ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

GESTIONE DI UN DATA LAKE STRUTTURATO ATTRAVERSO IL RICONOSCIMENTO SEMANTICO DEI DATI ACQUISITI

Relazione finale in
BIG DATA

Relatore
Dott. ENRICO GALLINUCCI

Presentata da
ANNA GIULIA LEONI

Terza Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

Big Data

Data Lake

Metadati

Web Semantico

Ontologie

Ai miei genitori

Indice

Introduzione

Negli ultimi anni, a causa della crescita tecnologica e di un numero sempre maggiore di dispositivi connessi ad Internet, si è assistito ad un incremento smisurato nella generazione di dati relativi a qualsiasi ambito che ha dato vita al fenomeno dei Big Data. Le caratteristiche attribuite a questi dati hanno evidenziato la necessità di riorganizzare i processi aziendali e di sviluppare nuove soluzioni, in contrapposizione a quelle tradizionali, che permettessero di cogliere le opportunità dei Big Data e di controllare la complessità derivante dalla gestione di questi dati. Per rispondere a quest'esigenza ci si è rivolti all'utilizzo di architetture scalabili ed efficienti, cioè ad architetture parallele, oggi alla base dei principali framework utilizzati per memorizzare ed analizzare Big Data. Pur utilizzando questi framework, senza definire un'organizzazione interna al Data Lake (repository in cui i dati vengono memorizzati) ed adeguate politiche di governance, abilitate da una solida base di metadati, c'è il rischio che il valore dei dati vada perdendosi, mano a mano che il loro volume aumenta, e che i dati diventino difficilmente reperibili nel Data Lake.

Il progetto di tesi è volto a predisporre i sistemi aziendali ad una corretta gestione futura dei Big Data e si colloca all'interno di un processo di reingegnerizzazione molto più ampio, che prevede l'adozione di un Data Lake come unico punto d'ingresso per i dati aziendali, memorizzati nel loro formato originale. L'obiettivo principale del progetto di tesi è sviluppare un approccio estendibile e parametrizzabile che permetta di indicizzare, nella maniera più automatica possibile, i singoli file presenti nel Data Lake sulla base del loro tipo di evento tramite l'estrazione di metadati strutturali, di processo e semantici, che consentano e facilitino il recupero dei file quando rilevanti rispetto a determinate interrogazioni. I file che il caso di studio prende in esame sono file XML che fanno riferimento ad eventi generati in ambito sanitario, con un

focus particolare sugli eventi relativi all'AD/RSA. Per portare a termine quest'obiettivo viene definita un'architettura per Data Lake ed un modello per l'archiviazione di metadati che sia scalabile e flessibile e possa fungere da base per tutti i sistemi aziendali: un'ontologia. La possibilità di tarare i parametri del processo di estrazione e visualizzare i risultati ottenuti è resa disponibile tramite un'interfaccia web.

La tesi è suddivisa in tre capitoli. Il primo capitolo fornisce una panoramica sui concetti, lo stato attuale della ricerca e le tecnologie esistenti correlate al mondo dei Big Data e del Web Semantico con un focus specifico sull'importanza assunta dalla definizione di un'architettura per Data Lake e da una corretta gestione dei metadati. Il secondo capitolo descrive il contesto ed il caso di studio alla base del progetto di tesi, l'obiettivo che si pone e l'approccio proposto per realizzarlo. Il terzo capitolo illustra le tecnologie utilizzate per implementare l'approccio proposto, i test eseguiti per valutarne efficacia e performance ed il risultato ottenuto. Seguono conclusioni, sviluppi futuri e ringraziamenti.

Capitolo 1

Stato dell'arte

Il capitolo è dedicato a fornire una panoramica relativa a concetti, stato attuale della ricerca e tecnologie esistenti correlate al lavoro di tesi. Gli argomenti trattati sono connessi al mondo dei Big Data e del Web Semantico con un focus particolare sull'importanza assunta dalla definizione di un'architettura per Data Lake e da una corretta gestione dei metadati.

1.1 Big Data

Le definizioni esistenti in letteratura per descrivere il concetto di *Big Data* sono molteplici. Il McKinsey Global Institute si riferisce ai Big Data come a "dataset la cui dimensione va oltre la capacità dei software comunemente utilizzati per catturare, memorizzare, gestire ed analizzare dati" [1]. Da questa definizione è evidente che il confine di quando un dataset diventa Big è labile: definire i limiti, sia hardware che software, di un'elaborazione tradizionale è relativo, perché può variare da settore a settore a seconda di software disponibili e tipiche dimensioni dei dataset, e cambia nel tempo, perché con l'avanzare della tecnologia aumenta potenzialmente anche la dimensione dei dataset qualificabili come Big. Il Gartner Group, invece, si riferisce ai Big Data come ad un "patrimonio informativo ad elevato volume, elevata velocità e varietà che richiede forme innovative ed efficienti di elaborazione in modo da migliorare la comprensione delle informazioni ed il processo decisionale" [2]. Da questa definizione emergono, in una sola frase, sfide ed opportunità dei Big Data. Un modo diverso di definire i Big Data è dato dalle cosiddette "3V",

ossia un insieme di caratteristiche che vengono generalmente riconosciute ed attribuite oggi ai Big Data. Tali caratteristiche sono riportate di seguito [3].

Volume Il volume fa riferimento alla quantità, all'ammontare dei dati generati. L'ordine di grandezza dei dati prodotti oggi va dai Terabytes (1TB = 10^{12}B) ai Zettabytes (1PB = 10^{21}B). Questo incremento smisurato è conseguenza di un mondo sempre più connesso in cui proliferano continuamente nuove sorgenti dati che spaziano tra Internet Of Things, Social Media ecc. Si stima che entro il 2025 si produrranno 163ZB di dati, cioè 10 volte tanto i dati che si producevano nel 2016 [4].

Velocità La velocità fa riferimento alla frequenza alla quale i dati vengono generati ed analizzati (periodica, real-time) ma anche a variazioni di velocità ed alla combinazione di serie di dati che arrivano con velocità diverse. Gestire in modo opportuno l'ingestion e la suddivisione del carico di lavoro di questi flussi dati, permette alle aziende di soddisfare il loro bisogno di ottenere risposte in tempi brevi per essere competitive sul mercato.

Varietà La varietà fa riferimento all'eterogeneità nella struttura dei dati. Si possono distinguere dati strutturati, semi-strutturati e non strutturati. I dati strutturati hanno un modello/schema che li descrive e sono i più semplici da maneggiare in fase di analisi; sono presenti, ad esempio, in fogli di calcolo e database relazionali. I dati semi-strutturati hanno una struttura auto-descrittiva, cioè sono caratterizzati da tag/chiavi che ne descrivono gli elementi, e possono essere parzialmente organizzati come modelli gerarchici; XML e JSON ne sono un esempio. Tutti i dati che non hanno una struttura predefinita ricadono nella classificazione di dati non strutturati; sono tipicamente dati di natura testuale ma anche immagini, audio e video che non possono essere utilizzati nell'immediato per eseguire analisi. Si stima che fino all'80% dei dati aziendali siano non strutturati e che in questi che risieda il maggior valore.

Nel tempo sono state proposte altre caratteristiche che prendono in considerazione aspetti che non possono essere ignorati. Tra queste:

Veracità La veracità, o veridicità, fa riferimento principalmente all'inaffidabilità intrinseca di alcune sorgenti dati (Social Media), ma anche a problemi di qualità e rumore. Questa caratteristica, coniata da IBM, vuole evidenziare come sia necessario fare i conti costantemente con dati incerti quando si tratta di Big Data. In questi casi la fiducia che l'azienda ripone nei dati influenza il modo in cui li utilizzerà nel processo decisionale.

Valore Il valore fa riferimento alla rilevanza delle informazioni che è possibile estrarre dai dati. Sulla base della definizione di Oracle, i Big Data sono spesso caratterizzati da "low value density". Questo vuol dire che i dati ricevuti in forma originale (raw data) hanno di solito un valore basso rispetto al loro volume. Un elevato valore può essere ottenuto analizzando grandi volumi di tali dati. È possibile infatti applicare su questi modelli/algoritmi sofisticati che danno risultati più accurati che in passato permettendo di scoprire informazioni/trend che non sarebbero emersi altrimenti e che possono essere usati per prendere decisioni di business.

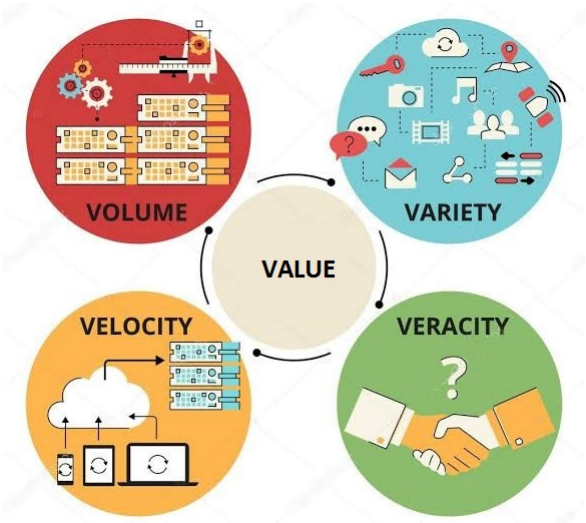

Figura 1.1: "V" dei Big Data, adattato da [5]

Per essere considerati Big Data, i dati devono possedere un sottoinsieme delle caratteristiche presentate (Figura 1.1).

1.1.1 Sfide ed Opportunità

Da quanto è emerso fino ad ora è evidente che i Big Data abbiano un grande potenziale da cui, però, derivano altrettante sfide da affrontare [6, 7] (Figura 1.2).

Sfide legate ai dati Sfide direttamente collegate alle caratteristiche dei Big Data sopra citate.

Sfide legate al ciclo di vita dei dati Sfide incontrate nella gestione del ciclo di vita dei dati, processo iterativo che coinvolge diverse fasi.

- Acquisizione ed archiviazione: comprende l'acquisizione dei dati dalle sorgenti e la loro archiviazione per la generazione di valore futuro.
- Estrazione e pulizia: comprende la trasformazione e la pulizia dei dati, operate per ridurne l'inaffidabilità.
- Integrazione ed aggregazione: comprende l'integrazione e l'aggregazione dei dati provenienti da sorgenti diverse tramite l'utilizzo di tecniche tradizionali, basate sullo schema dei dati, e semantiche.
- Analisi e creazione del modello: comprende l'esecuzione di analisi più o meno semplici sui dati che possono portare alla creazione di un modello che li descriva; a seconda dell'esigenza è possibile utilizzare due diversi approcci:
  - Batch: il focus è sul volume dei dati, il dataset analizzato è statico, processato in un unico blocco e la computazione può richiedere molto tempo ma fornisce un risultato preciso.
  - Near real-time (streaming): il focus è sulla velocità dei dati, il dataset analizzato è un flusso dati dinamico, processato una finestra alla volta, e la computazione fornisce un risultato approssimato ma in tempi brevi.

  Per eseguire queste operazioni non si può prescindere dall'avere competenze adeguate ed una grande capacità di archiviazione e calcolo.

- Interpretazione: comprende rendere risultati/modelli ottenuti disponibili a coloro che si occuperanno di prendere le decisioni di business; questa operazione può comportare l'utilizzo di strumenti e tecniche visuali per meglio comprendere la conoscenza risultante.

Sfide legate alla gestione dei dati Sfide incontrate nell'accedere, nel gestire e nel governare i dati.

- Sicurezza e privacy: l'integrazione delle tecnologie necessarie a gestire il ciclo di vita dei Big Data porta maggiori rischi per la sicurezza dei dati e problemi di privacy che si possono verificare nel momento il cui si riesce ad inferire nuova conoscenza.

- Costi: l'acquisto delle risorse di archiviazione e di calcolo necessarie dipende dal budget aziendale ed impatta l'efficienza della soluzione Big Data che si intende implementare.

- Data governance: è necessario stabilire un insieme di strategie, processi, regole da definire a monte dell'utilizzo dei dati per poterne garantire la governabilità e la qualità nel tempo.

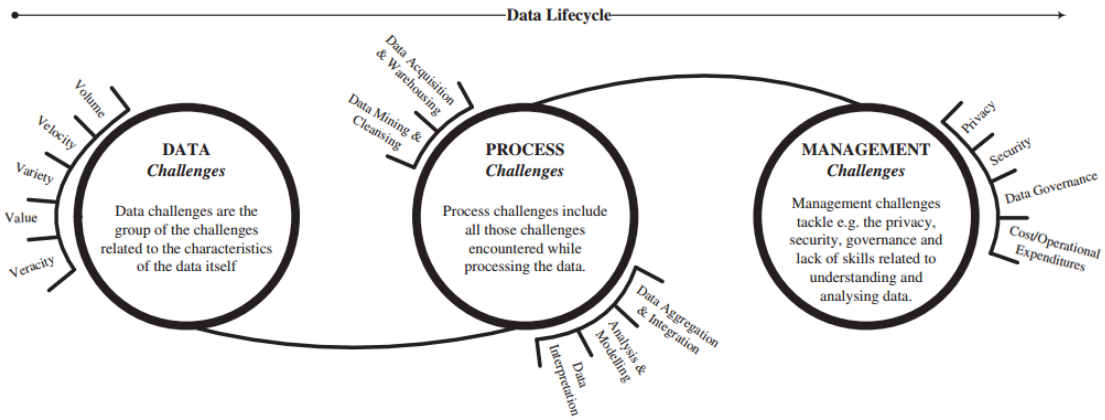

Figura 1.2: Sfide dei Big Data, adattato da [7]

Riuscire a gestire in maniera adeguata tutti questi aspetti è importante perché permette alle aziende di porsi in una posizione avvantaggiata rispetto

ai concorrenti che, non sfruttando i Big Data, non riescono ad avere la stessa comprensione del mercato e dei dati di business. Per farlo, però, è necessario abbandonare le tecnologie ed i framework tradizionali, fondati su un approccio di tipo relazionale, e muoversi verso architetture più scalabili ed efficienti. Per rispondere a quest'esigenza ci si è rivolti all'utilizzo di architetture parallele.

1.1.2 Architettura Big Data

L'architettura comunemente utilizzata in ambito Big Data è il *cluster computing*. Il cluster computing rappresenta una soluzione economica per scalare orizzontalmente. Il parallelismo è ottenuto collegando in rete più macchine, da poche a centinaia, ognuna delle quali offre spazio di archiviazione e capacità di calcolo locali. L'hardware utilizzato per comporre il cluster è *commodity hardware*. Al contrario di come si può pensare il termine non è sinonimo di hardware di scarsa qualità ma viene usato per indicare hardware disponibile ad un prezzo accessibile e con caratteristiche simili da diversi produttori.

Apache Hadoop Apache Hadoop [8] è un framework open-source che permette di eseguire computazioni distribuite di grandi dataset in cluster di computer utilizzando semplici modelli di programmazione. Piuttosto che affidarsi all'hardware per garantire alta disponibilità ed affidabilità, la libreria stessa è progettata per rilevare e gestire tutti i problemi a livello applicativo garantendo la disponibilità del servizio. L'analista si deve preoccupare "solo" di progettare la sua analisi che verrà eseguita all'interno di un ambiente runtime già predisposto interfacciandosi al cluster come se fosse un unica macchina.

Il framework si compone di quattro moduli di base e di una serie di progetti collegati volti a fornire ulteriori servizi. I moduli di base sono:

- Common: insieme di librerie ed utility a supporto di tutti gli altri moduli.
- HDFS (Hadoop Distributed File System): file system distribuito che fornisce accesso ad alta velocità ai dati delle applicazioni; viene approfondito in seguito, sempre all'interno di questa sezione.
- YARN (Yet Another Resource Negotiator): gestore delle risorse del cluster; si occupa di soddisfare le richieste di risorse, dello scheduling e

dell'allocazione di processi; opera secondo un pattern master-slave fornendo i suoi servizi principali tramite due tipi di demoni: il *resource manager*, il master, e i *node manager*, gli slave.

- MapReduce: modello/paradigma di programmazione basato su una struttura dati di coppie chiave-valore che viene utilizzato per eseguire analisi parallele di grandi dataset; si appoggia su YARN e prevede l'esecuzione di due operazioni: Map e Reduce; l'operazione di Map è ampiamente parallelizzabile, avviene in maniera isolata su ogni record del dataset in input e consiste nell'applicazione di una funzione *f*, che genera una lista di coppie chiave-valore; l'operazione di Reduce consiste nell'applicazione di una funzione *g* che restituisce in output sempre una lista di coppie chiave-valore aggregata per chiave; prima di poter essere eseguita è necessario raccogliere in un unico punto le coppie aventi stessa chiave.

HDFS HDFS è un file system distribuito progettato per la memorizzazione di file di grandi dimensioni, tipicamente dai GB ai TB. Il tipo di accesso che utilizza è chiamato *streaming data access* e comporta un modello di accesso ai file *write-once-read-many*. Normalmente un dataset, una volta che viene memorizzato su HDFS, viene poi acceduto nel tempo per eseguire diversi tipi di analisi che coinvolgeranno interamente, o in gran parte, il dataset: per questo motivo l'accento è posto sulla velocità di lettura più che sulla bassa latenza di accesso ai file. HDFS è in esecuzione su un cluster di commodity hardware in cui il failure dell'hardware è la norma: se si verificano dei problemi HDFS stesso deve individuarli, gestirli in modo automatico ed in maniera trasparente nei confronti dell'utente.

L'unità minima di dato che viene letto/scritto su disco prende il nome di *blocco*. Il blocco ha una dimensione che può variare da 64MB a 1GB (128MB è la dimensione di default) tuttavia, se un file ha dimensione minore, occupa su disco solo lo spazio necessario. Lavorare per blocchi consente di spezzare i grandi file su più dischi e gestire la loro replicazione in maniera più semplice. I blocchi HDFS sono volutamente grandi per garantire una velocità in lettura elevata e ridurre al minimo il costo della ricerca di tutti i blocchi che compongono un file.

I nodi in un cluster HDFS sono di due tipi ed operano secondo un pattern master-slave.

- Il *namenode* (NN) è il master e viene interrogato per sapere dove si trovano i file perché gestisce il namespace del file system; mantiene in modo persistente l'albero e i metadati relativi a file e cartelle che memorizza mentre la mappa di distribuzione dei blocchi, cioè la posizione sui datanode di ciascun blocco per ogni file, viene mantenuta in memoria.

- I *datanodes* (DNs) sono gli slave e gestiscono la memorizzazione ed il recupero effettivo dei blocchi che compongono il file; periodicamente comunicano al NN la lista dei blocchi che memorizzano.

All'interno del cluster si può decidere di implementare un meccanismo di federazione configurando NNs aggiuntivi con un loro namespace in modo da migliorare la scalabilità, perché la dimensione della memoria dei NNs limita il numero di file e blocchi indirizzabili, la disponibilità, perché il failure di uno dei NN non influisce sulla disponibilità dei namespace gestiti dagli altri NNs e la sicurezza, perché ogni namespace è indipendente dagli altri.

HDFS non è la soluzione migliore quando:

- C'è necessità di accedere ai dati con bassa latenza: se si presenta questo bisogno è meglio optare per HBase, progetto collegato ad Hadoop ed implementato sopra HDFS, sviluppato per ottimizzare l'accesso casuale ed in tempo reale ai dati.

- I file da memorizzare sono molto piccoli: il NN può andare in saturazione.

1.2 Data Lake

Il concetto di *Data Lake*, coniato da James Dixon nel 2010, è emerso nella recente letteratura come modo popolare per risolvere alcune delle sfide introdotte dai Big Data [9, 10]. Definito come un "sistema o repository di dati salvati nel loro formato originale" viene spesso messo in relazione al concetto di Data Warehouse per meglio evidenziarne le caratteristiche che lo rendono adatto, al contrario di quest'ultimo, al ruolo di repository per Big Data che gli è stato attribuito (Tabella 1.1).

Data Lake e Data Warehouse condividono l'essere un repository per i dati. Il Data Lake archivia i dati nella loro forma grezza. I dati vengono mantenuti così come arrivano per evitare di perdere informazioni che ora non si ritengono interessanti ma potrebbero rivelarsi preziose in futuro per rispondere ad esigenze che al momento non sono ancora emerse. Essendo progettato su piattaforme distribuite come Hadoop ha una capacità di storage potenzialmente infinita che può essere aumentata al bisogno ad un costo relativamente basso. I dati archiviati nel Data Warehouse, al contrario, rispondono ad esigenze di business ben precise e devono sposare il modello scelto per rappresentare i dati rendendolo inadatto all'archiviazione dei Big Data.

Per il motivo di cui sopra si può dire che il Data Lake abbia una barriera d'ingresso per i dati molto bassa. Segue infatti l'approccio schema-on-read: la struttura dei dati viene definita solo nel momento in cui questi vengono utilizzati, non quando vengono salvati nel repository. In questo modo è possibile archiviare velocemente grandi quantità di dati senza pagare il costo e la complessità dei procedimenti di ETL da attivare per poterli salvare nel Data Warehouse che segue un approccio schema-on-write. Cambiare il modello su cui si basa il Data Warehouse inoltre, può essere molto costoso.

La varietà di dati contenuta nel Data Lake può essere analizzata utilizzando la tecnica che si ritiene più adatta per rispondere a specifiche esigenze ma anche per fare analisi esplorative. Questo richiede lo sviluppo di pipeline di analisi più o meno complesse da parte dei data scientists. I dati contenuti nel Data Warehouse, invece, sono dati strutturati, ottimizzati per l'accesso tramite linguaggio SQL e da parte di strumenti di Business Intelligence. Fare analisi esplorative a partire dal Data Warehouse è escluso visto che, al contrario, viene spesso associato al concetto di information silo.

	DATA WAREHOUSE	**DATA LAKE**
DATI	Puliti Strutturati	Grezzi Strutturati, semi-strutturati, non strutturati
SCHEMA	Schema-on-write	Schema-on-read
MEMORIZZAZIONE & COSTO	Grandi volumi di dati a costi moderati	Volumi di dati estremi a basso costo
AGILITA'	Configurazione fissa, pochi cambiamenti	Flessibile, cambiamenti anche radicali
ACCESSO	SQL e strumenti di BI	Programmi ad-hoc
UTENTI	Professionisti del business	Data scientists
VANTAGGI	Maturo Facile da usare Performance, sicurezza ed integrazione garantite	Meno maturo e meno facile da usare Possibilità di eseguire analisi complesse ed analisi esplorative Altamente scalabile

Tabella 1.1: Data Warehouse vs. Data Lake

Adottare una soluzione non esclude l'altra: entrambe possono essere utilizzate ed integrate per rispondere alle diverse necessità aziendali. In ogni caso, il solo utilizzo del Data Lake non rappresenta una soluzione a tutti i problemi dei Big Data. Senza un'adeguata architettura ed adeguate politiche di governance, abilitate da una solida base di metadati, i Data Lake spesso non riescono a mantenere le promesse iniziali.

1.2.1 Architettura

Non esiste un'architettura per Data Lake valida universalmente ma esistono un insieme di best practice generalmente riconosciute da adottare per dargli un'organizzazione sensata. Il Data Lake dovrebbe essere diviso in *zone*, o aree, che servono per funzioni specifiche. Tale separazione può essere fisica, se vengono create in server o cluster dedicati, o logica, se il Data Lake viene diviso in cartelle con determinati privilegi d'accesso. Ogni azienda che sviluppa una soluzione di Data Lake tende a dare un nome diverso alle zone anche se lo scopo per cui vengono create è lo stesso. Di seguito (Figura 1.3) viene presentata un'architettura di riferimento, che può essere semplificata o complicata a piacere sottraendo od aggiungendo ulteriori zone, con la denominazione delle aree proposta da Zaloni [11].

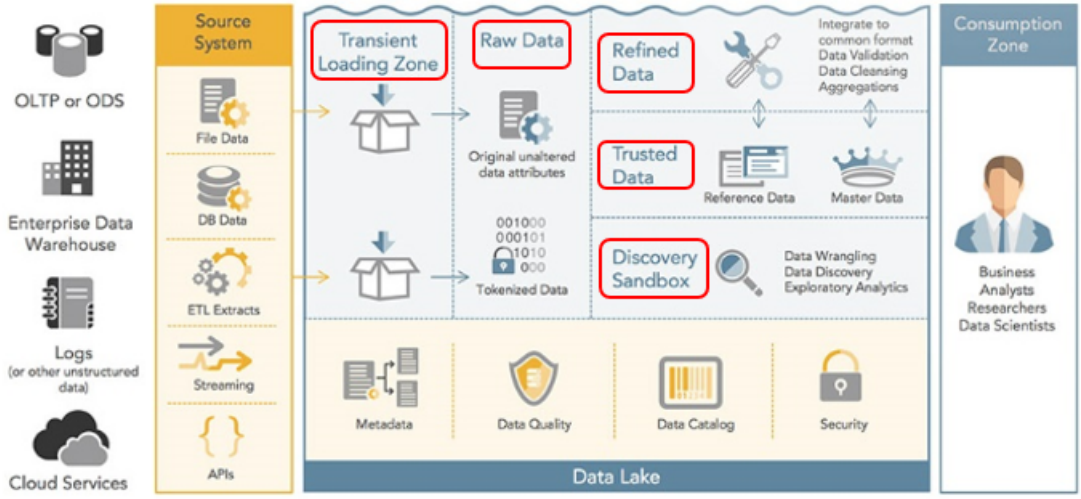

Figura 1.3: Esempio di architettura di riferimento per un Data Lake [11]

La **Transient zone** rappresenta il punto di ingresso dei dati nel Data Lake. È prevista generalmente nelle soluzioni rivolte ad aziende che operano in settori altamente regolamentati come zona di passaggio in cui i dati possono essere tokenizzati o mascherati per proteggere, ad esempio, dati sensibili.

Se non è presente la Transient zone i dati arrivano nel Data Lake nella **Raw zone**. Questa zona contiene i dati nel loro formato originale o così come arrivano dalla Transient zone.

Nella **Refined zone** è possibile memorizzare dataset trasformati a partire dai dati grezzi che potrebbero essere necessari per diversi casi d'uso. In questa zona si possono trovare dati "semilavorati" che, a seguito di operazioni di pulizia, aggregazione e validazione, sono stati ricondotti ad una struttura comune, ad esempio tabellare, prima di essere salvati.

La **Trusted zone** contiene master e reference data. I dati di questa zona sono preziosi perché il loro valore è riconosciuto e condiviso all'interno dell'azienda. Sono inoltre ampiamente utilizzati dai tool di reportistica e data visualization che rendono disponibili i risultati delle analisi in maniera comprensibile, ed eventualmente visuale, ai soggetti aziendali e, per questo motivo, dovrebbe essere ottimizzata per la lettura.

La **Sandbox** è un'area a cui i data scientists possono attingere per sperimentare con i dati, allo scopo di far emergere il loro valore. È possibile osservare i dati per comprendere le loro caratteristiche ed estrarre actiona-

ble patterns, cioè informazioni che l'azienda può usare a suo favore in fase decisionale , ma anche trasformare i dati da grezzi in un formato idoneo ad essere utilizzato dall'algoritmo che si intende applicare . La Sandbox può essere alimentata da tutte le altre aree.

1.2.2 Metadata management

Strutturare in maniera sensata il Data Lake (Figura 1.4) non basta per gestire con successo i dati al suo interno perché, senza una solida base di metadati, il rischio che si corre è che si trasformi in un *Data Swamp* (Figura 1.5). Il Data Swamp è un Data Lake in cui, a causa di un sistema di metadata management inesistente o non funzionale, diventa difficile distinguere i dati compromettendone l'usabilità. Il loro valore va perdendosi mano a mano che il volume aumenta perché, nonostante la comune infrastruttura sottostante, la mancanza di una modalità condivisa per etichettarli rende complicato recuperare i dati rilevanti rispetto alle analisi da effettuare. Senza riuscire ad accedere ai dati è impossibile integrare e condividere le informazioni perciò, quello che si ottiene, sono nuovi information silo [12].

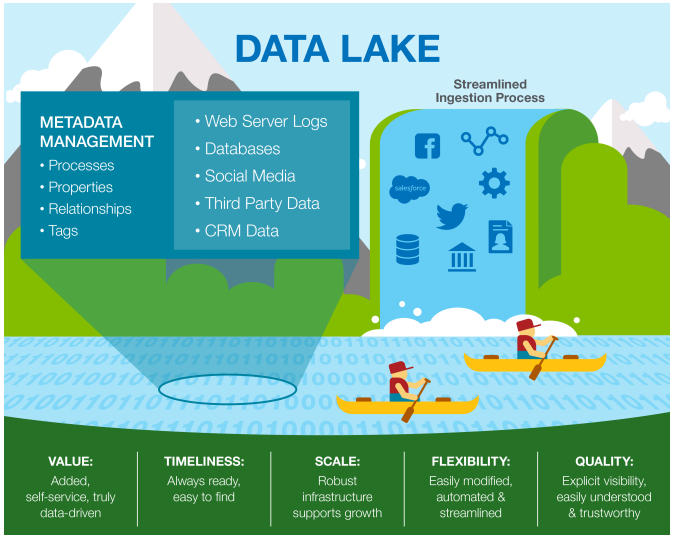

Figura 1.4: Caratteristiche del Data Lake [12]

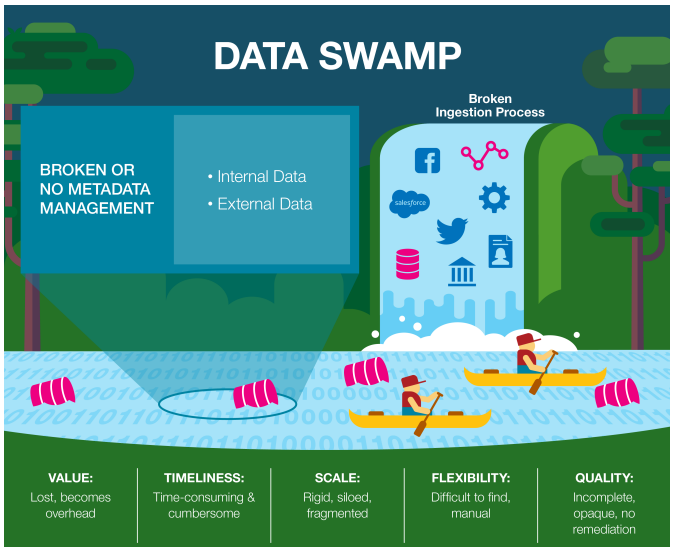

Figura 1.5: Caratteristiche del Data Swamp [12]

Metadati I *metadati* sono dati che descrivono altri dati (content data). In questo contesto assumono un ruolo importante proprio perché i dati non vengono caricati in un repository integrato con uno schema unico.

Sono un elemento chiave per:

- Garantire visibilità ai dati fornendo un catalogo comune che li renda facilmente accessibili specificando anche criteri di ricerca.
- Garantire che le analisi avvengano su dati che hanno un certo livello di qualità e quindi possano essere considerate affidabili.
- Determinare se un dataset è coerente con il contesto dell'analisi che si vuole effettuare.
- Governare il ciclo di vita dei dati, la loro permanenza all'interno del Data Lake e le informazioni relative alle trasformazioni che hanno subito.
- Garantire un certo livello di sicurezza e privacy, definendo politiche di accesso ai dati, mascherando dati sensibili, ecc.
- Favorire la loro integrazione vista l'eterogeneità nella struttura.

Il processo per l'estrazione e la gestione dei metadati dovrebbe:

- Garantire un trade-off tra flessibilità e governabilità, evitando di diventare più costoso e dispendioso in termini di tempo rispetto al valore che si riesce ad estrarre dai dati.

- Essere pervasivo ed attivato già dalla fase di ingestion in modo da poter cogliere opportunità di riutilizzo e condivisione dei dati.

- Essere il più automatico possibile per costruire un'architettura scalabile in grado di stare al passo con la crescita dell'azienda.

1.3 Web Semantico

Il termine Web Semantico è stato coniato da Tim Berners-Lee per indicare "una collezione di tecnologie standard per realizzare il Web of Data", un'evoluzione del web esistente in cui i dati nel web sono annotati esplicitamente con metadati semantici in un formato adatto all'interrogazione e all'interpretazione automatica da parte dei computer [13].

Sebbene il mondo del Web Semantico sia slegato da quello dei Big Data, esso fornisce gli strumenti per rappresentare in maniera strutturata ma flessibile una determinata conoscenza. Pertanto, come si mostrerà nella Sezione 1.4, le tecnologie su cui si basa vengono utilizzate come mezzo per realizzare l'integrazione tra dataset eterogenei ed interrogare il Data Lake.

I principi su cui il Web Semantico si fonda sono [14]:

- Rendere disponibili ed accessibili, in un formato standard, i dati e le loro relazioni nel web, siano essi strutturati, semi-strutturati o non strutturati.

- Descrivere la semantica dei dati attraverso un formalismo in modo che possano essere processati automaticamente dai computer.

Questi principi sono stati tradotti nelle tecnologie che seguono.

RDF *RDF* (Resource Description Framework) è il formato standard utilizzato per rappresentare i dati nel web. Le risorse, e le relazioni tra queste, sono modellate rispettivamente come nodi ed archi di un grafo etichettato e sono identificate in maniera univoca tramite *URI* (Universal Resource Identifier).

L'unità base per rappresentare un'informazione in RDF è lo statement cioè una tripla soggetto-predicato-oggetto. Il soggetto è una risorsa, il predicato è una relazione o proprietà e l'oggetto può essere un'altra risorsa od un valore.

Una tra le sintassi più utilizzate per rappresentare i dati in formato RDF è quella XML-based (RDF/XML) mentre uno degli approcci più comuni per memorizzarli è usare un triplestore, molto simile ad un database relazionale ma ottimizzato per il salvataggio ed il recupero di triple [15].

SPARQL Essendo la conoscenza distribuita rappresentata come un grafo, per accedere ai dati si devono utilizzare linguaggi di interrogazione più complessi rispetto a SQL: lo standard che viene usato è *SPARQL* (SPARQL Protocol and RDF Query Language). Oltre ad essere un linguaggio di interrogazione è anche un protocollo di comunicazione tra il client che effettua le query SPARQL e gli endpoint RDF raggiungibili via HTTP, vale a dire i triplestore.

Con SPARQL si può avere accesso al modello concettuale dei dati usando la stessa modalità utilizzata per avere accesso ai dati stessi, triple in questo caso. E' possibile, tramite appositi costrutti, verificare se un certo pattern di dati esiste (ASK), con risultato true o false, ma anche ottenere un grafo RDF che descrive una risorsa (DESCRIBE) o il risultato di un'interrogazione (CONSTRUCT), con risultato RDF/XML. Questi costrutti si rivelano utili perché non tutte le risorse sul web vengono descritte utilizzando lo stesso schema o le stesse proprietà a causa dell'"assunzione del mondo aperto" del Web Semantico [16].

Ontologie Un'*ontologia* è una specifica esplicita e formale di una concettualizzazione condivisa. Permette di superare le differenze terminologiche proprie del linguaggio naturale, che si possono riscontrare nella definizione di uno stesso concetto nell'ambito di un dato dominio, favorendo l'integrazione e la costruzione di una conoscenza distribuita. Consiste in un insieme di concetti, relazioni, assiomi, individui, asserzioni e fatti ed è strettamente legata alla

logica del primo ordine. Una macchina in grado di interpretare un linguaggio ontologico (reasoner) sa inferire conoscenza e verificare automaticamente la correttezza delle assunzioni su un dominio.

Non esiste un miglior linguaggio ontologico in assoluto: viene scelto, a seconda dei casi, sulla base di un trade-off tra espressività ed esigenza di ragionare ricorsivamente su tutte le istanze del dominio in un tempo e con una modalità che sia accettabile. Più il linguaggio ontologico è espressivo più richiede un lavoro computazionale elevato. I più importanti linguaggi ontologici riconosciuti come standard sono RDF Schema ed OWL.

RDF Schema è il linguaggio ontologico più semplice. Permette di descrivere il vocabolario riferito al dominio che si vuole modellare in termini di classi, proprietà, gerarchie tra classi (superclassi e sottoclassi) e tra proprietà (superproprietà e sottoproprietà). E' possibile vincolare la classe di soggetti ed oggetti delle triple definendo restrizioni sulle proprietà, rispettivamente di dominio e di range, oppure riferirsi come oggetto di una tripla ad un intero statement (reificazione) [17]. Un esempio di modellazione tramite RDF Schema è visibile in Figura 1.6.

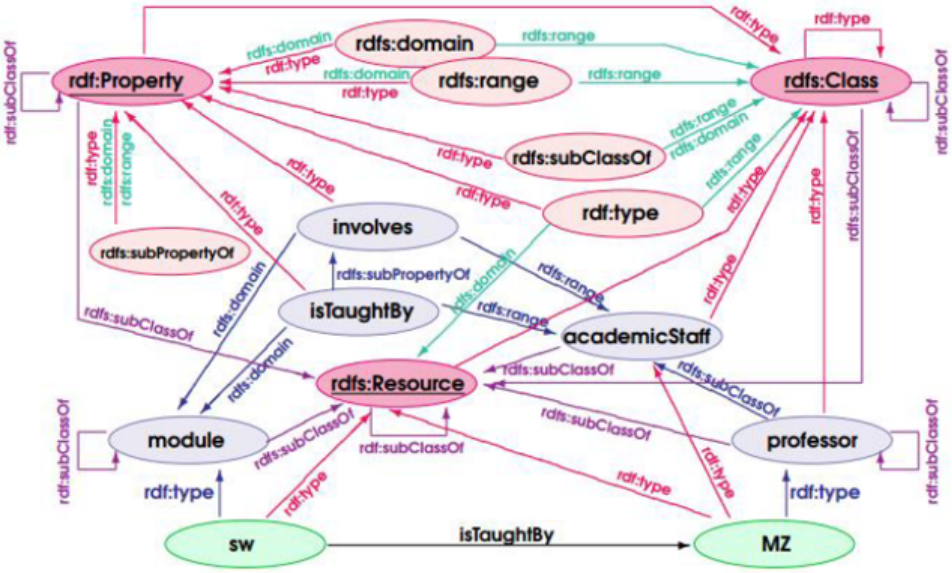

Figura 1.6: Esempio di modellazione tramite RDF Schema

Per modellare il dominio in maniera ancora più espressiva si può utilizzare **OWL** (Web Ontology Language). OWL aggiunge al vocabolario di RDF Schema la possibilità di esprimere, ad esempio, per le proprietà nuovi tipi, nuove restrizioni sulla cardinalità e sul tipo di valori che possono assumere, mentre per le classi disgiunzioni e combinazioni booleane [18].

1.4 Soluzioni esistenti per governare il Data Lake

In relazione alla necessità di definire un'architettura per il Data Lake ed un sistema di metadata management funzionale i ricercatori e le aziende che hanno deciso di cogliere le opportunità dei Big Data hanno risposto proponendo nuovi framework e strumenti software.

1.4.1 Letteratura scientifica

GOODS [19] è un'architettura progettata da Google per organizzare e gestire i dati all'interno del Data Lake. Lavorando in background, raccoglie, aggrega e mantiene aggiornato un catalogo centralizzato di metadati (Figura 1.7). Il catalogo contiene metadati creati a partire da dataset provenienti da sistemi di archiviazione eterogenei che vengono combinati con quelli generati dalle pipeline di produzione e dall'analisi del contenuto dei dataset. Questi metadati consentono di integrare ed inferire relazioni tra i dataset e forniscono agli utenti e ai servizi una vista unificata del contenuto del Data Lake. Poiché non è scalabile comparare ogni coppia di dataset nel Data Lake per scoprire le relazioni che li legano GOODS opera in modo *post-hoc* e si basa sui segnali dell'infrastruttura sottostante per scoprire i loro legami. Le relazioni tra i dataset possono emergere da metadati già estratti relativi alla loro struttura e provenienza, dall'appartenenza del dataset ad un cluster logico, ecc. e contribuiscono all creazione di un grafo della conoscenza. Come anticipato, all'interno del catalogo i dataset correlati possono essere raggruppati in cluster che diventano entità di prima classe a cui associare metadati. Versioni diverse di dataset dello stesso tipo conterranno metadati simili ed utilizzando i cluster è possibile riferirsi a queste come ad un singolo dataset logico. As-

sociare metadati ad ogni singolo file, considerando la scala e l'eterogeneità del Data Lake di Google, sarebbe proibitivo così come ricercare singoli dataset. Per questo motivo, il motore di ricerca incorporato nel sistema, lavora a livello di cluster. Il fatto che il motore di ricerca sia, tra tutti, lo strumento usato più frequentemente per navigare il Data Lake conferma l'importanza di riuscire a recuperare i dataset che il Data Lake contiene.

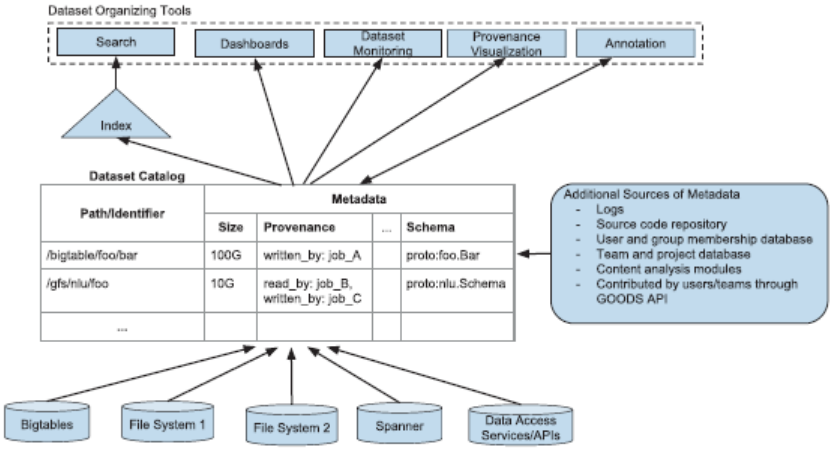

Figura 1.7: Panoramica dell'architettura di GOODS [19]

Constance [20] e GEMMS [21], rispetto a GOODS, introducono esplicitamente il concetto di metadati semantici.

Constance si pone come obiettivo quello di (i) estrarre metadati da fonti eterogenee memorizzandoli in un modello che sia flessibile ed estensibile (un'ontologia), (ii) annotare i dataset con informazioni semantiche per evitare ambiguità, favorire la loro integrazione con dataset correlati semanticamente e riassumerne il contenuto con un insieme di concetti salienti, (iii) fornire un supporto per interrogare il Data Lake e (iv) un'interfaccia grafica per monitorare la qualità dei dataset ed il risultato del processo di gestione dei metadati. I componenti principali della sua architettura (Figura 1.8) sono due. Si occupano, in cascata, rispettivamente dell'estrazione di metadati strutturali e semantici e sono lo *Structural Metadata Discovery* ed il *Semantic Metadata Matching (SMM)*. L'output è un grafo della conoscenza in cui emergono i metadati estratti e le loro relazioni che può essere interrogato.

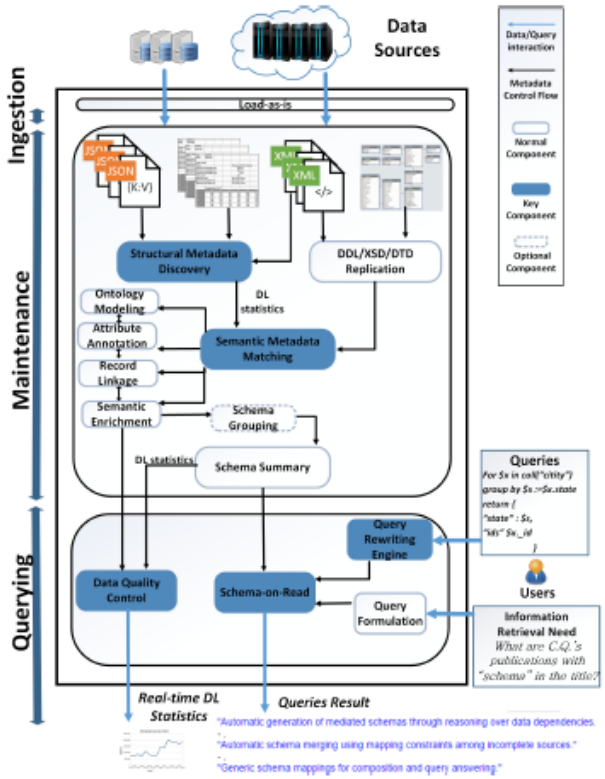

Figura 1.8: Panoramica dell'architettura di Constance [20]

GEMMS si inserisce nel contesto di Constance focalizzandosi sul metadata management. Il modello che propone per rappresentare i metadati è basato su coppie chiave-valore a cui è possibile allegare annotazioni di solito rappresentate come URI ad elementi di un ontologia. Le funzionalità che GEMMS fornisce sono essenzialmente tre: estrazione dei metadati, trasformazione dei metadati nel modello proposto e loro archiviazione all'interno di un catalogo. Il componente della sua architettura (Figura 1.9) che garantisce la flessibilità e l'estensibilità del sistema è il *Parser*, che si occupa di leggere la struttura interna del dataset per l'estrazione di metadati. Il Parser è specificatamente costruito per un tipo di file: quando il *Media Type Detector* rileva il tipo del file l'*Extractor* crea un'istanza del Parser corretto. Se vuole gestire un tipo di file che attualmente non viene gestito basterà creare un nuovo Parser che se ne occupi da invocare quando quel tipo viene rilevato.

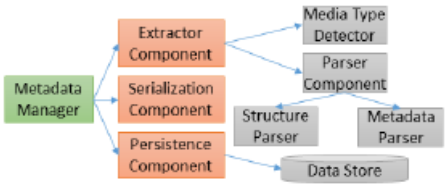

Figura 1.9: Panoramica dell'architettura di GEMMS [21]

KAYAK [22, 23] è un framework nato per aiutare i data scientist a definire ed ottimizzare le operazioni volte a preparare i dati per essere analizzati (Figura 1.10). Il sistema di metadata management al suo interno ha due componenti: il *Metadata Catalog* ed il *Metadata Collector*. Il Metadata Catalog si occupa dell'archiviazione di *intra-dataset* e *inter-dataset* metadata. Quelli intra-dataset riguardano, ad esempio, struttura e statistiche relative al singolo dataset. Quelli inter-dataset, invece, specificano relazioni tra dataset o attributi contenuti in dataset diversi: tra queste relazioni la *joinability* e l'*affinity* misurano rispettivamente la percentuale di valori comuni tra gli attributi di due dataset e la forza semantica di una relazione tra dataset. L'esistenza di relazioni tra dataset può essere utilizzata per eseguire analisi complesse. Il contenuto del Metadata Catalog è suddiviso in categorie ispirate a schemi di metadati esistenti ma è comunque estendibile. Il Metadata Collector si occupa di popolare il Metadata Catalog estraendo intra-dataset metadata resi accessibili attraverso un'interfaccia utente.

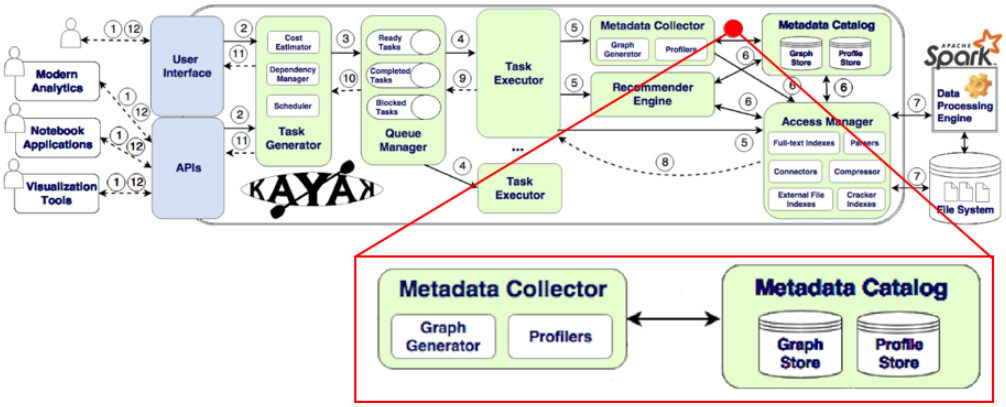

Figura 1.10: Panoramica dell'architettura di KAYAK [22]

Gli approcci incontrati finora si focalizzano più che altro su dati strutturati e semi-strutturati. A partire dalla proposta di Zaloni [24] è stato presentato un approccio semantico per correlare dataset valido su dati non strutturati che si concentra sull'intersezione tra quelli che vengono definiti *business* e *technical* metadata [25]. I business metadata catturano regole di business e significato dei dati in modo che tutti possano interpretarli in maniera concorde. I technical metadata catturano il formato e lo schema dei dati. In una sorgente non strutturata l'intersezione potrebbe consistere in un insieme di parole chiave generalmente adottate per dare un'idea del suo contenuto. Utilizzando una rappresentazione basata su un grafo della conoscenza l'approccio proposto consente la costruzione di viste, riguardanti uno o più argomenti di interesse per l'utente, ottenute estraendo ed unendo dati provenienti da diverse sorgenti sulla base di vocabolari più o meno espressivi (ontologie). Per mappare una sorgente non strutturata all'interno del modello ed integrarla con le altre sorgenti si procede per step. Dopo aver creato un grafo con le parole chiave che la rappresentano come nodi, utilizzando un dizionario (ad esempio BabelNet), si derivano le similarità lessicali con le parole chiave delle altre sorgenti. La similarità emerge se hanno almeno un *lemma* in comune. Se la similarità, secondo la metrica scelta, risulta maggiore di una soglia fissata si esegue l'unione dei nodi in uno unico che sarà collegato ad entrambe le sorgenti.

1.4.2 Strumenti open-source e commerciali

Teradata Kylo [26] è un framework open-source utilizzato per implementare un Data Lake su Hadoop, integrato con *Spark* ed *Hive*. Tramite interfaccia grafica è possibile definire un insieme di metadati che vengono memorizzati in un metadata server integrato che traccia dei cambiamenti nel tempo, permette di eseguire ricerche testuali e di estendere lo schema associando ai dataset business metadata. I business metadata sono propri del dominio di riferimento mentre, i technical metadata, riguardano la struttura del dataset. Oltre a questi vengono raccolti anche metadati che derivano dal processing distribuito (stato del processo, tempo di esecuzione, ecc.). Per definire come deve avvenire l'ingestion dei dati e le operazioni che devono essere eseguite su questi (pulizia, validazione, ecc.), Kylo usa **Apache NiFi** [27], un progetto open-source che

automatizza il passaggio dei dati da un sistema all'altro tramite un approccio visuale (drag and drop).

Oltre che con Kylo, NiFi si può integrare con **Apache Atlas** [28]. Atlas è uno strumento open-source per la data governance scalabile ed estensibile che, come Spark ed Hive, è integrato con l'ecosistema Hadoop. Permette di costruire un catalogo centralizzato che consente di avere una visione globale dei metadati a livello di business. La ricerca dei metadati all'interno del catalogo è supportata da un motore in grado di ricercarli efficacemente; è possibile eseguire ricerche full-text, per termini di un glossario (parola utile del dominio aziendale), sulla base di una classificazione attribuita ai dati (dati sensibili, log, ecc.), SQL-like. L'interfaccia di Atlas permette anche di visualizzare come i dati sono confluiti nel dataset considerato e come eventuali cambiamenti che si apportano al dataset influenzano i dataset che da questo attingono. Atlas può essere integrato con **Apache Ranger** [29], un altro progetto open-source collegato ad Hadoop, che permette di gestire in maniera centralizzata tutte le questioni legate alla sicurezza dei dati sul cluster. In Ranger, è possibile definire politiche di accesso ai dati che non si basano solo sul ruolo degli utenti che accedono al cluster ma anche sui metadati associati ai dataset. L'accesso ai dati può essere negato se non si è abilitati ad accedere a dati etichettati secondo una certa classificazione, dopo una certa data o da un certo paese. Tramite i metadati è possibile anche definire regole che impediscano di combinare due dataset.

DataIku [30] è uno strumento che, nella versione commerciale, può connettersi a dataset presenti nell'ecosistema Hadoop (su HDFS ed HBase) per svolgere operazioni ETL ed analisi complesse, occupandosi anche di gestire tutti i metadati associati (tipo dei dati, provenienza, qualità, ecc.). In fase di connessione ad un dataset, o di esecuzione di un'operazione sul dataset che origina in output un risultato con uno schema non ancora conosciuto dal sistema, è in grado di inferire automaticamente il tipo "tecnico" dei dati (int, float, ecc.), che chiama *storage type*, e il tipo semantico, che chiama *meaning*. Alcuni tra i tipi semantici già esistenti all'interno del sistema fanno riferimento all'ambito geospaziale e del web (latitudine/longitudine, indirizzo IP, indirizzo email, ecc.). Ogni meaning può rendere valida una colonna o meno per quel tipo semantico. E' possibile aggiungere nuovi meaning dichiarativi, per do-

cumentare la colonna, oppure come liste di valori o pattern che specificano i valori facenti parte del meaning.

Tutti gli strumenti sopra citati espongono le loro funzionalità anche attraverso API REST. Altre soluzioni commerciali per realizzare e governare il Data Lake sono state sviluppate dai principali colossi dell'informatica e sono fornite in cloud.

- Microsoft propone Azure Data Lake, basato su Hadoop ed integrato con altri servizi proprietari tra cui Azure SQL Data Warehouse, Power BI, Data Factory e Data Lake Analytics [31].

- Amazon propone una composizione di Amazon Web Services (AWS), quindi di servizi, che in fase di ingestion automaticamente rileva origine e formato dei dati suggerendo schemi e trasformazioni applicabili; i dati vengono salvati in Amazon S3 per analisi future, mentre i metadati, che comprendono anche tag definiti dagli utenti che aggiungono un contesto e permettono la ricerca dei dati all'interno del Data Lake, sono memorizzati su Amazon DynamoDB [32].

- Google propone la realizzazione del Data Lake su Google Cloud Platform (GCP) indicando Cloud Storage come repository per i dati, circondandolo da un insieme di servizi che si occupano dell'ingestion e dell'analisi dei dati; i metadati estratti vengono memorizzati in servizi come BigQuery per future ricerche ed analisi [33].

Capitolo 2

Progettazione del sistema

Il capitolo è dedicato ad illustrare il contesto ed il caso di studio alla base del progetto di tesi, il suo obiettivo e l'approccio proposto per realizzarlo, sviluppato tenendo conto di quanto emerso dal Capitolo 1.

2.1 Contesto, obiettivo e caso di studio

Contesto La tesi si inserisce all'interno di un processo di reingegnerizzazione dei sistemi aziendali, al fine di predisporli ad una corretta gestione futura dei Big Data. Tale processo prevede la migrazione da un approccio più tradizionale, che prevede un'architettura su tre livelli costituita da livello delle sorgenti, livello dell'alimentazione, che contiene l'Operational Data Store (ODS), e livello del warehouse, che contiene il Data Warehouse alimentato dall'ODS, ad uno più innovativo che prevede, in sostituzione all'ODS, l'adozione del Data Lake come unico punto d'ingresso per i dati. I dati contenuti al suo interno hanno un duplice utilizzo.

- Alimentare il Data Warehouse, utilizzato per rispondere, con vincoli di performance, alle esigenze aziendali correnti; al Data Warehouse attingono attualmente numerosi strumenti di Business Intelligence, Data Mining e moduli software sviluppati dall'azienda utilizzati in diversi ambiti (sanità, industria, logistica, ecc.). Questi strumenti e moduli permettono di trasformare i dati in conoscenza e prendere decisioni strategiche avendo una visione completa del proprio business oltre a definire e monitorare

processi ed obiettivi aziendali creando report e schede di valutazione on the fly rivolte sia all'area operativa che dirigenziale.

- Eseguire analisi esplorative per scovare pattern interessanti da utilizzare per cogliere nuove opportunità di business.

Alla luce di quanto emerso dal Capitolo 1 e dal contesto aziendale generale si possono fare le seguenti considerazioni.

- L'introduzione del Data Lake *as-is* non rappresenta una soluzione perché senza definire un'architettura ed adeguate politiche governance c'è il rischio che si trasformi in un Data Swamp.

- E' necessario definire un architettura per il Data Lake che tenga in considerazione il fatto che l'azienda opera in ambiti diversi; il Data Lake da realizzare è quindi *multi-tenant*.

- E' necessario definire un sistema di metadata management che consenta di realizzare una solida base di metadati, valida e condivisa da tutti i sistemi aziendali; per farlo bisogna definire

  - Un modello per l'archiviazione di metadati che sia estendibile e flessibile e consenta di indicizzare i dati contenuti nel Data Lake in maniera scalabile.
  - Un processo di estrazione ed aggiornamento di metadati che sia pervasivo; i metadati devono essere raccolti a partire dalla fase di ingestion e devono essere aggiornati, nella maniera più automatica possibile, mano a mano che i dati vengono utilizzati.

Obiettivo del prototipo Il prototipo sviluppato in questa tesi si colloca nella fase iniziale del processo di reingegnerizzazione ed è basato su un caso di studio specifico illustrato nel prossimo paragrafo. I suoi obiettivi, che stabiliscono i confini del progetto di tesi, possono essere riassunti nei punti che seguono.

- Progettare la struttura per il Data Lake.

- Fornire un modello per l'archiviazione di metadati che sia scalabile, estendibile, flessibile che funga da base per tutti i sistemi aziendali.

- Fornire un approccio che sia estensibile e parametrizzabile per indicizzare, nella maniera più automatica possibile, i singoli file che arrivano sul Data Lake tramite l'estrazione di metadati strutturali, di processo e semantici, in modo da consentire e facilitare il loro recupero quando rilevanti rispetto a determinate interrogazioni.

- Fornire le funzionalità in maniera coerente con gli altri moduli aziendali, quindi attraverso un'interfaccia web.

Il prototipo sviluppato non si occupa in alcun modo di garantire la privacy, la sicurezza e la qualità dei dati.

Caso di studio Il caso di studio preso in esame per lo sviluppo del prototipo è correlato all'ambito sanitario. I dati relativi a questo dominio giungono ai sistemi aziendali come file XML di eventi che si sono verificati (ad esempio un ricovero di un paziente, la prenotazione di una visita, ecc.) e, nell'ottica del processo di reingegnerizzazione, vengono salvati all'interno del Data Lake.

I file su cui si concentra il caso di studio fanno riferimento agli eventi descritti dalle RFC 115 e 118, reperibili al sito eCompliance di Regione Toscana [33], che contiene anche RFC non relative all'ambito sanitario. Le RFC definiscono un insieme di standard condivisi che descrivono il formato e la modalità di scambio delle informazioni che gli enti devono utilizzare nelle comunicazioni a Regione Toscana, con cui l'azienda lavora. Quelle esaminate contengono le seguenti tipologie di eventi.

- Eventi riconducibili all'Assistenza Domiciliare (AD) (RFC 115).
  - Segnalazione: si verifica quando il diretto interessato, o qualcuno titolato a farlo, si rivolge alla rete di accesso per segnalare un bisogno.
  - Presa in carico: si verifica quando viene esaminata, quindi presa in carico, la segnalazione.

  - Erogazione dell'AD: si verifica ogni qualvolta si ha l'accesso di un operatore al domicilio dell'assistito per l'erogazione di una prestazione.
  - Rivalutazione del caso: si verifica in concomitanza a scadenze programmate o in presenza di condizioni che la rendano necessaria (rientro del paziente da una fase di sospensione, variazioni nel quadro clinico del paziente, ecc.) per modificare, laddove opportuno, il percorso assistenziale.
  - Sospensione dell'assistenza: si verifica quando si presentano le condizioni per sospendere l'erogazione dell'assistenza (allontanamento temporaneo, ricovero in struttura residenziale, ecc.) .
  - Conclusione del percorso assistenziale: si verifica nel momento in cui termina il percorso assistenziale.
  - Cancellazione: si verifica quando viene inviato un messaggio errato a Regione Toscana per comunicare la cancellazione del messaggio precedentemente inviato.

- Eventi riconducibili alle prestazioni residenziali e semi-residenziali fornite dalle RSA (RFC 118).

  - Ammissione in RSA: si verifica al momento dell'ammissione dell'assistito nella struttura residenziale o semi-residenziale.
  - Dimissione da RSA: si verifica quando l'assistito viene dimesso dalla struttura residenziale o semi-residenziale.

Oltre ad una descrizione discorsiva degli eventi, le RFC contengono anche il loro schema ed alcuni file di esempio. Analizzando lo schema degli eventi si nota immediatamente la presenza di tag opzionali e *choice* (costrutto XML che consente la selezione di un sotto-albero piuttosto che di un altro) che possono potenzialmente generare molte versioni diverse per lo stesso tipo di evento. Diventa quindi fondamentale associare ai file che contengono questi eventi un insieme di metadati semantici che li identifichino indipendentemente dalla loro struttura, per poterli reperire in maniera semplice all'interno del Data Lake.

2.2 Struttura del Data Lake

Facendo riferimento all'architettura presentata alla Sottosezione 1.2.1 le zone che vengono considerate ai fini della tesi sono la Transient zone e la Raw zone. A loro volta, le zone, sono state divise in cartelle che possono essere utilizzate anche per dividere, a livello logico, i dati relativi ai diversi ambiti in cui l'azienda opera, all'interno dello stesso repository (Figura 2.1).

- La Transient zone conterrà i file che non hanno ancora metadati associati. Le cause possono essere due: i file devono ancora passare attraverso il processo di estrazione dei metadati (cartella *New*) oppure si è verificato un errore durante l'estrazione (cartella *Wrongly processed*).
- La Raw zone conterrà i file che sono passati con successo attraverso il processo di estrazione dei metadati (cartella *Correctly Processed*).

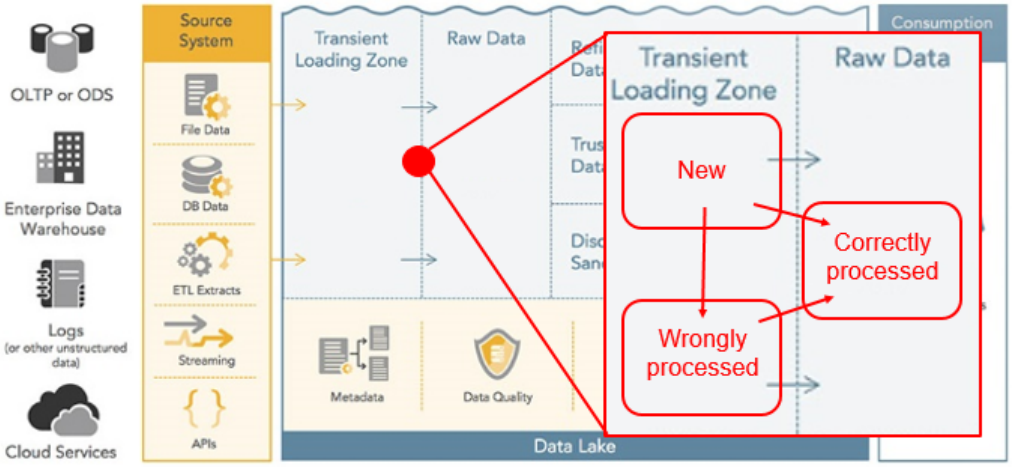

Figura 2.1: Architettura di riferimento per il Data Lake nel prototipo sviluppato nella tesi, adattata da [11]

2.3 Definizione dell'ontologia

Per rispondere ai requisiti di flessibilità ed estendibilità espressi in fase di analisi, come modello per organizzare la base di metadati è stata scelta un'ontologia. Per poter essere estesa a domini diversi rispetto a quello utilizzato

come caso di studio, l'ontologia è stata modellata come un insieme di concetti che fosse il più generale possibile (Figura 2.2).

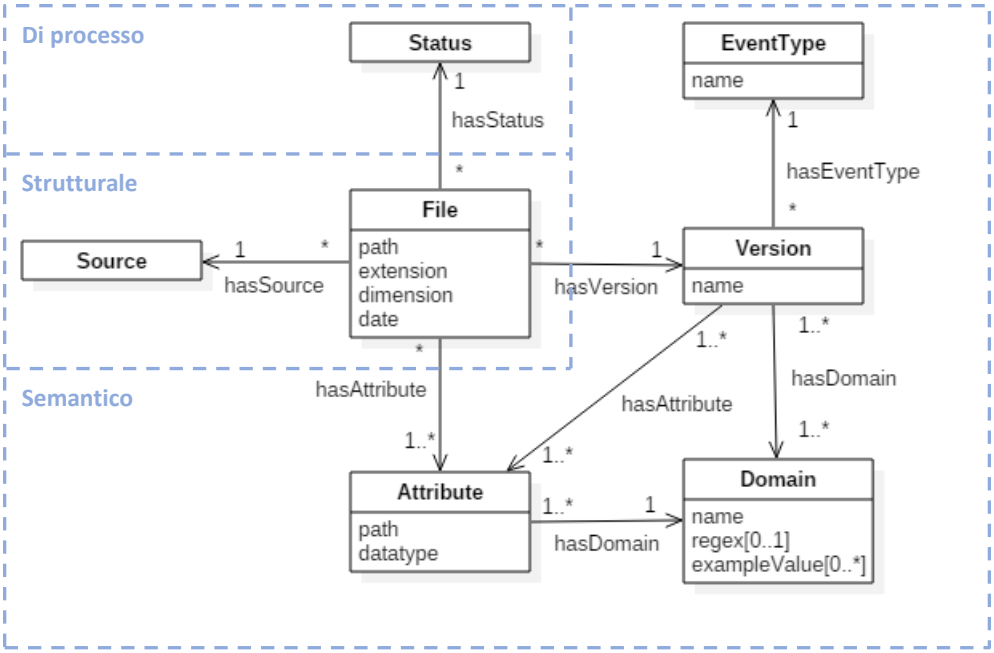

Figura 2.2: Modello dell'ontologia

Per la sua creazione sono stati considerati gli aspetti che seguono.

- Strutturale: riguarda le proprietà del file (posizione nel Data Lake, estensione, dimensione, data).
- Di processo: riguarda lo stato del file (Status).
- Semantico: riguarda la definizione di un insieme di concetti che descrivano il file (Attribute, Domain, Version, EventType, Source e loro proprietà).

Le classi sono state definite come disgiunte le une dalle altre e vengono elencate di seguito introducendo il concetto che modellano.

- *File*: singolo file che viene processato. Rappresenta un file XML, composto da uno o più attributi associati a dei domini; gli attributi, e i domini a questi collegati, sono importanti perché permettono di ricondurre il file

al tipo di evento, tramite la versione. Le proprietà che vengono mantenute sono il path, cioè il percorso in cui il file viene memorizzato all'interno del Data Lake (su HDFS), l'estensione, XML per i file del caso di studio, la dimensione e la data di creazione del file.

- *Status*: stato del file (caricato/non caricato nel Data Warehouse); quando un file passa attraverso il processo di estrazione dei metadati proposto gli viene assegnato lo stato "non caricato"; tale informazione potrà essere cambiata da futuri processi che si occuperanno dell'alimentazione del Data Warehouse.

- *Source*: sorgente che ha emesso il file; un file può essere emesso dalle sorgenti più svariate come programmi, sensori, ecc. In questo caso le sorgenti dati sono le applicazioni degli enti che inviano gli eventi a Regione Toscana.

- *Attribute*: singolo tag XML che è contenuto nel file o nella versione. Le proprietà che vengono mantenute sono il path, percorso di tag dalla radice alla foglia, e il datatype, tipo del valore contenuto dal tag, inferito durante il processo di estrazione dei metadati.

- *Domain*: dominio che descrive l'attributo o la versione. Un dominio rappresenta la classificazione che si può dare al valore di un attributo. Le proprietà che vengono mantenute sono il nome ed opzionalmente un'espressione regolare, cioè una sequenza di simboli che identifica quali sono le stringhe ammissibili, e/o un insieme di valori di esempio per il dominio.

- *EventType*: tipo di evento del file XML che può pervenire in versioni diverse ma comunque nel rispetto delle descrizioni alle RFC 115 e 118.

- *Version*: versione del file XML, descritta da un insieme di attributi e domini; la presenza o meno, all'interno del file XML, di tag opzionali e/o sotto-alberi di tag diversi può portare alla creazione di numerose versioni che fanno riferimento allo stesso tipo di evento. È il tramite che permette di collegare il file al tipo di evento a cui fa riferimento.

Gli attributi delle classi, nella modellazione in Figura 2.2, sono stati espressi all'interno dell'ontologia come *datatype properties*. Le datatype properties

sono proprietà che mettono in relazione risorse e valori, cioè istanze di una classe con tipi di dati (come possono essere string, date, integer, ecc.). Le relazioni tra classi sono state espresse invece come *object properties*. Le object properties sono proprietà che mettono in relazione risorse con altre risorse, cioè istanze di una classe con istanze di un'altra. Per vincolare soggetto ed oggetto delle proprietà sono stati definiti in maniera opportuna *domain* e *range* di appartenenza: un esempio è visibile nella Tabella 2.1.

Proprietà	**Tipo**	**Dominio**	**Range**
path	Datatype property	Attribute	string
datatype	Datatype property	Attribute	string
hasAttribute	Object property	File or Version	Attribute
hasDomain	Object property	Attribute or Version	Domain

Tabella 2.1: Esempio di domini e range per le proprietà che coinvolgono l'attributo

2.4 Processo di estrazione dei metadati

Il processo di estrazione dei metadati proposto è calato rigorosamente sul caso di studio ed è quindi volto all'estrazione di metadati strutturali, di processo e semantici, a partire dai file XML dell'ambito scelto. Lo scopo è quello di indicizzare i singoli file XML attribuendo a questi, nella maniera più automatica possibile, il tipo di evento a cui fanno riferimento (segnalazione, presa in carico, ecc.). Per fare questo, l'idea di fondo consiste nell'associare immediatamente al file i metadati strutturali e di processo per poi eseguire l'estrazione dei metadati semantici che permetteranno di associare il file al tipo di evento (Figura 2.3). Per associare il file al tipo di evento si cerca di associare gli attributi che il file contiene a quelli già presenti nell'ontologia e, se assenti, viene cercata, sulla base del loro nome e valore, una corrispondenza con i domini esistenti. Tramite gli attributi del file, ed i domini che gli sono stati associati, è possibile ricostruire la versione a cui il file fa riferimento e ricavare il tipo di evento a cui il file appartiene. Questo processo di estrazione può portare alla creazione di nuovi attributi, domini, versioni e tipi di evento.

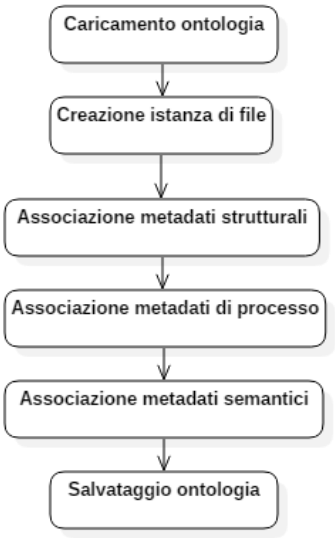

Figura 2.3: Panoramica dello scenario di successo del processo di estrazione dei metadati

Il processo di estrazione dei metadati semantici, oltre ad utilizzare il modello di ontologia progettato, è basato sull'utilizzo:

- Di una soglia ed una finestra di similarità da considerare per il calcolo della similarità degli attributi rispetto ai domini.

- Di una soglia ed una finestra di similarità da considerare per il calcolo della similarità di un file rispetto alle versioni.

Le modalità che si possono utilizzare per estrarre i metadati semantici sono due: una coinvolge l'interazione dell'utente, qualora il processo non sia in grado di prendere una decisione autonomamente (modalità di apprendimento) mentre l'altra è completamente automatica (modalità automatica). L'estrazione dei metadati semantici avviene in tre step, come visibile in Figura 2.4. Il primo step è sempre uguale mentre gli altri due differiscono a seconda della modalità scelta.

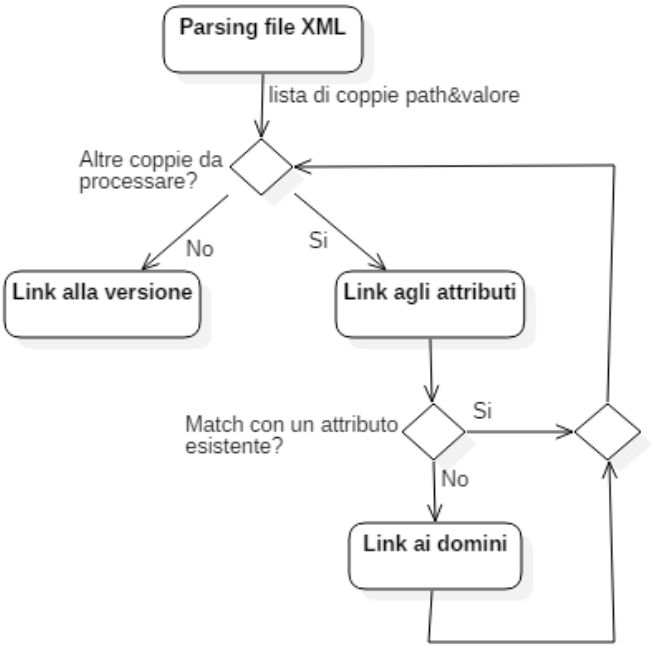

Figura 2.4: Descrizione di alto livello del processo di estrazione di metadati semantici

Link agli attributi In questo step si tenta di associare gli attributi del file agli attributi esistenti nell'ontologia sulla base del path ricavato dal parsing del file XML. Se all'interno dell'ontologia esiste già un attributo con quel path semplicemente si crea l'associazione tra file ed attributo, andando a modificare il datatype se questa volta l'attributo è comparso con un valore avente tipo "più largo" rispetto a quello già memorizzato. Se l'attributo non esiste si crea una nuova istanza di attributo, si associa al file, si memorizza il datatype inferito dal valore e si passa allo step successivo.

Link ai domini In questo step si tenta di associare ai nuovi attributi un dominio sulla base dei domini esistenti nell'ontologia e del valore con cui compare l'attributo. In primo luogo si calcola la similarità di path e valore dell'attributo con le proprietà di ogni dominio (la modalità con cui viene calcolata è descritta nella Sottosezione 2.5). Ottenuta una similarità nei confronti di ogni dominio

- Se tutti i domini hanno similarità $<$ soglia:

  - In modalità automatica non si associa nessun dominio all'attributo.
  - In modalità di apprendimento si richiede l'input dell'utente per la creazione di un'istanza di dominio di cui specificare nome, espressione regolare, eventuali valori di esempio; l'attributo verrà poi associato al dominio creato.

- Se almeno un dominio ha similarità >= soglia:
  - In modalità automatica si associa l'attributo al dominio che ha similarità più elevata a meno che non ve ne siano due che hanno la stessa similarità: in tal caso si verifica un errore.
  - In modalità di apprendimento si associa l'attributo al dominio che ha similarità più elevata a meno che non ve ne siano di più entro la finestra specificata: in tal caso si richiede l'input dell'utente per selezionare il dominio più adatto o crearne uno nuovo.
  - Indipendentemente dalla modalità, in caso di match perfetto (similarità = 1), il dominio viene assegnato automaticamente all'attributo: è impossibile che per due domini diversi esista un match perfetto perché non è possibile che l'ontologia memorizzi due risorse che hanno lo stesso URI.

Una volta associati tutti gli attributi del file ai domini si passa allo step successivo.

Link alla versione In questo step si tenta di associare al file una versione sulla base degli attributi che lo caratterizzano e dei domini che questi ultimi hanno matchato. In primo luogo si calcola la similarità tra attributi e domini che caratterizzano il file e quelli che caratterizzano ogni versione (la modalità con cui viene calcolata è descritta nella Sottosezione 2.5). Ottenuta una similarità nei confronti di ogni versione

- Se tutte le versioni hanno similarità < soglia:
  - In modalità automatica si verifica un errore.
  - In modalità di apprendimento si richiede l'input dell'utente per la creazione di un'istanza di versione di cui specificare il tipo di evento;

alla versione creata verranno poi associati domini ed attributi del file che si sta processando e il tipo di evento specificato.

- Se almeno una versione ha similarità >= soglia:
  - In modalità automatica si crea una nuova istanza di versione del tipo di evento di quella che ha similarità più elevata a meno che non ve ne siano due che hanno la stessa similarità: in tal caso si verifica un errore.
  - In modalità di apprendimento si crea una nuova istanza di versione del tipo di evento di quella che ha similarità più elevata a meno che non ve ne siano di più entro la finestra specificata: in tal caso si richiede l'input dell'utente che deve specificare la versione da cui recuperare il tipo di evento o crearne una nuova se ritiene che il file appartenga ad un tipo di evento che non esiste; alla versione così creata verranno poi associati domini ed attributi del file che si sta processando.
  - Indipendentemente dalla modalità, in caso di match perfetto (similarità = 1), la versione viene associata automaticamente al file.

Se l'estrazione dei metadati dal file si è conclusa correttamente l'ontologia viene salvata ed il file spostato tra quelli processati correttamente. Al contrario, se si verifica un errore, i metadati del file non vengono salvati (l'ontologia non viene salvata) ed il file viene spostato tra quelli processati erroneamente. Gli errori che possono dare origine a questo comportamento sono diversi. Tra questi: errori che si verificano durante il parsing del file XML, errori che si verificano in modalità automatica quando due domini/versioni hanno la stessa similarità o nessuna versione ha similarità sopra la soglia, ecc.

2.5 Misure di similarità

Le misure di similarità che vengono utilizzate all'interno del processo di estrazione dei metadati sono due e sono riportate di seguito.

Similarità di un attributo rispetto ad un dominio La similarità ripetto ai domini esistenti all'interno dell'ontologia viene calcolata per associare ogni attributo del file ad un dominio. Il calcolo della similarità si può scomporre in due parti, una delle quali usa la distanza di Levenshtein per calcolare una distanza tra stringhe. La distanza di Levenshtein [35] rappresenta il numero minimo di modifiche elementari che consentono di trasformare una stringa in un'altra. Come modifica elementare si intende la cancellazione di un carattere, la sostituzione di un carattere con un altro o l'inserimento di un carattere. Per ogni path e valore estratto dall'XML

- Si calcola la distanza di Levenshtein tra il nome del tag foglia, recuperato dal path, e il nome del dominio considerato e la si converte in similarità, normalizzata considerando la maggiore in lunghezza delle due stringhe (*similarityOnName*); tale similarità ha un peso di α nel calcolo della similarità totale.

- Se il dominio considerato ha un'espressione regolare associata si verifica se il valore la matcha (distanza 0); se il dominio non ha un'espressione regolare ma dei valori di esempio associati si calcola la distanza di Levenshtein tra valore estratto e valori di esempio e si mantiene la minima normalizzando il risultato considerando, anche in questo caso, il maggiore in lunghezza di questi due valori; se il dominio non ha nemmeno valori di esempio non ci sono abbastanza elementi per discriminare (distanza 0); la distanza calcolata viene convertita in similarità (*similarityOnRegexOrValues*) ed il suo peso nel calcolo della similarità totale è β.

Le distanze vengono convertite in similarità considerando la formula:

$$1 - distance$$

La similarità totale viene quindi calcolata come:

$$similarityOnName * \alpha + similarityOnRegexOrValues * \beta$$

α e β sono pesi che si possono definire nei limiti di $\alpha + \beta = 1$. Nel processo di estrazione proposto $\alpha = 0.5$ e $\beta = 0.5$.

Similarità di un file rispetto ad una versione La similarità rispetto alle versioni esistenti all'interno dell'ontologia viene calcolata per associare ogni file ad un tipo di evento. Gli attributi del file ed i domini che hanno agganciato vengono confrontati con attributi e domini che caratterizzano le versioni. Per ogni versione

- Per gli attributi del file presenti nella versione si assegna score 1 (*matchOnAttribute*).
- Per gli attributi del file che non sono presenti nella versione ma agganciano un dominio che caratterizza la versione si assegna uno score $0 < \delta < 1$ (*matchOnDomain*). Nel processo di estrazione proposto $\delta = 0.5$.
- Per i restanti attributi del file, che non hanno fatto match con nulla, si assegna score 0.

Lo score così calcolato viene normalizzato per il numero di attributi maggiore tra quello del file e della versione considerata.

$$\frac{(matchOnAttribute * 1 + matchOnDomain * \delta)}{max(numAttributeFile, numAttributeVersion)}$$

Capitolo 3

Implementazione e test del sistema

Il capitolo è dedicato ad illustrare le tecnologie utilizzate per implementare l'approccio proposto, i test eseguiti per verificarne il funzionamento ed il risultato ottenuto.

3.1 Tecnologie e loro utilizzo

3.1.1 Protégé ed Apache Jena

Protégé Protégé [36] è un framework open-source utilizzato per la definizione di ontologie sviluppato dallo Stanford Center for Biomedical Informatics Research della Stanford University School of Medicine. È basato su Java, è estensibile tramite plug-in, sviluppati per facilitare la creazione di sistemi intelligenti che richiedono ontologie complesse, e supporta le specifiche di RDF, RDF Schema ed OWL, definite dal W3C. Tramite un'interfaccia grafica è possibile creare, modificare, interrogare ontologie, verificare la loro consistenza e dedurre nuove informazioni.

Protégé è stato utilizzato all'interno del progetto per modellare l'ontologia e verificare la consistenza delle informazioni in essa contenute (ad esempio il rispetto dei vincoli di dominio e di range delle proprietà) a seguito del processo di estrazione dei metadati. L'ontologia modellata è stata salvata su file in formato RDF/XML e viene acceduta utilizzando le Jena API.

Apache Jena Apache Jena [37] è un framework open-source utilizzato per creare applicazioni basate sulle tecnologie del Web Semantico. Fornisce un insieme di API che permettono di lavorare sia a livello di RDF che di OWL, un triplestore per memorizzare le triple, un motore ed un server SPARQL per accedere alle triple, memorizzate rispettivamente in locale o ad un endpoint remoto, ed un motore di inferenza per verificare la consistenza ed inferire nuova conoscenza a partire dal modello.

Jena è una delle API Java più utilizzate per manipolare ontologie insieme alle OWL API. La scelta di utilizzare Jena rispetto a queste ultime è stata dettata dal fatto che le Jena API trattano l'ontologia come un insieme di triple RDF, direttamente accessibili tramite SPARQL (che è un RDF query language), possibilità che le OWL API non danno perché trattano l'ontologia come un insieme di assiomi. Le Jena API vengono utilizzate all'interno dell'applicazione per tutto quello che concerne la manipolazione dell'ontologia. Una volta caricato il modello ontologico serializzato in formato RDF/XML questo viene poi interrogato ed aggiornato continuamente durante il processo di estrazione dei metadati e salvato ogni volta che il processing di un file è andato a buon fine.

3.1.2 Java Server Faces e WildFly

Java Server Faces Java Server Faces (JSF) [38] è un framework basato sul design pattern architetturale Model-View-Controller (MVC) che semplifica lo sviluppo dell'interfaccia utente (UI) di un'applicazione Web abilitando il riuso e l'estensione dei componenti dell'interfaccia. Ne esistono diverse implementazioni: quella usata all'interno della tesi è **PrimeFaces**. Di seguito viene analizzato nelle sue caratteristiche per permettere di comprendere com'è strutturata l'applicazione sviluppata.

In JSF ogni pagina web (View) è associata ad un *managed bean*. I managed bean sono visibili all'interno del framework tramite annotazioni. Le loro proprietà e i loro metodi possono essere rispettivamente accedute ed utilizzati all'interno delle pagine web tramite un sistema di dependency injection e l'utilizzo di un Expression Language (EL). Esistono diversi tipi di managed bean: quelli più utilizzati all'interno del prototipo ricadono sotto la definizione di *backing bean*. I backing bean supportano la logica dell'interfaccia utente

ed hanno una relazione 1 ad 1 con una pagina web. Al contrario di quelli che vengono definiti *model bean*, le proprietà ed i metodi di get e set che contengono fanno riferimento alla pagina web e non ai modelli di dati utilizzati dall'applicazione. I modelli di dati utilizzati all'interno del prototipo vengono mantenuti come semplici *POJO* (Plain Old Java Object) e mappano i concetti contenuti nell'ontologia. Questi ultimi, la logica del processo di estrazione dei metadati e le classi di helper costituiscono il Model dell'applicazione. I backing bean forniscono anche una piccola quantità di logica simile ad un Controller tuttavia, il Controller principale di un'applicazione sviluppata tramite JSF è la *FacesServlet*, servlet di default del framework, che intercetta le richieste che arrivano all'application server e gestisce tutto il ciclo di vita delle pagine e dei backing beans. JSF mette inoltre a disposizione la logica per la conversione dei tipi, da e verso stringhe da visualizzare, tramite convertitori, e la validazione dell'input dell'utente, tramite validatori. Entrambe queste funzionalità sono state utilizzate per la costruzione della modalità di apprendimento così come l'utilizzo di chiamate AJAX, supportate anch'esse dal framework, per aggiornare dinamicamente la pagina web.

Alla luce di tutto questo i moduli di cui si compone l'applicazione sono due: un modulo web che contiene le pagine web (View) ed un modulo java che contiene i backing bean (Controller), le classi che mappano i concetti dell'ontologia, la logica del processo di estrazione dei metadati e classi di helper (Model).

WildFly WildFly [39] è un application server open-source che implementa le specifiche Java EE. Fornisce l'infrastruttura sulla quale viene eseguita l'applicazione web. Il deploy dell'applicazione sviluppata usando JSF è stato eseguito sull'istanza locale dell'application server.

3.1.3 HDFS

Come anticipato nella Sottosezione 1.1.2 HDFS è un file system distribuito che fornisce accesso ad alta velocità ai dati delle applicazioni utilizzato per l'archiviazione dei Big Data. All'interno del progetto di tesi HDFS è stato scelto per implementare il Data Lake e memorizzare i singoli file da processare. Per leggere/scrivere i file contenuti al suo interno dall'applicazione web sono state

usate le **File System API** compatibili con la distribuzione e la versione di Hadoop installata sul cluster (*Cloudera Express 5.13.1*). All'accesso alla pagina web dell'applicazione viene caricata la configurazione del cluster HDFS, specificata tramite due file (*core-site.xml* ed *hdfs-site.xml*), per consentire l'accesso al file system. Quando viene lanciato il processing i file sono letti in blocco e spostati tra quelli processati correttamente o non, a seconda dell'esito del processo di estrazione dei metadati.

3.2 Architettura funzionale

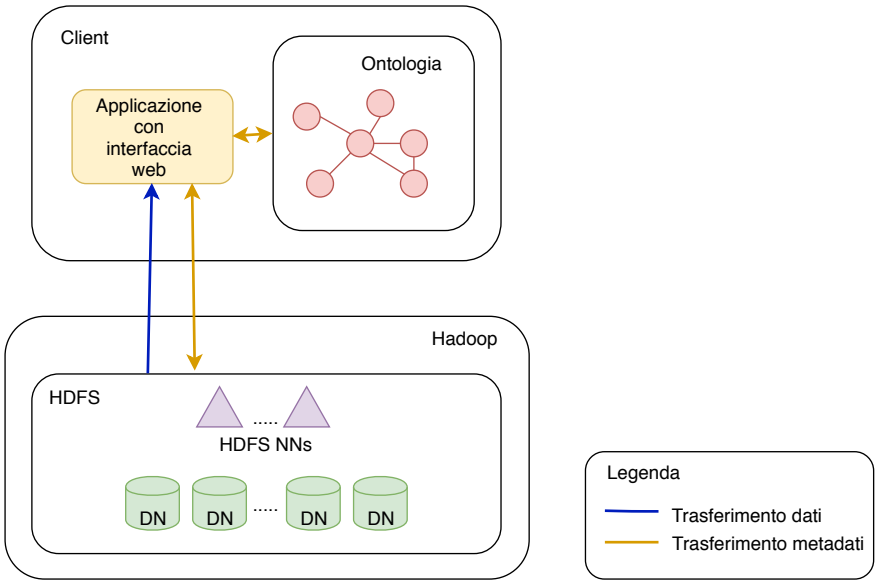

Figura 3.1: Architettura di alto livello del prototipo realizzato

L'architettura di alto livello relativa al prototipo realizzato viene mostrata in Figura 3.1. Le principali componenti sono il Client ed il cluster Hadoop. Il Client rappresenta l'istanza locale dell'application server su cui viene eseguito il deploy dell'applicazione web e mantenuta l'ontologia. All'interno del cluster Hadoop viene evidenziata la struttura di HDFS, il file system distribuito utilizzato per memorizzare i dati da processare e sul quale è stato implementato il Data Lake. La freccia blu rappresenta il trasferimento di dati, che avviene quando l'applicazione web legge da HDFS i file da processare. Le frecce arancioni, invece, rappresentano il trasferimento di metadati che avviene quando

le caratteristiche del file devono essere lette per essere inserite come proprietà nell'ontologia ma anche quando il file deve essere spostato, in base all'esito del processo di estrazione dei metadati, in una cartella piuttosto che in un'altra su HDFS. Lo scambio di metadati avviene inoltre tra applicazione web ed ontologia a seguito dell'aggiornamento o dell'interrogazione dell'ontologia.

3.3 Interfaccia

L'interfaccia è stata progettata per rendere il processo dei estrazione di metadati accessibile in maniera coerente agli altri moduli aziendali, che forniscono le proprie funzionalità via web. Di seguito vengono mostrate le pagine principali dell'applicazione.

Processing A partire dalla pagina "Processing", Figura 3.2, è possibile

- Settare la soglia e la finestra da considerare per il calcolo della similarità con i domini.
- Settare la soglia e la finestra da considerare per il calcolo della similarità con le versioni.
- Selezionare la modalità con cui eseguire il processing: automatica (Fast) o di apprendimento (Slow).
- Selezionare la cartella da cui reperire i file che devono passare attraverso il processo di estrazione dei metadati: cartella dei nuovi file (New files) o cartella dei file erroneamente processati (Wrongly processed files).
- Azionare il processo di estrazione dei metadati.
- Visualizzare il numero dei file processati, distinguendo quelli processati correttamente e non.
- Visualizzare il tempo totale di esecuzione del processo di estrazione, distinguendo tra quello di processing e di input/output.
- Navigare su altre pagine dell'applicazione web.

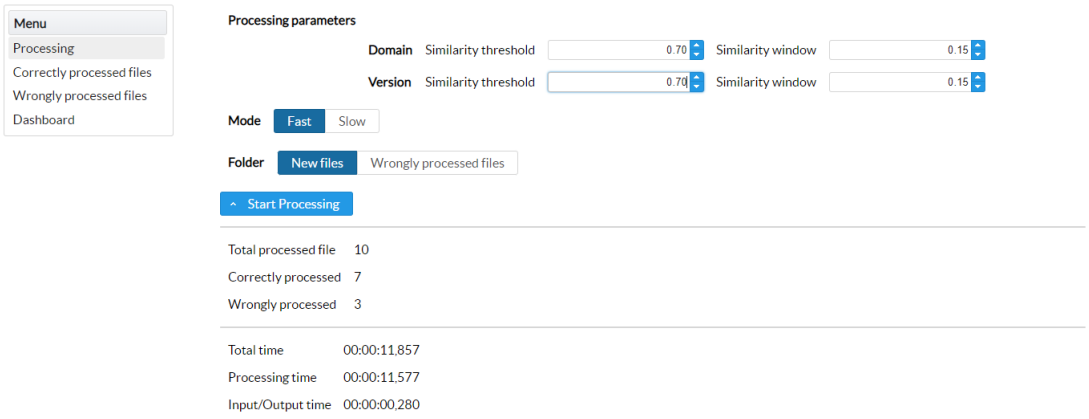

Figura 3.2: Pagina del processo di estrazione

All'avvio del processo di estrazione dei metadati, se questo viene eseguito in modalità di apprendimento, l'applicazione può generare una serie di dialog che richiedono l'input dell'utente per selezionare o specificare nuovi domini, versioni, tipi di evento.

In Figura 3.3 e 3.4 sono riportati rispettivamente due esempi relativi alla creazione di nuovo dominio ed alla selezione di un dominio esistente. In fase di selezione è possibile scegliere il dominio da agganciare all'attributo tra quelli proposti e mostrati per similarità decrescente, che ricadono all'interno della finestra di similarità definita, oppure creare un nuovo dominio. La dialog permette anche richiedere tutti i domini esistenti all'interno dell'ontologia: il soggetto incaricato all'aggiunta delle informazioni può essere a conoscenza del fatto che nell'ontologia esista già il dominio adatto per l'attributo ma, a causa della formula scelta per il calcolo della similarità, è possibile che questo non compaia all'interno della finestra. In Figura 3.5 è riportato invece un esempio relativo alla creazione di una nuova versione. Da questa dialog si può decidere di creare un nuovo tipo di evento a cui agganciare la versione.

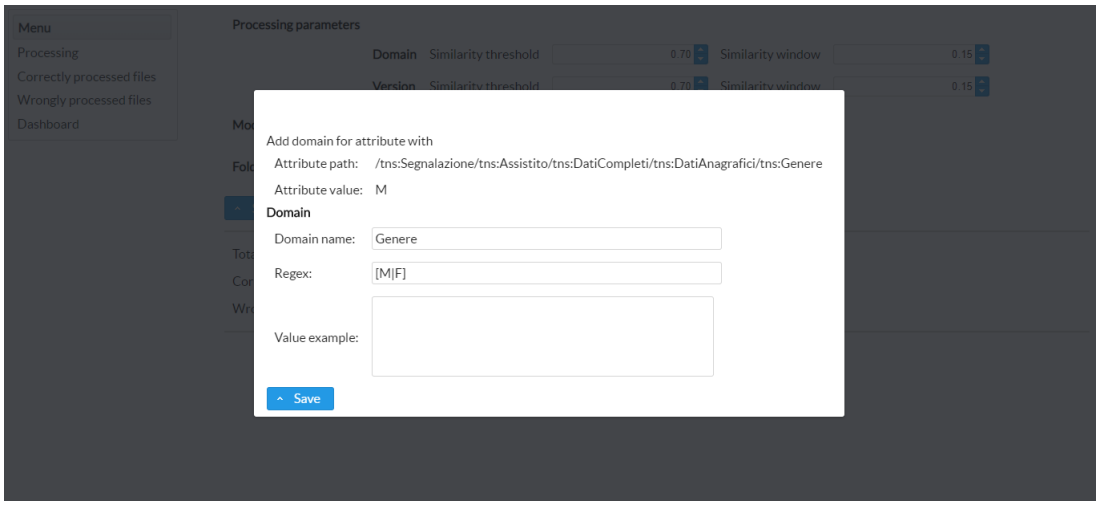

Figura 3.3: Dialog di creazione nuovo dominio

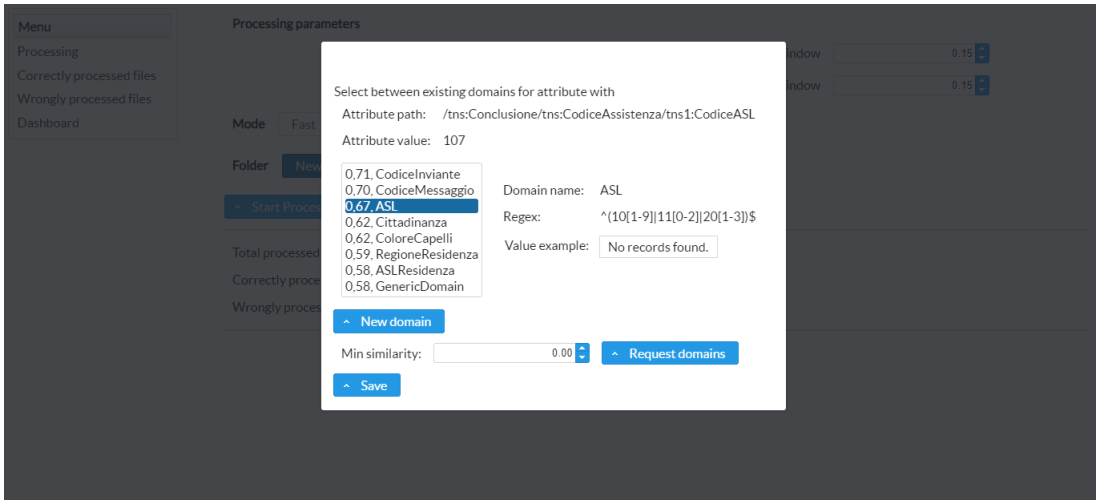

Figura 3.4: Dialog di selezione del dominio

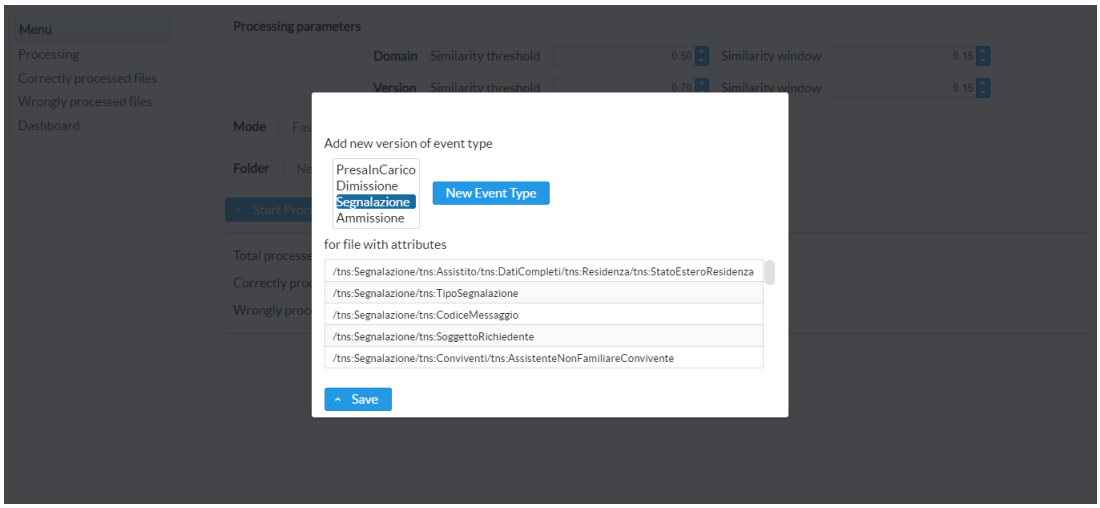

Figura 3.5: Dialog di creazione nuova versione

Esito del processing Navigando sulle pagine "Correctly processed files" e "Wrongly processed files" è possibile visualizzare l'esito del processo di estrazione per i file processati correttamente e non. Per i file processati correttamente si riporta la versione ed il tipo di evento a cui il file è stato associato (Figura 3.6). Per i file processati erroneamente si riporta l'errore che si è verificato ed ha impedito la corretta estrazione dei metadati (Figura 3.7).

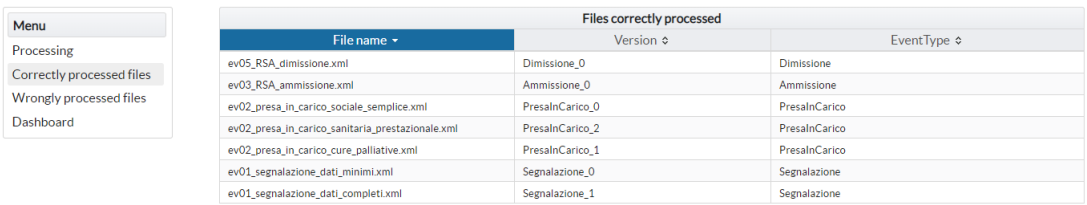

Figura 3.6: Pagina dei file processati correttamente

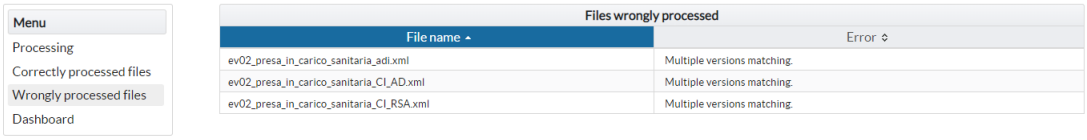

Figura 3.7: Pagina dei file processati erroneamente

Interrogazione dell'ontologia Navigando sulla pagina "Dashboard" è possibile cercare tra i file per i quali sono stati estratti i metadati quelli che soddisfano alcuni criteri di ricerca. In particolare, tramite SPARQL, è possibile interrogare l'ontologia per ottenere i file che sono caratterizzati da un tipo di evento (Figura 3.8 e Listato 3.1) e file che sono caratterizzati da un certo dominio (Figura 3.9 e Listato 3.2).

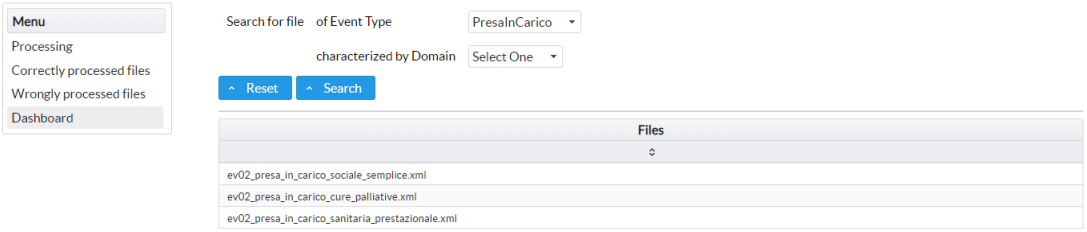

Figura 3.8: Ricerca file caratterizzati da un tipo di evento

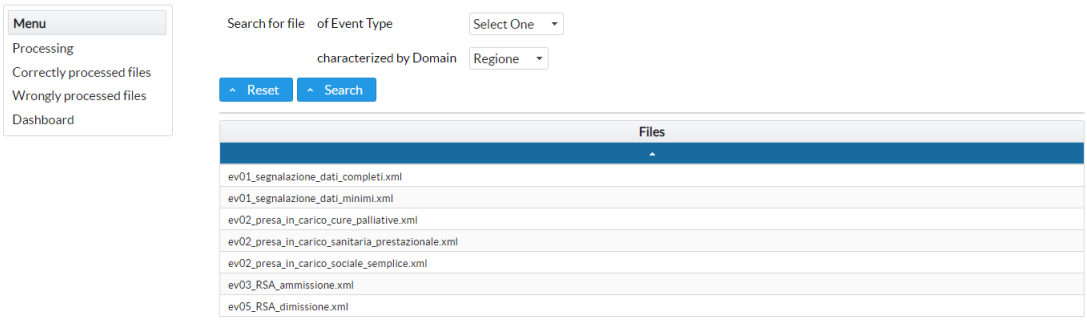

Figura 3.9: Ricerca file caratterizzati da un dominio

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ontology: <http://ontology#>

SELECT ?file
WHERE {
   ?file rdf:type ontology:File .
   ?file ontology:hasVersion ?version .
   ?version ontology:hasEventType ontology:PresaInCarico .
}
```

Listato 3.1: Esempio di query SPARQL per recuperare i file aventi tipo di evento PresaInCarico

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ontology: <http://ontology#>

SELECT ?file
WHERE {
   ?file rdf:type ontology:File .
   ?file ontology:hasVersion ?version .
   ?version ontology:hasDomain ontology:Regione .
}
```

Listato 3.2: Esempio di query SPARQL per recuperare i file caratterizzati dal dominio Regione

3.4 Test

I test che seguono sono stati eseguiti a partire da un'ontologia inizialmente popolata da un insieme di attributi e domini che fanno riferimento a versioni relative a quattro tipi di evento: segnalazione, presa in carico, ammissione e dimissione. Per ognuno di questi tipi di evento è stata aggiunta all'ontologia la versione più breve che li caratterizza (in termini di tag XML distinti che contiene). Le informazioni relative alle proprietà di attributi e domini sono state inserite manualmente e ricavate a partire dai file XSD, contenuti all'interno delle RFC 115 e 118, che descrivono lo schema dei tipi di evento per l'ambito scelto.

Le informazioni messe a disposizione da parte dell'azienda per l'esecuzione dei test sono limitate e corrispondono ad un totale di 9 tipi di evento e 25 file a questi relativi. Tale varietà e numero di file è sufficiente per condurre i test di efficacia ma non per l'esecuzione dei test di efficienza: in questo ultimo caso si è provveduto alla replicazione dei file.

Efficacia Al fine di valutare l'efficacia dell'approccio proposto è stata calcolata la similarità media con cui un file analizzato viene assegnato ad un tipo di evento. Il test consiste nel calcolare una media delle similarità distinguendo tra quella del tipo di evento corrispondente (indipendentemente dalla soglia minima settata e dal fatto che sia quello con similarità massima) e le similarità degli altri tipi di evento. Se un tipo di evento compare in più versioni, viene mantenuta la similarità più alta rispetto ad ogni tipo di evento.

I file processati per eseguire il calcolo sono diversi ma hanno come tipo di evento un tipo già presente nell'ontologia, inizialmente popolata. Solo uno dei file coincide perfettamente con la versione con cui è stata popolata l'ontologia mentre le altre versioni hanno un numero di tag XML più elevato. Il risultato che è stato ottenuto è che il tipo di evento che ha similarità maggiore è, in ogni caso, il tipo di evento corretto per il file processato. Ciò nonostante, come si può vedere nel risultato in Tabella 3.1, la similarità media rispetto al tipo di evento corretto non è così elevata.

Similarità media rispetto al tipo di evento corretto	0.58
Similarità media rispetto agli altri tipi di evento	0.18

Tabella 3.1: Similarità medie rispetto ai tipi di evento

La situazione descritta poco fa accade perché, in determinate circostanze, i file processati contengono un numero di tag molto maggiore rispetto alla versione presente nell'ontologia per lo stesso tipo di file. Questo fa si che la similarità complessiva risulti bassa. Avendo però un nucleo centrale di attributi che matcha perfettamente con quelli della versione corretta, la versione corretta rimane la più simile. Un esempio di questo è visibile in Tabella 3.2.

Abbreviazioni utilizzate
/PIC = /tns:PresaInCarico
/CA = /tns:CodiceAssistenza
/S = /tns:Sanitaria
/CP = /tns:CurePalliative
/VCP = /tns:ValutazioneCurePalliative
/BR = /tns:BisogniRilevati
TE = Tipo di Evento

Tag file da processare	**Match più elevato (Attributo o Dominio)**	**Sim.**	**# TE**	**TE corretto**
/PIC/tns:CodiceInviante	/PIC/tns:CodiceInviante	1	1	Si
/PIC/tns:DataInvio	/PIC/tns:DataInvio	1	1	Si
/PIC/tns:CodiceMessaggio	/PIC/tns:CodiceMessaggio	1	1	Si
/PIC/CA/tns:CodiceASL	/PIC/CA/tns:CodiceASL	1	1	Si
/PIC/CA/tns:CodiceRegione	/PIC/CA/tns:CodiceRegione	1	1	Si
/PIC/CA/tns:CodiceZona	/PIC/CA/tns:CodiceZona	1	1	Si
/PIC/CA/tns:CodiceProgetto	/PIC/CA/tns:CodiceProgetto	1	1	Si
/PIC/tns:DataPIC	/PIC/tns:DataPIC	1	1	Si
/PIC/S/CP/tns:TipoPercorso	TipoPercorso	1	1	Si
/PIC/S/CP/tns:Terminalita	AreaFinanziaria	0,7	1	No
/PIC/S/CP/VCP/tns:DataValutazione	Date	0,63	4	Si
/PIC/S/CP/VCP/SS1	CodiceInviante, CodiceMessaggio, GenericDomain, Float, Int	0,5	4, 4, 4, 1, 1	Si, Si, Si, No, No
/PIC/S/CP/VCP/tns:TipologiaCaregiver	TipologiaDimissione	0,76	1	No
/PIC/S/CP/VCP/tns:IndiceKarnofsky	CodiceInviante, CodiceMessaggio, Intero2Cifre	0,63	4, 4, 1	Si, Si, No
/PIC/S/CP/VCP /tns:ConsapevolezzaPazienteDiagnosi	TipoSegnalazione	0,65	1	No
/PIC/S/CP/VCP /tns:PatologiaResponsabile	CodiceInviante, CodiceMessaggio	0,62	4, 4	Si, Si
/PIC/S/CP/VCP/BR /tns:RischioInfettivo	EsitoPrimaLetturaBisogno, Iniziativa, TitoloStudio	0,63	1, 1, 1	No, No, No
/PIC/S/CP/VCP/BR /tns:BroncoaspDrenaggioPosturale	ZonaDistrettoAmmissione	0,63	1	No
/PIC/S/CP/VCP/BR /tns:GestOssigenoTerap	GenericDomain, PresenteAssente	0,62	4, 1	Si, No

Tabella 3.2: Esempio di file per cui la similarità complessiva risulta bassa

Il file processato corrisponde ad una presa in carico e la similarità rispetto al tipo di evento a cui corrisponde, che risulta dal processing in modalità automatica è 0,23, quindi piuttosto bassa. Il file contiene in totale 51 attributi. Di questi ne sono stati riportati in tabella solo alcuni, a titolo esemplificativo. In totale, 11 attributi hanno un match perfetto con attributi già presenti

nell'ontologia (8 in tabella), 1 ha un match perfetto con un dominio esistente (TipoPercorso) ed 1 ha un match elevato con il dominio corretto (Date) che, tuttavia, non viene assegnato all'attributo utilizzando la soglia di default per il calcolo della similarità tra attributi e domini (0,7). I restanti attributi, invece, hanno un match con un dominio sbagliato: per alcuni di questi, come i 30 attributi (3 in tabella) che contengono nel path il tag BisogniRilevati, l'attributo corretto esiste nell'ontologia (SiNo) ma non viene assegnato. Si evidenzia quindi un limite nell'approccio proposto che contribuisce a rendere bassa la similarità del file processato rispetto al tipo di evento corretto. Questo limite risiede nella modalità con cui viene calcolata la similarità degli attributi rispetto ai domini perché la similarità tra nome dell'attributo e nome di dominio ha lo stesso peso della similarità tra valore dell'attributo ed espressione regolare o valori di esempio che descrivono il dominio. Una possibile soluzione potrebbe essere dare un peso variabile all'espressione regolare in funzione di quanto sia significativa: alcuni domini, che hanno un formato particolare e facilmente riconoscibile (ad esempio Date), vengono penalizzati dal peso dato alla similarità sul nome, che può essere anche molto bassa. In questi casi sarebbe opportuno dare un peso maggiore alla similarità sull'espressione regolare.

Un altro test effettuato per valutare l'efficacia dell'approccio proposto è stato aggiungere progressivamente elementi di disturbo ad un file che ha una corrispondenza esatta con una versione nell'ontologia, senza andare a salvare il risultato. Gli elementi di disturbo sono presi da un altro tipo di evento. Nell'esempio in Tabella 3.3 sono stati aggiunti alla versione esistente di presa in carico elementi di disturbo provenienti dalla versione di dimissione con cui è stata popolata l'ontologia.

Attributo aggiunto	**Dominio a similarità più elevata**	**Dominio corretto**	**Similarità PresaInCarico**	**Similarità Dimissione**
-	-	-	1	0,458
/tns:CodiceEventoDimissione	CodiceEvento (0,77)	Si	0,923	0,462
/tns:CodiceEventoAmmissione	CodiceEvento (0,77)	Si	0,857	0,464
/tns:DataDimissione	Date (0,64)	Si	0,83	0,467
/tns:GiorniPresenza	GenericDomain (0,57)	No (Int)	0,813	0,469
/tns:TipologiaDimissione	TipologiaDimissione (1)	Si	0,765	0,471

Tabella 3.3: Esempio di aggiunta progressiva di elementi di disturbo ad un file con tipo di evento PresaInCarico da un file con tipo di evento Dimissione

L'andamento della similarità che ne deriva è visibile in Figura 3.10. Si può notare come la similarità rispetto al tipo di evento originale decresca ma rimanga comunque più elevata rispetto al tipo di evento di disturbo, la cui similarità cresce in maniera molto meno marcata. Questo fenomeno è principalmente dovuto alla radice del file XML (/tns:PresaInCarico), che rappresenta anche il tipo di evento e, nonostante gli altri attributi che descrivono entrambi i tipi di file siano associati agli stessi domini, gli attributi che nel path contengono la radice hanno un match perfetto solo con quelli del tipo di evento che viene esteso. Alla luce di questo, e tenendo in considerazione come viene calcolata la similarità di un file rispetto alle versioni, il file esteso viene sempre associato al tipo di evento corretto.

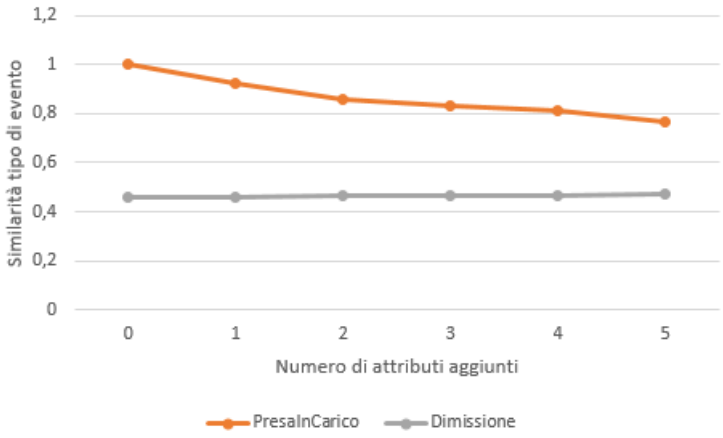

Figura 3.10: Andamento della similarità all'aggiunta di elementi di disturbo a PresaInCarico da Dimissione

Performance Questo test è stato eseguito per verificare le performance del processo di estrazione dei metadati in funzione di un numero crescente di file. In assenza di dati reali, i file sono stati creati duplicando uno di quelli a disposizione, presente tra gli esempi contenuti all'interno delle RFC. Il file che è stato duplicato corrisponde alla versione di segnalazione con cui è stata popolata l'ontologia. Le performance dell'approccio proposto, approssimate in secondi e riportate in Tabella 3.4, sono state calcolate in relazione al processing di 10, 100, 1.000 file aventi una dimensione di qualche KB, tenendo in considerazione diversi aspetti.

- Il processo di estrazione dei metadati viene eseguito in maniera *sequenziale*.

- Il deploy dell'applicazione web è avvenuto in *locale* pertanto i tempi riportati sono collegati alle caratteristiche del PC sul quale il test è stato eseguito ed alle caratteristiche del cluster. Il PC sul quale è stato eseguito è un Lenovo ThinkPad E540 con Windows 10 Pro 64bit, 8GB di RAM, processore Intel Core i5-4210M CPU @ 2.6 GHz. Il cluster è un cluster virtuale di 10 nodi; ogni nodo è equipaggiato con una CPU quad-core, 16GB di RAM, 1TB di disco; i nodi sono organizzati in due rack: la velocità intra-rack è 6GB circa mentre quella extra-rack 900MB; la distribuzione di Hadoop in esecuzione sul cluster è Cloudera Express 5.13.1.

- Il tempo totale comprende il tempo di processing e di input/output (IO); il tempo di IO comprende, a sua volta, il tempo di lettura/scrittura su HDFS e quello di lettura/scrittura dell'ontologia su file (non sul modello che viene invece acceduto durante tutta la fase di estrazione).

		Numero file		
		10	**100**	**1.000**
Secondi	*Totale*	7	37	418
	Processing	7	33	303
	IO	0	4	115

Tabella 3.4: Performance del processo di estrazione dei metadati per numero di file

Com'è possibile vedere dal grafico in Figura 3.11 mano a mano che il numero di file da processare aumenta il tempo totale aumenta in maniera non lineare e si assiste ad una degradazione delle performance non indifferente. Provando a testare l'approccio con 10.000 file il tempo totale d'esecuzione del processo ha superato le due ore. Questo avviene perché, come accennato nella Sottosezione 1.1.2, HDFS non è ottimizzato per l'accesso a tanti file di piccole dimensioni: l'accento è posto sulla velocità di lettura più che sulla bassa latenza di accesso ai file.

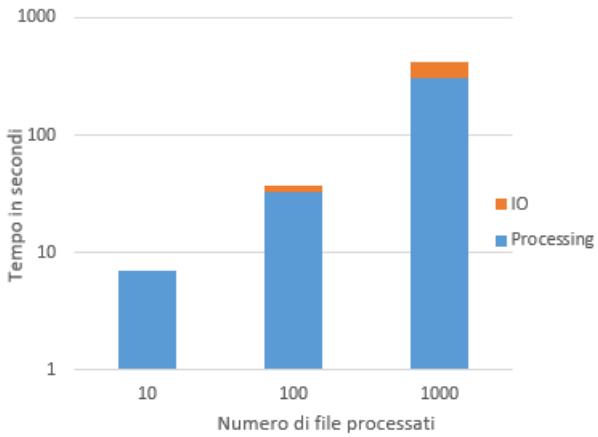

Figura 3.11: Grafico relativo alle performance del processo di estrazione dei metadati per numero di file

Il prototipo sviluppato è focalizzato più sugli aspetti di processo e di efficacia. Tuttavia, l'aspetto di performance è importante per un'eventuale messa a regime del sistema. In tal caso, è possibile adottare diverse soluzioni tra quelle proposte dalla comunità di Hadoop.

- Per accedere con bassa latenza ai dati memorizzati sul Data Lake è possibile utilizzare HBase. Il suo uso tuttavia è sconsigliato perché, non rimpiazzando il file system ma costituendo semplicemente un livello sopra HDFS, non può essere utilizzato per implementare il Data Lake.

- E' possibile combinare tanti piccoli file in un unico *Sequence file*, un formato di file binario che consiste di coppie chiave-valore. L'utilizzo di sequence file per combinare tanti piccoli file dovrebbe essere proposta fin da subito per memorizzare diversi file XML in un file unico, non solo per impacchettarli durante il processing. I sequence file, inoltre, permettono di ridurre il numero di interazioni IO con Hadoop passando allo scenario ideale per HDFS in cui i file memorizzati al suo interno sono pochi ed hanno una dimensione molto grande.

- La soluzione migliore è utilizzare un *object store* per Hadoop come *Apache Hadoop Ozone*. Apache Hadoop Ozone [40] è un nuovo sotto-progetto open-source di Hadoop in fase di sviluppo la cui prima alpha-release risale all'Ottobre 2018. Essendo Ozone un object store i dati al suo interno

vengono manipolati come oggetti, identificati da chiavi, a differenza di HDFS che li manipola come blocchi. Oggetti e chiavi sono organizzati in *buckets* ognuno dei quali ha un nome definito dall'utente. Le chiavi sono uniche all'interno di ogni bucket. Anche il sistema di metadati di Ozone, come i namespace di HDFS, è costruito sopra un livello di archiviazione a blocchi, Hadoop Distributed Data Storage (HDDS), tuttavia i suoi metadati, saranno distribuiti dinamicamente per supportare un elevato numero di bucket.

Anche il processo di estrazione dei metadati sviluppato può essere ottimizzato: è possibile aumentare le capacità del PC sul quale il processo viene eseguito, distribuire il processo di estrazione, gestendo in maniera adeguata la modifica concorrente dell'ontologia, oppure mantenere in memoria l'ontologia, evitando di salvarla dopo il processing di ogni singolo file e ricaricarla in caso di errori.

Conclusioni e Sviluppi futuri

Il lavoro svolto in questa tesi si colloca nella fase iniziale di un processo di reingegnerizzazione dei sistemi aziendali volto a predisporre questi ultimi ad una corretta gestione dei Big Data. L'obiettivo principale del lavoro svolto è stato lo sviluppo di un approccio estendibile e parametrizzabile che permettesse di indicizzare, nella maniera più automatica possibile, i singoli file presenti nel Data Lake sulla base del loro tipo di evento tramite l'estrazione di metadati strutturali, di processo e semantici, che consentissero e facilitassero il recupero dei file quando rilevanti rispetto a determinate interrogazioni. I file che il caso di studio ha preso in esame sono file XML che fanno riferimento ad eventi generati in ambito sanitario, con un focus particolare sugli eventi relativi all'AD/RSA. Per portare a termine quest'obiettivo è stato necessario definire un'architettura per Data Lake ed un modello per l'archiviazione di metadati che fosse scalabile e flessibile e potesse fungere da base per tutti i sistemi aziendali: un'ontologia. I risultati sono resi disponibili tramite un'interfaccia web.

I risultati che sono stati ottenuti a seguito dei test sono diversi. In primo luogo è stata evidenziata l'efficacia del processo di estrazione che, nonostante la diversità nel numero di tag contenuti nel file da processare rispetto alla versione presente nell'ontologia per lo stesso tipo di file, riesce comunque a rilevare come tipo di evento più simile il tipo di evento corretto a causa di un nucleo centrale di attributi che matcha perfettamente con quelli della versione del tipo di evento corretto. L'efficacia del processo è stata verificata anche tramite l'aggiunta di elementi di disturbo, presi da un altro tipo di evento, ad un file che ha una corrispondenza esatta con una versione inserita nell'ontologia. In questo caso è possibile notare come la similarità del file rispetto al tipo di evento originale decresca ma rimanga comunque più elevata rispetto al tipo

di evento di disturbo, la cui similarità cresce in maniera molto meno marcata: nonostante gli attributi che descrivono entrambi i tipi di file matchino gli stessi domini, gli attributi che nel path contengono la radice, che identifica il tipo di evento, hanno un match perfetto solo con quelli del tipo di evento che viene esteso. In secondo luogo sono state misurate le performance del processo di estrazione dei metadati in funzione di un numero crescente di file. Dai risultati ottenuti è emerso il principale limite di HDFS: non essere ottimizzato per l'accesso a tanti file di piccole dimensioni. È consigliato accorpare più file in un unico Sequence file o rivolgersi a soluzioni di storage alternative, come gli object store. Relativamente al processo di estrazione sono emersi limiti legati alla natura sequenziale del processing ed all'elevato numero di operazioni di IO che viene eseguito: in questo caso si può pensare di distribuire il processo di estrazione e mantenere l'ontologia in memoria evitando di leggerla/scriverla da/su file al termine di ogni singolo processing. Un limite che invece è emerso dai test di efficacia riguarda la modalità di calcolo adottata per associare un attributo ad un dominio che può contribuire ad abbassare la similarità rispetto al tipo di evento corretto. Una possibile soluzione potrebbe essere dare un peso variabile alla similarità calcolata sull'espressione regolare in funzione di quanto sia significativa, anziché dare a questa un peso pari alla similarità calcolata sul nome del dominio.

In seguito vengono evidenziati alcuni possibili sviluppi futuri. Il processo proposto all'interno del lavoro di tesi è volto ad indicizzare file XML tuttavia, come nell'approccio proposto GEMMS [21], un livello di intelligenza che salva metadati all'interno della stessa ontologia può potenzialmente essere creato per ogni tipo di file che il sistema deve essere in grado di processare. L'intelligenza finisce dove finisce la casistica considerata: se la complessità è maggiore dell'intelligenza il processo di estrazione non associa metadati al file. Poiché come evidenziato in GOODS [19] indicizzare singoli file può diventare molto costoso, si potrebbe decidere di indicizzare al loro posto cartelle che contengono versioni dello stesso tipo di evento, definendo l'appartenenza del file ad un tipo di evento sulla base della cartella che lo contiene. Si può pensare, inoltre, di eseguire il deploy dell'applicazione sul cluster di Hadoop rendendo disponibile l'interfaccia grafica utilizzata per settare i parametri del processo di estrazione su un nodo ad una certa porta.

Ringraziamenti

Desidero ringraziare tutti quelli che, a loro modo, mi hanno supportato durante questo percorso universitario e nella realizzazione della tesi.

Ringrazio in primo luogo il Dott. Enrico Gallinucci ed il Prof. Matteo Golfarelli per la loro pazienza, disponibilità ed i preziosi consigli a me dati durante tutto il periodo di sviluppo del progetto. Ringrazio anche Vladimiro Buda, Vincenzo Tresoldi e tutti i colleghi di Onit Group S.r.l. che mi hanno accolto fin dal primo giorno come una di famiglia e mi hanno aiutata a calarmi nelle dinamiche aziendali e nel dominio scelto come caso di studio nel progetto di tesi.

Ringrazio i miei amici, per avermi accompagnato in questi anni universitari, sempre presenti nei bei momenti ed in quelli più difficili.

Ringrazio tutta la mia famiglia ed in modo particolare i miei genitori, Massimo e Stefania, per avere sempre creduto in me e nelle mie capacità, spesso più di quanto facessi io stessa.

Anna Giulia Leoni

Elenco delle figure

Elenco delle tabelle

Listings

Bibliografia

[1] McKinsey Global Institute, *Big Data: The nextfrontier for innovation, competition and productivity*, Giugno 2011

[2] Gartner Group, *IT Glossary Big Data*, `https://www.gartner.com/it-glossary/big-data`

[3] Gandomi A., Murtaza H., *Beyond the hype: Big data concepts, methods, and analytics*, Dicembre 2014

[4] IDC White Paper, *Data Age 2025: The Evolution of Data to Life-Critical*, Aprile 2017

[5] *The 4 V's of big data*, `https://www.atg.world/view-article/The4Vsofbigdata-2321`, Maggio 2016

[6] Rabiul Islam J., Rakibul Islam R., Musfiqur R., *Big Data Characteristics, Value Chain and Challenges*, Maggio 2016

[7] Sivarajah U., et al., *Critical analysis of Big Data challanges and analytical methods*, Agosto 2016

[8] *Apache Hadoop*, `https://hadoop.apache.org/`

[9] Huang F., *Managing Data Lakes in Big Data Era*, Giugno 2015

[10] Miloslavskaya N. G., Tolstoy A., *Big Data, Fast Data and Data Lake Concepts*, Dicembre 2016

[11] O'Reilly report, Castanedo F., Gidley S., *Understanding Metadata: Create the Foundation for a Scalable Data Architecture*, Febbraio 2017

[12] *Data Lake vs. Data Swamp: pushing the analogy*, https://www.collibra.com/blog/blogdata-lake-vs-data-swamp-pushing-the-analogy/, Febbraio 2017

[13] Berners-Lee T., Hendler J., Lassila O., *The Semantic Web*, Maggio 2001

[14] Antoniou G., Groth P., van Harmelen F., Hoekstra R., *A Semantic Web Primer*, 2012

[15] *RDF*, https://www.w3.org/RDF/

[16] *SPARQL*, https://www.w3.org/2001/sw/wiki/SPARQL

[17] *RDFS*, https://www.w3.org/2001/sw/wiki/RDFS

[18] *OWL*, https://www.w3.org/OWL/

[19] Halevy A., Korn F., Noy N. F., Olston C., Polyzotis N., Roy S., Whang S. E., *Managing Google's data lake: an overview of the GOODS system*, 2016

[20] Hai R., Geisler S., Quix C., *Constance: An Intelligent Data Lake System*, Giugno 2016

[21] Quix C., Hai R., Vatov I., *GEMMS: A Generic and Extensible Metadata Management System for Data Lakes*, Giugno 2016

[22] Maccioni A., Torlone R., *Crossing the finish line faster when paddling the Data Lake with KAYAK*, Agosto 2017

[23] Maccioni A., Torlone R., *KAYAK: A Framework for Just-in-Time Data Preparation in a Data Lake*, Gennaio 2018

[24] O'Reilly report, Oram A., *Managing the Data Lake*, Settembre 2015

[25] Diamantini C., Lo Giudice P., Musarella L., Potena D., Storti E., Ursino D., *A New Metadata Model to Uniformly Handle Heterogeneous Data Lake Sources*, Settembre 2018

[26] *Teradata Kylo*, https://kylo.io/

[27] *Apache NiFi*, https://nifi.apache.org/

[28] *Apache Atlas*, https://atlas.apache.org/

[29] *Apache Ranger*, https://ranger.apache.org/

[30] *DataIku*, https://www.dataiku.com/

[31] *Azure Data Lake*, https://azure.microsoft.com/it-it/solutions/data-lake/

[32] *AWS Data Lake Solution*, https://docs.aws.amazon.com/en_us/solutions/latest/data-lake-solution/overview.html

[33] *Google Data Lake Solution*, https://cloud.google.com/solutions/build-a-data-lake-on-gcp

[34] *eToscana compliance ricerca RFC*, http://web.rete.toscana.it/eCompliance/portale/mostraCercaRFC

[35] Manning C., Schutze H., Raghavan P., *Introduction to Information Retrieval*, 2008

[36] *Protégé*, https://protege.stanford.edu/

[37] *Apache Jena*, https://jena.apache.org/

[38] *Java Server Faces*, http://www.javaserverfaces.org/

[39] *WildFly*, http://wildfly.org/

[40] *Apache Hadoop Ozone*, https://hadoop.apache.org/ozone/

[41] *Ozone Object Store*, https://it.hortonworks.com/blog/ozone-object-store-hdfs/, 14 Ottobre 2014