla qualità. Si noti che il tutto è stato realizzato combinando le statistiche estratte nei summaries e i risultati delle violazioni riscontrate tramite le API illustrate in precedenza.

In tabella 5.2 sono contenuti gli esempi che andremo ad analizzare e le relative statistiche. Prima di procedere all’analisi è bene fornire una spiegazione dei campi della tabella:

- **versione**: versione di DBpedia
- **stato**: stato di validità dell’AKP; *ok* per pattern validi; *ko* per pattern che presentano errori
- **AKP**: pattern minimale *<subjectType, predicate, objectType>*
- **frequency**: numero di triple nel dataset che si identificano nell’AKP
- **cardinalità**: cardinalità massime, medie e minime dell’AKP. In questo caso si tratta di cardinalità dirette in quanto lo studio riguarda i predicati di tipo *FunctionalProperty*
- **soggetti violati**: % soggetti non validi rispetto al totale dei soggetti coinvolti nell’AKP di riferimento; calcolata come  $\frac{\#invalid\_subjs}{\#subjs} * 100$ , con il totale dei soggetti ricavabile come  $\#subjs = akp\_frequency - \#invalid\_triples + \#invalid\_subjs$ .

*Esempio numerico:*

*Dato un AKP che rappresenta le seguenti 10 triple, ossia Totti con 5 date di nascita, Maldini 4 e Del Piero 1, si ha che  $\frac{2}{3}$  dei soggetti non sono validi. Utilizzando la formula, infatti, si otterrà che 66.6% di soggetti non sono validi (Totti e Maldini)*

- **triple valide**: % triple potenzialmente valide rispetto al totale delle triple dell’AKP di riferimento; calcolata come  $\frac{\#valid\_triples}{akp\_frequency} * 100$ , con il totale delle triple potenzialmente valide ricavabile come  $\#valid\_triples = akp\_frequency - \#invalid\_triples + \#invalid\_subjs$ .

*Esempio numerico:*

*Dato l’AKP dell’esempio precedente, si otterrà che  $\frac{3}{10}$  delle triple sono potenzialmente valide (considerato che un soggetto per essere valido debba comparire in una sola tripla, le triple valide sarebbero 1 per Del Piero, 1 tra le 5 di Totti e 1 tra le 4 di*

*Maldini). Utilizzando la formula, infatti, si otterrà una validità del 30% delle triple*

- **errori:** in merito al passaggio di versione, rappresentano nell’ordine il numero di soggetti non validi che vengono corretti, che vengono aggiunti e che restano invariati; ottenuti confrontando i risultati estratti con le API dalle due versioni del dataset

AKP		vers.	stato	freq.	cardinalità (M-a-m) <sup>1</sup>	sogg. violati	triple valide	errori (c-n-e) <sup>2</sup>
1	dbo:Train, dbo:weight, xmls:double	2015	ko	502	4 - 1 - 1	1.01%	98.40%	0 - 1 - 5
		2016	ko	568	4 - 1 - 1	1.07%	98.41%	
2	dbo:Continent, dbo:populationTotal, xmls:nonNegativeInteger	2015	ko	23	6 - 1 - 1	5.5%	78.2%	1 - 0 - 0
		2016	ok	17	1 - 1 - 1	0%	100%	
3	dbo:BusinessPerson, dbo:birthYear, xmls:gYear	2015	ko	144	2 - 1 - 1	0.7%	99.3%	1 - 0 - 0
		2016	ok	89	1 - 1 - 1	0%	100%	
4	dbo:Automobile, dbo:wheelbase, xmls:double	2015	ok	5235	1 - 1 - 1	0%	100%	/ - 9 - /
		2016	ko	5442	5 - 1 - 1	0.16%	99.7%	

<sup>1</sup> M=massima; a=media; m=minima

<sup>2</sup> c=corretti; n=nuovi; e=ereditati

Tabella 5.2 Esempi di studio AKPs delle versioni 2015 e 2016 di DBpedia

Partendo dall’AKP 1 si può notare dai summaries che il numero di triple (frequenza) per questo pattern nel dataset aumenta, mentre le cardinalità restano invariate e di conseguenza l’AKP rimane non valido. Confrontando per entrambe le versioni i soggetti coinvolti nelle violazioni si scopre che gli errori della versione 2015 persistono nella 2016. Inoltre, si riscontra l’aggiunta di un nuovo soggetto non valido. Sebbene il numero di soggetti non validi sia aumentato, esso è irrilevante rispetto al numero di nuove triple corrette aggiunte. Per questo motivo osservando gli indici di correttezza si nota un leggerissimo miglioramento di utilizzo dell’AKP.

L’AKP 2 mostra un caso in cui nel passaggio di versione vengono apportate delle correzioni che lo rendono valido. Analizzando le violazioni si riscontra che l’unico soggetto non valido presente nella versione 2015 scompare nella versione 2016 (come conferma la

diminuzione della frequenza di 6, ossia la cardinalità massima dell’AKP nel 2015), mentre le restanti triple restano invariate garantendo una correttezza del 100%.

L’AKP 3, analogamente al caso precedente, beneficia di correzioni che lo portano ad una validità del 100% delle triple. Osservando la frequenza nelle due versioni si può anche notare come, oltre agli errori rimossi, il numero di triple che fanno riferimento a questo pattern sia diminuito.

Infine abbiamo l’AKP 4, il quale presenta una situazione opposta rispetto alle precedenti in quanto passa da valido a non valido. Sebbene presenti un’alta frequenza di utilizzo rispetto agli altri esempi e che quindi la probabilità di riscontrare triple errate sia più alta, esso risulta privo di errori nella versione 2015. Con l’aggiunta di nuove triple nel passaggio di versione, però, vengono introdotti dei soggetti non validi che provocano un inevitabile abbassamento delle percentuali di correttezza.

Come si è visto dagli esempi presentati, la metodologia proposta permette di assegnare dei livelli di correttezza di un AKP in base alle triple e ai soggetti coinvolti nelle violazioni. Nel momento in cui sarà predisposto un processo automatico di analisi cross-datasets sarà possibile estendere ad ogni livello calcoli di questo tipo. Come già visto attraverso lo pseudocodice di questa metodologia di analisi, aggregando le informazioni raccolte per ogni AKP sarà anche possibile estrarre nuovi indici per attribuire scores sulla correttezza dell’intero dataset in termini di AKPs, entità e triple.

# **Capitolo 6**

## **Conclusioni**

Le conclusioni che si possono trarre dal lavoro svolto per questa tesi si possono dividere tra quelle relative a quanto emerso dagli esperimenti su DBpedia e quelle in merito alle potenzialità del tool sviluppato.

Per quanto riguarda i casi di studio di DBpedia, si è potuto osservare nel concreto come un passaggio di versione non sempre implichi un miglioramento della qualità. Basti pensare che sono stati sufficienti soltanto pochi esempi per poter individuare casi di correzione dei dati, casi di peggioramento e casi in cui la situazione è rimasta invariata. Inoltre è stata proposta e approfondita tramite degli esempi una possibile metodologia di analisi per valutare la correttezza del dataset.

In merito ai cambiamenti del dataset che vengono registrati tra le varie versioni è possibile fare un’ulteriore considerazione. La definizione di vincoli tramite shapes è un buon metodo per poter effettuare controlli di qualità dei dati nei passaggi da una versione all’altra di un dataset. Una volta validate le shapes sulla versione da cui sono state generate basterebbe riapplicarle alle versioni successive per effettuare un rapido controllo che evidenzierebbe le nuove violazioni. Nei passaggi di versione andranno anche ad aggiungersi le shapes generate dai possibili nuovi pattern, le quali di volta in volta andranno ad accrescere l’insieme dei vincoli delle versioni precedenti. In questo modo, di versione in versione, avverrà una progressivo arricchimento delle shapes che renderà più efficace e preciso il processo di validazione.

Un’ultima conclusione che si può trarre da questo lavoro è quella relativa alla facilità con cui è stato possibile individuare le violazioni dei vincoli di cardinalità grazie alle API sviluppate. Attraverso la generazione automatica delle SPARQL queries, infatti, ho risparmiato moltissimo tempo nella ricerca di esempi per i casi di studio che potessero essere significativi. Questo mi ha anche fatto capire come oltre al tempo che consentono di far risparmiare ad un utente SPARQL esperto, l’utilizzo di queste API può permettere davvero a chiunque di effettuare verifiche di questo tipo su qualsiasi dataset. Questo aspetto è fondamentale perché permette ad un esperto di dominio del dataset di effettuare una valutazione degli errori di cardinalità senza che gli sia richiesta alcuna competenza tecnica SPARQL. Inoltre Abstat ora dispone di un’apposita interfaccia che, sfruttando tali funzionalità, offre all’utente la possibilità di navigare tra i patterns ed esplorarne i dettagli delle violazioni in modo semplice ed intuitivo.

# Capitolo 7

## Sviluppi Futuri

Il lavoro svolto si presta ad essere portato avanti attraverso vari sviluppi futuri e costituisce un buon punto di partenza per nuovi progetti di evoluzione di Abstat.

Uno sviluppo molto interessante potrebbe essere realizzato nel momento in cui dovesse essere integrato un validatore SHACL che soddisfi tutte le esigenze di Abstat. Così come era stato proposto nel primo approccio, una volta integrato il validatore le shapes assumerebbero un ruolo centrale nella validazione. Questo aprirebbe la strada a degli sviluppi mirati alla personalizzazione delle shapes da parte dell’utente. L’esperto di dominio potrà quindi modificare i vincoli delle cardinalità generati automaticamente ed eventualmente aggiungerne di nuovi.

Un altro sviluppo potrebbe essere quello relativo all’implementazione di un algoritmo di previsione delle cardinalità massime basato sul machine learning. Attualmente la previsione è stata calcolata come il doppio della media soltanto per dare una scrematura grossolana dei risultati, ma l’introduzione di un nuovo calcolo che restituisca un intervallo di valori all’interno del quale debba cadere la cardinalità è fondamentale. Lo stesso concetto può essere esteso al calcolo delle cardinalità minime. Allo stato attuale la presenza del pattern implica una cardinalità minima pari a 1, ma per alcuni pattern potrebbe essere importante dare informazioni più precise. Ad esempio per pattern come *<dbo:FootballMatch, dbo:team, dbo:SoccerClub>*, sarebbe interessante prevedere che la cardinalità diretta minima debba essere pari a 2.

A beneficio delle shapes e delle previsioni delle cardinalità, un importante sviluppo da tenere in considerazione è quello relativo all’aggregazione di pattern che rappresentano gli stessi concetti. L’aggregazione verrebbe fatta sulla base della proprietà *owl:EquivalentClass*<sup>1</sup>, la quale indica che due classi distinte possono essere considerate equivalenti. Tale aggregazione permetterebbe di avere a disposizione per ogni pattern più dati, permettendo così di effettuare una previsione delle cardinalità più accurata. Un altro vantaggio del raggruppamento di pattern equivalenti è quello di ridurre il numero di AKPs, con conseguente riduzione del numero di shapes e maggior compattezza di un summary.

Infine, per riprendere quanto detto nella sezione degli esperimenti, sarebbe estremamente interessante l’implementazione di un processo di analisi per attribuire al dataset degli indici di correttezza rispetto alle sue versioni. Con la sua realizzazione sarebbe possibile valutare la correttezza del dataset a livello globale, di AKP o delle singole entità. Inoltre sarebbe possibile estrarre informazioni riguardanti l’andamento del tasso di errori nei passaggi di versione.

---

<sup>1</sup><https://www.w3.org/TR/owl-ref/#equivalentClass-def>

# **Elenco delle figure**

2.1	Semantic Web Stack . . . . .	6
2.2	Linked Open Data Cloud 2019 - Focus su DBpedia . . . . .	10
4.1	Esempio shape . . . . .	23
4.2	Approccio iniziale . . . . .	27
4.3	Soluzione finale . . . . .	32
4.4	Architettura Abstat . . . . .	33



# **Elenco delle tabelle**

5.1	Summaries delle versioni 2015 e 2016 di DBpedia - confronto AKPs functional properties . . . . .	47
5.2	Esempi di studio AKPs delle versioni 2015 e 2016 di DBpedia . . . . .	49



# Bibliografia

- [1] Batini, C. (2006). Monica scannapieca data quality concepts, methodologies and techniques.
- [2] Batini, C., Cappiello, C., Francalanci, C., and Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):16.
- [3] Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., et al. (2004). Owl web ontology language reference. *W3C recommendation*, 10(02).
- [4] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.
- [5] Berners-Lee, T. J. (1992). The world-wide web. *Computer networks and ISDN systems*, 25(4-5):454–459.
- [6] Brickley, D. (2004). Rdf vocabulary description language 1.0: Rdf schema. <http://www.w3.org/TR/rdf-schema/>.
- [7] Cimiano, P. (2006). *Ontologies*. Springer.
- [8] Debattista, J., Auer, S., and Lange, C. (2016). Luzzu—a framework for linked data quality assessment. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 124–131. IEEE.
- [9] Debattista, J., Lange, C., Auer, S., and Cortis, D. (2018). Evaluating the quality of the lod cloud: An empirical investigation. *Semantic Web*, (Preprint):1–43.
- [10] Di Noia, T., De Virgilio, R., Di Sciascio, E., and Donini, F. M. (2013). *Semantic Web: Tra ontologie e Open Data*. Apogeo Editore.
- [11] Flemming, A. (2010). Quality characteristics of linked data publishing datasources. *Master's thesis, Humboldt-Universität of Berlin*.
- [12] Gayo, J. E. L., Fernández-Álvarez, D., and García-González, H. (2018). Rdfshape: An RDF playground based on shapes. In *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*.
- [13] Gayo, J. E. L., Prud'Hommeaux, E., Boneva, I., and Kontokostas, D. (2017). Validating rdf data. *Synthesis Lectures on Semantic Web: Theory and Technology*, 7(1):1–328.

- [14] Guéret, C., Groth, P., Stadler, C., and Lehmann, J. (2012). Assessing linked data mappings using network measures. In *Extended semantic web conference*, pages 87–102. Springer.
- [15] Juran, J. and Godfrey, A. B. (1999). Quality handbook. *Republished McGraw-Hill*, pages 173–178.
- [16] Klyne, G. (2004). Resource description framework (rdf): Concepts and abstract syntax. <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.
- [17] Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., and Zaveri, A. (2014). Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758. ACM.
- [18] Mendes, P. N., Mühleisen, H., and Bizer, C. (2012). Sieve: linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. Citeseer.
- [19] Mihindukulasooriya, N., Poveda-Villalón, M., García-Castro, R., and Gómez-Pérez, A. (2015). Loupe-an online tool for inspecting datasets in the linked data cloud. In *International Semantic Web Conference (Posters & Demos)*.
- [20] Mihindukulasooriya, N., Rizzo, G., Troncy, R., Corcho, O., and García-Castro, R. (2016). A two-fold quality assurance approach for dynamic knowledge bases: The 3cixty use case. In *(KNOW@ LOD/CoDeS)@ ESWC*.
- [21] Principe, R. A. A., Spahiu, B., Palmonari, M., Rula, A., De Paoli, F., and Maurino, A. (2018). Abstat 1.0: Compute, manage and share semantic profiles of rdf knowledge graphs. In *European Semantic Web Conference*, pages 170–175. Springer.
- [22] Rashid, M. R. A., Rizzo, G., Torchiano, M., Mihindukulasooriya, N., Corcho, O., and García-Castro, R. (2019). Completeness and consistency analysis for evolving knowledge bases. *Journal of Web Semantics*, 54:48–71.
- [23] Ruan, T., Dong, X., Li, Y., and Wang, H. (2015). Kbmetrics-a multi-purpose tool for measuring quality of linked open data sets. In *International Semantic Web Conference (Posters & Demos)*.
- [24] Ruckhaus, E., Baldizán, O., and Vidal, M.-E. (2013). Analyzing linked data quality with liqueate. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 629–638. Springer.
- [25] Rula, A., Maurino, A., and Batini, C. (2016). Data quality issues in linked open data. In *Data and Information Quality*, pages 87–112. Springer.
- [26] Spahiu, B., Maurino, A., and Palmonari, M. (2018). Towards improving the quality of knowledge graphs with data-driven ontology patterns and shacl. In *ISWC (Best Workshop Papers)*, pages 103–117.
- [27] Spahiu, B., Porrini, R., Palmonari, M., Rula, A., and Maurino, A. (2016). Abstat: ontology-driven linked data summaries with pattern minimalization. In *European Semantic Web Conference*, pages 381–395. Springer.

- [28] Zaveri, A., Kontokostas, D., Sherif, M. A., Bühlmann, L., Morsey, M., Auer, S., and Lehmann, J. (2013). User-driven quality evaluation of dbpedia. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 97–104. ACM.
- [29] Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., and Auer, S. (2016). Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93.

