ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Tesi di Laurea
in
Reti di Calcolatori M

Progetto e Strumenti di Federazione di Endpoint SPARQL per Enterprise Architecture

Candidato:

Francesco Maria Sprotetto

Relatore:

Chiar.mo Prof. Ing.
Antonio Corradi

Correlatori:
Dott. Mirco Casoni
Ing. Maria Seralessandri
Dott. Luca Assirelli

Anno Accademico 2014/2015

Sessione I

Indice

Indice delle Figure

Indice delle Tabelle

Introduzione

Lo scopo dell'Enterprise Architecture è quello di fungere da supporto per le strategie di Business, fornendo strumenti e informazioni utili e di supporto alla funzione del decision-macking.
Obiettivi significativi nel contesto IT consistono nel definire processi di raccolta ed elaborazione delle informazioni utili a garantire una evoluzione sostenibile e in linea con gli obiettivi business del sistema informativo.
Il risultato finale, almeno in teoria, è che l'architettura sarà meno costosa, strategicamente più efficace e più reattiva ai cambiamenti nel tempo.
Il Semantic Web permette di gestire i contenuti web in modo molto innovativo. Non separato dal Web "tradizionale", introduce il significato dei dati, ne è un'estensione che aggiunge una nuova funzionalità alle macchine, che diventano in grado di comprendere ed elaborare i dati, che fino ad ora semplicemente visualizzavano.
A nostro parere, il Semantic Web risulta essere una disciplina abilitante all'Enterprise Architecture, soprattutto nella gestione della grande mole di dati non strutturati che rappresentano un'azienda e che sono prodotti dall'azienda stessa, attraverso un linguaggio standard (W3C) chiamato SPARQL.
Questo supporto è molto importante nell'ambito del Data Analysis, cioè la disciplina che si occupa di analizzare la grande mole di dati prodotti dalle aziende. Infatti uno degli obiettivi principali di questa disciplina è quello di produrre dei documenti riassuntivi, ad esempio i report, che possano essere di supporto ai manager nel prendere decisioni strategiche per l'azienda.

L'oggetto di studio di questa tesi si focalizza sull'aspetto di federazione, in quanto significativa nell'ambito Enterprise Architecture, ma ancora non sviluppata nel contesto delle ontologie e del Semantic Web, il quale però offre la possibilità di fare interrogazioni distribuite.
Partendo da questo presupposto l'obiettivo è stato quello di elaborare un modello logico e fisico che permettesse la creazione di una federazione di endpoint SPARQL.

Tale modello quindi dovrà essere accettato dai singoli componenti della federazione, i quali esporranno i propri dati in un formato condiviso e accessibile.

La tesi è composta di 8 capitoli, così suddivisi:

Capitolo 1 - Background e motivazioni:

Un'introduzione all'ambito dell'IT Operation e dell'IT Governance. Una successiva definizione del concetto di Enterprise Architecture e di Semantic Web, ed infine un approfondimento sulle specifiche, tecnologie e framework utilizzati nello sviluppo dell'elaborato di tesi.

Capitolo 2 – Data Governance e Ownership:

Definizione del concetto di Ownership, con conseguente distinzione e approfondimento sulle differenze tra Ownership dei dati e Ownership del modello.

Capitolo 3 – Federazione di endpoint SPARQL: Software selection.

Definizione e discussione sul concetto della *Federazione di Endpoint SPARQL*, soprattutto nell'ambito Enterprise Architecture. Panoramica sulle specifiche attuali relative allo SPARQL 1.1 ed infine una discussione e comparazione tra i middleware che implementano la federazione.

Capitolo 4 – Federazione di endpoint SPARQL: Architettura logica.

Definizione e spiegazione delle ipotesi di lavoro individuate. Successiva comparazione tra le varie ipotesi con giustificazione della scelta finale.

Capitolo 5 – Caso sperimentale: ontologie e modello dati

Realizzazione di un'ontologia ad hoc per la sperimentazione e definizione dei relativi endpoint utilizzati. Spiegazione dettagliata del modello dati.

Capitolo 6 – Caso sperimentale: soluzione applicativa

Dettagliata spiegazione e giustificazione della realizzazione dell'architettura logica scelta come ipotesi di lavoro, evidenziando la struttura logica e fisica, la sequenza delle chiamata, i test effettuati e lo stack software utilizzato.

Capitolo 7 – Enterprise Architecture e analisi dei dati. Casi d'uso di scrittura dati

Discussione sull'importanza del Data Analisys nell'ambito dell'EA, con evidente argomentazione sul processo ETL e sui benefici che il modello federato, da noi sviluppato, può portare a questa tipologia di attività, in termini di modello, di standard utilizzabili via world wide, e soprattutto in termini di performance.

Approfondimento sui casi d'uso relativi alla possibilità di scrivere dati su endpoint federati, analizzando quali possono essere i problemi presenti in questa tipologia di feature.

Capitolo 8 – Conclusioni: Risultati ottenuti e proposte future

Argomentazione dei risultati ottenuti dalla sperimentazione effettuata. Inoltre due possibili proposte di implementazioni, sia in termini di gestione della federazione, sia in termini di costrutto per lo standard SPARQL 1.1.

1. Background e motivazioni

Il seguente capitolo spiega i concetti principali della disciplina dell'Enterprise Architecture e del Semantic Web, evidenziando in particolare quali tecnologie risultano attualmente abilitanti, e quindi a supporto, dell'Enterprise Architecture. Verranno inoltre analizzate tecnologie centrali nel Semantic Web, quali le ontologie, ed infine tutte i framework analizzati e utilizzati per la sperimentazione .

1.1 IT Operation e IT Governance

"L'IT Governance è un insieme di competenze e responsabilità del consiglio di amministrazione e del management esecutivo. È parte integrante della politica aziendale ed è costituita dalla direzione, dalla struttura organizzativa e dai processi in grado di assicurare che l'IT sostenga ed estenda gli obiettivi e le strategie dell'organizzazione" [MAR06][SAS13].

Inserita all'interno di un più ampio modello organizzativo che definisce un insieme di regole a ogni livello, che disciplina la gestione aziendale, le relazioni tra i vari attori e gli obiettivi che l'impresa si pone, l'IT Governance è diventata uno dei principali processi decisionali dell'impresa sostenendo il raggiungimento degli obiettivi, adeguandosi dinamicamente ad ogni nuova esigenza del mercato.

L'IT Governance, in particolare, si occupa di:

- ***Strategic Alignment/Allineamento Strategico:*** per garantire che gli obiettivi aziendali vengano supportati correttamente dall'IT;
- ***Value Delivery/Erogazione del Valore:*** per assicurare che l'IT produca un ritorno economico;
- ***Risk Management/Gestione del Rischio:*** per gestire i rischi degli investimenti IT;

- ***Resource Management/Gestione delle Risorse:*** garantire che vengano rispettati i requisiti preposti con il minor utilizzo di risorse (umane ed economiche);
- ***Performance Measurement/Misurazione delle Performance:*** valutare le prestazioni del sistema di IT.

Figura 1 - Le aree dell'IT Governance.

L'*IT Governance*, quindi, non è da intendersi né come un mero sistema di osservazione e valutazione, né come un regolamento che stabilisce esclusivamente ruoli e responsabilità. È piuttosto un processo che sfrutta direttive, standard e strategie affinché il settore dell'IT di un'azienda risulti efficace ed efficiente.

Per ciò che riguarda l'*IT Operation*, nel settore IT esistono molte definizioni diverse, motivate dal fatto che i venditori e le singole organizzazioni creano spesso le proprie definizioni personalizzate di tali processi e dei servizi ai fini della commercializzazione dei propri prodotti.
Una delle possibili definizioni dell'*IT Operation* consiste nelle attività di definizione e applicazione dei processi di Service Management che consentono all'IT di essere un abilitatore per il Business, permettendo una forte integrazione tra le strategie corporate e la creazione di valore tramite l'erogazione dei servizi IT.
Tipicamente l'IT Operation incluse la gestione, pianificazione, progettazione, realizzazione, la costruzione, la distribuzione, la verifica, l'installazione di istanze,

l'esecuzione e la manutenzione. Essi si sforzano di definire processi comuni e le procedure, le politiche, i ruoli, le responsabilità, la terminologia, le best practice e gli standard per la gestione di un'impresa.

In questi termini risulta molto importante il supporto dell'Enterprise Architecture, soprattutto nell'ambito della gestione aziendale, con l'obiettivo di fornire una visione più ampia e strutturata dell'azienda, in particolare nella struttura e nell'impiego delle proprie risorse IT [MAR06][SAS13][PRI09].

1.1.1 Data Quality

Il Data Quality può essere definito come la pratica che garantisce che il dato sia accurato e utilizzabile per lo scopo previsto. Proprio come nella gestione della qualità ISO 9000 nei processi di produzione, gli strumenti di Data Quality andrebbero sfruttati in ogni fase del ciclo di produzione e distribuzione dei dati. Tale ciclo inizia nel momento in cui il dato "entra" in azienda, o vi si accede attraverso flussi di caricamento dei datawarehouse, considera i vari punti di integrazione o scambio/arricchimento dei dati, fino al punto immediatamente precedente a dove il dato sarà caricato all'interno dei sistemi di destinazione.
La qualità dei dati è un elemento imprescindibile per aumentarne l'efficacia nei processi aziendali [MAR06][SAS13].
La qualità del dato è legata strettamente a *cosa voglio dal dato*, ad esempio analizzando un'applicazione, vengono stabiliti i requisiti principali e secondari, e si cerca di soddisfarli in pieno. Se vi è il soddisfacimento dei soli requisiti core, la qualità del dato risulta essere bassa; al contrario, se vi è il soddisfacimento di tutti i requisiti, la qualità del dato risulta essere alta.
Uno dei principali scopi del Data Analisys è proprio quello di agire sui dati, tipicamente non strutturati, trasformandoli in informazioni. Queste informazioni vengono successivamente trattate e trasformate a loro volta in conoscenza.

1.1.2 Data Management

Il Data Managment è un insieme di processi e tecnologie che aiutano a definire, uniformare e gestire i dati condivisi da tutte le aree aziendali fornendo una vista dei dati relativi ai clienti, prodotti, fornitori, asset/location. I dati sono gestiti tipicamente in un singolo punto, spesso chiamato “hub”. L’Hub agisce come unico punto di pubblicazione e condivisione di questi dati critici, in modo coerente.
Questi processi assicurano che utenti diversi, in aree e con funzioni diverse, utilizzino la stessa versione dei dati. Senza Data Management, ad esempio, un cliente che stipula una polizza con una compagnia, potrebbe ricevere un’offerta promozionale per la stessa polizza della stessa assicurazione. Questo capita quando il dato relativo al cliente è registrato in modo corretto nel sistema, ma non nella lista dei destinatari di marketing. Quindi è fondamentale disporre di un ambiente che permetta alle organizzazioni di disporre di una “vista unica e certificata” della realtà, garantendo cosi la consistenza del dato di tutti i processi “core” [MAR06] [SAS13].

1.1.3 Risk Management

Il Risk Management è il processo mediante il quale si misura o si stima il rischio e successivamente si sviluppano delle strategie per governarlo.
Si occupano di gestione del rischio sia le grandi imprese che hanno dei team appositi, sia le piccole imprese che praticano informalmente la gestione del rischio.
Di regola, le strategie impiegate includono il trasferimento del rischio a terze parti, l'evitare il rischio, il ridurre l'effetto negativo ed infine l'accettare in parte o totalmente le conseguenze di un particolare rischio. Essa focalizza sui rischi derivanti da cause fisiche o legali come ad esempio, disastri naturali, incendi, morti e processi penali. La gestione del rischio finanziario, invece, focalizza sui rischi governabili usando strumenti di trade finanziario.
Occorre notare che recentemente il concetto di rischio tende ad ampliarsi in "rischio/opportunità", dove insieme ad impatti negativi (minacce) sono associati anche potenziali impatti positivi (opportunità) da perseguire [MAR06] [SAS13].

1.1.4 Data Policies

Data Policies può essere definita come set di linee guida per garantire la corretta gestione delle informazioni di una determinata con particolare riferimento alla sicurezza, qualità e privacy dei dati.
Una Data Policies descrive formalmente come il monitoraggio di attività di business deve essere effettuato per assicurare che i dati siano accurati, accessibili, coerenti e protetti. La politica stabilisce chi è responsabile dei dati e delle informazioni, e quali procedure devono essere utilizzate per gestire queste responsabilità [MAR12].

1.2 Enterprise Architecture

Una corretta definizione dell'Enterprise Architecture è individuabile in una serie di attività dirette a descrivere un'azienda, la sua mission e il suo contesto e tutto ciò che le è necessario per raggiungere gli obiettivi prefissati [FEL13][FRA13][AUE08].
Nel contesto di un'organizzazione si parla di Architettura di Business e di Architettura IT. L'architettura di Business ha come ambito la struttura organizzativa dell'organizzazione, il modello di funzionamento, la mission e gli obiettivi a breve e lungo termine, nonché i processi ed i servizi erogati ai clienti.
L'architettura IT ha come componenti le applicazioni, i servizi tecnologici, i dati gestiti, l'hardware e il software.
L'unione di questi due aspetti costituisce l'Enterprise Architecture di un sistema informativo.
Il ruolo dell'Enterprise Architecture è fondamentale sotto molti aspetti: in primo luogo si focalizza nel raggiungimento di obiettivi, quali allineamento tra IT e Business, fornendo strumenti per analizzare e gestire questi due elementi. In secondo luogo supporta la gestione dell'architettura IT, analizzando i sistemi informativi attraverso punti di vista diversi, non solo funzionali, ma anche tecnici (manutenibilità, estendibilità, riuso, ecc.). Fornisce inoltre strumenti necessari per la definizione di una roadmap di evoluzione dell'architettura, esaminando gli impatti dal punto di vista dei clienti, delle tecnologie e dei costi.

L'Enterprise Architecture può essere utilizzata per analizzare l'intero parco applicativo e tecnologico, identificare ridondanze e sprechi, oppure impostare dei criteri di priorità degli investimenti in relazione agli obiettivi di business.
In un periodo in cui fusioni e acquisizioni trasformano le aziende in entità più grandi e con sistemi disomogenei, l'Enterprise Architecture fornisce una visione generale dei suoi processi di business e delle risorse globali che li supportano.
In conclusione l'EA è una disciplina in grado di apportare benefici alla realtà aziendale, se è condotta nel rispetto di alcune linee guida fondamentali che prevedono impegno del management, coinvolgimento e partecipazione di tutte le componenti organizzative, concretezza degli obiettivi e dei risultati [FEL13][FRA13][AUE08].

In ***figura 2*** è rappresentato un possibile scenario di esempio dell'utilizzo dell'Enterprise Architecture, nel caso di una fusione di due aziende, le quali hanno due sistemi informativi diversi, offrono dei servizi diversi, ma possono avere dei particolari servizi in comune (un tipico esempio di servizi in comune è la gestione del badge).

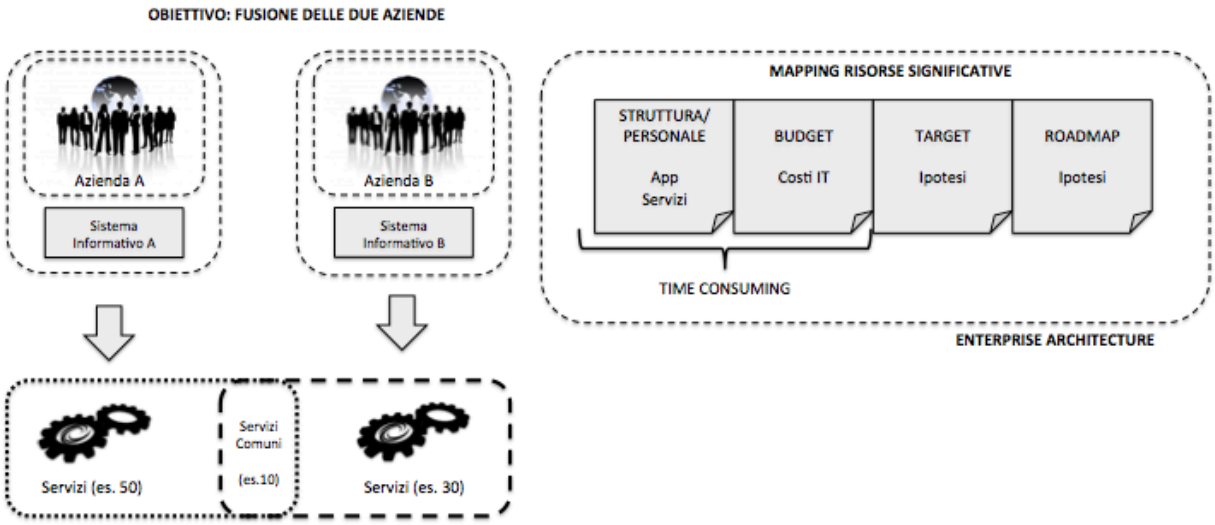

Figura 2 - Fusione aziendale

Le domanda che sicuramente verranno poste sono:

- *deve essere effettuata Fusione?*
- *Se si, quali sono le roadmap per eseguirla?*
- *Se no, quali sono le conseguenza risultanti, in particolar modo l'impatto sul business delle due aziende?*

Le risposte a queste tipologie di domande scaturiscono dall'utilizzo della funzione dell'Enterprise Architecture.
Quindi il compito dell'Enterprise Architecture è quello di fotografare la visione dell'azienda. In particolare deve evidenziare qual è la struttura aziendale, il personale, il budget a livello di macro (dato il mapping IT), costo dell'intero IT, disegnare lo scenario target e la roadmap per arrivare alla fusione delle due aziende.
Questa fotografia viene creata in concordo con il CIO (*Chief information officer*, il quale dirige strategicamente i sistemi informativi in azienda, in modo che si adattino al meglio ai processi aziendali e costituiscano un elemento di vantaggio competitivo a supporto delle attività di produzione).
Il risultato di questa cooperazione fornisce tutte le informazioni evidenziando qual è il miglior modo per convergere al risultato finale.
Infine se le due aziende avessero già una struttura interna tramite Enterprise Architecture, la parte di struttura e budget (in figura evidenziata come *time consuming*) risulterebbe già presente, quindi il processo di fusione si rispecchia nella decisione del target e della roadmap, sempre in collaborazione con il CIO.

La parte riguardante *l'IT Governance* si occupa di ufficializzare quelle che sono le policy target, i processi target e tutte quelle attività interne di adeguamento concordate con il CIO.
Tipicamente queste funzioni si rispecchiano nei processi delle due aziende che devono essere uniti, piuttosto che separati o eliminati.
L'IT Governance si occupa di ufficializzare la lista delle attività in roadmap (definita in precedenza dall'EA). Si occupa di definire che tipologia di operazione compiere, come, ad esempio, manutenzione, evoluzione, dismissione, o nessuna operazione.

Per ognuna di queste si occupa di ingaggiare fornitori per creare i progetti per eseguire queste funzionalità, quini gestisce le gare di assegnazione dei progetti, collabora alla definizione delle KPI di fusione.
Quando tutta questa parte viene esaurita e quindi i progetti sono in esecuzione, l'IT Governance svolte una funzione importantissima quale quella di monitoring.
Relativamente all'esempio di fusione delle due aziende (viste in figura), supponendo che la prima azienda offra 50 servizi, la seconda offra 30 servizi e 10 servizi di questi risultano essere in comune.
L'EA, in collaborazione con il CIO, si occupa, ad esempio, di capire quali dei 10 servizi in comune deve essere mantenuto, piuttosto che modificato o eliminato.
Quindi il tutto si rispecchia nel nascere di un nuovo progetto ad hoc, con la relativa gara di assegnazione. Il progetto può essere successivamente assegnato al PM (Project Manager), con il monitoraggio dell'eventuale PMO (Project Management Office), o altrimenti gestito dall'IT Governance.

L'IT Operation si occupa di eseguire tutte le parti operative del progetto creato in precedenza dall'IT Governance.

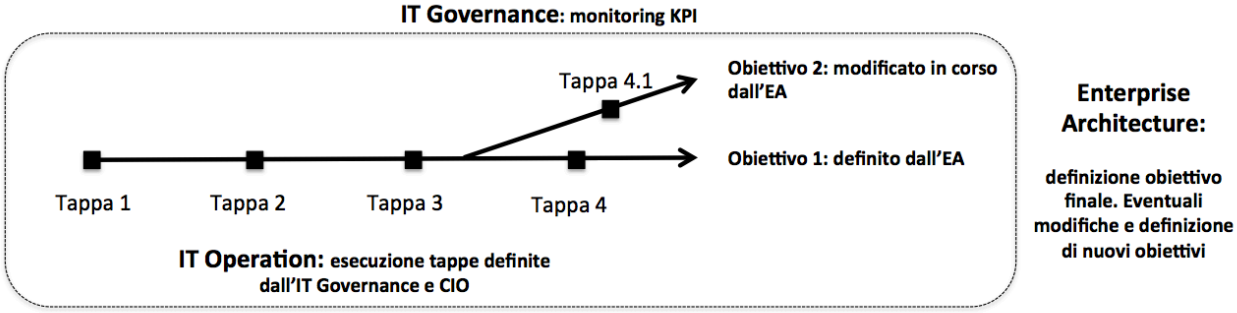

Figura 3 - Ruolo dell'Enterprise Architecture, IT Governance e IT Operation.

In conclusione, i ruoli principali di queste discipline sono:

- *L'Enterprise Architecture* si occupa di fornire una visione azienda allo scopo di capire le modalità di raggiungimento dell'obiettivo;
- *L'IT Governance* si occupa di monitorare che i processi dettati dall'Enterprise Architecture rispettino le KPI definite;

- *L'IT Operation* si occupa di eseguire le varie tappe per il completamento dei processi definiti, quindi sostanzialmente è concentrata sulla parte operativa.

1.3 Semantic Web

Con il termine *Semantic Web*, termine coniato dal suo ideatore, Tim Berners-Lee, si intende la trasformazione del World Wide Web in un ambiente in cui i documenti pubblicati (pagine HTML, file, immagini, e così via) sono associati ad informazioni e dati (metadati) che ne specificano il contesto semantico in un formato adatto all'interrogazione e all'interpretazione (es. tramite motori di ricerca) e, più in generale, all'elaborazione automatica [TOM13][EMA09][W3C15].
Lo scopo del Semantic Web è la cooperazione tra computer e persone, in modo tale che le macchine possano essere maggiormente di supporto agli esseri umani nell'esecuzione e nell'automazione dei compiti; la realizzazione del Semantic Web è possibile solo dando un significato ben definito all'informazione, in modo che le macchine possano raggiungere quel grado di comprensione che abilita funzionalità avanzate di ragionamento e capacità di rispondere a domande complesse [TOM13][EMA09][W3C15].
Quindi l'obiettivo principe del Semantic Web è di permettere alle macchine l'elaborazione delle informazioni attraverso l'impiego della logica; quest'obiettivo è raggiungibile se i computer hanno accesso a collezioni di informazioni e se possono portare a termine operazioni di ragionamento automatico mediante l'uso di insiemi di regole di inferenza o algoritmi riconducibili alle discipline di intelligenza artificiale.
Per far questo, il Semantic Web ha come punto di partenza e trova il suo fondamento sulla rappresentazione della conoscenza. La sfida del Semantic Web è quella di ideare un linguaggio logico adatto a esprimente sia i dati che le regole per il ragionamento automatico sui di essi; questo linguaggio deve ovviamente poter coesistere con i diversi sistemi di rappresentazione della conoscenza già esistenti e deve essere sufficientemente espressivo da abilitare il ragionamento su scala Web.

1.3.1 La pila del Semantic Web

Per meglio illustrare cosa serve per questa disciplina, mostriamo la "*pila del Semantic Web*", la quale stabilisce i diversi livelli necessari a completare un'implementazione o un sistema di comunicazione, in modo che il problema sia "modularizzato" a livelli e a ogni livello sia associato uno standard opportuno [TOM13][EMA09][W3C15].

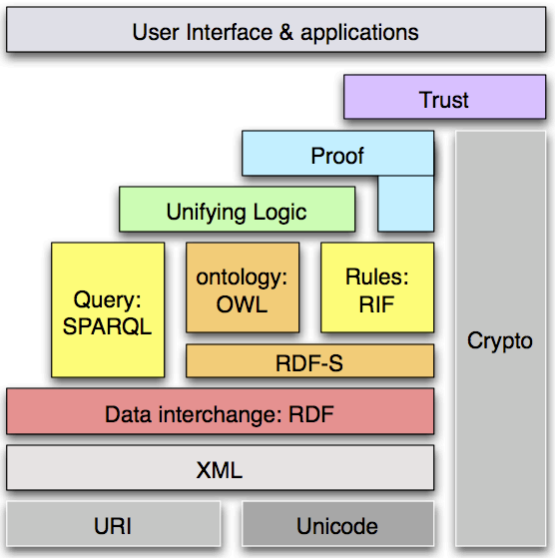

Figura 4 - La pila del Semantic Web.

Lo stack tecnologico del Semantic Web, ***in figura 4***, ha come livelli base *Unicode e URI*, i quali rappresentano rispettivamente un supporto agli strati superiori e una rappresentazione univoca per identificare risorse nel web.
Nel livello ancora superiore troviamo il linguaggio XML utilizzato per la marcatura dei testi, quale standard W3C.
Lo strato dei "modelli di base" è composto dall'RDF, RDF Schema e dall'Ontologia.

L'*RDF (Resource Description Framework)* è certamente un elemento fondamentale e centrale nel Web Semantico. Si tratta di un modello minimalista per la rappresentazione di risorse e le relazioni tra esse, estendendo il concetto di link alle

relazioni tra risorse sul Web. Una descrizione elementare (un enunciato) assume la forma ***soggetto-predicato-oggetto***, dove il predicato è una semplice relazione binaria. Un enunciato viene normalmente denominato come una tripla RDF, come ad esempio "Antonio Corradi" (soggetto) "insegna" (predicato) "Reti di Calcolatori M" (oggetto) [TOM13][EMA09][W3C15].

L'*RDF-S (RDF Schema)* fornisce gli elementi necessari e definire, sulla base di RDF, un elementare linguaggio ontologico. Comprende i costrutti per descrivere vocabolari. In altre parole, meccanismi per descrivere gruppi di risorse collegate e le proprietà che li legano tra loro. Rispetto all'Object Oriented, RDFS aggiunge potenzialità per descrivere in maniera molto ricca le proprietà che legano le risorse, considerate elementi di prima classe del linguaggio, cosi come lo sono in RDF, perché RDFS ci fornisce la possibilità di descrivere diverse caratteristiche di tali proprietà [TOM13][EMA09][W3C15].

L'*ontologia* rappresenta una concettualizzazione condivisa di un certo dominio. Nasce come un accordo consensuale sulla definizione di concetti e relazioni che caratterizzano la conoscenza del dominio stabilito, garantendo la possibilità di applicare regole d'inferenza (ragionamento) sia per stabilire nuove asserzioni deducibili (nuova conoscenza sulla base di quella a disposizione), sia per organizzare e recuperare in modo intelligente ed efficiente le informazioni presenti sul web [TOM13][EMA09][W3C15].
Quindi il livello ontologico fornisce gli strumenti per definire ontologie, utilizzando RDF e RDF-Schema come linguaggio per la modellazione.

Il livello immediatamente superiore, denominato **Livello Logico**, utilizza le asserzioni e le ontologie definite al livello inferiore per derivarne nuova conoscenza. Rende, quindi, quello che era un linguaggio dichiarativo, con limitate capacità espressive, un linguaggio logico completo con inferenze e funzioni, le quali consentono ad applicazioni RDF diverse di connettersi tra loro [TOM13][EMA09][W3C15].

L'ultimo livello è **Trust e Proof,** il cui scopo è quello di definire un linguaggio universale per rappresentare le dimostrazioni, ovvero una sequenza di formule ciascuna derivata applicando regole di inferenza, assiomi, definizioni e formule. Questo livello è tutt'oggi ancora in fase di studio del W3C [TOM13][EMA09][W3C15].

Parlando di URI e URL si fa spesso confusione. Nel corso degli anni '90 venne definita una suddivisione fra gli URL, quali identificatori che specificavano la localizzazione di una risorsa e gli URN, ovvero nomi delle risorse che erano indipendenti dalla localizzazione delle risorse stesse. Secondo questa visione definita "classica" dunque un URI era sia un URL che un URN. Nella visione contemporanea questa differenza si è un po' persa e gli identificatori Web sono in generale definiti come URI che definiscono dei sottospazi, cioè i namespace. Secondo questa visione, dunque, gli URL sono un tipo di URI che identificano una risorsa attraverso un meccanismo di accesso primario come, ad esempio, la sua localizzazione "http". Dunque un URI http è un URL. Gli URI abbracciano quindi un campo più ampio di applicazione e possono riferirsi a risorse di diversi tipi che siano identificate in maniera univoca.

1.4 Semantic Web nell'ambito Enterprise Architecture

La tecnologia Semantic Web può avere diverse applicazioni nell'ambito dell'Enterprise Architecture. Innanzitutto offre un linguaggio di *modeling* generale e flessibile come il linguaggio RDF e OWL. Questo permette di gestire la struttura aziendale e i relativi dati in modo flessibile e facilmente evolvibile, infatti unire al grafo RDF informazioni aggiuntive relative ad un ambito aziendale significa aggiungere semplicemente delle triple al grafo [PHI10][SUI08].
Ha inoltre come funzione quella di permettere di integrare, di distribuire e di federare informazioni Enterprise. In un contesto aziendale costituito da una molteplicità di unità di business (es. gestione marketing e gestione clienti), ognuno con una sua

propria strategia, tipicamente vi è un utilizzo di strumenti ad hoc consolidati che si adattano bene alle loro esigenze di gestione. Questa soluzione non permette di avere delle informazioni distribuite e condivise tra le varie unità.

Una tecnica utilizzabile per ovviare a questa mancanza è l'utilizzo, da subito, di un modello condiviso tra le varie unità di business. Il modello RDF ha come scopo quello di semplificare molto l'integrazione fornendo un'infrastruttura diretta a creare una rete collegata di informazioni, dando la possibilità di usufruire di questi dati sia all'interno dell'ambito aziendale, sia distribuendo queste informazioni sull'internet pubblico.

In relazione a quest'ultimo aspetto è importante sottolineare la possibilità di identificare tutte le risorse tramite identificatori univoci come gli URI, che sono web-compatibili, consentendo di riferire le informazioni relative al modello in modo univoco e universale a prescindere dalla loro origine e collocazione, aspetto cardine per l'integrazione e la distribuzione.

Per usufruire di queste tipologie di risorse il Semantic Web, secondo lo standard W3C, offre la possibilità di accedere in modo universale a questi tramite un linguaggio standard del W3C chiamato SPARQL.

Quindi il Semantic Web permette la gestione di relazioni semantiche tra i diversi elementi dell'EA, tra cui le relazioni tra diverse fonti di dati.

Le connessioni che in precedenza erano gestite tramite software personalizzato, possono essere descritte in modo standard. Le imprese possono utilizzare SPRQL per aggiungere regole e vincoli commerciali relativi al loro dominio, piuttosto che interrogare queste fonti per analizzare i dati e trarne indicazioni utili agli aspetti aziendali.

La tecnologia Semantic Web può essere sfruttata per fornite tutte le informazioni utili dell'azienda e rendere queste informazioni, normalmente disponibili solo in formati proprietari e quindi difficili da integrare, disponibili agli stakeholder aziendali. Questo rende fortemente collegate fra loro l'Enterprise Architecture come disciplina e il Semantic Web come tecnologia di applicazione.

1.5 Specifiche

Dopo un'attenta analisi della "pila del Semantic Web", analizziamo nel dettaglio il protocollo standard per l'interrogazione dei dati strutturati in RDF: SPARQL.
Questo protocollo, elemento centrale della nostra sperimentazione, fornisce la possibilità di interrogare dati via web, i quali sono esposti tramite un servizio chiamato endpoint.
Inoltre nello sviluppo di nuove ontologie, si ha la tendenza al riuso di ontologie standard, le quali definiscono concetti generali e riusabili. In questo senso analizzeremo l'ontologia FOAF, la quale ha lo scopo di descrivere persone, organizzazioni, ecc., con le loro attività e le relazioni con altre persone, organizzazioni ed oggetti.
Infine analizzeremo SKOS, ovvero una famiglia di linguaggi formali creata per rappresentare glossari, classificazioni, tassonomie e qualsiasi tipo di vocabolario strutturato.

1.5.1 SPARQL 1.1

SPARQL (acronimo di: Simple Protocol and RDF Query Language) è un linguaggio di interrogazione per Resource Description Framework (RDF) reso standard dal Data Access Working Group, gruppo di lavoro del consorzio W3C, che lo ha reso raccomandazione ufficiale il 15 gennaio 2008 [W3C12] [W3C08].
Tutte le specifiche relative a SPARQL sono definite in tre Recommendation differenti del W3C:

- *SPARQL Query Language for RDF*: documento principale in cui viene definito il linguaggio di interrogazione e vengono presentati tutti gli elementi che compongono una query SPARQL;
- *SPARQL Protocol for RDF*: documento in cui vi è descritta l'interfaccia WSDL 2.0 e il corrispondente binding HTTP e SOAP per l'implementazione di uno SPARQL query service e dei corrispondenti clienti;

- *SPARQL Query Results XML Format:* documento che identifica la struttura del documento XML contenente i risultati di una query SPARQL di tipo SELECT o ASK.

SPARQL è un elemento chiave del web semantico e consente di estrarre informazioni dalle basi di conoscenza distribuite sul web.
RDF descrive i concetti e le relazioni su di essi attraverso l'introduzione di triple (soggetto-predicato-oggetto); se tali triple hanno degli elementi in comune emerge un grafo di conoscenza.
SPARQL non fa altro che ricercare dei sotto-grafi corrispondenti alla richiesta dell'utente che effettua la query [BOB11].
L'elemento chiave di RDF sono le URI, che identificano le risorse in maniera univoca, consentendo a chi usa SPARQL di scrivere query ben definite e non ambigue.
L'elaborazione in SPARQL avviene introducendo due informazioni: il grafo dei dati (presente sul Web), e il grafo di query (descritto attraverso triple dall'utente). L'output può essere di più tipi, ma principalmente si utilizzano interrogazioni di tipo esistenziale (esiste o meno il sotto-grafo ricercato?) o tabellare (elencami i risultati possibili).
SPARQL contiene funzionalità per l'esecuzione di query su pattern graph obbligatori e facoltativi con le loro congiunzioni e disgiunzioni.
SPARQL supporta anche l'esecuzione di query attraverso un grafico sorgente RDF.
I risultati delle query SPARQL possono essere gruppi di risultati o grafi RDF [BOB11][W3C12][W3C08].
Da un punto di vista sintattico, SPARQL può ricordare SQL (Structured Query Language), il linguaggio per interrogare basi di dati, anche sei due modelli di rappresentazione sottostanti presentano notevoli differenze.
Un database relazionale è caratterizzato da record organizzati in tabelle e il processo di identificazione degli oggetti informativi memorizzati tramite record avviene tramite le primary e foreign key. In RDF, ogni risorsa è identificata da un URI [W3C12] [W3C08] [BOB11].

Il linguaggio RDF prevede diverse tipologie di notazioni, tramite le quali rappresentare le basi di conoscenza. Le più importanti e più utilizzate sono la notazione *Turtle*, la ***notazione N3***, la notazione ***N-Triples***e la notazione *RDF/XML* (rif. Appendice A).

I costrutti principali del linguaggio SPARQL sono:

- ***SELECT:*** ha esattamente lo stesso significato del costrutto SQL e indica cosa vogliamo ottenere in risposta all'interrogazione;
- ***FROM:*** permette di indicare quali sono le sorgenti dati verso cui indirizzare le richieste per ottenere risultati;
- ***WHERE***: contiene tra partentesi graffe la path expression che costituisce la vera e propria interrogazione; ovviamente, questo pattern di triple deve contenere tutte le variabili indicate nella SELECT, ma nulla vieta di utilizzare altre variabili per completare il percorso di enunciati necessario all'interrogazione;
- ***PREFIX:*** viene utilizzato per definire gli IRI delle risorse che saranno utilizzate;
- ***CONSTRUCT:*** è un tipo di query. Restituisce un grafo RDF che istanzia un graph pattern. Può essere usato, ad esempio, per trasformare un vocabolario in un altro;
- ***ASK:*** è una tipologia di query. Il risultato è un valore booleano, e indica se l'informazione richiesta è presente o meno nella base di conoscenza;
- ***DESCRIBE***: è una tipologia di query. Viene utilizzata quando non si conosce il modello dei dati. L'interpretazione della query di tipo DESCRIBE è: "restituiscimi tutto quello che sai in relazione alle risorse indicate";
- ***UNION***: si specifica due o più graph pattern diversi e si chiede che il risultato dei loro binding sia unito (nel senso insiemistico del termine);
- ***OPTIONAL***: serve ad indicare che per alcune porzione del graph pattern rappresentato può anche non esserci un binding nella soluzione finale;
- ***FILER***: serve a filtrare dall'insieme delle soluzione tutti i binding che soddisfano una determinata condizione;

- ***ORDERBY***: è un modificatore della query, analogamente al SQL permette di ordinare la sequenza dei risultati di una query in modalità crescente o decrescente;
- ***LIMIT:*** serve ad indicare quanto deve essere grande la sequenza dei risultati;
- ***OFFSET***: specifica l'elemento a partire dal quale devono essere restituiti i risultati.

Partendo da un semplice esempio, cerchiamo di dimostrare il funzionamento di una semplice query SPARQL.
Supponiamo di avere una piccola base di conoscenza rappresentata nella ***tabella 1***.

Soggetto	Predicato	Oggetto
Gestione Magazzino	**a**	**Application**
Word Processor	**a**	**Application**
Virtuoso	**a**	**Middleware**

Tabella 1 - Esempio di base di conoscenza

In seguito si esegue la query in figura

```
SELECT *
WHERE  ?app a Application.
```

Figura 5 -Esempio di query SPARQL

Il risultato delle query è visibile nella ***tabella 2***. Di notevole importanza è il placeolder *?app*, il quale, a differenza del linguaggio SQL, permette di filtrare i risultati nella clausola WHERE.

Soggetto	Predicato	Oggetto
Gestione Magazzino	**a**	**Application**
Word Processo	**a**	**Application**

Tabella 2 - Risultato query SPARQL.

I termini RDF possono essere IRI (espressa in maniera esplicita o tramite i prefissi) o dei literal. L'unico che ha la possibilità di essere un literal è l'oggetto della tripla.
Un ulteriore aspetto legato a SPARQL è la possibilità di creare e inserire delle ***regole di inferenza***, definita come una regola che permette di passare da un numero finito di proposizioni assunte come premesse a una proposizione che funge da conclusione.
Un semplice esempio di regola, definite in pseudocodice, è visibile in ***figura 6***, definito sulla base di conoscenza vista in precedenza. C'è da tener conto del fatto che la sintassi delle regole è strettamente legata al middleware utilizzato, ed inoltre non tutti i middleware disponibili sul mercato supportano l'inferenza.

```
IF   ?app  a   Middleware
THEN ?app  a   Application.
```

Figura 6 - Esempio di regole di inferenza.

Questa regole di inferenza sono alla base dei *reasoner* (rif. Appendice C).
In particolare, applicando la regola di inferenza vista in precedenza, il risultato della query precedente si modificherà, come visibile nella ***tabella 3***.

Soggetto	Predicato	Oggetto
Gestione Magazzino	**a**	**Application**
Word Processor	**a**	**Application**
Virtuoso	**a**	**Application**

Tabella 3 - Risultato query con regola di inferenza.

Le triple definite tramite il linguaggio RDF vengono memorizzate tipicamente in uno storage chiamato Triple Store.

Il *Triple Store* è un database costruito appositamente per il salvataggio e il recupero di triple ed è molto simile a un database relazionale poiché memorizza le informazioni in una base dati e le recupera tramite un linguaggio di query.

I Triple Store funzionano come applicazioni server che espongono dei servizi accessibili generalmente via HTTP ed interrogabili via SPAQRL.

Questa funzionalità è mediata da Middleware, i quali espongono questa tipologia di servizio come Endpoint SPARQL. La funzione del Middleware è quella di ottenere la query in ingresso, elaborarla, accedere al Triple Store recuperando i dati opportuni, restituire il risultato nei vari formati possibili (il classico formato è XML).

In ***figura 7*** vi è una semplice rappresentazione di come è strutturato logicamente un Endpoint SPARQL.

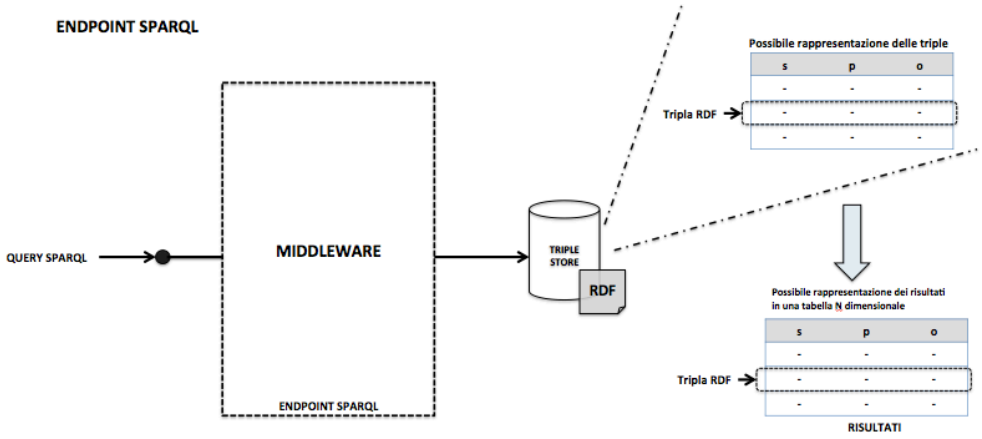

Figura 7 - Endpoint SPARQL.

1.5.2 Endpoint SPARQL

Un *endpoint SPARQL* può essere definito come un servizio Web da cui poter accedere ai dati attraverso delle interrogazioni (query) scritte in SPARQL, che è un linguaggio e protocollo d'interrogazione di dati in RDF. In termini più tecnici, si può dire che SPARQL consente di fare graph pattern matching all'interno di dati RDF.

A seconda dei dataset interrogabili tramite un endpoint, possiamo dividerli in due macro-categorie:

- *endpoint specifici:* l'interrogazione avviene ed è possibile solo sui dataset che sono precaricati nell'endpoint;
- *endpoint generici*: l'interrogazione avviene ed è possibile su qualsiasi fonte dati RDF vie Web (definito anche endpoint general purpose).

I risultati di una determinata interrogazione sono ottenuti in formato XML o RDF, ed è possibile gestire questi risultati in modo dinamico.

Un endpoint SPARQL permette di separare la richiesta proveniente dall'esterno dai dati interni all'applicazione, questo tramite l'implementazione di un componente software che espone all'esterno un'interfaccia SPARQL, ma che internamente gestisce i dati con la tecnologia più appropriata alle proprie esigenze (ad esempio tramite RDF o anche mediante l'utilizzo di SQL).

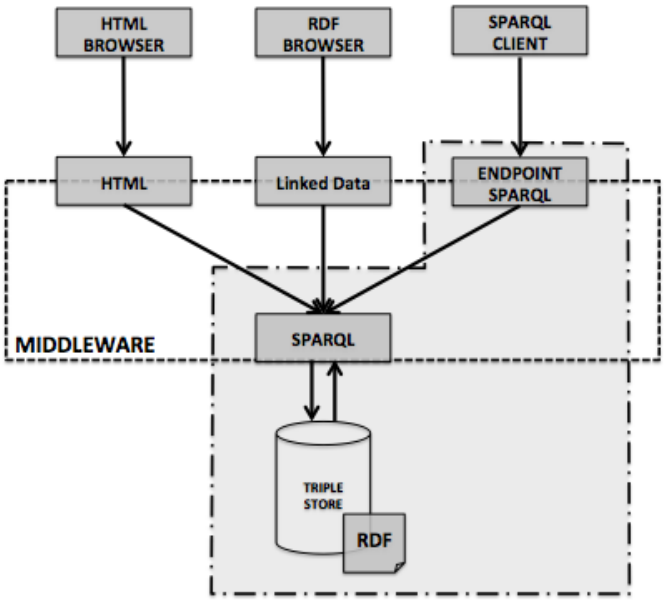

Figura 8 - Endpoint SPARQL.

Alcuni dei browser del Semantic Web (come Marbles), di soluzioni storage (come Sesame) e di application framework (come Joseki), espongono nativamente un endpoint SPARQL con cui è possibile accedere ai dati internamente gestiti.

Tuttavia, quando l'esigenza è rendere facilmente accessibili sul Web grandi quantità di dati, oggi contenuti in database relazionali, può essere utile disporre di strumenti

che mappino i modelli relazionali dei database in RDF e viceversa [W3C12][BOB11][W3C08].

1.5.3 FOAF

La comunità scientifica ha creato il progetto *FOAF (Friend of a Friend)* al fine di definire un vocabolario (ontologia) RDF per esprimere i metadati relativi a persone, ai loro interessi, relazioni e attività.
Fondata da Dan Brickley e Libby Miller, FOAF è una iniziativa completamente in linea con gli obiettivi della comunità del Web Semantico, per la creazione del cosiddetto "Web of Data".
FOAF contiene le primitive per la descrizione di una persona e dei legami di conoscenza con i suoi amici e colleghi. Sempre più spesso, sul Web le persone inseriscono sul proprio sito il proprio file foaf.rdf.
Essendo un'applicazione RDF, FOAF può rivendicare i noti vantaggi di raccogliere e aggregare dati facilmente. E come tutti i vocabolari (ontologie), RDF può essere facilmente combinato con altri vocabolari, consentendo la cattura di un insieme molto ricco di metadati.
Il suo successo è inoltre dovuto al suo utilizzo nei Social Network, come ad esempio *LiveJournal* (http://livejournal.com), il quale crea in automatico i file FOAF per ciascuno dei suoi utenti.
Utilizza come namespace http://xmlns.com/foaf/0.1, mentre per la creazione del file FOAF è possibile utilizzare FOAF-a-matic [FOF14][EMA09][TOM13].

1.5.4 SKOS

SKOS (Simple Knowledge Organizzation System) è un linguaggio per descrivere semplici strutture di conoscenza sul Web.
Questo linguaggio aggiunge a RDF/RDF-Schema alcune classi e molti predicati utili a far migrare nel Web semantico raccolte preesistenti di termini correlati, come tesauri, tassonomie, soggettari.

La classe basilare è `skos:Concept`, alla quale appartengono tutti i concetti indipendenti dai termini usati per descriverli.
SKOS adotta l'approccio same-syntax, ovvero segue la notazione RDF [MAT06][EMA09][TOM13].

1.6 Tecnologie utilizzate

1.6.1 Database

Un *database (relazionale)* è un insieme di dati strutturati, ovvero i dati sono suddivisi in tabelle (o entità) elementari e collegate poi tra loro da altri dati, chiamati relazioni, per mantenerne la coerenza.
Le logiche di modellazione sono derivate dalla teoria degli insiemi e quindi teoricamente la struttura di un Database è definibile in modo scientifico.
A queste teorie si sono nel tempo affiancate regole e buone abitudini che aiutano a creare una struttura efficiente rispetto al contesto in cui questa viene utilizzata: in alcuni casi si arriva addirittura a denormalizzare i dati (cioè sostanzialmente a replicarli in più tabelle) per migliorare le performance a scapito della modellazione.
Il più piccolo database possibile è composto, ovviamente, da un'unica tabella con una sola colonna senza relazioni: una specie di elenco di informazioni, che per quanto elementare è una delle strutture dati più comuni anche al di fuori dell'IT.
L'estrazione dei dati presenti in un database avviene utilizzando un linguaggio di interrogazione chiamato SQL [PAT13][JNA13][THE08].

1.6.2 Ontologie

Un'ontologia è la formalizzazione di una conoscenza, quindi dal punto di vista logico è sostanzialmente un archivio d'informazioni, ma queste sono strutturate e utilizzate in modo molto differente rispetto al classico approccio di modellazione E/R.

Le logiche di modellazione di questa conoscenza non sono basate su teorie matematiche, ma sulla capacità di esprimere concetti che possono evolvere in modo non lineare grazie all'uso di una struttura di base molto elementare chiamata ***tripla***, che è formata da tre elementi: **soggetto**, **predicato** e **oggetto**.
Nelle ontologie sono quindi esprimibili due tipi di relazione fra dati che non sono facilmente esprimibili usando i database:

- L'ereditarietà tra concetti (anche di diverse ontologie);
- La composizione di più concetti (rappresentati anche su diverse ontologie).

Entrambi i concetti sono, in verità, molto comuni dopo l'avvento della programmazione OOP, ma sono decisamente più "esotici" nell'uso della modellazione dati.
L'estrazione dei dati presenti in un'ontologia avviene utilizzando il linguaggio d'interrogazione **SPARQL**, quando questa ontologia è espressa tramite gli OWL e più in generale quando questa è espressa utilizzando le tecnologie e gli standard del Semantic Web [EMA09][TOM13].

1.6.3 Uso efficiente delle ontologie

Questo documento parla in modo particolare di ontologie, ma prima di entrare nel loro dettaglio è utile capire quando è consigliabile usarle e quando invece sia più conveniente usare un normale database.
Sicuramente il campo di utilizzo più naturale delle ontologie è l'analisi dati.
Le ontologie sono, infatti, la struttura dati più corretta su cui applicare classificazioni (anche a posteriori del primo censimento dati), logiche induttive attraverso software che possono arricchire il dato originale con nuove informazioni e, soprattutto, logiche di simulazione per capire come determinati "indici" possono evolvere nel tempo.
Le ontologie supportano meglio queste situazioni, perché la strutturazione del dato è meno rigida rispetto a quella presente nei normali database.
Mentre per quanto riguarda l'operatività ordinaria, ovvero quella presente nelle normali applicazioni di tutti i giorni, sicuramente i database relazionali svolgono

ancora un ruolo fondamentale, tuttavia stanno emergendo dei casi di utilizzo delle ontologie anche in questi scenari come, per esempio, per gli ***Open Data*** e per i ***Big Data*** *(rif. Appendice B)*.
Le ontologie, infatti, utilizzano gli URI all'interno dei dati, per cui queste diventano anche il miglior mezzo per indirizzare dei dati che sono di uso pubblico.
Allo stesso tempo, poter lavorare su un modello condiviso e su dati remoti, apre alla possibilità di utilizzare moli di dati estreme: ad esempio la possibilità di fare analisi dati sui social network per tutto ciò che riguarda la propria azienda e di fornire il risultato dell'analisi direttamente sui propri applicativi.

1.6.4 Database noSQL

I ***database noSQL*** sono una tipologia di database in cui la persistenza dei dati è caratterizzata dal fatto di non utilizzare il modello relazionale, di solito usato dai database tradizionali (RDBMS). L'espressione noSQL fa riferimento al linguaggio SQL, che è il più comune linguaggio d'interrogazione dei dati nei database relazionali. Questi archivi di dati il più delle volte non richiedono uno schema fisso (schemaless), evitano spesso le operazioni di unione (join) e puntano a scalare in modo orizzontale. Gli accademici e gli articoli si riferiscono a queste basi di dati come memorizzazione non strutturata [FRF13][PAT13][JNA13].

1.6.5 OWL

Il ***Web Ontology Language OWL*** è un linguaggio di markup semantico per la pubblicazione e la condivisione di ontologie sul World Wide Web. OWL è sviluppato come estensione del vocabolario di RDF (Resource Description Framework) e deriva dal DAML + OIL Web Ontology Language. Lo scopo è quindi quello di descrivere il significato semantico dei dati, presenta strumenti più espressivi di RDFS per esprimere le informazioni semantiche dei dati di cui ne costituisce una estensione [TOM13][EMA09][W3C04].

La maggiore espressività, rispetto ad RDFS è dovuta alla capacità di descrivere:

- vincoli di cardinalità (es . una persona ha una sola madre e un solo padre);
- possibilità di indicare che due classi definite in schemi differenti rappresentano lo stesso concetto. Proprietà molto utile all'interno di un ambito distribuito come il Web, per collegare risorse definite in due schemi indipendenti. È il caso di "owl:sameAs";
- possibilità di indicare che due istanze, definite separatamente, rappresentano lo stesso concetto;
- possibilità di indicare che una proprietà è transitiva;
- la possibilità di definire una nuova classe come combinazione di più classi esistenti.

La definizione e la realizzazione di queste (e altre) capacità sono lo scopo del gruppo di lavoro sui linguaggi per la definizione di ontologie.

1.7 Prodotti Software

1.7.1 Apache CXF

Apache CXF è un framework sviluppato e mantenuto dalla fondazione Apache. Il suo nome deriva dalla fusione di due progetti (Celtix e Xfire) e l'obiettivo è quello di fornire delle astrazioni e strumenti di sviluppo per esporre varie tipologie si servizi web.

Le potenzialità del framework si possono riassumere nelle seguenti:

- Netta separazione dall'interfaccia JAX-WS e l'implementazione;
- Semplicità di definire servizi web e client tramite annotazioni Java;
- Alte performance;
- Alta modularità dei suoi componenti (che rende possibile esporre servizi stand-alone o su servlet container).

Questo framework offre una set di feature molto ampio, in particolare riguardo alle seguenti aree:

- Supporti standard al Web Service (SOAP, WS-Addressing, WS-Policy, WS-Security);
- Supporto a Corba;
- Api per sviluppo di Web Service (JAX – WS);
- Api per sviluppo di RESTful Web Service (JAX – RS);
- Supporto al framework Maven, utilizzato nel nostro progetto;
- Supporto allo sviluppo di servizi basati su Java EE;
- Supporto alla programmazione JavaScript per servizi client e server.

Infine è spesso usato con altri prodotti software per creare infrastrutture di tipo SOA [APA14].

1.7.2 Apache LOG4J

Apache LOG4J è una libreria Java facente attualmente parte del progetto di Apache Software Fondation.

Log4j è un tool per la gestione del logging in java, con la possibilità di implementare diversi livelli di logging, in ordine diverso e decrescente di severità:

- *OFF*: è il livello più alto di severità, ed è di norma utilizzato per disattivare i log;
- *FATAL*: segnala un errore importante che potrebbe causare un prematuro termine dell'applicazione;
- *ERROR*: segnala un errore di esecuzione o una condizione imprevista;
- *WARN*: di norma usato per segnalare ogni condizione inaspettata o anomalia di esecuzione, che però non necessariamente comportano un errore;
- *INFO*: usato per segnalare eventi di esecuzione (es. startup/shutdown);
- *DEBUG*: usato principalmente nella fase di debug del programma;
- *TRACE*: identifica e aggiunge ai log delle informazioni dettagliate sul programma [APAL15].

1.7.3 JUNIT

JUnit è un framework per il testing in Java. Consente di eseguire lo Unit Testing, ovvero di effettuare verifiche di funzionamento di piccole porzioni, o unità, di codice (ad esempio un metodo, una classe, un componente).
Può essere integrato nella maggior parte degli ambienti di sviluppo. Nel nostro progetto è stato integrato nel progetto maven, utilizzando la versione 4.11.[JUI14]

1.8 Framework utilizzati

1.8.1 Apache Jena

Il più diffuso Framework per il Semantic Web per Java, open source, è ***Jena***. Opera mantenendo in memoria tutto il modello RDF, fornendo un insieme di API per manipolarlo secondo il paradigma a grafo.
Nelle ultime versioni sono stati integrati anche dei reasoner RDF e per OWL, permettendo cosi di manipolare la gerarchia della classi, di creare e rimuovere istanza e concetti e di inferire nuove triple in base ai dai presenti nel modello.
È inoltre capace di garantire la persistenza dei dati sia interpretando le varie sintassi XML/RDF sia interfacciandosi con diversi database relazionali.
Jena supporta SPARQL grazie all'aggiunta di una libreria ARQ, ed inoltre esiste anche un'estensione chiamata Joseki che implementa un server capace di esporre un endpoint SPARQL con cui è possibile pubblicare e interrogare i dati all'interno del modello gestito da Jena.
L'architettura Jena è molto articolata, come si evince dalla ***figura 9***, e presenta diverse funzionalità, tra cui è di nostro interesse la parte relativa alle API SPARQL.

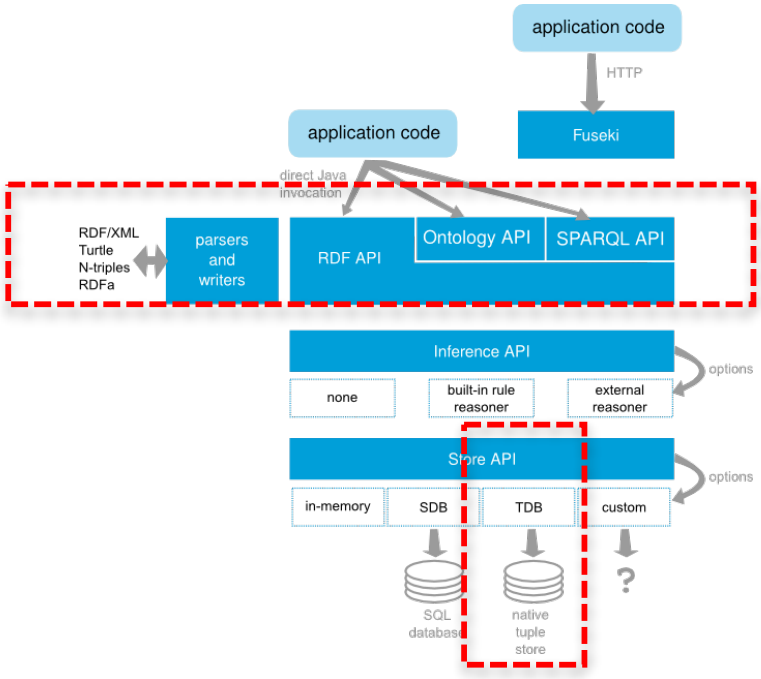

Figura 9 - Architettura Apache Jena

Le classi di interesse sono:

- "Query": classe che rappresenta la query dell'applicazione;
- "QueryExecution": classe che rappresenta l'esecuzione di una query;
- "QueryExecutionFactory": classe usata per ottenere una istanza di QueryExecution a partire da un oggetto Query e un riferimento al servizio SPARQL.

Per ciò che riguarda la query SELECT:

- "QuerySolution": un oggetto che contiene la soluzione della query;
- "ResultSet": un iteratore su tutte le risposte alla query;
- "ResultSetFormatter": una classe in grado di trasformare un ResultSet in vari modi, tra cui semplice testo, formato CVS o in grafo RDF [APA15][APJ14].

1.8.2 Sesame

Sesame è un progetto open source che rappresenta uno dei più diffusi storage RDF per il linguaggio Java, aspetto certamente positivo in ambito Enterprise, poiché Java risulta essere il linguaggio più diffuso ed utilizzato.

Propone un livello di astrazione intermedio che disaccoppia le funzionalità di inserimento e interrogazione di RDF dallo specifico database sottostante.

Offre un'API Java stile JBDC ed una interfaccia HTTP RESTful che supporta il protocollo SPARQL. Contiene le implementazioni di un in-memory triplestore e di uno store su disco, ma può anche essere usato come libreria di API per altri triplestore come Mulgara e AllegroGraph. Proprio come Jena non supporta configurazioni di High Availability (HA).

Uno schema della sua architettura interna è illustrato in ***figura 10***:

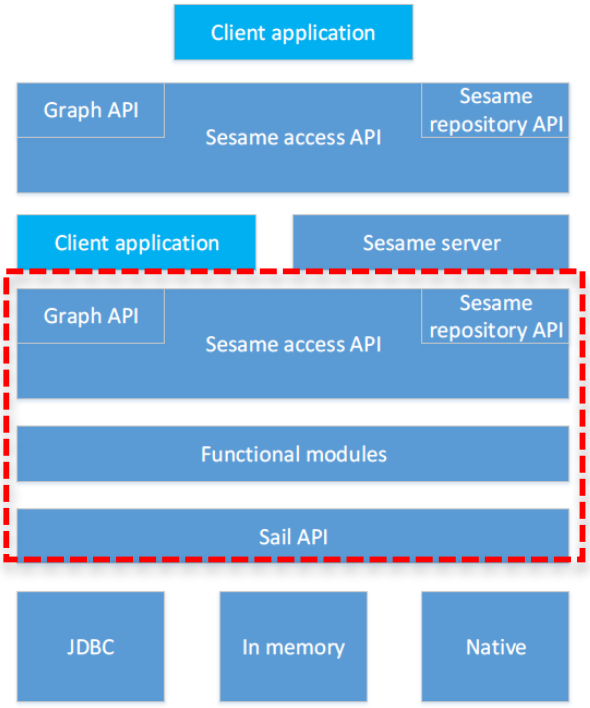

Figura 10 - Architettura di Sesame.

Le Sail API (Storage And InferenceLayer) forniscono un livello di astrazione rispetto al formato di archiviazione utilizzato (cioè se i dati sono memorizzati in un RDBMS, in memoria o in file, per esempio) e fornisce il supporto all'inferenza.

Lo strato superiore al Sail API racchiude i moduli funzionali, dove risiedono il query engine, i moduli di amministrazione ed il modulo per l'esportazione dell'RDF.
Nello strato successivo troviamo le API per l'accesso a questi moduli funzionali, composte da due parti separate:

- *l'API Repository* offre accesso ad alto livello, come ad esempio l'esecuzione di query, la memorizzazione di file RDF, l'estrazione di RDF etc.;
- Le *Graph API* forniscono invece un supporto molto più a grana fine per la manipolazione di RDF, come l'aggiunta e la rimozione di singole triple e la creazione di piccoli modelli RDF direttamente dal codice. Le due API si completano a vicenda in termini di funzionalità e sono in pratica spesso utilizzate insieme.

Queste API forniscono accesso diretto ai moduli funzionali di Sesame, ad un programma client e al componente successivo dell'architettura di Sesame, il "Sesame server". Si tratta di un componente che fornisce una interfaccia HTTP ed RMI alle API di Sesame. Quindi per client che usano i protocolli HTTP od RMI troviamo nuovamente le API di accesso, tramite le quali è possibile comunicare con Sesame, non più come libreria, ma come verso un server remoto [SES15].

1.8.3 AllegroGraph

AllegroGraph è uno storage per dati RDF che ha ottime performance e la capacità di gestire grandi moli di triple. Supporta ontologie scritte in prolog, in RDFS e in OWL, scelta fatta per ottimizzare il reasoner interno e garantire ottime prestazioni.
Il linguaggio d'interrogazione è SPARQL, anche se in AllegroGraph è utilizzato un suo sottosistema chiamato Twinql, aderente anch'esso allo standard W3C, includendo ottimizzazioni di query e gestione dei named-graph.
Il motore di AllegroGraph registra al suo interno le triple "soggetto-predicato-oggetto" in modo efficiente, assegnando ad ognuna un indice, chiamato UPI, tramite il quale gestisce le query in modo efficiente e veloce.

Permette inoltre di gestire la transazionalità delle operazioni, gestendo le quattro caratteristiche della proprietà ACID (Atomicità, Consistenza, Isolamento e Durabilità) (rif. Appendice F).

AllegroGraph mette a disposizione molte feature per manipolare i dati all'interno del triplestore, permettendo l'utilizzo di applicativi Java o linguaggi come Lisp.

L'utilizzo di applicativi Java è sempre un punto a favore per ogni framework utilizzabile in ambito Enterprise, proprio perché in quest'ambito risulta essere il linguaggio più utilizzato.

Fornisce inoltre la possibilità di utilizzo del protocollo HTTP, con ampio supporto alla funzionalità REST.

In ***figura 11*** è mostrata l'architettura del framework.

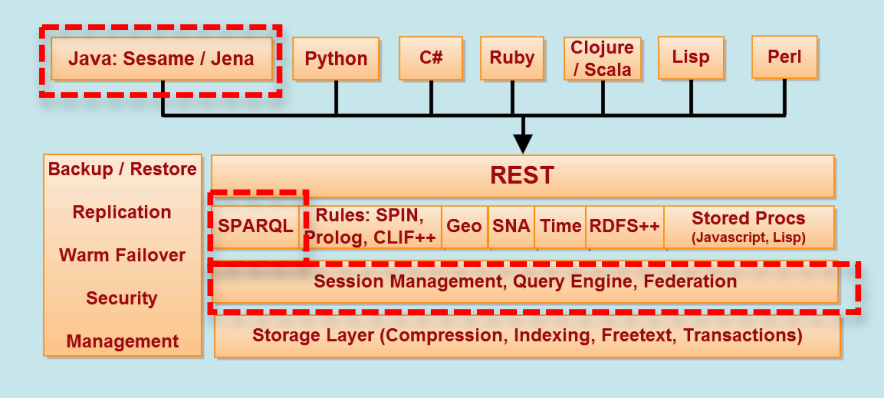

Figura 11 - Architettura di AllegroGraph.

In termini di federazione, in AllegroGraph è gestita tramite un'istanza virtuale di uno store, nel quale sono memorizzati e gestiti tutti gli store facenti parte della federazione.

In particolare il meccanismo di federazione di AllegroGraph si combina con la flessibilità del triple store per creare una connessione "multiple store" (o nella stessa istanza di AllegroGraph, o in istanze multiple di AllegroGraph sulla stessa macchina, o in istanze multiple di AllegroGraph in un sistema cluster) gestendo il tutto come se fosse un singolo store.

Tipicamente quando viene creato un repository AllegroGraph federato, viene creato un indice virtuale per gli store della federazione e mantenuto nelle sessione del client per facilitarne l'elaborazione e migliorare le prestazioni [ALL15].

1.8.4 Virtuoso

Openlink Virtuoso è un data server multi-modello, esso offre una soluzione, indipendente dalla piattaforma, per la gestione dei dati, l'accesso e l'integrazione.
L'architettura ibrida del server Virtuoso consente di offrire le funzionalità del server tradizionale all'interno di un offerta unica che copre le seguenti aree:

- Relational Data Management;
- RDF Data Management;
- XML Data Management;
- Free Text Content Management & Full Text Indexing;
- Document Web Server;
- Linked Data Server;
- Web Application Server;
- Web Services Deployment (SOAP or REST).

Virtuoso è quindi un database ibrido che combina in un unico sistema le funzionalità di un tradizionale RDBMS, ORDBMS, virtual database, RDF, XML, web application server e file server.
Piuttosto che avere server dedicati per ogni funzionalità sopra elencate, Virtuoso fornisce un accesso trasparente alle fonti di dati esistenti, che sono in genere banche dati di fornitori di database diversi.
Come DBMS relazionale offre un po' tutte le caratteristiche che offrono gli RDBMS tipici come MySQL e simili. Ovviamente supporta il linguaggio SQL, inoltre mette a disposizione molte funzioni utili.
Per quanto riguarda invece le sue funzionalità come Triplestore, il modello di dati utilizzato è il modello a grafo, che è quello utilizzato dalla maggior parte dei

Triplestore che non si basano su database relazionali. È possibile interrogarlo tramite il linguaggio SPARQL, eseguire update tramite SPARUL ed è possibile importare ed esportare dati in formato RDF.

I formati di serializzazione RDF supportati sono HTML+RDFa, RDF-JSON, N3, Turtle, TriG, TriXeRDF/XML (rif. Appendice A).

Virtuoso ha guadagnato un notevole interesse poiché è utilizzato per ospitare molti importanti Linked open Date (ad esempio, DBpedia). Questo ha permesso l'evolversi di una community di supporto molto estesa che ha assegnato un ulteriore punto a favore per questo framework.

Nella ***figura 12*** è mostrata l'architettura di Virtuoso.

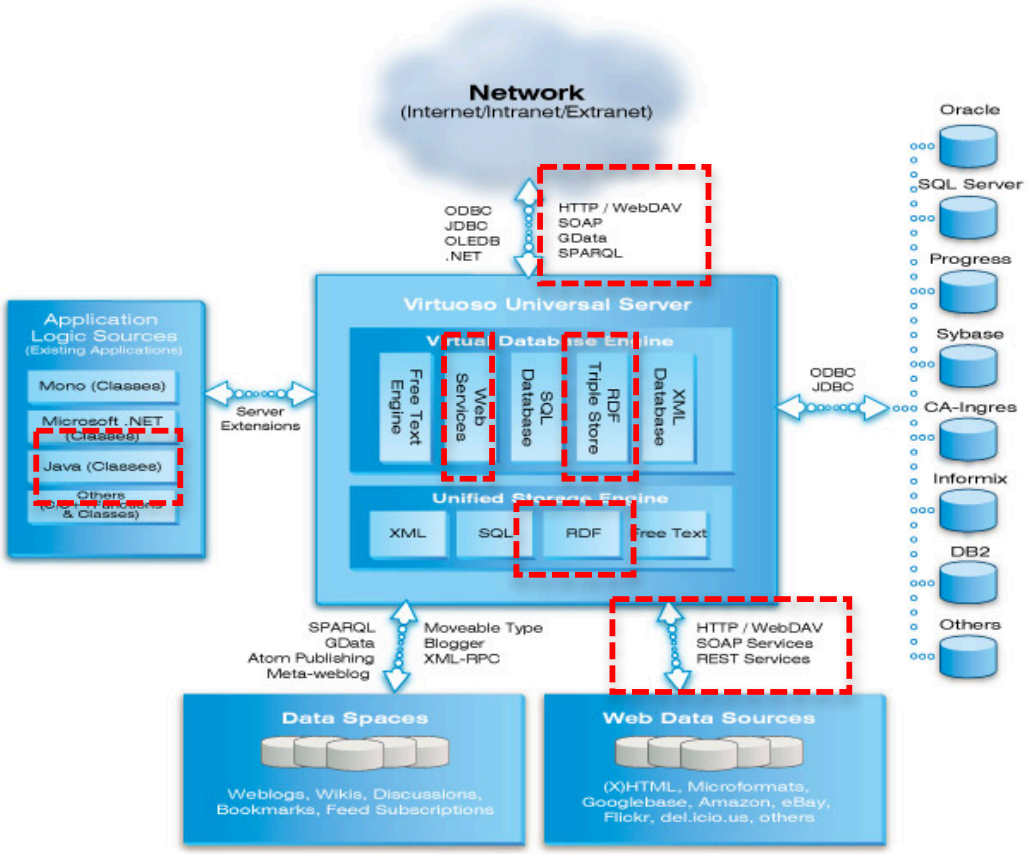

Figura 12 - Architettura di Virtuoso.

2. Data Governance e Ownership

Il seguente capitolo affronta un aspetto molto importante, quello dell'ownership. Ci si è soffermati particolarmente nell'analizzare l'ownership a livello di modellazione dei dati e l'ownership a livello del singolo dato, cercando di capire quali fossero le differenze e le criticità tra queste due tipologie di ownership.

Il patrimonio informativo di un'organizzazione, inteso come l'insieme delle informazioni relative a tutte le operazioni che fanno parte o influenzano la sua attività, degli oggetti (prodotti e servizi) ai quali queste si applicano e dei soggetti (client e stackolder) che vi partecipano, è l'asset di maggior valore di cui si possa disporre, ed è altresì il più difficile da creare e gestire [SAS13][MED14].

È da questa constatazione che nasce il concetto di ***Data Governance***, che si può definire come l'insieme delle attività volte a gestire persone, processi, metodologie e tecnologie dell'informazione al fine di realizzare un costante e corretto trattamento di tutti i dati che abbiano importanza per un'organizzazione. La strategia di Data Governance prevede la convergenza di varie funzionalità, dal Data Quality, Data Management, Risk Management, la Data Polices.

L'importanza del *Data Governance* risiede nella possibilità di capire come vengono utilizzati i dati a livello aziendale, quali sono i vincoli che bisogna considerare, qual è il conteso normativo, chi è responsabile dei dati e chi può utilizzare i dati stessi; una volta che queste problematiche sono state allineate, si procede all'impostazione e alla definizione delle regole sul modello dei dati e sui dati stessi.

Sempre nell'ambito della gestione e dell'utilizzo del dato, un aspetto importante da considerare è la Data Federation. Può essere definita come la capacità di integrare virtualmente i dati che sono archiviati in luoghi diversi senza dover predisporre un archivio fisico dove mantenere il risultato. Può essere visto come una specializzazione del Data Integration, dove l'archiviazione viene fatta in un luogo "terzo", quindi secondo una visione più distribuita. L'importanza di questa pratica risiede nel possibilità di disporre di dati sempre aggiornati, infatti, i processi di Data

Integration (spesso noti come ETL) sono eseguiti periodicamente e rappresentano sempre e solo una fotografia del sistema in un dato momento.
Il focus della nostra tesi si basa proprio su questo aspetto, mettere a disposizione uno strumento che fornisca la possibilità di federare dati e fornire supporto al Data Analisys e al decision-macking, in ambito Enterprise Architecture.
Tutte queste pratiche mettono in evidenza molte problematiche nella definizione dell'ownership a livello di modello e a livello di dato. Cerchiamo di analizzare ed evidenziare queste problematiche [SAS13][MED14].

2.1 Definizione di Ownership

L'Ownership dei dati è il diritto (anche legale) al controllo di un insieme di dati sia in termini di accesso (in lettura e scrittura) sia in termini di distribuzione del dato verso altri soggetti (normalmente sulla base di una qualche policy).
Definire un'ownership sui dati permette di dettare delle regole precise su come questi debbano essere gestiti e organizzati, delineando tutte le policy per la creazione, la modifica, la cancellazione e la condivisione dei dati stessi, sia nell'ambito interno all'organizzazione, ad esempio tra le varie unità di business, sia nell'ambito delle terze parti, come potrebbero essere gli stackolder [SAS13][MED14].

2.2 Ownership a livello di modellazione dei dati

Nella definizione di un modello dati bisogna tener conto di molti aspetti importanti, dall'acquisizione della conoscenza del dominio d'interesse, al riuso di risorse esistenti, alla pianificazione del modello in termini di organizzazione e integrazione.
La creazione di un nuovo modello dati parte dall'analisi del dominio da modellare, cercando di individuare entità e concetti che sono già stati precedentemente modellati, con l'obiettivo della massima integrazione e del riuso. Un classico esempio, in questo senso, potrebbe essere il noto vocabolario FOAF, il quale esprime concetti, come Person, del tutto generali e del tutto riusabili.

Quindi nella maggior parte dei casi di modellazione, si cerca di estendere modelli preesistenti o di comporre tra di loro parte di essi per definire e modellare il proprio dominio.

Nel caso in cui, non esiste nulla in termini di entità e concetti preesistenti che possano essere relazionati al proprio dominio, si definisce un modello di dati in cui entità e concetti vengono creati ex-novo, con l'unico grande problema della conoscenza approfondita del domino, non sempre semplice.

La condizione necessaria per il riuso di modelli già formalizzati è l'utilizzo di uno standard (ad esempio le ontologie), altrimenti tecnologie e soluzioni proprietarie risulterebbero un grande scoglio per l'integrazione, interoperabilità e riuso.

Estendere il modello dati permette al progettista di specializzare solo alcuni aspetti che sono tipici del dominio da modellare, definendo una ownership parziale, proprio perché ci si integra in domini già esistenti dei quali non si possiede l'ownership. Sul mercato Enterprise i grandi Vendor di Enterprise Architecture si stanno muovendo in questo senso, cercando di creare e fornire un modello che risulti del tutto estendibile.

La difficoltà si presenta nella "pratica realizzativa", infatti non avendo un controllo totale su modello quando si effettuano delle specializzazioni, estendere il modello stesso potrebbe risultare molto arduo, con il forte rischio di eliminare delle feature classiche. Difatti quello che verrebbe fatto è una progettazione verticale sul modello dei dati, evidenziando una evoluzione arbitraria del tutto, complicando di gran lunga lo sviluppo applicativo.

In un modello di dati totalmente nuovo l'ownership risulterebbe essere totale, in quando la decisione e la gestione è demandata al progettista.

Quando si parla invece di utilizzo di un modello dati si manifestano diverse possibilità. Si pensi infatti ad un gruppo operativo che utilizza un determinato modello di dati, è intuibile che chi ha bisogno di "agganciarsi" e utilizzare quel modello deve adeguarsi e rispettare le regole di ownership definite sul modello stesso, altrimenti andrà incontro a forti limitazioni.

Una possibilità diversa riguarda "l'accordo" sul modello dei dati, quindi definire a priori, di comune accordo, concetti e regole sull'utilizzo di un modello condiviso.

La federazione di endpoint, sviluppata in questa tesi, si basa proprio su questo "accordo", definendo un modello dei dati comune che verrà condiviso tra tutti i partecipanti alla federazione stessa.

2.3 Differenza tra ownership di modello e ownership dei dati

Nei paragrafi precedenti è stata evidenziata l'importanza dell'ownership sia a livello di modello che di dato.

L'ownership sul modello dei dati si riferisce alla possibilità e ai diritti di creazione, la modifica, l'integrazione e la condivisione di un modello dati in un dominio d'interesse.

L'ownership di dato si riferisce alle politiche relative al dato singolo, in sostanza ingloba tutte le policy di lettura, modifica, cancellazione e condivisione del dato, sia se questo viene utilizzato internamente all'organizzazione, sia se questo viene condiviso con terzi.

Questo permette di evidenziare un aspetto importante quale la proprietà del modello e del dato. L'ownership relativa al dato può essere associata al creatore del modello dati, il quale genera sia il modello dei dati sia i dati stessi, ma può essere, ad esempio, associata ai consumatori del dato, i quali pur non essendo proprietari del modello, rivendicano la proprietà dei dati.

L'importanza dell'ownership, quindi, è centrale quando si parla di dati e di chi li possiede, gestendo problemi importanti come dare valore ai dati, gestire la privacy, gestire le policy.

Per evidenziare ulteriormente questi aspetti, si pensi alla definizione dell'ownership a livello di processo, dove c'è il bisogno di capire "*chi deve fare cosa*"; vengono separate le ownership relative a chi deve sviluppare il modello, chi invece deve operare sul dato (producer e consumer del dato), chi infine deve manutenere il dato stesso.

Nell'ambito dell'azienda in cui è stato sviluppato questo lavoro di tesi, Imola Informatica, si sta cercando di utilizzare un approccio più centralizzato, cercando di avere un unico archivio d'informazioni in cui tutti i gruppi operativi inseriscono i

propri dati. Quest'obiettivo, tutt'ora non del tutto raggiunto, evidenzia dei problemi importanti. Un dato è fatto da una classe, da delle proprietà ed appartiene ad un namespace. Se si potessero individuare delle regole tramite le quali dividere le classi in "partizioni" di proprietà, cosi che gli utilizzatori, siano essi degli applicativi o degli analisti di dati, possano censire queste proprietà. Fornendo questa feature a livello di ownership, si potrebbe prevedere una strumentazione a livello applicativo che, in base all'ownership stessa, attivi dei profili personalizzati permettendo di agire su dati condivisi.

Altri problemi inerenti sono emersi nello sviluppo del modello dati utilizzato nell'implementazione della federazione. Si è cercato di sperimentare sul campo il concetto del Data Governance cercando di individuare un modello esaustivo che rappresentasse la struttura dell'Università, purtroppo senza successo.

Quindi la sperimentazione è proseguita nella realizzazione di un modello ontologico ad hoc per rappresentare, in primis le organizzazioni, come ad esempio la Pubblica Amministrazione, specializzando e focalizzando poi il tutto nella realizzazione del modello per le Università Italiane.

Nella realizzazione sono emerse molte situazioni eterogenee di ownership, sia sul modello che sui dati. In particolare nella definizione delle organizzazioni (la descrizione dettagliata verrà data successivamente) la ownership sul modello e sui dati risultava essere parziale, infatti è stata specializzata ulteriormente. Anche nella definizione della porzione di modello ontologico relativo alla Pubblica Amministrazione, la ownership è risultata parziale. Nella costruzione del modello Università, invece, la ownership sul modello è risultata totale, proprio per il fatto di non avere nessun modello ontologico esistente che rappresentasse le Università stesse. Se si parla di dati di Università, qui è stata separata l'ownership appartenente alle varie università scelte, proprio per evidenziare e marcare quanto spiegato ampiamente nei paragrafi precedenti.

3. Federazione di endpoint SPARQL: software selection

In questo terzo capito è stato esplorato il concetto di federazione, in particolare l'importanza della federazione in ambito Enterprise Architecture. Sono stati successivamente analizzati nel dettaglio gli attuali standard disponibili per la gestione della federazione di endpoint SPARQL, ed infine si è passati all'selezione del middleware da utilizzare per la sperimentazione.

3.1 L'importanza della Federazione nell'EA

In ambito Enterprise Architecture, soprattutto dal punto di vista IT, si ha la necessità di disporre di una mole di dati come input per funzionalità centrali, come ad esempio il decision-making.
La mole di dati prodotta in ambito aziendale risulta essere eterogenea e per di più proveniente da fonti diverse, come ad esempio dati riguardanti l'ambito produttivo e dati riguardanti l'ambito amministrativo, ma l'aspetto determinante è la correlazione esistente tra questi dati, che supporta il concetto di federazione.

"*La federazione (dal latino foedus, **'patto, alleanza, unione'**) è un ente costituito dall'associazione di più enti che, in tal modo, pur mantenendo la propria organizzazione, danno vita ad organi unitari per perseguire scopi comuni"* [DIZ14][SGU14].

Nel lavoro di tesi la definizione di questo "patto" è stato uno dei momenti più importanti anche per definire a livello tecnico un modello sul quale costruire una federazione. Difatti nel Data Governance esistono dei modelli standard che soddisfano vari aspetti, ma vi è un'assenza di modelli che definiscano un'organizzazione o un'unità organizzativa, in particolare non esiste nulla che permetta l'implementazione del concetto di federazione.

Questa motivazione ha spinto la nostra sperimentazione alla definizione di una nostra ontologia capace di raggruppare e collegare tra loro parti di un'organizzazione e i loro dati, rappresentando, di fatto, "il patto" per poter costruire una federazione.

3.2 Specifiche SPARQL 1.1: concetto di federazione

Il concetto di federazione è stato introdotto nello standard SPARQL dalla versione 1.1, tramite un costrutto chiamato SERVICE, implementato seguendo un modello distribuito, in cui i risultati vengono restituiti come intersezione dei risultati di ogni singola interrogazione.

3.2.1 La clausola Service

La specifica SPARQL 1.1 definisce la clausola SERVICE utilizzata per effettuare *query sparql federate* fornendo soluzioni basate sul merge dei risultati ottenuti da fonti diverse [W3C13].

```
PREFIX  foaf:http://xmlns.com/foaf/0.1/

SELECT *
{
?p a foaf:Person.
{
SELECT *
WHERE
{      SERVICEhttp://dbpedia.org/sparql{
?p a foaf:Person.
                              }

                     } limit 5
}
UNION
{
SELECT *
WHERE
{      SERVICEhttp://lod.openlinksw.com/sparql{
?p a foaf:Person.
                              }

                     } limit 5
}
}
```

Figura 13 - Esempio dell'utilizzo della clausola SERVICE

La query di esempio, in ***figura 13***, cerca sugli endpoint *specificati esplicitamente* nella clausola SERVICE tutti gli elementi che rispondono al pattern `?p a foaf:Person`. Il risultato ottenuto consegue l'unione dei risultati ottenuti dalle singole interrogazioni, questo perché nella query è stata utilizzata la parola chiave UNION; infatti se questa non fosse stata utilizzata, il risultato della query, come configurazione standard nell'uso della clausola SERVICE, sarebbe stato l'intersezione di tutti i risultati di ogni singola interrogazione.

Questo approccio, dal nostro punto di vista, presenta alcune problematiche.

In primo luogo il progettista delle query deve focalizzare il proprio sforzo funzionale sulla conoscenza esaustiva di tutto il dominio in cui agiranno le query, sia in termini di modello dei dati che dei dati stessi, e quindi non solo sulla conoscenza piena dello standard SPARQL 1.1.

Questo sforzo viene decretato dai modelli diversi presenti su ogni singolo endpoint utilizzato.

È chiaro che in un ambito Enterprise, costituito da molte fonti dati eterogenee ed ampie, questa soluzione presenta evidenti limiti di usabilità.

In secondo luogo un ulteriore aspetto penalizzante è la mancanza di un modello tecnico e di un modello logico su cui basare l'interrogazione stessa, rafforzando la tesi appena descritta. Infatti viene evidenziata la necessità di conoscere i modelli eterogenei da interrogare e quindi alla necessità, da parte dei query Architect, di conoscenze molto ampie e trasversali.

Inoltre si evidenzia l'ulteriore necessità di personale qualificato con competenze verticali sui singoli domini.

Infine il modello d'interrogazione, a livello logico, può essere accostato più ad un modello distribuito che non ad un modello federato. Infatti nella clausola service deve specificare ogni singolo endpoint sul quale effettuare la query, il pattern in comune tra loro ed infine la modalità di gestione del risultato.

In un modello d'interrogazione federato, ci saremmo aspettati l'uso di un'ontologia comune sui singoli endpoint e una maggiore trasparenza sul numero di endpoint coinvolti, cioè su "*chi fa parte della federazione*".

Si pensi alla possibilità di aggiungere un nuovo endpoint da interrogare o alla possibilità che esso modifichi il proprio modello dei dati.
Questo comporta, con l'attuale specifica, la riscrittura di tutte le query per introdurre le modifiche, ma soprattutto si deve introdurre una qualche processo organizzativo per capire come e quando queste situazioni si verificano e chi le deve gestire, operazione non banale e comunque molto costosa.

3.3 Middleware che implementano la Federazione

Dopo l'analisi approfondita delle feature messe a disposizione dal linguaggio SPARQL 1.1, la sperimentazione è proseguita cercando di individuare sul mercato dei middleware che supportavano tali feature.
La comparazione tra i middleware individuati è stata fatta seguendo delle linee guida molto importanti e centrali nel nostro obiettivo, veicolati anche dalle politiche interne all'azienda in cui si è svolta la tesi. In particolare si è cercato un middleware che principalmente supportasse, in modo nativo, la feature di federazione di endpoint. Inoltre si è utilizzati come ulteriori parametri di comparazione la possibilità di avere una significativa community di supporto, la possibilità di utilizzare Java come linguaggio per l'estensione del prodotto, la maturità del prodotto stesso sul mercato, la possibilità di avere una Licenza Open Source (o comunque senza particolari restrizioni tecniche), ed infine, ma non meno importante, la possibilità di usufruire dell'esperienza diretta degli sviluppatori dell'azienda in cui si è svolta la tesi.
Dopo un'attenta analisi e studio, si è passati alla definizione dei pro e dei contro di ogni prodotto, sia da un punto di vista generale, sia per ciò che concerne l'utilizzo di questi prodotti in ambito EA, ed infine si è proseguiti con la comparazione dei seguenti prodotti.

- Jena di Apache Software Fondation [APA15];
- Sesame di Aduna Software [SES15];
- AllegroGrapf di FrancInc. software [ALL15];
- Virtuoso di Openlink [VIR15].

3.3.1 Jena: analisi dei pro e dei contro.

Un primo aspetto da sottolineare in Jena, è l'operare mantenendo in memoria tutto il modello RDF. In ambito Enterprise l'uso forte e intenso della memoria può, però, presentare dei problemi di gestione dei dati e problemi di scalabilità. Si presuppone, infatti, che in quest'ambito vi siano grossi moli di dati da analizzare e gestire, quindi l'uso di Jena potrebbe risultare poco adatto.
Un secondo aspetto importante è l'integrazione di Fuseki [APJ14], il quel non risulta essere J2EE compliant, ma si presenta più come un server stand-alone. L'azienda in cui è stata sviluppata la tesi ha modificato questo framework per permetterne l'installazione su container J2EE (operazione fondamentale per i contesti Enterprise) e per fornire il supporto a query SPARQL maggiori di 2000 caratteri.
Infatti le interrogazioni in SPARQL sugli endpoint vengono veicolate tramite HTTP con chiamate GET, e alcuni browser presentano dei limiti di utilizzo.
Questa si presenta come una forte motivazione per non utilizzare Jena in ambito Enterprise, anche se risulta essere molto utile e ottimo nelle fasi di studio e sperimentazione.
L'architettura Jena è molto articolata e presenta diverse funzionalità, tra cui è di nostro interesse la parte relativa alle API SPARQL.
Oltretutto bisogna evidenziare la dismissione del SDB a favore del TDB, il quale viene portato avanti e gestito tutto sul filesystem, aspetto piuttosto negativo se si ha bisogno di utilizzare questo prodotto in ambito enterprise.
Rispetto agli altri framework analizzati in seguito, è stata valutata la possibilità dell'utilizzo diretto di questo framework, questo perché Jena può essere considerato il "mattoncino" alla base di tutti gli altri framework. Inoltre mette a disposizione una vasta gamma di funzionalità, risulta semplice da utilizzare, è totalmente scritto in Java, ed il fatto di essere il più utilizzato da un gran numero di applicazioni, si rispecchia in una community di supporto e discussione molto attiva.

3.3.2 Sesame: analisi dei pro e dei contro.

Purtroppo Sesame, come anche Jena, non ha il supporto alle configurazioni di High Availability (HA) (rif. Appendice E). Quando si opera in contesti Enterprise, l'High Availability (HA) è uno degli aspetti più importanti, soprattutto dal punto di vista di chi si occupa dell'esercizio del sistema. Anche se non è del tutto centrale e fondamentale per chi svolge Data Analisys, lo è per chi progetta e fornisce un framework che supporti in pieno endpoint SPARQL con tutte le relative feature.
Questo potrebbe essere una forte motivazione per non selezionare Sesame come framework quando si opera in ambiti Enterprise.
Se i progettisti di Sesame avessero sfruttato un'architettura J2EE, gestendo il tutto in un classico container, avrebbero fornito, oltre alle feature tipiche del modelli a container, l'High Availability (HA) e la possibilità di manutenere facilmente il server.
Questi aspetti sono molto importati e soprattutto positivi per spingere alla diffusione di un prodotto software in ambito Enterprise.
Mantenendo invece l'approccio architetturale descritto nel paragrafo 1.8.2, l'unico modo di usarlo è impacchettandolo in un appliance fornito As-Is al cliente e gestito completamente dal fornitore.
Questa politica è molto lontana da quella Open Source e da un classico modello di business di una società di consulenza, quindi potrebbe risultare come punto a sfavore per l'utilizzo del framework.

3.3.3 AllegroGraph: analisi dei pro e dei contro.

Il motore di AllegroGraph registra al suo interno le triple "soggetto-predicato-oggetto" in modo efficiente, assegnando ad ognuna un indice, chiamato UPI, tramite il quale gestisce le query in modo efficiente e veloce.
Dal punto di vista Enterprise, questo potrebbe risultare uno svantaggio. Infatti la gestione delle triple all'interno potrebbe risultare inefficiente quando si parla di grande mole di dati, si pensi all'ambito del Data Analisys o dei Big Data.

Permette inoltre di gestire la transazionalità delle operazioni, gestendo le quattro caratteristiche della proprietà ACID (Atomicità, Consistenza, Isolamento e Durabilità) (rif. Appendice F).

Queste proprietà, soprattutto nell'ambito della federazione, potrebbero essere ingestibili e non soddisfacibili. Si pensi alla garanzia dell'Atomicità in ambito federato, dovrebbe essere garantita non solo su ogni nodo, ma come un'unica transazione, con ovvi problemi di performance, impossibilità d'implementazione e gestione di una transazione distribuita. Inoltre bisognerebbe capire quanto l'atomicità ha senso in ambito federato, infatti, più che di atomicità delle transazioni si cerca di gestire e garantire il Data Quality.

Questo aspetto è collegato anche alla proprietà di consistenza, infatti se in DBMS viene a mancare un dato, questa proprietà viene meno, mentre in un Triple Store questo aspetto non è vero. Si pensi ad una qualsiasi tabella del DBMS nella quale si vuole aggiungere una nuova proprietà, se questa fosse senza dato non avrebbe senso aggiungerla, mentre nel triple store potrebbe aver senso e caratterizzare ulteriormente il concetto da esprimere aggiungendo informazioni.

Quindi il Data Quality risulta esser un aspetto più importante e centrale quando si parla di Triple Store. Infine, in un contesto federato, è difficile garantire una proprietà forte come quella ACID, questo perché non tutti i triple store sono fatti allo stesso modo (sia a livello hardware che software), ed inoltre il modello della federazione evolve in modo eterogeneo e incontrollato, aspetti che da soli evidenziano l'impossibilità di gestione e garanzia della proprietà ACID.

AllegroGraph mette a disposizione molte feature per manipolare i dati all'interno del triple store, permettendo l'utilizzo di applicativi Java o linguaggi come Lisp.

Infine, uno svantaggio chiaro, che ci ha convinto nell'escludere AllegroGraph, è riscontrabile nella creazione della federazione ogni qual volta si crea una sessione client, specificando i triplestore da federare. Inoltre, nel caso in cui si volesse aggiungere o eliminare un triple store della federazione, vi è la necessità di ravvio della sessione con il client, aspetto alquanto sfavorevole soprattutto nel caso di federazioni altamente dinamiche.

3.3.4 Virtuoso: analisi dei pro e dei contro.

Virtuoso ha guadagnato un notevole interesse poiché è utilizzato per ospitare molti importanti Linked Open Data (ad esempio, DBpedia). Questo ha permesso l'evolversi di una community di supporto molto estesa che ha assegnato un ulteriore punto a favore per questo framework.

Virtuoso viene offerto sia in una versione Open Source che in una commerciale. La versione Open Source ha creato molto aspettative per questo framework, anche se in ambito Enterprise avere un supporto commerciale risulta essere un elemento prezioso. Si pensi ad una società di consulenza informatica, la quale potrebbe offrire al cliente una soluzione di base per il prodotto richiesto, e poi è il cliente stesso ad avere la possibilità di acquistare l'assistenza commerciale strettamente funzionale alle proprie esigenze.

Un'altra caratteristica molto interessante di Virtuoso è la facilità nell'esporre le proprie funzionalità come Web Services. Infatti, può essere interrogato sia utilizzando SOAP che REST.

Un forte svantaggio da associare al framework Virtuoso è dettato dal linguaggio di sviluppo. Virtuoso è sviluppato in C, linguaggio che risulta difficile da maneggiare e gestire. Questo è stata un'ulteriore motivazione che ci ha spinto a non selezionale la soluzione di estensione di Virtuoso tramite plug-in, soluzione che verrà descritta in seguito.

Inoltre, come gli altri framework analizzati, Virtuoso non fornisce nessun funzionalità che permetta la gestione e la federazione di endpoint SPARQL. Offre, come da specifica di linguaggio Sparql 1.1 del W3C, la possibilità di utilizzare la clausola SERVICE tramite l'interfaccia standard SPARQL, con cui si espone l'endpoint.

3.3.5 Comparazione tra middleware

Dopo un'attenta analisi delle specifiche di ogni prodotto visionato, dei suoi vantaggi e dei suoi svantaggi, si è passati alla comparazione pesata dei vari middleware basandosi su KPI seguendo la linea guida definita in precedenza, avendo come aspetto principale la maturità dei prodotti rispetto alle KPI.
Questa comparazione ha avuto come obiettivo principale l'individuazione di un prodotto software che soddisfi, almeno in parte, i nostri requisiti, ma soprattutto evidenzi anche il fatto che, dal punto di vista della federazione, non esiste ad oggi sul mercato un prodotto maturo.
I pesi sono stati assegnati evidenziando come la KPI ***federazione*** fosse fondamentale nella comparazione. Come secondo parametro importante è stato utilizzato ***l'esperienza diretta*** *nell'azienda in cui si è sviluppata la tesi*. Infine le altre KPI sono state valutate tutte allo stesso livello di importanza. Nei paragrafi successivi sono state analizzate nel dettaglio tutte le KPI utilizzate nella comparazione.

I livelli di assegnazione del voto sono stati cosi definiti, rifacendosi al modello di Maturità CMMI:

- *0 – INCOMPLETE;*
- *1 – INITIAL;*
- *2 – MANAGED;*
- *3 – DEFINED;*
- *4 – QUANTITATIVELY MANAGED;*
- *5 – OPTIMIZING.*

3.3.5.1 KPI Federazione

Il KPI di federazione risulta essere il più importante indicatore nella comparazione, il tutto rispecchiato nell'assegnazione di un peso maggiore. Questo KPI è stato utilizzato per evidenziare e capire qual è l'attuale maturità dei middleware in relazione alla funzionalità di federazione.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: supporto della clausola service;*
- *1 – INITIAL: oltre al supporto della clausola service vi è almeno un feature di condivisione del modello dati (modello dati comune); questo permette l'azione sulla federazione in maniera trasparente in termini di membri;*
- *2 – MANAGED: trasparenza in termini di membri;*
- *3 – DEFINED: implementazione delle linee guida di gestione utilizzando il middleware;*
- *4 – QUANTITATIVELY MANAGED: misurabilità;*
- *5 – OPTIMIZING: processo di miglioramento continuo (ad esempio community di supporto).*

In relazione ai quattro middleware analizzati, il voto assegnato è stato 0 per Jena, AllegroGraph e Virtuoso, in quanto questi middleware non supportano nessuna feature di federazione, ma soltanto le specifiche di SPARQL 1.1.
Per ciò che riguarda, invece, Sesame, è l'unico a gestire la federazione tramite un repository creato in relazioni ai repository da gestire nella federazione utilizzando la classe RepositoryManagerFederation messa a disposizione nella api java. Questa gestione, però, non presenta la possibilità di gestire la condivisione del modello dati, ed inoltre presenta lo svantaggio della ridefinizione delle federazione ad modifica del repository.

3.3.5.2 KPI Community

Il KPI definita Community rispecchia la presenza di una community di supporto al middleware utilizzato, in particolare evidenziando la maturità di quest'ultima.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: community assente;*
- *1 – INITIAL: presenza di documentazione;*
- *2 – MANAGED: presenza di forum non presidiato;*
- *3 – DEFINED: presenza di forum presidiato;*
- *4 – QUANTITATIVELY MANAGED: issue traker e definizione di roadmap;*
- *5 – OPTIMIZING: suggerimento e possibili evoluzioni.*

Rispetto alla KPI Community il voto minore è stato assegnato ad AllegroGraph, in quanto non esiste una community attiva o quanto meno non presidiata, ma è disponibile solo la documentazione relativa al middleware.
Per ciò che riguarda Jena e Sesame, invece, vi è un'ampia community con possibilità di supporto diretto, issue traker e la definizione di roadmap.
La community più completa è risultata essere quella di Virtuoso, ben definita e ampia, con issue traker, definizione di roadmap, suggerimenti e possibili evoluzioni, e con la possibilità di supporto diretto online tramite chat.

3.3.5.3 KPI Supporto commerciale

Il KPI di *supporto commerciale* si riferisce precisamente alla possibilità di avere un qualsiasi supporto commerciale dalla società fornitrice del middleware.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: assenza di supporto commerciale;*
- *1 – INITIAL: possibilità di utilizzo del sistema mail per contattare il supporto;*
- *2 – MANAGED: possibilità di una consulenza generica sul prodotto;*
- *3 – DEFINED: system integrator, tramite partner;*

- *4 – QUANTITATIVELY MANAGED: possibilità di definizione e associazione di SLA (Service Level Agreement) alle installazioni;*
- *5 – OPTIMIZING: possibilità di definire soluzioni ad hoc.*

Il middleware Apache Jena purtroppo non fornisce nessuna tipologia di supporto commerciale, a differenza di AllegroGraph e Sesame che offrono rispettivamente la possibilità di avere delle consulenze generali sul prodotto e la possibilità di integrazione del sistema tramite partner aggiunti.
Il supporto commerciale completo viene fornito soltanto dal middleware Virtuoso, il quale offre, oltre alle soluzioni presenti negli altri middleware, la possibilità di definire e realizzare soluzioni ad hoc.

3.3.5.4 KPI Endpoint SPARQL e accesso a Triple Store

Il KPI di *Endpoint SPARQL e accesso al Triple Store* risulta essere molto importante. Riguarda la possibilità di esposizione di un endpoint tramite middleware e la conseguente possibilità di accedere tramite questo ad almeno un Triple Store.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: supporto alle specifiche SPARQL 1.1 e disponibilità di API;*
- *1 – INITIAL: specifiche SPARQL 1.1 e almeno un Triple Store associato, disponibilità di API e di uno store non Enterprise;*
- *2 – MANAGED: presenza di un installer monolitico;*
- *3 – DEFINED: presenza di un installer con personalizzazioni forti sull'esposizioni (ad esempio utilizzo di REST, SOA, ecc.) e la ulteriore possibilità di scelta del Triple Store da utilizzare;*
- *4 – QUANTITATIVELY MANAGED: disponibilità di un tool di monitoraggio;*

- *5 – OPTIMIZING: disponibilità di un tool per estendere parti del prodotto per soluzioni ad hoc.*

Rispetto a questa KPI, solo Virtuoso mette a disposizione un middleware completo, fornendo la possibilità di esporre un endpoint, sia standard che custom. Fornisce tutto il supporto necessario, la possibilità di gestire sia Triple Store singoli che multipli, ed infine, ma non meno importante, fornisce dei tool per gestire, monitorare e realizzare soluzioni ad hoc.
Mentre Jena e Sesame, in questo senso, danno la sola possibilità di esposizione dell'endpoint SPARQL che rispetta le specifiche 1.1.
AllegroGraph, infine, fornisce un installer monolitico per la configurazione dell'endpoint SPARQL.

3.3.5.5 KPI Uso di Java

Il KPI *Uso di Java* rappresenta la possibilità di utilizzare nel middleware il linguaggio di programmazione Java come linguaggio di sviluppo. Questo aspetto è importante, in quanto spesso i contesti Enterprise sono Java Oriented, quindi rappresenta un plus la possibilità di utilizzo di questo linguaggio.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: impossibilità di estensione Java;*
- *1 – INITIAL: API di interazione Java;*
- *2 – MANAGED: Wrapper utilizzabile come black box;*
- *3 – DEFINED: middleware scritto interamente in Java;*
- *4 – QUANTITATIVELY MANAGED: disponibilità plug-in Java;*
- *5 – OPTIMIZING: SDK integrabile in Eclipse.*

Analizzando a fondo i quattro middleware, è stato constatato che l'unico a non avere un buon supporto per Java è Virtuoso, questo perché il middleware è stato sviluppato

in C#. Virtuoso mette comunque a disposizione la possibilità di interazione con API Java.
Il voto più alto in questa comparazione è stato assegnato a Jena, questo è giustificato dal fatto che questo middleware è totalmente sviluppato in Java. Anche il middleware Sesame è sviluppato in Java, ma la differenza sostanziale tra questi due middleware risiede nel fatto che Jena mette a disposizione dei plug-in integrabili, mentre Sesame solo le API base.
AllegroGraph, infine, mette a disposizione solo delle API Java utilizzabili come black box.

3.3.5.6 KPI Esperienza Imola Informatica

Il KPI *Esperienza Imola Informatica* ha avuto molto importanza nella comparazione, tanto da associargli un peso quasi uguale alla feature principale di federazione. Questo perché avere la possibilità di un supporto diretto di tecnici specializzati permette di sfruttare l'esperienza diretta, la professionalità e i consigli, utilizzandoli come bagaglio per la scelta del middleware.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: nessuna esperienza;*
- *1 – INITIAL: prove di concetto (PoC);*
- *2 – MANAGED: almeno un progetto sviluppato;*
- *3 – DEFINED: più progetti sviluppati;*
- *4 – QUANTITATIVELY MANAGED: esistenza di un gruppo di persone che utilizza stabilmente quel middleware;*
- *5 – OPTIMIZING: sviluppo parti del middleware.*

In relazione ai middleware analizzati e all'esperienza dei tecnici dell'azienda in cui si è svolta la tesi, è emerso che Virtuoso e Jena sono i più utilizzati. In particolare con Virtuoso sono stati utilizzato per verificare la sua adozione come Triple Store e come

esposizione dell'endpoint SPARQL. Per ciò che riguarda Jena, invece, è stato utilizzato come middleware core per sviluppare una molteplicità di soluzioni, fornendo ampio supporto e soluzioni custom.
Infine per i middleware Sesame e AllegroGraph sono state fatte solo delle PoC (Proof of Concept).

3.3.5.7 KPI Maturità del prodotto

Il KPI *Maturità del prodotto* è stato utilizzato per analizzare le varie versioni dei middleware presenti attualmente, e la conseguente maturità in termini di stabilità.

I livelli di assegnazione riferiti al modello CMMI sono cosi definiti:

- *0 – INCOMPLETE: versione non stabile;*
- *1 – INITIAL: versione 1.0 del prodotto (prima versione stabile);*
- *2 – MANAGED: versione stabile con fix nell'ultimo anno;*
- *3 – DEFINED: almeno una versione stabile con update programmati;*
- *4 – QUANTITATIVELY MANAGED: casi d'uso di successo documentati;*
- *5 – OPTIMIZING: esiste almeno un caso che affronta volumi di dati.*

Dall'analisi è scaturito che sia AllegroGraph che Sesame forniscono delle versioni stabili del prodotto, con rilasci tipicamente annuali.
Mentre per il middleware Jena vi è un maggior frequenza e supporto a livello di release, abbastanza frequenti e programmate.
Infine Virtuoso, anche per questa KPI, ha ottenuto il voto maggiore, in quanto oltra alla forte community fornisce casi d'uso reali e di successo, dettagliatamente documentate e supportate.

3.3.6 Risultato della comparazione

Nella ***tabella 4*** vi è il prospetto riassuntivo delle KPI appena descritte e discusse, con specificato il dettaglio dei valori associati.

	Peso	*Jena*	*AllegroGraph*	*Virtuoso*	*Sesame*
Federazione	***0,3***	***0***	***0***	***0***	***1***
Community	*0,1*	4	2	**5**	4
Supporto commerciale	*0,1*	0	2	**5**	3
Esp. endpoint Sparql	*0,1*	1	2	**5**	1
Uso di Java	*0,1*	4	2	**1**	3
Esperienza Imola Informatica	*0,2*	3	1	**3**	1
Maturità del prodotto	*0,1*	3	2	**4**	2
	1	***1,8***	***1,2***	***2,6***	***1,8***

Tabella 4 - Comparazione Middleware

Valutando in parallelo, però, tutte le features messe a disposizione dai framework analizzati, si è giunti alla constatazione che nessuno di essi fornisce un supporto totale e soddisfacente alla federazione, soprattutto analizzando il tutto secondo il principio della condivisione del modello dati, centrale nella nostra sperimentazione, lasciando emergere i problemi descritti in precedenza.

Si è giunti a questa affermazione, in quanto dall'analisi è emerso che, per ogni singolo framework relativo alla feature di federazione, emergevano delle mancanze. Tutti supportano la specifica SPARQL 1.1, quindi è evidente l'emergere dei problemi descritti nel paragrafo delle specifiche; nessuno dei framework prevede l'implementazione e il supporto ad un patto federativo, elemento principale e trainante della nostra tesi. Infine nessun dei framework mette a disposizione una feature di federazione trasparente all'endpoint stesso.

Sesame e Jena si avvicinano molto alle esigenze da noi ricercate in un framework, perdendo, però, qualche punto principalmente nella non implementazione della feature di federazione, successivamente nell'esposizione dell'endpoint SPARQL, aspetto per noi molto importante per rendere indipendente l'endpoint stesso, ed infine nella community di supporto disponibile.

Nel definire una soluzione si è scelto di utilizzare il framework di Openlink Virtuoso. Questa scelta è giustificata principalmente dalla possibilità di usufruire del supporto e dell'esperienza su questo framework direttamente dal personale specializzato dell'azienda in cui si è svolta la tesi, la quale ha scelto Virtuoso perché dispone di una community molto ampia e offre delle feature molto interessanti come la possibilità di gestire in modo più efficiente la scalabilità in contesti Enterprise e una potenziale velocità di esecuzione delle query superiore a quella degli altri prodotti. Inoltre questo framework fornisce sia la versione Open Source e sia la versione commerciale per funzionalità più importanti e complete. Infine, risulta essere il prodotto più maturo attualmente sul mercato.

3.3.7 Performance

Per ciò che riguarda le performance, i framework sono stati comparati utilizzando come linea guida le regole dettate dalle Benchmark Specification [CHR15].

I benchmark analizzati sono qui elencati:

- *The Berlin SPARQL Benchmark [CHR08];*
- *Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints [CHR09];*
- *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data [MOH11];*
- *AllegroGraph RDFStore Benchmark [ALL14];*
- *Virtuoso Benchmark [VIR14].*

Il ***Berlin SPARQL Benchmark*** si occupa del confronto delle prestazioni di vari Triple Store RDF, confrontando le prestazioni di sistemi di storage che espongono endpoint SPARQL via protocollo SPARQL.
Questo Benchmark si occupa del confronto prestazionale sia dei sistemi Triple Store nativi ed inoltre sistemi che riscrivono le query SPARQL in query SQL.

Benchmarking the Performance of Storage Systems thatexpose SPARQL Endpoints si occupa anch'esso di confrontare le prestazioni di sistemi di storage in architetture. È costituito su di un particolare caso d'uso e-commerce, in cui un insieme fornitori forniscono una vasta gamma di prodotti, e dove i consumatori inviano recensioni sui prodotti.
Il mix di query di riferimento illustra la ricerca e la navigazione dei prodotti da parte dei consumatori.
Vengono quindi presentati i risultati ottenuti confrontando Store RDF (Sesema, Virtuoso, Jena TDB, Jena SDB e altri) e store come un Server D2R che utilizza database relazionali per mappare dati RDF.

DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data si occupa di analizzare le prestazioni di query generate da umani o applicativi, eliminando il mappaggio di queste su database relazionali. In particolare sono state analizzate la prestazioni di Virtuoso, Sesame, Jena TDB, AllegroGraph e BigOWLIM, sia a livello singolo, sia tenendo in considerazione query log mining, clustering e SPARQL feature analysis.

I benchmark di ***AllegroGraph e Virtuoso*** sono forniti direttamente dalle società produttrici. In questi benchmark, molto dettagliati, è possibile analizzare tutte le prestazioni di questi due middleware in varie tipologie di configurazioni, singoli, in Clustering; inoltre è possibile analizzare le prestazioni in termini di caricamento dei dati, di esecuzione di una singola query da singolo client o da multi client, e altri tipologie di prestazioni significative.

La comparazione è stata effettuata tenendo conto dei risultati ottenuti nelle operazioni di *Load del dataset* (con la dimensione delle triple variante tra 1M, 25M e 100M), *Execute di un set di query, sia su single che su multi client*.

La ***tabella 5*** evidenzia le differenze, in termini di ***tempo di load***, misurato in gg[hh:mm:sec], dei vari framework nel caricamento di un dataset rispettivamente di 1M di triple, 25M di triple ed infine 100M di triple.

	1M	*25M*	*100M*
Sesame	*00:02:59*	*12:17:05*	*3:06:27:25*
Virtuoso	***00:00:34***	***00:17:15***	***01:03:53***
AllegroGraph	*00:00:52*	*00:36:49*	*00:48:30*
Jena TDB	*00:49*	*00:16:53*	*01:34:14*

Tabella 5 - Comparazione performance: Load dataset.

Le ***tabelle 6 e 7*** evidenziano le differenze, in termini ***esecuzione delle query*** in un situazione di ***singolo client*** e di ***multi client***, dei vari framework con un dataset rispettivamente di 1M di triple, 25M di triple ed infine 100M di triple.
I risultati sono stati ottenuti misurando l'esecuzione del set di query su ogni store e riportando i risultati come rapporto Query per ora (Query Mixed per hour – QMpH)

	1M	*25M*	*100M*
Sesame	*18,094*	*1,343*	*254*
Virtuoso	***17,424***	***12,972***	***4,407***
AllegroGraph	*4,075*	*493*	*656*
Jena TDB	*4,450*	*353*	*81*

Tabella 6–Esecuzione set query su singolo client.

Nell'analizzare l'esecuzione su multi-client si è considerato solo in caso in cui il set di triple è 1M, analizzando le risposte all'aumentare dei client in esecuzione.

	Number of client (dataset 1M)		
	2	***8***	***64***
Sesame	*19,057*	*18,295*	*16,517*
Virtuoso	***28,985***	***32,668***	***33,339***
AllegroGraph	*5,861*	*7,453*	*7,888*
Jena TDB	*6,752*	*8,453*	*8,664*

Tabella 7– Esecuzione set query su multi-client.

Dall'analisi dei risultati appena mostrati si evince subito che il framework Openlink Virtuoso risulta avere i dati medi migliori. Un aspetto molto importante per ottimizzare una applicazione che utilizza endpoint SPARQL, indipendentemente da come è realizzato l'endpoint stesso, è il tempo di risposta all'esecuzione di una query o di un set di query. In questo Virtuoso ha mostrato dei risultati nettamente migliori rispetto agli altri framework analizzati, in particolare nel caso di multi-client, dove si evince la capacità del framework di supportare un set multiplo di query da più client (valore di 33,339 QMpH).

4. Federazione di Endpoint SPARQL: architettura logica

In questo quarto capito sono state esplorate nel dettaglio le ipotesi di lavoro individuate nella fase di studio. Sono stati successivamente definiti e discussi i parametri per valutare e comparare queste soluzioni. Infine sono state analizzate le performance di ogni ipotesi di lavoro.

Dopo un'attenta analisi e comparazione dello standard SPARQL 1.1 e dei middleware presenti sul mercato si è giunti alla conclusione che non vi è a tutt'oggi nessuna standard di implementazione per la feature di federazione.
Dopo aver preso atto di queste problematiche, la sperimentazione è proseguita nell'individuazione di possibili ipotesi di lavoro sviluppabili per implementare la *federazione di endpoint SPARQL*.
In particolare, ne sono state individuate tre: evoluzione su singolo endpoint, creazione di un plug-in, ed infine la soluzione tramite Web Service.
Analizziamo nel dettagli il funzionamento di ogni singola ipotesi di lavoro individuata.

4.1 Ipotesi di lavoro: evoluzione su singolo endpoint

La prima soluzione è incentrata sull'evoluzione della logica interna di un endpoint SPARQL, passando dalla classica logica di mera esecuzione della query per l'estrazione dei dati ad una logica che prevedeva i seguenti passaggi:

1. *intercettazione della query* (scritta come se si interrogasse un solo endpoint);
2. *riscrittura della query, attraverso della logica applicativa cablata, inserendo gli statement necessari ad interrogare tutti i nodi della federazione;*
3. *ed esecuzione sui vari endpoint e restituzione del risultato in base a quanto previsto dalle specifiche della clausola SERVICE di SPARQL 1.1.*

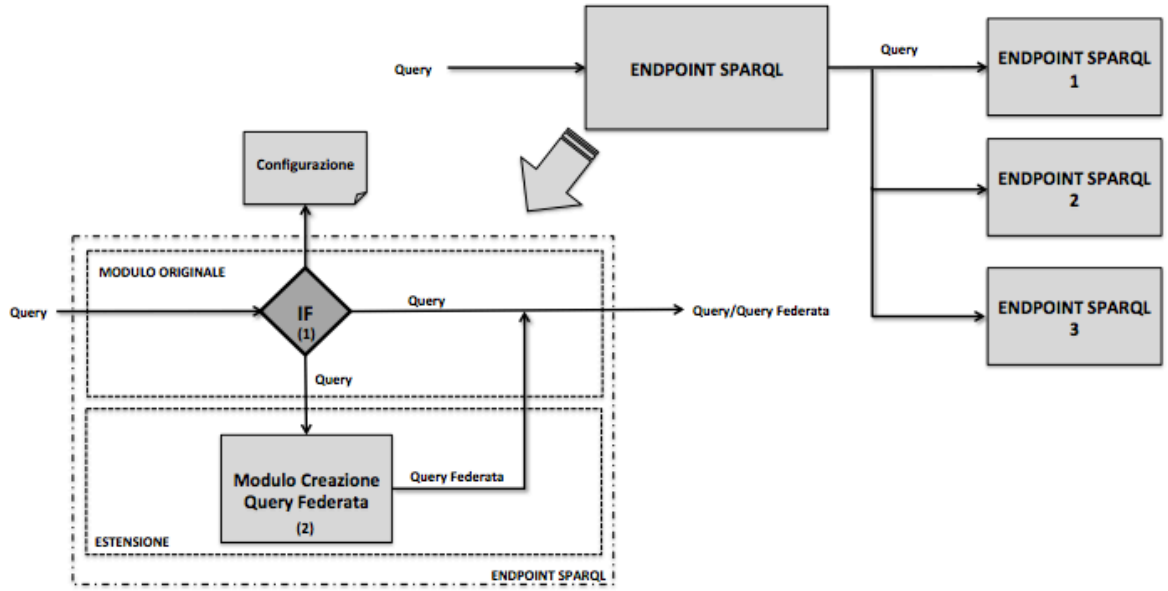

Figura 14 - Ipotesi di lavoro: evoluzione su singolo endpoint

Nello specifico il modulo IF:

- legge una configurazione per conoscere gli endpoint SPARQL che fanno parte della federazione;
- analizza la query in ingresso:
  - se questa deve essere federata la invia al modulo di Creazione Query Federata;
  - se questa non deve essere federata esegue la query sull'endpoint specifico;

Il modulo di Creazione Query Federata ottiene in ingresso la query, la trasforma in query federata applicando la clausola SERVICE e specificando tutti gli endpoint della federazione presenti nella configurazione.

Nella figura seguente una possibile manipolazione della query effettuata dal modulo.

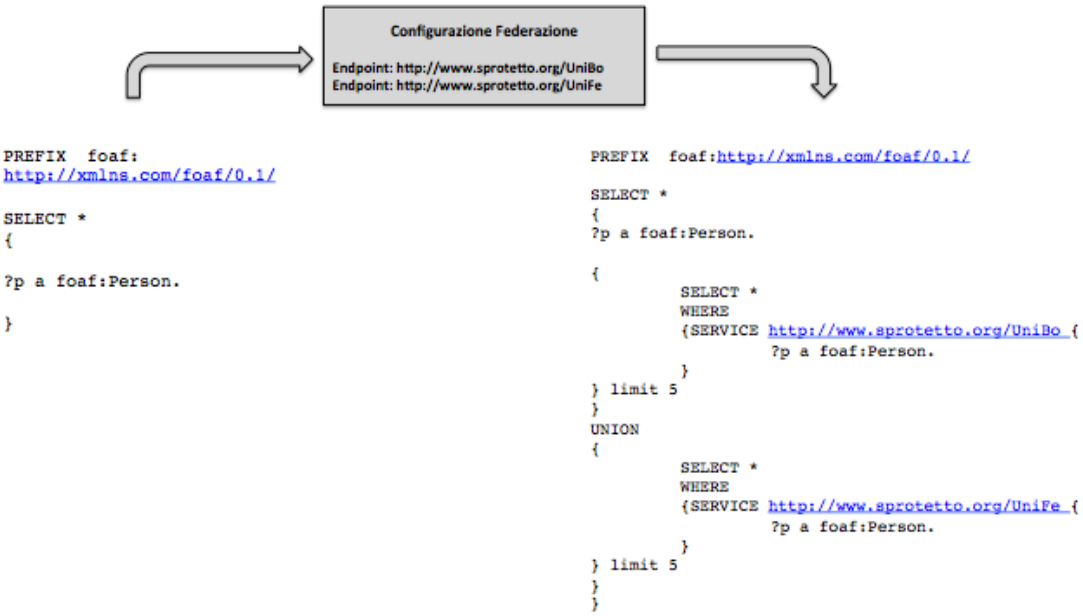

Figura 15 - Esempio di manipolazione della query.

Questa prima soluzione permette di fornire in ingresso qualsiasi tipologia di query SPARQL, con l'unico vincolo di non poter utilizzare la clausola SERVICE, sia perché questa viene utilizzata come costrutto principale per la manipolazione, sia perché renderebbe la manipolazione e la query finale altamente complesse.
Questo aspetto è importante in quanto non permette l'utilizzo di tutti i costrutti conformi allo standard SPARQL 1.1, conferendo il valore aggiunto più al prodotto che alla soluzione.
Un altro aspetto da considerare è la difficoltà nella manipolazione della query stessa, la quale potrebbe comportare una complessità di implementazione non banale, presentando problemi di gestione e performance importanti.

4.2 Ipotesi di lavoro: Plug-in

La seconda soluzione è incentrata sulla definizione di un plug-in, nel quale implementare tutte le funzionalità per la gestione della federazione.
Vediamone una rappresentazione logica in figura.

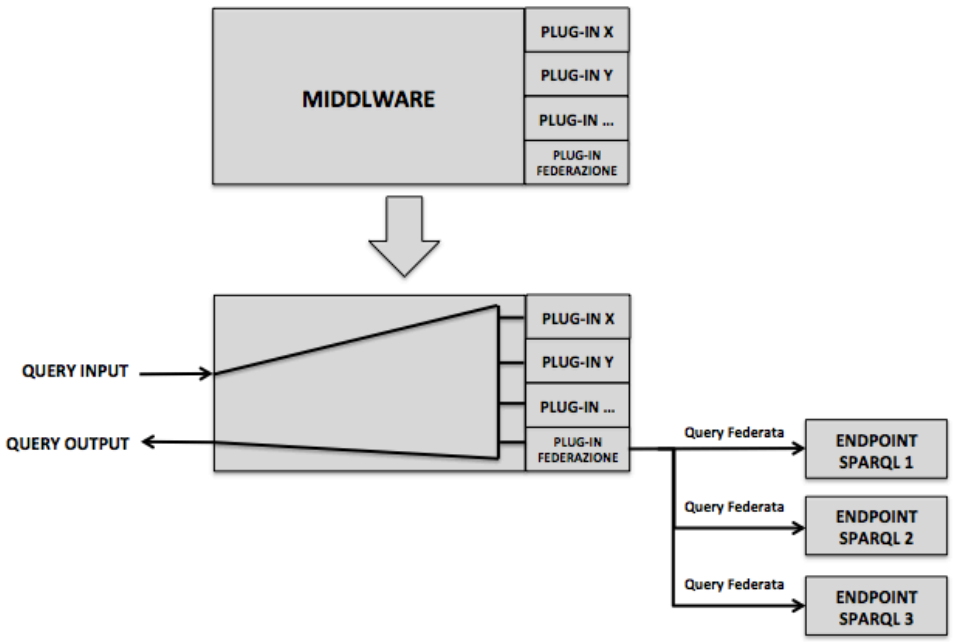

Figura 16 - Ipotesi di lavoro: Plug-in.

In questa soluzione vi è la possibilità di creare un plug-in che gestisca la federazione (in ***figura 16*** rappresentato logicamente), prendendo in ingresso una query generale, applicando alla stessa le eventuali altre regole degli altri plug-in, ed infine federarla secondo la configurazione installata, restituendo in uscita il risultato ottenuto.

Questa modalità permette di utilizzabile in contemporanea altri plug-in, ma soprattutto permette di configurare il plug-in stesso secondo le specifiche e le esigenze di utilizzo.

Questa configurazione può essere effettuata rispettando le linee guida di sviluppo del framework utilizzato, oppure tramite un'ontologia che descriva la composizione della federazione (scelta da noi preferita perché basata su tecnologie standard).

Anche questa soluzione presenta problemi simili alla prima, anche se l'uso di un'estensione realizzata secondo le specifiche di prodotto rende sicuramente preferibile questa soluzione rispetto alla prima per diversi motivi:

- risulta più facile la procedura di installazione;
- lo sviluppo segue specifiche ufficiali;
- non sono necessarie particolari indicazioni in termini di manutenzione ed esercizio del sistema.

Inoltre dal confronto delle due soluzioni si evince che dal punto di vista prettamente accademico si ottiene lo stesso risultato, mentre dal punto di vista IT la preferenza ricade su questa seconda soluzione perché permette la distribuzione del codice a prescindere dal vendor che sviluppa il framework. Infatti l'eventuale evoluzione del plug-in è collegata alla versione del framework rendendo più facile la gestione del codice stesso.
Inoltre nella gestione delle licenze nel primo caso è strettamente legata a framework sul quale si sviluppa l'estensione, mentre nel secondo caso è legata agli sviluppatori del plug-in, i quali possono essere diversi dallo sviluppatore del framework (es. Virtuoso potrebbe essere Open Source, mentre il plug-in potrebbe avere un costo imposto dai suoi sviluppatori diversi da quelli di Virtuoso, quindi gestione delle licenze separate).
Allo stesso tempo, però, si perde completamente di indipendenza dal prodotto e quindi alla fine anche questa soluzione risulta più orientata ad architetture ad hoc piuttosto che a soluzioni IT orientate ai servizi e alla riduzione dei lock-in.

4.3 Ipotesi di lavoro: Web Service

La terza soluzione, da noi scelta e implementata, si basa sulla creazione di un servizio applicativo nel quale conferire tutte le funzionalità relative alla federazione e richieste per la soluzione.
Questo servizio è stato realizzato tramite l'utilizzo della tecnologia basata su Web Service.
In ***figura17*** una rappresentazione logica del funzionamento.

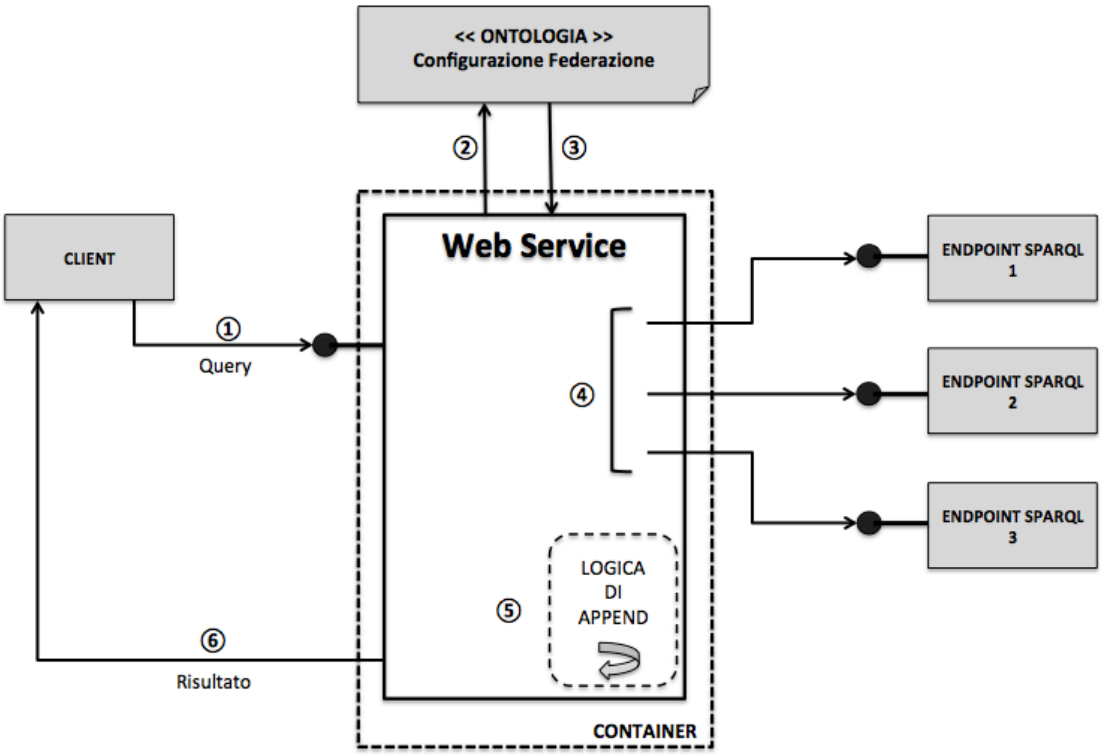

Figura 17 - Ipotesi di lavoro: Web Service.

La soluzione prevede che:

1) Il client invia al Web Service la query generale da eseguire sulla federazione;
2) Il Web Service legge la configurazione della federazione, gestita su un endpoint esterno tramite un'ontologia;
3) Ottiene la lista degli endpoint che fanno parte della federazione;
4) Interroga gli endpoint della lista;
5) Applica la logica di APPEND senza gestione dei duplicati, da noi scelta come soluzione di default, a tutti i risultati di tutti gli endpoint;
6) Restituisce il risultato al chiamante.

I vantaggi di questa soluzione, rispetto alle altre due discusse in precedenza, sono molteplici:

- Vi è una totale separazione dei componenti dell'architettura logica e dei ruoli che questi devono ricoprire, eliminando di fatto il lock-in tecnologico e fornendo una struttura sicuramente più adatta all'integrazione in stile SOA, trend in corso in tutte le Enterprise;

- Vi è la possibilità di interrogare la federazione utilizzando lo standard SPARQL 1.1, risolvendo il problema della non possibilità di utilizzare la clausola SERVICE presentato nelle soluzioni precedenti;
- Non vi è alcuna manipolazione della query, problema forte nella prima soluzione;
- Nessun impatto sulla logica di gestione, dettato dall'aggiunta o eliminazione di entità dalla federazione;
- La struttura modulare della soluzione permette l'utilizzo degli endpoint sia nella federazione, sia come singoli endpoint;
- L'uso di un WS consente la sua installazione in un container J2EE standard e quindi di risolvere alla base i problemi di scalabilità orizzontale e verticale della nostra soluzione grazie alle caratteristiche dell'architettura scelta.
- Infine questa soluzione non introduce un overhead tecnologico alle normali tecnologie Enterprise, poiché sono già pienamente utilizzate in quest'ambito e quindi totalmente e facilmente integrabili.

4.4 Elementi per la valutazione delle ipotesi di lavoro.

La valutazione delle soluzioni appena illustrate è stata fatta anche in base ad alcuni parametri che abbiamo ritenuto significativi sia a livello di ricerca pura che a livello di contesto aziendale:

- *Rapidità di sviluppo;*
- *Integrazione (intesa come facilità di integrazione);*
- *Facilità di Manutenzione;*
- *Potenzialità evolutive.*

Questi parametri possono assumere i valori (soggettivi) basso, medio e alto, in base alle caratteristiche degli elementi tecnologici e architetturali delle singole soluzioni che sono emersi dalla documentazione dei prodotti utilizzati o dalle prove sperimentali effettuati durante la ricerca.

4.4.1 Rapidità di sviluppo

Per ***rapidità di sviluppo***, s'intende la facilità di realizzazione della soluzione a prescindere dalla semplicità di manutenzione della stessa, per cui questo parametro diventa fondamentale per eventuali dimostrazioni e prove di concetto, mentre risulta secondario per soluzioni Enterprise o commerciali.

		Jena	*AllegroGraph*	*Sesame*	*Virtuoso*
Rapidità di sviluppo	***Evo. su singolo endpoint***	*ALTO*	*MEDIO*	*MEDIO*	*BASSO*
	Plug-in	*BASSO*	*MEDIO*	*MEDIO*	*ALTO*
	Web Service	*ALTO*	*ALTO*	*ALTO*	*ALTO*

Tabella 8 - Elementi per la valutazione delle ipotesi di lavoro: Rapidità di Sviluppo.

La valutazione di questo parametro sui singoli casi è descritta di seguito:

- *Estensione del singolo endpoint*: in questo caso è stato assegnato un valore alto a Jena, rispecchiando il fatto di essere totalmente sviluppato in Java, ampiamente utilizzato e supportato dalla community;
- *Plug-in*: in questo caso il valore alto è stato assegnato a Virtuoso semplicemente perché è l'unico framework, tra quelli analizzati, ad avere una struttura a plug-in;
- *Web Service*: il valore alto qui è assegnato ad ogni framework, in quanto la struttura della soluzione è indipendente da quello che viene utilizzato.

4.4.2 Facilità di Integrazione

Il parametro ***facilità di integrazione*** è riferito alla possibilità di interagire tra i singoli componenti evitando la scrittura di adattatori e utilizzando protocolli standard e diffusi.

		Jena	AllegroGraph	Sesame	Virtuoso
Facilità di Integrazione	***Evo. su singolo endpoint***	*MEDIO*	*MEDIO/ BASSO*	*MEDIO/ BASSO*	*ALTO*
	Plug-in	*BASSO*	*MEDIO*	*MEDIO*	*ALTO*
	Web Service	*ALTO*	*ALTO*	*ALTO*	*ALTO*

Tabella 9 - Elementi per la valutazione delle ipotesi di lavoro: facilità di integrazione.

La valutazione di questo parametro sui singoli casi è descritta di seguito:

- *Estensione del singolo endpoint*: questa soluzione appare la meno integrabile di tutte, in quanto l'estensione avviene direttamente su codice (di proprietà altrui) e quindi prevede dei passaggi formali (anche extra IT) per la sua distribuzione. Inoltre più un prodotto è complesso e più sarà difficile identificare i punti d'integrazione sul codice e quindi per questo motivo si è indicato Virtuoso come il più integrabile.
- *Plug-in*: l'architettura a plug-in è prevista solo in Virtuoso, per cui questo prodotto è l'unico facilmente integrabile attraverso questa soluzione, mentre per gli altri si dovrebbe prevedere qualche forma di libreria da distribuire a parte e da integrare tramite modifiche al codice originale;
- *Web Service*: l'assegnazione di un valore alto a tutti è giustificata dal fatto che gli endpoint e il WS possono essere sviluppati con qualsiasi tecnologia. Inoltre la forte integrazione è giustificata dall'utilizzo dell'architettura SOA, di per sé orientata a risolvere i problemi di integrazione.

4.4.3 Facilità di Manutenzione

Il parametro ***facilità di manutenzione*** è inteso come indicazione dell'effort per risolvere eventuali bug o per piccole evoluzioni che non stravolgano la logica applicativa o l'architettura della soluzione.

		Jena	*AllegroGraph*	*Sesame*	*Virtuoso*
Facilità di Manutenzione	***Evo. su singolo endpoint***	*BASSO*	*BASSO*	*BASSO*	*BASSO*
	Plug-in	*BASSO*	*BASSO*	*BASSO*	*ALTA*
	Web Service	*ALTO*	*ALTO*	*ALTO*	*ALTO*

Tabella 10 - Elementi per la valutazione delle ipotesi di lavoro: facilità di manutenzione.

La valutazione di questo parametro sui singoli casi è descritta di seguito:

- *Estensione del singolo endpoint*: è stato assegnato un valore basso a tutti i framework perché in questa soluzione si prevede la loro modifica diretta da far approvare dal gruppo di sviluppo ufficiale, oppure da mantenere nel tempo seguendo lo stesso ritmo di rilasci del prodotto: operazione ovviamente molto difficile e costosa;
- *Plug-in*: anche in questo caso la manutenibilità è strettamente legata al fatto che la soluzione a plug-in è applicabile solo nel caso si utilizzasse Virtuoso, per cui questo framework è l'unico ad ottenere una valutazione alta in questo contesto;
- *Web Service*: in questo caso la facilità di manutenzione risulta essere estremamente alta, indipendentemente dal framework di supporto utilizzato, questo perché siamo noi gli sviluppatori in toto del prodotto e quindi siamo noi i responsabili della manutenzione.

4.4.4 Potenzialità Evolutive

Il parametro ***potenzialità evolutive*** è riferito alla velocità di sviluppo di nuove logiche di business senza impattare sull'architettura in modo significativo e senza introdurre sessioni di test di non regressione particolarmente lunghe.

		Jena	*AllegroGraph*	*Sesame*	*Virtuoso*
Potenzialità Evolutive	***Evo. su singolo endpoint***	*BASSO*	*BASSO*	*BASSO*	*BASSO*
	Plug-in	*MEDIO /BASSO*	*MEDIO/ BASSO*	*MEDIO/ BASSO*	*ALTO*
	Web Service	*ALTO*	*ALTO*	*ALTO*	*ALTO*

Tabella 11 - Elementi per la valutazione delle ipotesi di lavoro: potenzialità evolutive.

La valutazione di questo parametro sui singoli casi è descritta di seguito:

- *Estensione del singolo endpoint*: qualsiasi prodotto utilizzato in questa soluzione ha, a nostro parere, un basso grado di potenzialità evolutive, perché il codice prodotto dovrebbe essere ufficialmente approvato dal gruppo di sviluppo ufficiale o dovrebbe essere adattato e testato sulle singole versioni rilasciate;
- *Plug-in*: in questo caso l'unico prodotto a poter supportare nuove feature attraverso dei plug-in è Virtuoso, mentre per gli altri framework è ipotizzabile la creazione di estensioni attraverso API e librerie, le quali risultano sicuramente più difficili da estendere;
- *Web Service:* le potenzialità di sviluppo risultano alte a prescindere dal framework utilizzato. Infatti si potrebbe evolvere il prodotto creato aggiungendo funzionalità ulteriori nella gestione delle federazione senza impattare sulla tecnologia degli endpoint. Inoltre, se la federazione fosse implementata negli endpoint come standard, basterebbe semplicemente dismettere il componente software, con vantaggi maggiori rispetto alle prime due soluzioni. Si pensi infatti alla necessità di dover eliminare tutti i plug – in sviluppati ad hoc oppure dismettere tutto il codice creato.

4.5 Valutazione costi

Nella valutazione dei costi delle singole ipotesi di lavoro sopra discusse, si è tenuti conto principalmente di due tipologie di costi; nello specifico di costi diretti, intesi come i costi di sviluppo della soluzione applicativa, e di costi indiretti, intesi come i costi di manutenzione dell'applicazione stessa.
La stima è stata fatta in riferimento al nostro contesto di sperimentazione, utilizzato il modello dati sviluppato ad hoc, e i sistemi informativi selezionati.

4.5.1 Ipotesi di lavoro: evoluzione su singolo endpoint

Relativamente alla prima ipotesi di lavoro evoluzione su singolo endpoint l'analisi è stata realizzata dimostrando la fattibilità e la possibile realizzazione della soluzione (PoC – Proof of Concept).
In questo caso le modifiche avvengono hard coded sull'endpoint, quindi le possibili modifiche sono limitate dall'endpoint stesso.
Questa soluzione presenta una serie di impatti importanti.
In primis il lock-in verso una particolare versione del middleware, infatti tutte le modifiche vengono effettuate in base alla versione del middleware utilizzata, questo comporta il fatto che, al rilascio di una nuova versione, le modifiche debbano essere riapplicate nuovamente.
Un ulteriore impatto è il costo evolutivo. Quando viene effettuata una modifica al middleware, anche se questa risulta essere minima, bisogna predisporre dei test da effettuare. Questi test comportano un costo almeno pari a quello dello sviluppo della modifica, con la conseguenze che al rilascio della nuova versione deve essere rifatta non solo la modifica del middleware, ma tutti i relativi test.
Un ulteriore problema di questa soluzione è il lock-in verso il middleware, quindi la soluzione sviluppata risulta essere di difficile migrazioni verso altri middleware e piattaforme.

Infine se vi è mancanza di documentazione, si avrà l'ulteriore lock-in relativo al personale che sta sviluppando quella particolare patch per quella particolare versione del middleware.
Quindi l'impatto in termini di IT può risultare importante, come ad esempio nel caso in cui si hanno endpoint distribuiti, rischiando di avere un lock-in sulla versione del middleware perdendo quindi di assistenza e avendo forti problemi organizzativi.
In conclusione questa soluzione è interessante per la realizzazione di PoC, in tempo breve si riesce a definire una soluzione immediata, ma con il rischio di avere una soluzione molto complicata e con problemi descrittivi e di realizzazione.

4.5.2 Ipotesi di lavoro: Plug-in

Nell'ipotesi di lavoro Plug-in i costi di realizzazione sono stati quantificati in circa 10 giorni. Questa stima, però, può essere facilmente sconvolta in base al fatto di utilizzare questa soluzione in un contesto a bassa innovazione o ad alta innovazione tecnologica. Infatti, in un contesto a bassa innovazione tecnologica il prodotto viene acquistato da un vendor e utilizzato direttamente, mentre in un contesto ad alta innovazione tecnologica adottare questa soluzione può ripercuotersi sulle performance dal punto di vista della gestione IT.
Anche in questa tipologia di soluzione si possono presentare dei problemi e impatti importanti.
In particolare la presenza di lock-in sul middleware, questo è intuibile in quanto il plug-in è un componente strettamente legato al middleware sul quale viene sviluppato.
Un secondo problema è il lock-in sulla versione, che tipicamente non dovrebbe presentarsi nelle varie evoluzioni del middleware, ma che in realtà potrebbe presentare problemi di compatibilità anche forti.
Il lock-in sul personale in questa soluzione non è presente, questo perché lo sviluppo del plug-in avviene tramite un gruppo di lavoro ufficiale, con la conseguenza di riuscire a generare e mantenere l'assistenza.

Infine in questo caso non ho libertà di scelta e implementazione del middleware, infatti sono strettamente legato al middleware utilizzato. Questo però comporta un contenimento dei costi di sviluppo, dei costi di manutenzione.

4.5.3 Ipotesi di lavoro: Web Service

Nell'ipotesi di lavoro Web Service il costo di realizzazione della soluzione è stato quantificato in circa 15/20 giorni. Questo perché il prodotto non viene acquistato direttamente da un vendor, ma deve essere realizzato interamente, quindi bisogna tener conto sia dei costi realizzativi, sia dei costi di manutenzione interna.
Inoltre, se l'utilizzatore ha una forte vocazione all'integrazione, questa soluzione fornisce la possibilità di integrarsi in diversi scenari, e conseguentemente impattare sui costi di manutenzione che risultano evidentemente più alti rispetto alle altre due ipotesi di lavoro discusse.
In questa soluzione il totale disaccoppiamento permette di eliminare qualsiasi tipologia di lock-in relativa al middleware, comportando l'eliminazione del lock-in sulla versione.
Vi è inoltre l'eliminazione del lock-in sul personale e sull'assistenza, infatti chi si occupa di sviluppare queste tipologie di soluzioni tipicamente fornisce assistenza e supporto gratuita.
Infine questa soluzione è utile ed efficace se si ha una situazione aziendale particolarmente strutturata, quindi con la presenza di un IT. Infatti in scenari troppo piccoli potrebbe essere inefficiente o addirittura non sviluppabile.
In scenari troppo grandi, invece, il Web Service potrebbe diventare obsoleto, presentando quindi costi di gestione e di manutenzione (manutenzione interna, documentazione, test) che possono essere gestiti in modo efficiente se si ha un IT.

5. Caso sperimentale: ontologie e modello dati

In questo quinto capitolo vi è descritta la prima parte del caso sperimentale, in particolare la parte riguardante le ontologie create ad hoc, come l'ontologia per la gestione della federazione e l'ontologia relativa all'università, e la parte relativa al modello dati per la gestione dei risultati.

Nell'implementazione del concetto di federazione si è partiti con la ricerca di un modello dati idoneo alla sperimentazione. La ricerca ha prodotto come scelta il modello dati che rappresenta le Università Italiane. Questo modello è stato interamente da noi definito ed implementato, in una versione semplificata e non sicuramente esaustiva nella rappresentazione delle Università. Questa scelta è giustificata da fatto di non avere a tutt'oggi un modello ontologico che rappresenti le Università Italiane, ed inoltre la scelta è stata di carattere puramente accademico.
Oltre a ciò, è stata sviluppata un'ulteriore ontologia utile alla gestione degli elementi facenti parte della federazione.

5.1 Ontologia Univ

Nella sperimentazione è stata creata un'ontologia, denominata *Univ*, la quale incorpora il modello dei dati da condividere su ogni endpoint facente parte della federazione.
Questa ontologia definisce formalmente, anche se non in modo esaustivo, l'ambito Universitario, della Pubblica Amministrazione e delle Organizzazioni in generale; infatti sono stati definiti tre namespace principali:

@prefix	*org:*	*http://www.sprotetto.org/org#*
@prefix	*PA:*	*http://www.sprotetto.org/PA#*
@prefix	*Univ:*	*http://www.sprotetto.org/Univ#*

oltre che ai namespace di FOAF e SKOS:

@prefix foaf: http://xmlns.com/foaf/0.1/
@prefix skos: http://www.w3.org/2004/02/skos/core#

Una visione generale e completa di tutta l'ontologia sviluppata è visibile nella seguente figura:

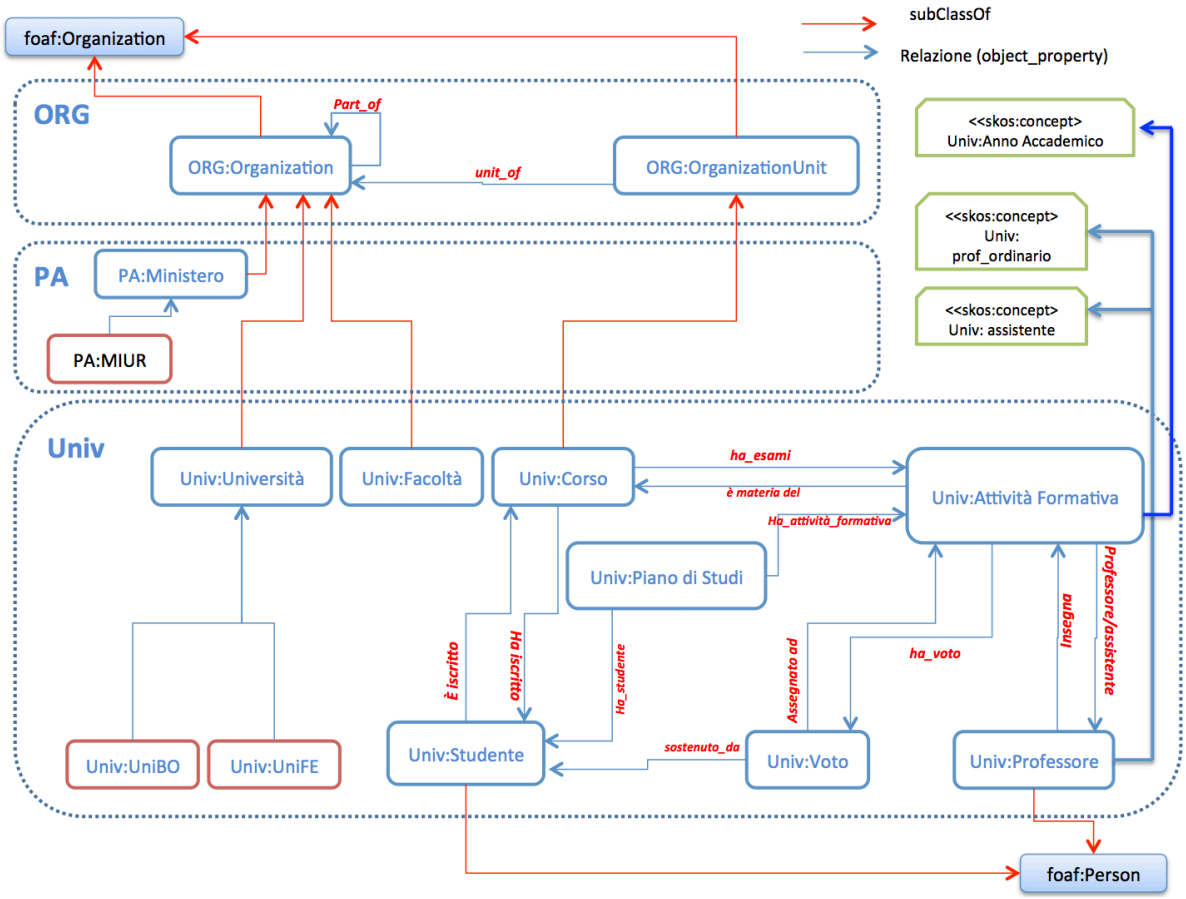

Figura 18 - Ontologia che descrive l'Università (Univ).

La prima parte dell'ontologia è stata implementata per definire nello specifico il concetto di *Organization*, troppo generale nel vocabolario foaf.
Sono state definite due sottoclassi in relazione `subClassOf`, ereditando dalla classe `foaf:Organization`, ed inoltre sono state definite rispettivamente due proprietà importanti; sulla classe `org:Organization` è stata definita la proprietà `is_part_of`, la quale si riferisce alla possibilità che una organizzazione faccia

parte di altre organizzazioni, mentre sulla classe *org:OrganizationalUnit*è stata definita la proprietà *unit_of*con la quale si esprime la relazione di partizione tra *org:OrganizationalUnit* e *org:Organization*.

La seconda parte dell'ontologia definisce il namespace *PA* (Pubblica Amministrazione), nella quale è stata definita la classe *PA:Ministero* come *subClassOf* di *org:Organization*.

L'ultimo namespace *Univ* contiene il core dell'ontologia. In particolare si è creata questa parte di ontologia per modellare i concetti che concernono l'ambito universitario, poiché ad oggi non vi è un'ontologia standard preesistente.
Sono state definite le classi Studente e Professore, le quali ereditano in modo diretto (*subClassOf*) dalla classe *foaf:Person*.
In relazione a queste due classi, sia per Univ:Studente che per Univ:Professore sono state utilizzate le proprietà *foaf:firstname* e *foaf:surname* ereditate dalla classe *foaf:Person*.
Alla classe *Univ:Professore* è stata aggiunta una proprietà *Univ:ruolo*, tramite la quale avviene il collegamento tra l'istanza del professore, dipendentemente dal ruolo di quest'ultimo, e l'istanza del concetto *skos:prof_ordinario*o *skos:assistente*.
La classe *Univ:Professore* è in relazione con la classe *Univ:Attività_Formativa* tramite la proprietà *Univ:Insegna* con la quale si dichiara che un professore insegna una determinata attività formativa, ed inversamente tramite le proprietà *Univ:professore* e *Univ:assistente*, si assegna all'attività formativa il professore e l'assistente.
Alla classe *Univ:Attività_Formativa* è stato associato lo skos concept *skos:Anno_accademico*, il quale identifica il legame tra l'istanza dell'attività formativa e l'anno accademico in cui questa è attiva.
La classe *Univ:Attività_Formativa* è inoltre legata alle classi *Univ:Voto*, *Univ:Piano_di_Studi* e *Univ:Corso*.

Univ:Voto rappresenta i voti assegnati alle attività formative, infatti come DataTypeProprerty si ha *Votazione*, *lode* (flag false o true), *nome dell'attività formati*va e *la data di registrazione*, mentre la proprietà *Univ:assegnato_a* identifica il legame tra l'istanza di Voto e quella di Attività Formativa. Oltre ciò la classe *Univ:Voto* è legata alla classe *Univ:Studente* tramite la relazione *Univ:sostenuto_da,* questa identifica la relazione tra una determinata istanza di voto di un esame e lo studente che ha ottenuto quella votazione.
La classe *Univ:Piano_di_studi* identifica il piano di studi di un determinato studente. È in relazione diretta con *Univ:Attività Formativa* e *Univ:Studente*tramite le relazioni *Univ:ha_attività_formativa* e *Univ:ha_studente*, che identificano rispettivamente l'insieme delle attività formative che compongono un piano di studi e il piano di studi di un determinato studente.

Anche la classe *Univ:Corso*, definita come sottoclasse di *org:OrganizzationalUnit*, è in relazione sia con *Univ:Studente* che con *Univ:Attività Formativa*, infatti con *Univ:Studente* la relazione identifica il fatto che uno Studente sia iscritto ad un Corso di Laurea e che lo stesso abbia degli studenti iscritti, mentre la relazione con *Univ:Attività Formativa* identifica il fatto che un Corso di Laurea abbia un insieme di attività formativa, mentre un'attività formativa è materia di un determinato corso di Laurea.
Infine sono state definite altre due classi importanti, quali *Univ:Università* e *Univ:Facoltà*, sono in relazione *subClassOf* con *org:Organizzation*.
Non sono state implementare relazione dirette tra queste due classi e con altre classi perché l'inferenza può essere gestita utilizzando le relazione *org:unit_of* e *org:part:of*.

5.2 Ontologia Federazione

Il servizio di federazione prevede una configurazione centralizzata, in cui un endpoint gestore contiene al suo interno una ontologia che descrive sia la federazione, sia i membri che ne fanno parte.
L'ontologia sviluppata per questa gestione è l'ontologia ***Federazione***, la quale ha il seguente namespace:

@prefix fed: http://www.sprotetto.org/federazione#.

È costituita da due classi principali:

- ***endpoint***: identifica un singolo endpoint;
- ***federazione***: identifica le informazioni della federazione.

La relazione che lega le due classi è espressa dalla proprietà `fed:memeber_of`, specificando l'appartenenza di un determinato endpoint ad una federazione.
Nel nostro caso di studio, l'istanza della classe federazione che identifica la nostra federazione è `fed:Univ_ITA`.
Sono state create due istanze di endpoint, chiamate rispettivamente `fed:UniBo` e `fed:UniFe`, le quali rappresentano rispettivamente gli endpoint che espongono le ontologie relative all'università di Bologna e all'università di Ferrara.
Infine sull'endpoint `fed:UniBo` è presente anche l'ontologia che gestisce la federazione.

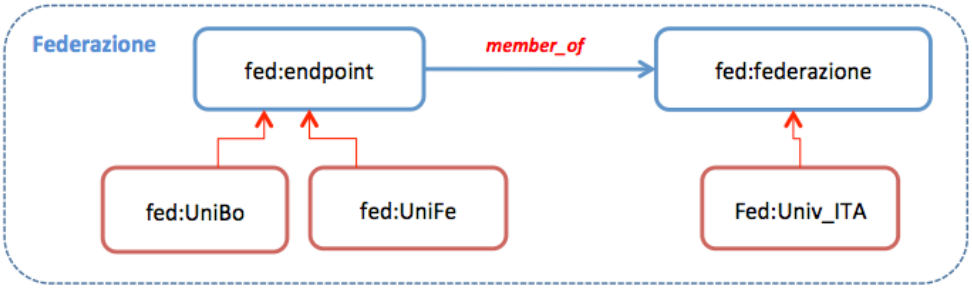

Figura 19 - Ontologia che descrive la federazione (Fed).

Questa soluzione potrebbe essere utilizzata e proposta come standard in SPARQL per la definizione di una federazione. Infatti, si potrebbero coniare dei termini specifici e definire un vocabolario (ontologia) secondo lo schema RDF per esprimere i metadati relativi agli endpoint e alle federazione, fornendo uno strumento standard e facile da implementare.

5.3 Endpoint Sparql: UniBo e UniFe

La federazione costruita per sperimentare il progetto di tesi è composta da due endpoint, ***UniBo*** e ***UniFe***, simulati su due macchine virtuali diverse con sistema operativo Linux CentOs ver. 7.0, che rappresentano rispettivamente l'Università di Bologna e l'Università di Ferrara. Questi due endpoint, per poter essere federati, condividono il modello dei dati avendo come namespace condiviso

@prefix Univ: http://www.sprotetto.org/Univ#.

Sui due endpoint sono state inserite le rispettive ontologie, avendo come istanze comuni solo quelle relative alla parte di ontologia riferita all'Organizzazione (org:http://www.sprotetto.org/org#) e quella riferita alla Pubblica Amministrazione (PA:http://www.sprotetto.org/PA#). Per la parte riguardante l'università, sono state create istanze di *studente*, *professore, piano di studi, voto, attività formative, corso di laurea e università* diverse per UniBo e per UniFe.
Infine l'endpoint UniBo è stato utilizzato come endpoint gestore dell'ontologia Federazione (*fed:http://www.sprotetto.org/federazione#*) descritta in precedenza.

5.4 Modello dei dati

Nell'individuare un modello dati per il progetto, si è partiti dal principio di avere un insieme di dati destrutturati e dal bisogno di individuare un modello che fosse il più

```
PREFIX  univ:http://www.sprotetto.org/Univ#

SELECT ?professore ?studente
{
WHERE {
      ?professore a univ:Professore .
      ?studente a univ:Studente .
} LIMIT 100
```

Figura 20 - Esempio query SPARQL su Univ.

Chiave

Valore

flessibile possibile, sia in termini di ordinamento che di numero di righe e colonne ottenibili nel risultato.

Il modello scelto si basa sulla rappresentazione di tutti i risultati ottenuti secondo una coppia ***chiave – valore***, dove la *chiave* rappresenta ciò che si vuole ottenere nella clausola SELECT, e *valore* rappresenta l'insieme dei dati relativi ad una determinata chiave, ottenendo una visione matriciale dei risultati.

Cercheremo di rendere più chiaro il tutto tramite un semplice esempio di query sparql.

Si suppone di interrogare la federazione Univ con la seguente query sparql:

Il risultato ottenuto è il seguente:

Professore	Studente
AntonioCorradi	FrancescoSprotetto
FoschiniLuca	GiuseppeRossi
TortonesiMauro	
GentiliniLuca	

Figura 21 - Tabella risultati query SPARQL su Univ (rappresentazione logica).

Come si può notare in figura, il risultato è rappresentato secondo un modello matriciale, dove ***professore*** e ***studente*** risultano essere le chiavi.

Nel progetto sono state implementate due classi per gestire questo modello:

- *Field.java* rappresenta la coppia chiave – valore;
- la classe *Record.java* implementa la visione matriciale, quindi l'insieme dei record ottenuti come risultato delle query.

La scelta di questo modello è giustificata dal fatto di poter gestire i risultati nel modo più generale possibile, in simbiosi con la modalità di esposizione dei risultati degli endpoint SPARQL, i quali rappresentano gli stessi come una lista di righe con valori per ogni chiave specificata nella clausola SELECT.
La scelta di non utilizzare come modello EDT (Enterprise Data Model) o un modello bastato su pattern DTO (Data Transfert Object), che risulta essere più corretto dal punto di vista formale, ma più vincolante nella struttura dei dati, è giustificata dal fatto di poter restare generali, sia per ciò che concerne la tipologia di query, sia per il discorso fatto sui risultati, avendo come focus principale non la modellazione dei dati, ma il Data Analysis.

6. Caso sperimentale: soluzione applicativa

In questo sesto capitolo vi è descritta la seconda parte del caso sperimentale, in particolare la soluzione applicativa. Si è partiti dalla descrizione, nel dettaglio, di tutta l'architettura, soffermandosi sugli aspetti chiave della soluzione. In seguito si è analizzata la sequenza di esecuzione delle chiamate e la gestione della logica e del risultato. Infine si è discusso dei test effettuati sulle componenti della soluzione, e sullo stack software utilizzato.

Per ciò che riguarda la soluzione architetturale la scelta è ricaduta sulla terza soluzione, per una serie di motivi molto importanti:

- Soluzione flessibile e ottima a livello architetturale;
- Soluzione fortemente modulare e non legata a prodotti software (aspetto importante per un'azienda di consulenza informatica);
- Soluzione facilmente estendibile e integrabile con altre tecnologie, dettata anche dall'uso dell'architettura SOA.
- Facilità di manutenzione, legata alla parte software sviluppata;
- Facilità nell'aggiungere o eliminare funzionalità, senza impattare sull'architettura e sui componenti.

In ***figura 22*** è illustrata la soluzione architetturale sviluppata, evidenziando il flusso delle chiamate e le varie parti che compongono il progetto.

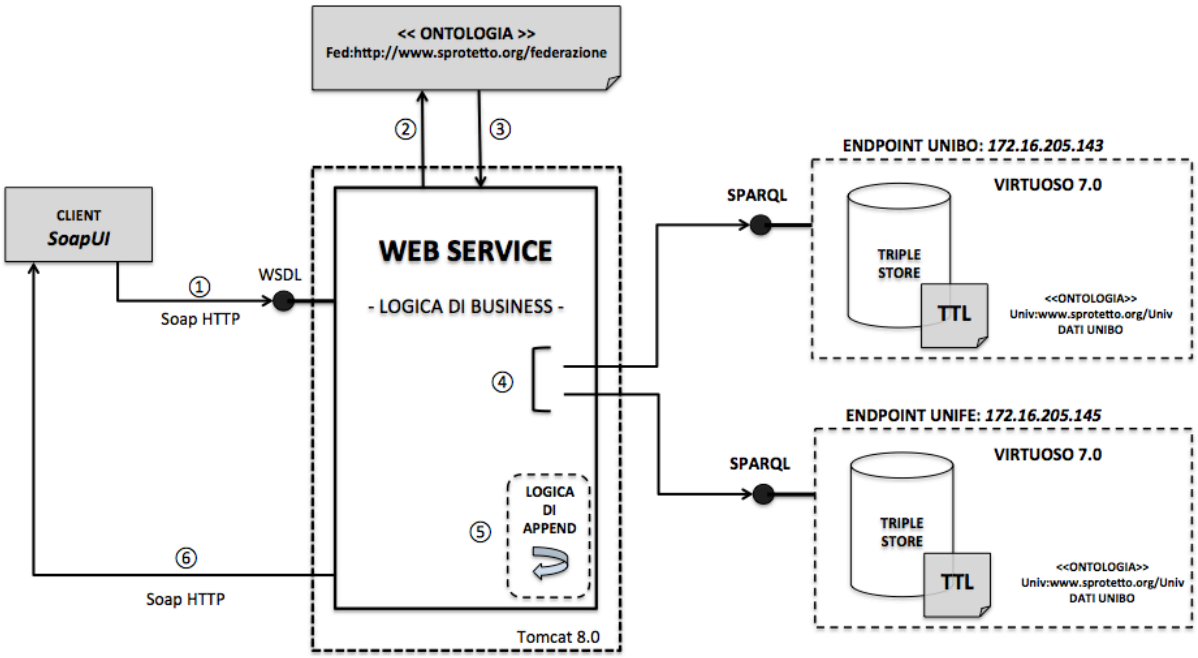

Figura 22 - Architettura soluzione applicativa.

Le componenti di questo servizio sono:

- un'ontologia, chiamata *Federazione*, utilizzata come gestore della struttura delle federazione; viene invocata dal Web Service per ottenere la lista degli endpoint;
- un endpoint che gestisce la federazione tramite un'ontologia Federazione;
- un'ontologia di riferimento, chiamata *Univ*, che rappresenta i dati accessibili sugli endpoint e che permette la condivisione del modello tra tutti gli endpoint della federazione; in figura si nota come sia UniBo che UniFe hanno lo stesso namespace (Univ:www.sprotetto.org/Univ);
- un *Web Service* per l'esposizione delle funzionalità invocabili sulla federazione, il quale:
    - effettua una chiamata SPARQL per ottenere la lista degli endpoint della federazione;
    - replica su tutti gli endpoint della lista la query in ingresso;
    - applica la logica di append ai risultati restituiti dai singoli endpoint;
    - restituisce il risultato al chiamante come risposta Soap HTTP;

- vari *endpoint* che espongono i dati accessibili dal servizio e che partecipano alla federazione (nel nostro caso sono stati utilizzati due endpoint, in particolare UniBo e UniFe, descritti in seguito);

Analizziamo ora nel dettaglio la sequenza delle chiamate necessarie per interrogare una federazione gestita tramite questo modello architetturale.

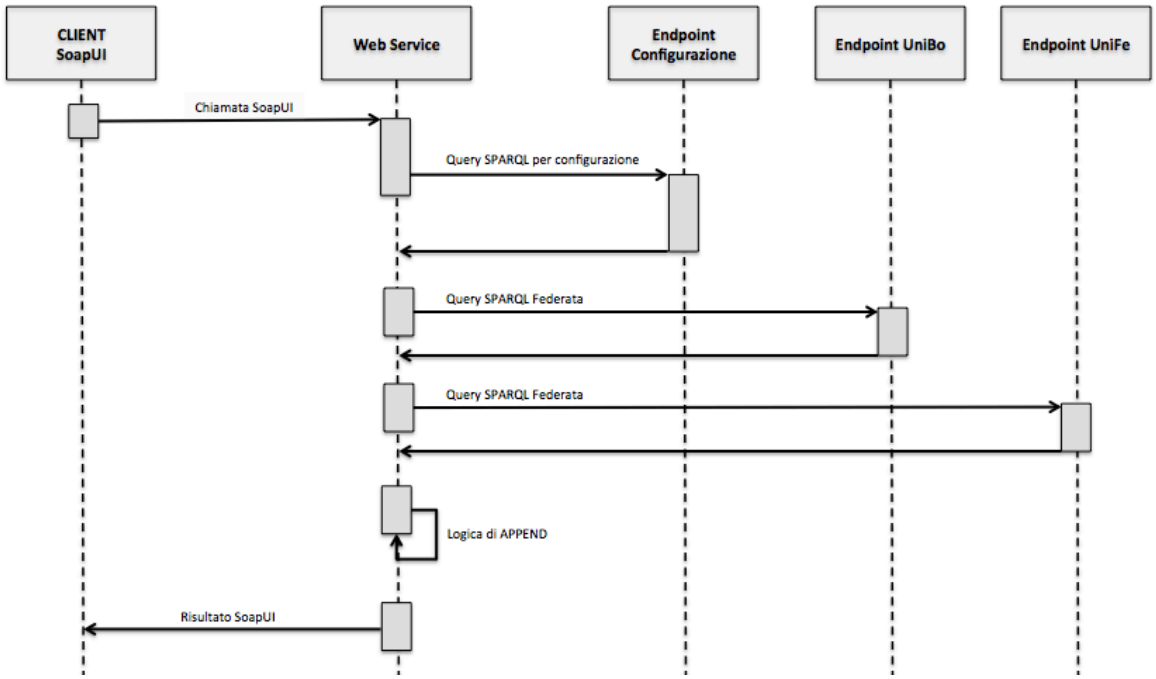

Figura 23 – Sequence Diagram soluzione applicativa.

Il client effettua una chiamata Soap specificando il nome della Federazione e la query da eseguire. Nel nostro progetto è stato utilizzato il supporto di SoapUI come client dell'applicazione, ma è possibile implementare lo stesso in qualsiasi altro modo, questo perché viene utilizzato come protocollo SOAP, standard votato all'integrazione.
Il Web Service elabora la chiamata, identificando la Federazione sulla quale operare. In seguito effettua un'interrogazione sull'endpoint che gestisce la federazione tramite una query SPARQL, la quale restituisce l'elenco, ovvero gli URL, degli endpoint che compongono la federazione.

In seguito il Web Service replica la query in ingresso su tutti gli endpoint ottenuti, in modo del tutto sequenziale.
Qui potrebbero sorgere alcuni problemi di gestione:

- gestire l'eventuale time-out degli endpoint; si potrebbe scegliere una soluzione che, in caso di time-out, escluda l'endpoint, piuttosto che replicare la query sull'endpoint soggetto a time-out dopo aver operato su tutti gli altri. È comunque una scelta che si allontanava molto dal focus del nostro lavoro;
- Gestire l'eventuale caduta dell'endpoint; si potrebbe scegliere di escludere subito l'endpoint, piuttosto che attendere che lo stesso sia nuovamente online e replicare la query, con ovvi e gravi impatti sulle performance. Anche in questo caso il focus è lontano dall'obiettivo della tesi;
- Gestire l'eventuale restituzione di triple "errate"; si potrebbe replicare nuovamente la query o eliminare il risultato.

Dopo aver replicato la query su tutti gli endpoint, il Web Service attende il risultato da ogni singolo endpoint e fonde i risultati applicando una logica di APPEND.
Anche in questo caso vi possono essere problemi di gestione, come ad esempio la logica da applicare al risultato. Infatti si poteva applicare la logica dell'intersezione, come succede con la clausola SERVICE, effettuare una UNION dei risultati gestendo i duplicati, oppure come nel nostro caso fare l'APPEND senza gestire i duplicati. In ogni caso questa gestione non risulta vicina al focus della nostra tesi, quindi poteva essere utilizzata qualsiasi soluzione.
A questo punto il Web Service spedisce il risultato tramite pacchetti Soap al chiamante.

6.1 Servizio WS SparqlFed

Il progetto è implementato tramite un Web Service seguendo l'approccio Java First per privilegiare il tempo di realizzazione.
Il core della logica di Business è stato implementato tramite il servizio ***sparqlFed***.
Questo servizio implementa un'unica funzione chiamata sparqlFed, la quale prende come parametri di ingresso *l'identificativo della federazione* sulla quale eseguire

l'interrogazione e la *query generale*. Il sistema esegue una prima interrogazione all'endpoint gestore, ottenendo l'insieme degli endpoint della federazione specificata, successivamente replica la query su ognuno di questi endpoint, restituendo come risultato finale l'unione dei singoli risultati ottenuti da ogni endpoint.
Il Web Service è stato realizzato come progetto maven.

6.1.1 Progetto Maven

Nello sviluppo del Web Service si è utilizzato Apache Maven, nella sua versione 3.2.3, come software di project management basato sul project object model (POM).
Il progetto è stato diviso individuando il modello dei dati, la Business Logic e la Presentation, quest'ultima tralasciata in quanto non centrale nel progetto, e inoltre sono stati implementati i test unitari per il servizio.

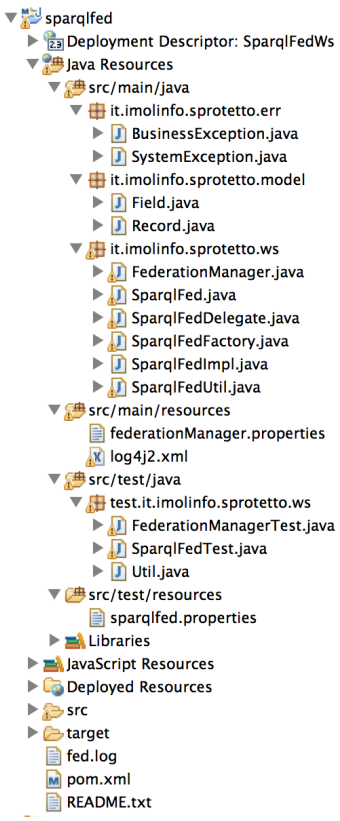

Figura 24 - Progetto Maven.

6.1.2 Business Logic

La Business Logic rappresenta il core della soluzione applicativa sviluppata.
La classe che implementa il servizio di federazione è ***SparqlFedImpl,*** la quale:

- ha come parametri di ingresso una ***String fed*** che rappresenta il nome della federazione, e una ***String query*** che rappresenta la query da eseguire sulla federazione;
- analizza la correttezza dei parametri in ingresso;
- interroga il gestore della federazione per ottenere l'elenco degli endpoint, ovvero i loro URL, utilizzando il Federation Manager (classe FederationManager.java);
- itera sugli endpoint per eseguire la query in ingresso;
- ottiene il Resultset, lo formatta secondo il modello dati scelto, e restituisce una lista di record in uscita, i quali rappresentano l'APPEND dei risultati di ogni endpoint.

La classe ***FederationManager*** si occupa di ottenere, dato in ingresso il nome della federazione, la lista di tutti gli endpoint che fanno parte della federazione stessa.
Altri aspetti importanti nella progettazione del servizio di federazione sono stati l'utilizzo della Delegate e della Factory.
SparqlFedDelegate è la classe che utilizza la factory per la creazione dell'istanza del servizio e che nasconde all'utilizzatore come questo sia implementato. Questa tipologia di progettazione consente in una possibile modifica futura del servizio e della sua implementazione senza impattare sull'utilizzatore.
SparqlFedFactory è la classe che implementa il classico pattern Factory, tramite il quale l'istanza della classe che implementa il servizio viene creata. Questa specifica è stata resa configurabile tramite un file di properties esterno preventivamente caricato, nel nostro caso *sparqlfed.properties*.

6.1.3 Gestione degli Errori

Nel progetto sono stati gestiti due tipologie di errori: gli errori di sistema e gli errori relativi alla logica di Business.
Il Package *it.imolinfo.sprotetto.err* contiene le classi tramite le quali sono stati gestiti queste due tipologie di errore; precisamente la classe SystemException gestisce errori relativi al sistema, mentre la classe BusinessException gestisce errori relativi alle funzionalità di business sviluppate nel servizio.

6.2 Test: JUnit e SoapUI

Nello sviluppo dell'applicazione sono stati effettuati due tipologie di test unitari:

- test relativi alla logica di business dell'applicazione tramite l'utilizzo di JUnit;
- test relativi all'applicazione WS tramite l'utilizzo di SoapUI.

6.2.1 Test unitari sulla logica di Business

I test effettuati utilizzando JUnit hanno avuto lo scopo di testare lo logica di business implementata. Questi test, in generale, sono implementati in quanto, in fase di progettazione e sviluppo software, si potrebbe dover sviluppare solo la logica di business o parte di essa, quindi fare dei test specifici permette al progettista di avere del codice corretto e che produca i risultati attesi.
Nella figura sottostante è rappresentata la struttura dell'applicazione.

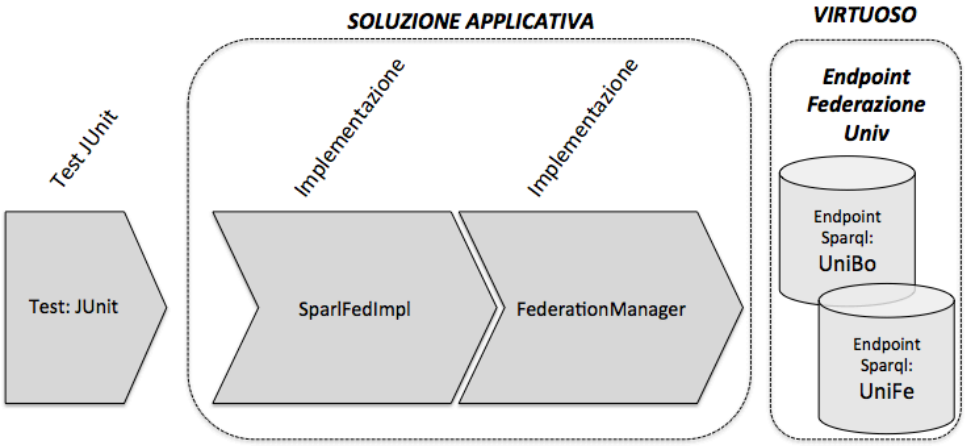

Figura 25 - Diagramma test unitari logica di Business.

Vediamo un'analisi dettagliata di tutti i test che sono stati implementati.

Sulla classe *FederationManager* sono stati effettuati 2 test:

- *testFedNull*: test sul parametro che specifica la federazione; questo parametro non deve risultare nullo, altrimenti il test fallisce.
- *testFindUnivITA*: test sulla funzionalità di ricerca degli endpoint sulla federazione specificata (nel nostro caso UnivITA); questo test fallisce se non vengono trovati endpoint nella federazione.

Sulla classe *SparqlFedImpl* sono stati effettuati i test:

- *testFedNull*: test sul parametro che specifica la federazione; questo parametro non deve risultare nullo, altrimenti il test fallisce.
- *testFedQueryNull*: test sul parametro che specifica la query da effettuare; questo parametro non deve risultare nullo, altrimenti il test fallisce.
- *testFedQueryVuota*: test sul parametro che specifica la query da effettuare; questo parametro non deve risultare vuoto, altrimenti il test fallisce.
- *testQueryUniBo*: test che interroga la federazione UnivITA con lo scopo di ottenere come risultato l'elenco dei professori ordinari che sono istanze dell'endpoint UniBo. Il test fallisce se non ci sono risultati o se il numero dei risultati non è conforme alla assert dichiarata.

- *testQueryUniFe*: test che interroga la federazione UnivITA con lo scopo di ottenere come risultato l'elenco dei professori ordinari che sono istanze dell'endpoint UniFe. Il test fallisce se non ci sono risultati o se il numero dei risultati non è conforme alla assert dichiarata.
- *testQueryUniv:* test che interroga la federazione UnivITA con lo scopo di ottenere come risultato l'elenco dei professori ordinari sia dell'endpoint UniBo che dell'endpoint UniFe. Il test fallisce se non ci sono risultati o se il numero dei risultati non è conforme alla assert dichiarata.

6.2.2 Test unitari sull'applicazione WebService

I test sul servizio Web Service SparlFed sono stati effettuati sia per testare il protocollo, in questo caso il protocollo SOAP, sia per la logica del servizio.
Lo strumento utilizzato per la creazione dei test è SoapUI nella sua versione 4.1.

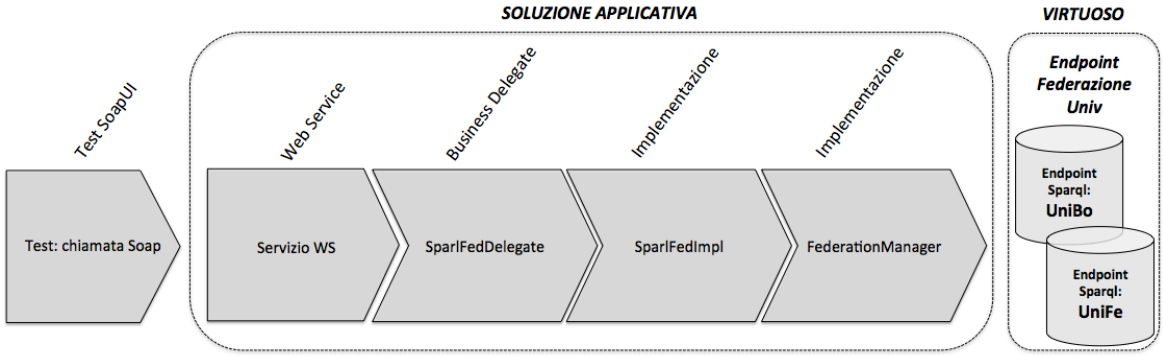

Figura 26 - Diagramma test unitari applicazione Web Service.

Nello specifico sono stati creati tre test:

- Test "*elenco professori ordinari UniBo*": lo scopo di questo test è quello di interrogare la federazione, ottenendo come risultato la sola lista dei professori ordinari dell'università di Bologna;

```
PREFIX  univ:http://www.sprotetto.org/Univ#
PREFIX  org:http://www.sprotetto.org/org#

SELECT ?nome
{
WHERE {
      ?orgorg:is_part_ofuniv:UniBo .
      ?corsoorg:unit_of ?org .
      ?insegna univ:è_materia_del ?corso .
      ?nome univ:insegna ?insegna .
      ?nome a univ:Professore .
      ?nome univ:ruoliuniv:prof_ordinario .
} limit 100
```

```
PREFIX  univ:http://www.sprotetto.org/Univ#
PREFIX  org:http://www.sprotetto.org/org#

SELECT ?nome
{
WHERE {
      ?orgorg:is_part_ofuniv:UniFe .
      ?corsoorg:unit_of ?org .
      ?insegna univ:è_materia_del ?corso .
      ?nome univ:insegna ?insegna .
      ?nome a univ:Professore .
      ?nome univ:ruoliuniv:prof_ordinario .
} limit 100
```

Figura 27 - Query "elenco professori ordinari UniBo".
Figura 28 - Query "elenco professori ordinari UniFe".

- Test "*elenco professori ordinari UniFe*": lo scopo di questo test è quello di interrogare la federazione, ottenendo come risultato la sola lista dei professori ordinari dell'università di Ferrara;

- Test "*elenco professori ordinari UniBo e UniFe*": lo scopo di questo test è quello di interrogare la federazione, ottenendo come risultato la sola lista dei professori ordinari dell'università di entrambe le università;

```
PREFIX  univ:http://www.sprotetto.org/Univ#

SELECT ?nome
{

WHERE {

      ?nome a univ:Professore .
      ?nome univ:ruolouniv:prof_ordinario .

} limit 100
```

Figura 29 - Query "elenco professori ordinari UniBo e UniFe".

6.3 Stack Software

Il servizio di Federazione è stato realizzato utilizzando i seguenti strumenti software:

- Macchine virtuali Linux create su VMware.
- Openlink Virtuoso per l'implementazione e la gestione degli endpoint;
- SoapUI per simulare il client e per utilizzare il protocollo SOAP;
- Java, J2EE per il core del Web Service;
- Tomcat come container in cui installare il Web Service;
- API di Apache CXF per il Web Service;
- API Jena per la gestione delle query Sparql;
- API Log4j per il logging;
- APIJUnit per la parte di test.

Nella figura seguente vie è una rappresentazione schematica dello stack software appena elencato.

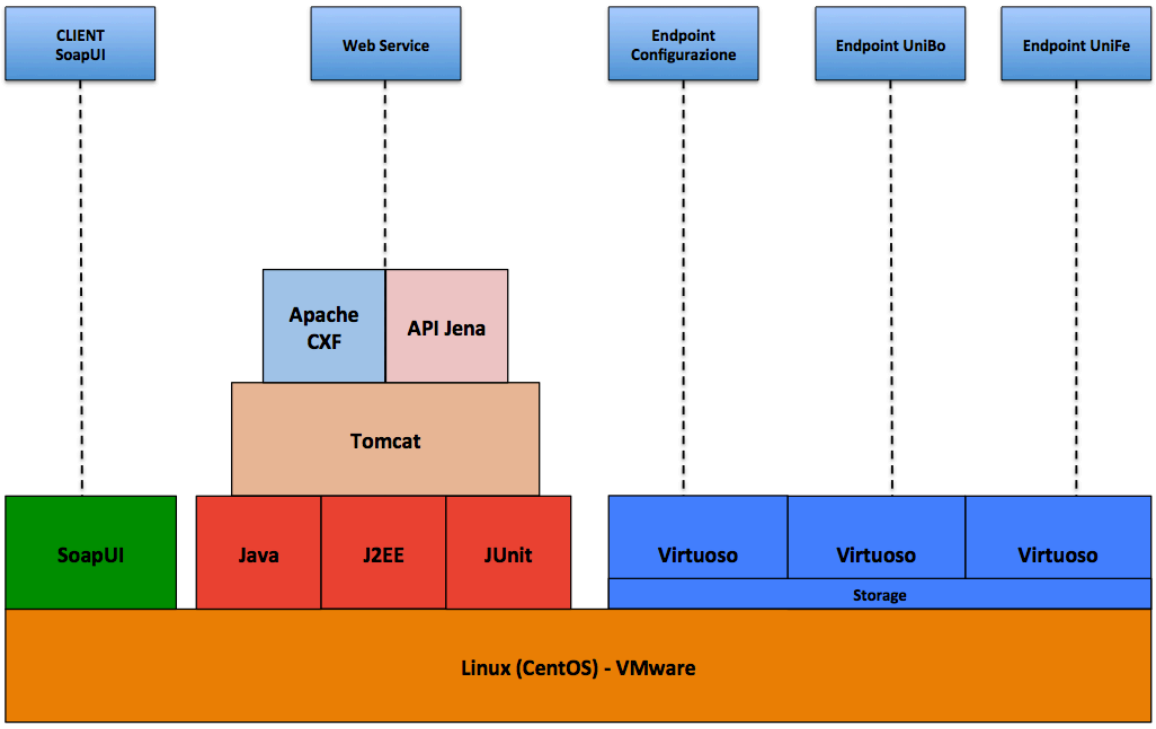

Figura 30 - Stack Software soluzione applicativa.

7. Enterprise Architecture e Data Analisys. Casi d'uso di scrittura dati

In questo settimo capitolo si è analizzata nel dettaglio l'analisi dei dati nell'ambito dell'Enterprise Architecture. Si è discusso in particolare di quali sono gli attuali procedimenti disponibili per il Data Analisys, come il processo ETL, cercando di evidenziare quali sono i benefici che la federazione apporta a questo tipo di disciplina.

Nella disciplina dell'Enterprise Architecture risulta molto importante l'aspetto che riguarda l'analisi dei dati aziendali.
I manager che prendono decisioni importanti hanno bisogno di informazioni sulle quali basare le proprie scelte; queste informazioni risultano spesso eterogenee e per di più provenienti da fonti diverse, riguardando una molteplicità di entità di business.
Oltretutto queste informazioni tipicamente dovrebbero essere fornite in modo veloce, dovrebbero essere attendibili e non ambigue.
Nell'ambito del Data Analisys si effettuano dei procedimenti laboriosi di elaborazione di queste informazioni al fine di fornire report completi e utili alla funzione del decision-making.

Questi report, tipicamente tabelle, grafici o lettere, file di diverso genere, vengono fornite dopo un procedimento composto da:

- estrazione dei dati da fonti eterogenee (interne ed esterne all'ambito aziendale);
- processamento dei dati;
- caricamento dei risultati su store aziendali;
- fornitura dei risultati.

Il processo estrazione, processamento e caricamento dei dati è chiamato ETL (Extraxt, Transform and Load) [ALE12].

7.1 Procedimento ETL

L'acronimo ***ETL (Extract, Transform and Load)*** è utilizzato per descrivere i processi con cui i dati sono estratti dai sistemi sorgente, sono trasformati attraverso operazioni di pulizia, unificazione, formattazione e in fine sono caricati all'interno del data warehouse [ALE12].

Le tre fasi sono sintetizzabili:

- ***Extraxt:*** fase di estrazione dei dati "grezzi" (non strutturati) dalle fonti;
- ***Trasfrom:*** trasformazione che consiste nell'integrazione dei dati provenienti da fonti divere ed eterogenee, nell'applicazione delle regole derivanti dai requisisti di business e, nel controllo e applicazione delle regole di Data Quality;
- ***Load:*** fase di caricamento dei dati puliti e integrati all'interno del data warehouse [ALE12];

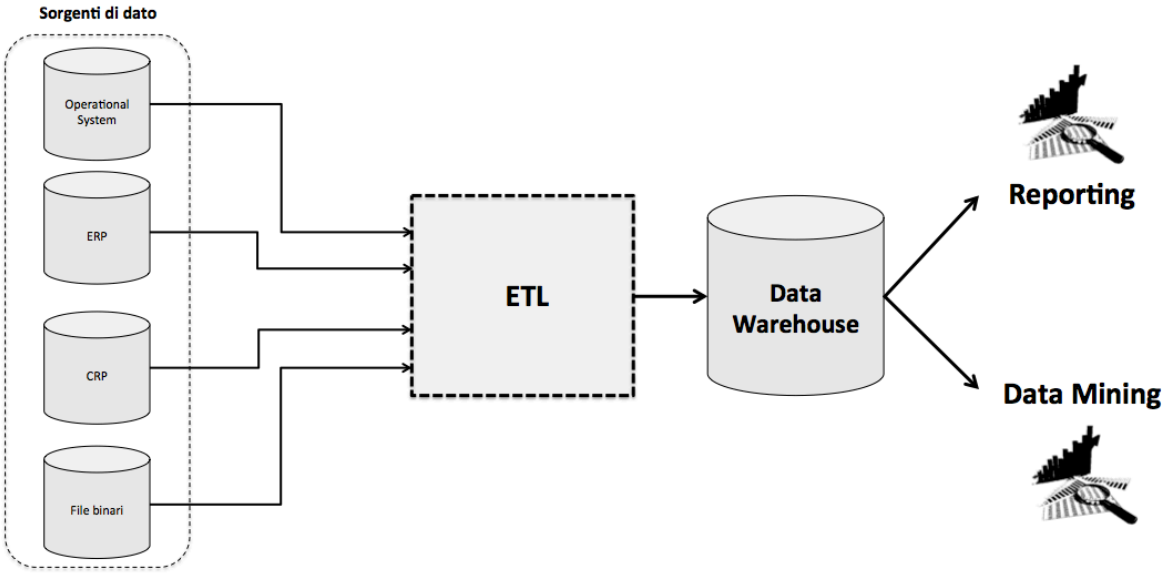

Figura 31 - Processo ETL.

7.1.1 Fase di estrazione

La fase di estrazione consiste nell'acquisizione dei dati dalle fonti, che possono essere numerose e tra loto eterogenee: possono infatti basarsi su tecnologie diverse e presentarsi si come fonti dato relazionali, sia non relazionali.
Per la fase di estrazione è quindi importante avere gli strumenti necessari a connettersi a ciascuna fonte dato differente.
La fase di estrazione deve tener conto delle caratteristiche della base di dati sorgente e dei singoli campi da estrarre, già esaminate attraverso l'operazione *di analisi delle fonti dati*.
Uno dei fattori più importanti da considerare è il volume di dati da estrarre al primo caricamento e nei caricamenti periodici successivi. In base a tale volume di dati si fornisce la possibilità di decidere quale comportamento adottare durante l'estrazione dei dati.
Se i volumi sono ridotti è accettabile estrarre sempre tutti i dati dalla fonte, e solo in seguito effettuare il filtraggio all'interno dell'ETL, per portare avanti soltanto quelli rilevanti per la procedura.
Spesso può capitare che nelle basi dati delle sorgenti vi siano volumi ragguardevoli di record che non è necessario processare; il filtraggio preventivo, ove è possibile, evita l'allungamento inutile delle fasi di recupero e di trasformazione dei dati.
Alcune operazioni, svolte all'interno della fase di trasformazione, richiedono che i dati siano ordinati secondo un chiave (ad esempio chiave di business). Anche se è possibile utilizzare la funzionalità di ordinamento inclusa in molti strumenti ETL, spesso è conveniente estrarre i dati già ordinati, sfruttando i motori messi a disposizione della fonti dati (ad esempio l'operatore Order By di SQL).
Ovviamente questo è possibile in presenza di fonti relazionale oppure di specifiche funzionalità di ordinamento in presenza di fonti non relazionali [ALE12].

7.1.2 Fase di trasformazione

La fase di trasformazione è sicuramente la più complessa giacché in essa vi è il controllo delle regole di Data Quality, implementando i requisiti di business e gestendo l'integrazione dei dati provenienti da fonti diverse.
La fase di trasformazione diventa molto complessa se le regole da controllare e applicare sono molteplici. Infatti suddividere l'attività legata alla Data Quality dal resto delle trasformazioni, consente di ottenere due processi distinti e più semplice.
La reazione di un insieme di dati ripuliti, separata dalle trasformazioni di business, ne permette l'utilizzo anche al di fuori dei processi di alimentazione del data warehouse. Infatti il data set, una volta ripulito, può essere impiegato per fornire un feedback ai sistemi sorgente, che potranno utilizzare i dati sistemati secondo le regole di qualità, al fine di eliminare le anomalie (***figura 32***).

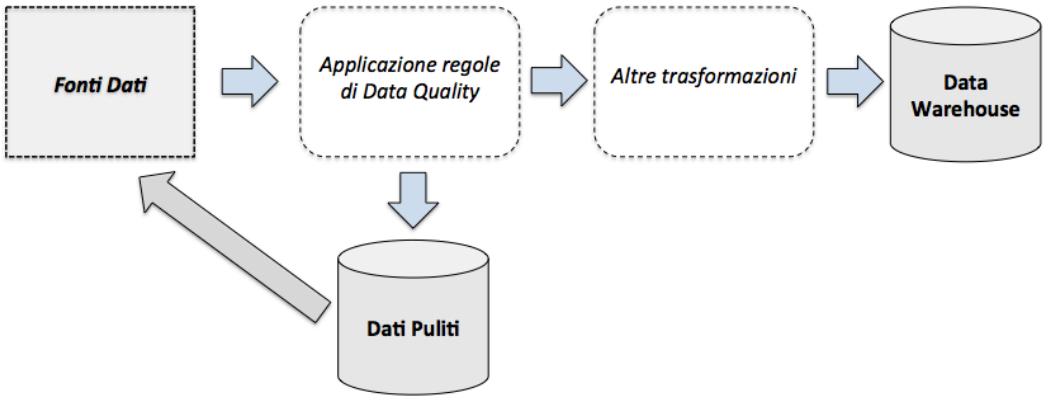

Figura 32 - ETL: processo di trasformazione dei dati.

Esiste un trade-off tra la verifica della qualità dei dati e la velocità di esecuzione dell'ETL.
La trasformazione dei dati può avvenire secondo due diversi approcci:

- Manipolazione attraverso query SQL;
- Trasformazioni gestite all'interno dell'ETL con i componenti disponibili nella piattaforma.

Nel primo caso non si utilizzano gli strumenti ETL, spostando il carico del lavoro sulla base dati.
Nel secondo caso, invece, le trasformazioni sono svolte tramite strumenti messi a disposizione dai software ETL, utilizzando la base dati solo per salvare i dati elaborati.
I fattori che entrano in gioco nella decisione circa la modalità da adottare, riguardano principalmente l'utilizzo delle risorse hardware (memoria, disco fisso, ecc,) e il carico di lavoro che è possibile attribuire all'RDBMS o al server ETL [ALE12].

7.1.3 Fase di caricamento

Una volta definita la mappatura tra i dati delle sorgenti e i campi delle tabelle di destinazione, può avvenire la fasi di caricamento (load).
Gli aspetti fondamentali, da tenere in considerazione nell'implementare il caricamento, riguardano l'integrità dei dati e i tempi di caricamento.
L'integrità dei dati deve essere garantita nell'ETL sia attraverso la corretta generazione delle chiavi primarie delle dimensioni (cioè le chiavi surrogate), sia attraverso la corretta determinazione delle foreign key delle tabelle dei fatti.
Molto spesso, infatti, per motivi legati al miglioramento delle performance, le foreign key sono omesse, oppure disabilitate in fase di caricamento, facendo venir meno un controllo molto importante sulla coerenza dei dati.
Le operazioni di caricamento avvengono solitamente in orari nei quali non c'è operatività degli utenti; non per questo, però, la performance di esecuzione è un elemento da trascurare. È possibile, infatti, che la finestra temporale disponibile per caricare i dati sia limitata da altre operazioni, quali backup, manutenzione ecc. Quindi è necessario sfruttare ogni meccanismo a disposizione per migliorare la velocità di inserimento e caricamento dei dati [ALE12].

L'ETL è certamente la componente più importante e più complessa di un sistema aziendale, e la sua complessità deriva da diversi fattori:

- La molteplicità delle fonti dato;

- I dati possono essere incompleti, errati, disomogenei e difficilmente riconciliabili;
- I dati devono essere interpretati attraverso "regole di business" propri di ciascuna azienda.

Possiamo elencare alcuni strumenti utilizzati per la realizzazione delle procedure relative all'ETL, identificabili in tre categorie:

- ***Applicazioni sviluppate ad hoc esterne agli RDBMS***: l'azienda può decidere di sviluppare un sistema di ETL utilizzando risorse interne o consulenti esterni. Si tratta però di una soluzione costosa;
- ***Procedure sviluppate ad hoc con gli strumenti inclusi nei RDBMS***. Tutti gli RDBMS offrono strumenti semplici per importare dati che possono costituire la prima fase di un sistema ETL interamente costruito tramite SQL;
- ***Soluzione applicative di ETL***: soluzioni che includono la possibilità di connettersi a fonti dati diverse (database, file di testo, XML, noSQL), offrendo funzionalità di trasformazione e spesso sono programmabili ed estendibili attraverso la scrittura di codice (Java, C#, VB, .NET). questa categoria di strumenti di ETL consente di costruire veri e propri workflow di trasformazione dei dati, che hanno le caratteristiche di essere più facilmente comprensibili e manutenibili anche da parte di chi deve prendere parte al loro sviluppo in fasi avanzate del progetto [ALE12].

In ***figura 33*** viene messo in evidenza come ognuna delle fonti dato tipicamente presenta una eterogeneità del dato e del modello del dato, una interfaccia diversa per accedervi ed infine protocolli diversi per la interrogazione e il reperimento.
Questa problematica si rispecchia in una maggiore difficoltà nel reperimento dei dati, infatti avere protocolli diversi presuppone diverse implementazioni e chiamate, avere un modello diverso presuppone struttura dei dati diversi e quindi maggior complessità nell'interrogazione.

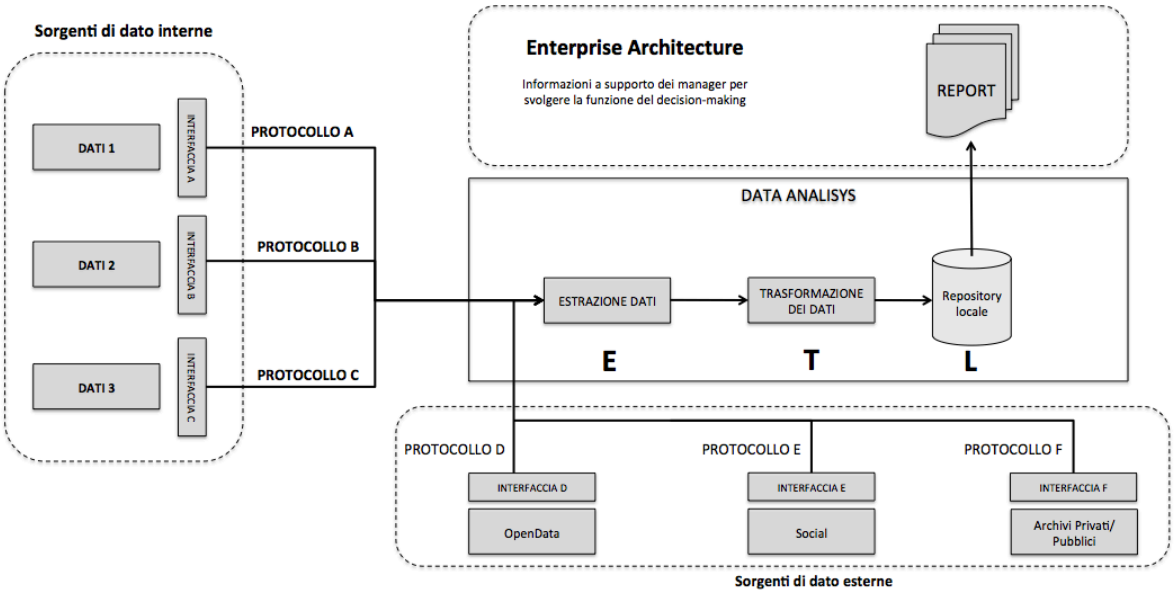

Figura 33 - Procedura ETL classica.

7.2 Procedimento Live

La possibilità di poter interrogare un insieme di dati in modo omogeneo e utilizzando protocolli standard, renderebbe più veloce e meno complesso il procedimento ETL prima descritto, soprattutto per ciò che riguarda l'Extract.

Il Semantic Web mette a disposizione tecnologie che possano supportare questa tipologia di problematica, in particolare il protocollo standard SPARQL.

Pertanto la disomogeneità dei dati può essere risolta condividendo un modello tra le fonti dati interni all'azienda, dando una visione unica e omogenea dei dati aziendali.

Da questo la possibilità di utilizzare la federazione come feature per rendere omogenei i dati interni all'azienda e risolvere, almeno in parte, il problema della complessità analizzato precedentemente. Infatti se i dati aziendali fossero federati potrebbero essere interrogati in modo più semplice, con un solo protocollo e con la possibilità di accedervi live (vedi figura). Inoltre adottare la federazione comporterebbe lo spostamento della complessità della fase di trasformazione e di normalizzazione sugli endpoint, diluendo di fatto l'effort tra tutti i partecipanti alla federazione stessa.

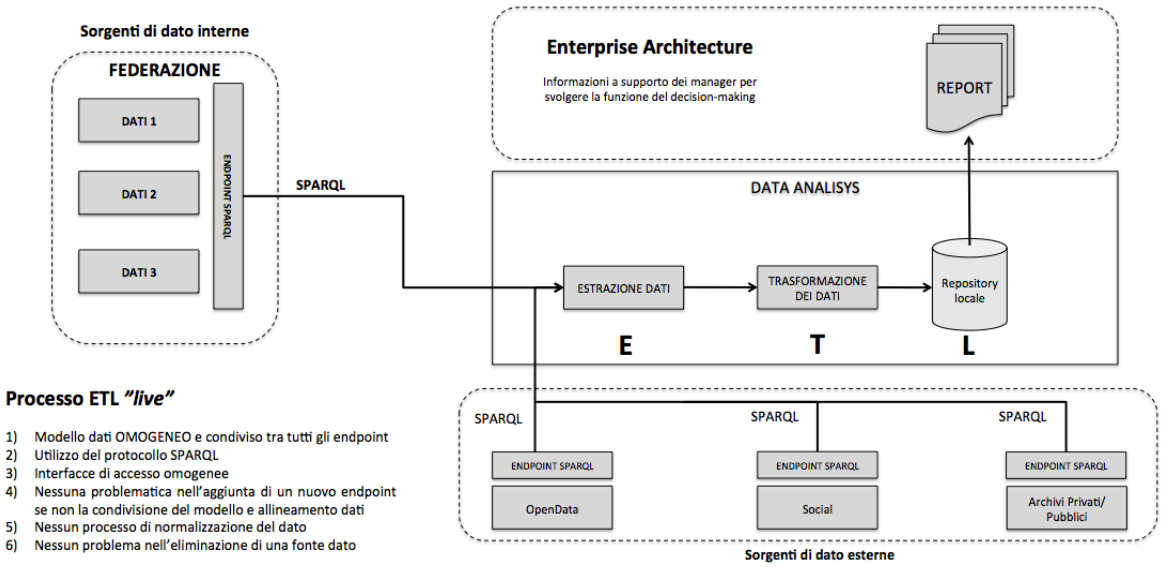

Figura 34- Procedura ETL con federazione.

L'idea è quindi di andare verso l'utilizzo in ambito Enterprise Architecture di modelli e tecnologie il più standard possibile, con l'obiettivo di avere dati strutturati in modo omogeneo, utilizzare protocolli standard come lo SPARQL al fine di sostenere il più possibile il principio d'integrazione.

Questo discorso è ulteriormente valido anche quando si parla di Big Data (rif. Appendice B), infatti si pensi alla mole di dati interni all'azienda coadiuvati da dati esterni (analisi di mercato, social, ecc), l'utilizzo di un modello federato basato sulla condivisione del modello dati potrebbe risultare un valido supporto al miglioramento, almeno in parte, delle tre V (Velocità, Varietà, Volume). Infatti velocizzerebbe il reperimento dei dati con la possibilità di azione live, permetterebbe una miglior gestione della Varietà dei dati, siano essi strutturati o meno, tramite la condivisione di un modello.

7.3 Performance

Dal punto vista delle performance, il processo che abbiamo chiamato "live" permette un netto miglioramento nella procedura di estrazione e normalizzazione dei dati, passando da una misurazione in termini di ***giorni*** ad una in termini di ***minuti***.

Questo miglioramento è dettato dall'omogeneità del modello condiviso tra gli endpoint della federazione ed inoltre dall'utilizzo di un protocollo standard come SPARQL pensato per interrogazioni via world wide e non intranet (come ad esempio SQL).
Soffermandoci su due scenari in particolare, si evince in modo immediato i notevoli benefici che questo tipo di approccio può apportare in termini di performance.

Il primo scenario riguarda il classico processo ETL, quindi nella creazione di un report in base ad determinate esigenze.
Quello che succede è che l'estrazione dei dati e la conseguente normalizzazione, importante per rendere i dati omogenei e processabili, viene effettuata e terminata in un tempo stimabile in giorni (stimati riferendoci al nostro caso specifico).
Nel caso in cui si dovesse avere bisogno di una nuova reportistica con quesiti largamente diversi da quelli posti in precedenza, il processo di estrazione e normalizzazione deve essere nuovamente compito, con ovvi costi in termini di tempo quantificabili in giorni (stimati riferendoci al nostro caso specifico).

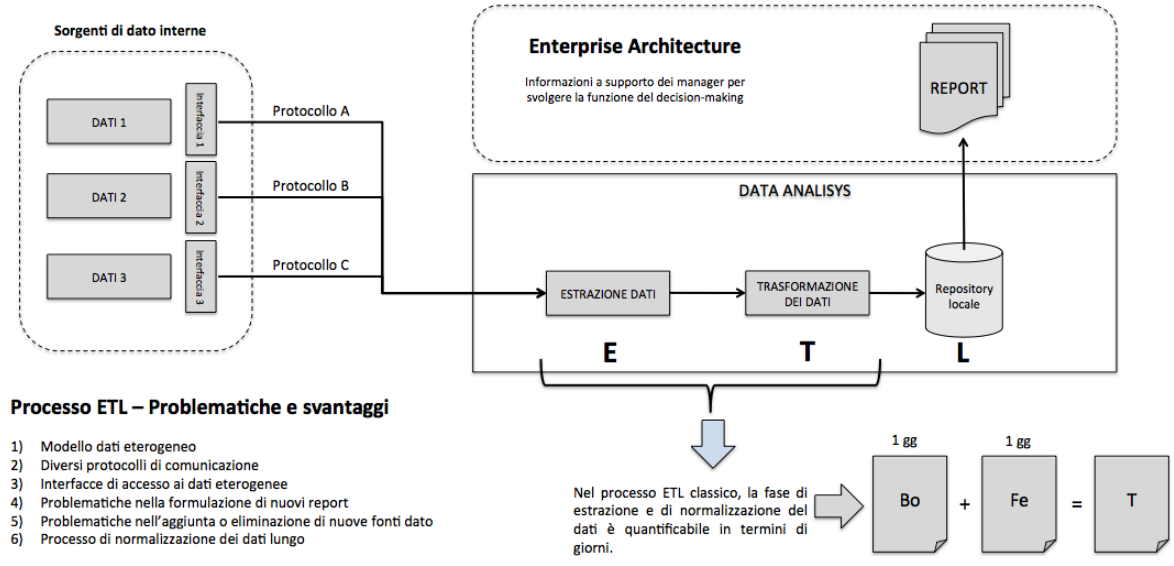

Figura 35 - Processo ETL classico: performance

Il procedimento live, che utilizza quindi la federazione delle fonti dalle quali estrarre i dati, permetterebbe accorciare questi tempi quantificando il tutto in minuti, infatti il

processo di estrazione viene fatto utilizzando interrogazioni SPQRQL, mentre il processo di normalizzazione dei dati non deve essere fatto, questo perché gli endpoint condividono il modello dei dati i quali, quindi, risultano già normalizzati.

Un secondo scenario, collegato a quanto appena discusso, è il caso in cui nelle fonti dato venissero aggiunti o eliminati degli endpoint. Questo, nel processo ETL classico, comporta un impatto forte, infatti tutti i servizi "vecchi" dovrebbero essere totalmente riavviati, rieseguendo estrazione e normalizzazione, con una conseguente regressione.
Nel caso live, invece, l'endpoint o gli endpoint che entrano a far parte della federazione, non devono far altro che esporre il proprio endpoint e condividere il modello dei dati, il quale comporta un primo costo iniziale (che nel nostro caso specifico è stimabile in circa 20gg), ma successivamente le operazioni possono essere svolte in termini di minuti.
Quindi è facile dedurre come questa tipologia di feature sia fortemente abilitante per le funzioni principali dell'EA.

Per concludere, con un modello di Enterprise Architecture utilizzato con il Semantic Web si cerca di aggredire quelli che sono i costi inerenti alla trasformazione e normalizzazione dei dati, dipendente anche dal contesto in cui si opera, dai sistemi informativi utilizzati, dalla mole di dati da elaborare. Inoltre avere un'evoluzione di un modello unico sugli endpoint permette di non avere regressione su modelli vecchi, con un conseguente unico costo inteso nell'evoluzione stessa del modello, con l'ulteriore beneficio di avere dati sempre allineati e disponibili.

7.4 Casi d'uso di scrittura dati

La funzionalità di scrittura in ambito federato concerne la capacità di poter creare, modificare o eliminare un dato nella federazione secondo il modello dei dati condiviso.

Vi sono però molti aspetti funzionali critici. Prima di tutto il problema l'ownership del dato, problema non prettamente IT, il quale deve essere affrontato caso per caso, quindi rimandiamo la gestione di questo aspetto in altra sede.
Il problema centrale nel nostro studio è capire quale sia il "pattern" migliore per la funzionalità di scrittura.
La prima soluzione emersa dalla ricerca riguarda il non implementare la funzionalità di scrittura. Questo significa che non è possibile scrivere in modo federato sugli endpoint, tuttavia ognuno di essi implementa la scrittura singola con le proprie regole, permettendo in seguito la lettura federata.
Una seconda soluzione riguarda la possibilità di scrittura in broadcast, mentre una terza soluzione riguarda la gestione della scrittura federata tramite un aggregatore centralizzato.

7.4.1 Caso d'uso: Soluzione con scrittura in broadcast

La soluzione di scrittura in broadcast è la soluzione più semplice da implementare. Prevede che un endpoint possa scrivere dei dati su se stesso e su tutti gli altri endpoint della federazione. La scrittura su se stesso può risultare anche opzionale nel caso in cui l'endpoint risulta essere il gestore delle federazione.
Nel caso della nostra federazione, l'endpoint "gestore" effettua una scrittura in broadcast sugli endpoint UniBo e UniFe, e opzionalmente può scrivere su se stesso.

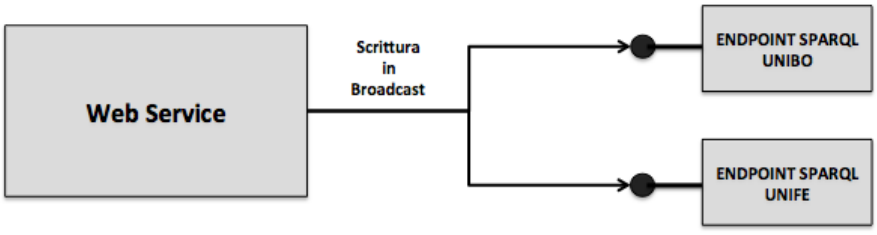

Figura 36 - Caso d'uso in scrittura: broadcast.

Questo "pattern" di soluzione presenta ovviamente dei pregi e dei difetti.

I pregi sono:

- Facilità di implementazione;
- Se viene effettuata la scrittura su di un endpoint questa viene replicata automaticamente su tutti.

I difetti di questa soluzione sono:

- I dati in scrittura sono replicati, quindi aumento la complessità in lettura dettata dall'eliminazione dei duplicati; tuttavia è possibile anche che questa complessità sia voluta;
- La gestione del nodo è demandata al nodo stesso, quindi si potrebbero presentare situazioni di disallineamento dovute a time-out o alla caduta del nodo;
- Realizzare questa soluzione nell'ambito dell'EA potrebbe non aver senso, infatti questa soluzione è più vicina al caso di Operation che al Data Analisys;

7.4.2 Caso d'uso: Soluzione con scrittura su singolo endpoint aggregatore

La soluzione con endpoint aggregatore è molto diversa da quella in broadcast. Prevede che un componente centralizzato, l'aggregatore, gestisca la funzionalità di scrittura sulla federazione. In particolare l'aggregatore analizza il dato in ingresso da scrivere, individua l'endpoint sul quale effettuare la scrittura, e successivamente invia una notifica dell'evento a tutti gli endpoint della federazione stessa.

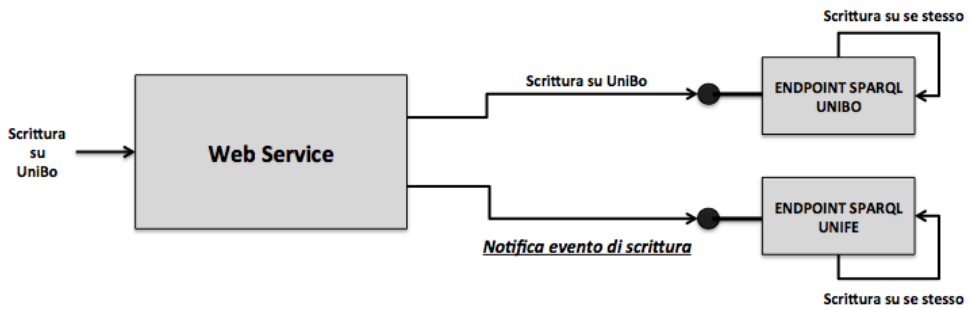

Figura 37 - Caso d'uso in scrittura: endpoint aggregatore.

Anche questa soluzione prevede dei pregi e dei difetti.

I pregi sono:

- Soluzione funzionalmente migliore, infatti l'endpoint scrive solo su se stesso o in federazione, quindi, ad esempio, potrei entrare nel merito del dato e del fornitore (es. UniFe) e scrivere solo sull'endpoint di UniFe (regola di Business Logic).

I difetti di questa soluzione invece sono:

- Ottima soluzione, ma sposta la complessità nel gestore o addirittura nell'utente;
- Se la complessità fosse spostata lato client, diventando quasi una soluzione diretta, è l'utente stesso che detta le regole di gestione allontanandosi di fatto dal nostro caso di studio;
- Difficoltà nella realizzazione della logica, poiché questa deve entrare nel merito del dato, il quale può risultare non completo. Questo comporta la difficoltà nelle decisioni da prendere per quel dato con possibili scarti e default.

8. Risultati ottenuti e proposte future

In questo ultimo capitolo sono stati messi in luce i risultati ottenuti dalla sperimentazione effettuata, evidenziando i pro e i contro della ipotesi di lavoro selezionata.
Infine sono state fatte due proposte future per poter implementare la federazione come costrutto standard del linguaggio SPARQL 1.1 e introdurre un vocabolario universale per la gestione della federazione.

Il lavoro di tesi ha avuto come focus principale la realizzazione della federazione di endpoint SPARQL, in quanto questo concetto risulta essere molto significativo nell'ambito dell'Enterprise Architecture.
È un concetto importante e interessante, soprattutto nell'ambito della ricerca, in quanto non ancora pienamente sviluppato in ambito Enterprise.
L'Enterprise Architecture risulta essere una disciplina molto giovane e in pieno sviluppo, tanto che i dipartimenti IT interni alle aziende cercano di sperimentare il suo valore, sviluppando processi e strumenti atti a supportare e realizzare la sua funzione.
Particolarmente significativi sono i dipartimenti IT delle società bancarie e assicurative italiane, le quali fungono da pionieri e propagatori per questa disciplina. Questo aspetto è legato sia alle imposizioni legislative che le indirizzano verso questa utilizzazione, sia dalle fusioni e dalle aggregazioni aziendali, tema forte e centrale nel mercato finanziario.
In questi scenari è centrale e fondamentale capire quali sono le scelte e le decisioni migliori allo scopo di rendere minimi gli sprechi, valorizzare le aree o i servizi che risultano essere più redditizi.
Questi aspetti sono molto favorevoli per introdurre una disciplina come l'Enterprise Architecture, il cui obiettivo principale è proprio quello di supportare i processi decisionali.

Prima dell'avvento della disciplina dell'Enterprise Architecture, il ruolo decisionale era delegato tipicamente ad uffici specializzati nell'analisi del dato e nella conseguente produzione di reportistica sui singoli e significativi aspetti aziendali. In seguito vi erano dei manager che si occupavano di analizzare questi report al fine di prendere le decisioni core per l'azienda.

Al giorno d'oggi la situazione è cambiata, diventando molto più complessa che in passato. Questo aspetto è dettato dal volume di dati cospicuo che un'azienda produce, soprattutto per la loro eterogeneità e la conseguente difficoltà nell'elaborazione in tempi ragionevoli.

Quindi i dipartimenti IT cercano di adattarsi alla nuova situazione, fornendo nuove metodologie e strumenti da affiancare a quelli già esistenti. Proprio in questo contesto si inseriscono due discipline importanti come l'Enterprise Architecture e il Semantic Web, le quali apportano rispettivamente le metodologie e gli strumenti appena citati.

L'Enterprise Architecture si concentra fortemente sulla proiezione dei dati, ovvero cerca di studiare e capire la situazione aziendale attuale cercando di fare delle ipotesi su quali e quanti possano essere i percorsi evolutivi possibili, e quali di questi presentano le caratteristiche più sostenibili a livello aziendale.

Per l'IT la sostenibilità non è solo di tipo economico, infatti l'IT si focalizza anche sulla sostenibilità tecnica e tecnologica, essendo un settore in continua evoluzione e difficilmente hardware e software restano sul mercato per lunghi periodi.

Allo stesso tempo il Semantic Web è in grado di arricchire i dati estratti dalle singoli fonti dati con dei dati descrittivi (chiamati metadati), che aiutano ad individuare le relazioni esistenti facendo emergere nuove informazioni.

Queste relazioni, inoltre, possono essere rappresentate sia utilizzando i classici report, sia tramite grafi, ovvero delle reti, abilitando di fatto l'introduzione di nuove discipline, come la Social Network Analysis e altre discipline legare alla Intelligenza Artificiale, come reti neurali, machine learning, mapreducing, ecc.

L'obiettivo di questa tesi è superare oltremodo questi aspetti, individuando un archetipo classico di aggregazione: la federazione.

8.1 Federazione e Enterprise Architecture

Le aziende, o i loro uffici, svolgono tipicamente la loro missione in modo autonomo.

Molte volte, però, succede che si creano delle collaborazioni (es. Joint Venture) al fine di raggiungere uno scopo comune.

Questa collaborazione permette di considerare queste aziende come una federazione.

Un aspetto importante in questo contesto è cercare di velocizzare l'analisi dei dati, quindi l'individuazione e la definizione di un modello comune è un modo molto efficiente per farlo.

A livello IT ciò è esprimibile attraverso le ontologie e il Semantic Web, questo perché sono tecnologie sicuramente adatte a descrivere dati anche destrutturati, come possono essere i contenuti RAW, o i contenuti audio/video, o contenuti parziali.

Il modello dei dati comune deve, inoltre, essere recepito dai singoli membri della federazione e deve essere reso accessibile, per consentirne l'utilizzo da parte degli Enterprise Architect.

Esistono oramai molti middleware che mettono a disposizione la possibilità di interrogare ontologie tramite un linguaggio standard, come lo SPARQL, e la tesi si è concentrata nell'analisi di tale linguaggio per capire se questo supportasse nativamente l'archetipo della federazione o se era comunque possibile realizzarlo.

Dopo un attento studio si è giunti alla conclusione che di fatto quello che in SPARQL è definito come **«federated query»,** implementato tramite la clausola **«service»,** risulta essere un costrutto che permette di interrogare delle fonti dato distribuite.

A partire da questo costrutto è possibile implementare una federazione, la quale non risulta nativamente supportata dal linguaggio, ovvero gli scrittori della query devono conoscere esattamente sia la lista dei membri della federazione, sia i singoli modelli da essi esposti.

Com'è evidente questa situazione è in parte limitante, in quanto sposta molto del tempo di progettazione su aspetti poco rilevanti come la riscrittura della query per adattarsi alla lista delle fonti da interrogare e al loro modello dati.

Questo scenario è decisamente non adatto ad ambienti in cui agisce la disciplina dell'Enterprise Architecture, la quale si propone di ridurre i tempi di analisi dei dati in favore dell'analisi dei possibili scenari a tendere.
Nella tesi sono state quindi indagate le possibili azioni correttive, vale a dire uniformare il modello dati delle singole fonti e creare una ontologia che descriva la federazione stessa, per ridurre i tempi di progettazione delle query.
Questa pratica è esattamente il lavoro di un Enterprise Architect ed interessante notare come questa si sia sviluppata:

- Ipotizziamo che esista un archetipo che leghi alcuni elementi della nostra organizzazione: ***la federazione***;
- Quindi la decisione di descriverlo, arricchendo i dati della nostra organizzazione con dei meta dati: ***il modello che descrive la federazione***;
- Cerchiamo inoltre di normalizzare l'input per l'analisi chiedendo alle singole fonti dato di esporre, tramite Endpoint SPARQL, un ***modello dati condiviso***, ma non imponiamo ad esse di sviluppare modelli dati complementari per i loro specifici scopi;
- A parte lo standard del linguaggio e il modello condiviso, non esistono altre prescrizioni tecniche o tecnologiche, per cui questo scenario è anche facilmente adattabile nel tempo in base alle soluzioni che si presenteranno sul mercato (l'unico punto di reale fallimento è la scelta dello standard);
- Realizziamo la logica di aggregazione dei dati in un unico punto: un Web Service che implementa la ***«union»*** delle singole fonti dato, facilitando la realizzazione di altre logiche da applicare ai dati raccolti in base alle nuove discipline.

Dato che in questo scenario il ritmo di aggiornamento dei dati sulle singole fonti e il tempo di recupero del dato stesso risultano essere decisamente determinanti, l'adozione di un modello dati in comune semplifica di molto la situazione, in quanto non è più il singolo ufficio di Data Analysis a normalizzare l'insieme dei dati, ma sono le singole fonti dato che si divido il lavoro, garantendo dei tempi di esecuzione

decisamente inferiori proprio perché si occupano solo dei dati da loro prodotti e non di tutto l'insieme.
Questi processi di decentramento delle attività sono un trend molto interessante che pongono anche dei temi di carattere organizzativo, che ovviamente non sono oggetto della tesi, ma che potrebbero essere anch'essi supportati da Enterprise Architecture e tecnologie legate al Semantic Web.

8.2 Possibili evoluzioni per SPARQL

In ottica di ricerca IT si è anche ipotizzato l'evoluzione del linguaggio SPARQL con l'introduzione di un nuovo costrutto: ***il foreach***.
Un tipo di costrutto presente in diversi linguaggi di programmazione, che consentirebbe l'implementazione della nostra logica di federazione usando solo i middleware per il Semantic Web.
È interessante notare che anche l'introduzione delle ***«federated query»*** è stata fatta su richiesta della community e non perché progettata direttamente dal gruppo di lavoro ufficiale.
Questa situazione potrebbe portarci ad ipotizzare che esiste negli utilizzatori finali di SPARQL una certa spinta all'aggregazione di diversi endpoint, ma che questa spinta non sia ancora stata recepita e indirizzata sia in termini di metodo che in termini tecnologici.
Sarebbe quindi interessante porre la questione alla comunità scientifica che si occupa di Semantic Web per verificare l'attendibilità di questa ipotesi e quindi decidere se investire nello sviluppo sia del linguaggio che dei middleware ad esso associato realizzando costrutti più potenti proprio in termini di aggregazione di informazioni.

8.2.1 Il costrutto foreach

Il lavoro di tesi ha portato all'analisi e alla definizione di un possibile costrutto per implementare la federazione secondo un accordo definito nel modello, da

```
foreach (List literal)
     QUERY SPARQL GENERALE
Filter By (condizione)
```

Figura 38 - Costrutto foreach

proporre come aggiunta allo standard Sparql 1.1 del W3C, dove attualmente la federazione è implementata tramite la clausola SERVICE, con le possibilità descritte ampiamente in precedenza.

Un possibile realizzazione del costrutto è la seguente:

Analizzando nel dettaglio il costrutto, questo prevede di specificare:

- *List Literal*: in ingresso il costrutto prevede o una classica lista di literal, oppure una query generale che abbia come risultato una lista di literal;
- *query sparql generale*: definizione della query generale da eseguire per ogni literal;
- *condizione*: permette la definizione di filtri generali o specifici da applicare all'esecuzione della query.

Questa soluzione permette di effettuare una query SPARQL del tutto generale, intendendo con questo la possibilità di utilizzare tutti i costrutti attualmente disponibili nello standard, permettendo anche l'utilizzo della stessa clausola SERVICE.

Inoltre, tramite questo costrutto, la federazione è gestibile sia a livello applicativo, tramite ad esempio l'utilizzo di un Web Service come implementato in questo lavoro di tesi, ma soprattutto diventa gestibile a livello di endpoint SPARQL, quindi totalmente indipendente dal framework di gestione dell'endpoint (es. Openlink Virtuoso), questo perché il costrutto integrato nel linguaggio è disponibile da qualsiasi endpoint che utilizzi la tecnologia SPARQL.

8.2.2 Ontologia come standard per definire una federazione

Nel progetto per gestire le informazioni concernenti la federazione, si è pensato di utilizzare l'ontologia come tecnologia esistente e che inoltre fosse il più generale possibile.

Questo ha permesso la definizione di un vocabolario specifico, in cui sono stati definiti due termini chiave, identificati come classi, per definire la federazione:

- ***Endpoint***: identifica un endpoint;
- ***Federazione***: identifica una federazione.

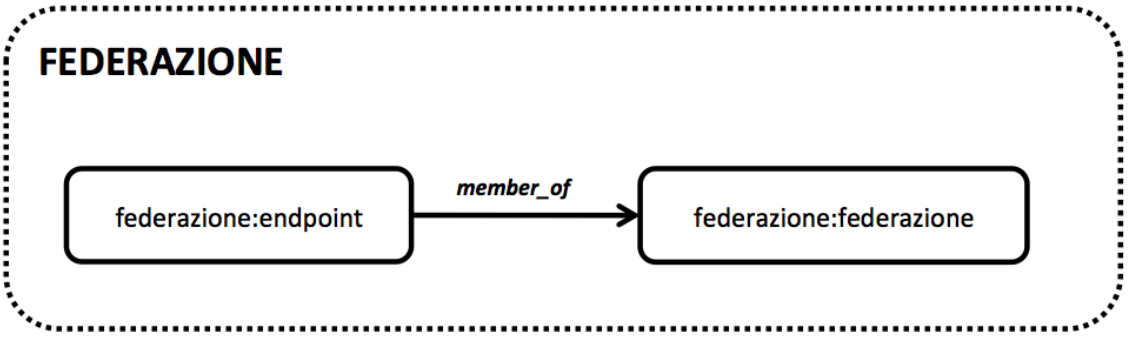

Figura 39 - Ontologia Federazione.

È inoltre stata definita la relazione fondamentale ***member_of***, tramite la quale si identifica la relazione di appartenenza degli endpoint alla federazione.

Questa soluzione potrebbe essere un punto di partenza per definire un vocabolario generale con il quale identificare una federazione, in particolare identificando tutte le informazioni relative agli endpoint (nome, location, framework, namespace gestiti, ecc.) e tutte le informazioni relative alle federazione (nome federazione, membri della federazione, location della federazione, ecc).

8.3 Federazione dei dati e processi di scrittura

La tesi si è concentrata soprattutto sugli aspetti di lettura del dato piuttosto che su quelli di scrittura.
In uno scenario di decentramento delle responsabilità, ovvero sono i singoli membri della federazione a produrre i loro dati e a mantenerli, non si è voluto forzare lo studio su processi di scrittura che sono sicuramente possibili sulla carta, ma poco applicabili nella realtà.
L'unico processo a risultare realmente efficace è quello della notifica, ovvero il supporto di uno scenario con la gestione degli eventi in cui il modello prevede tra i propri dati anche la definizione di determinati eventi da notificare in broadcast a tutti i membri della federazione per poi delegare a loro eventuali attività in risposta di questi eventi.
Questo tipo di operatività può risultare, però, decisamente più orientata alla gestione in real time piuttosto che all'analisi dei dati e quindi non strettamente collegabile all'Enterprise Architecture.
Per questa ragione non sono stati realizzate «Proof Of Concept» dei casi di scrittura, questo per evitare che potessero spostare l'attenzione della nostra ricerca su aspetti più vicini all'IT Operation che non ai processi di «decision-making».

8.4 Risultati ottenuti

Riassumendo quanto detto e provato sperimentalmente, possiamo dire che il Semantic Web è sicuramente una tecnologia abilitante per la disciplina di Enterprise Architecture e che l'archetipo della federazione di fonti dati può essere utile per rendere più semplice e veloce l'analisi del dato, in quanto si delega ai singoli membri della federazione una parte del lavoro e la sua normalizzazione ed esposizione su modelli condivisi e standard riconosciuti.

Possiamo anche dire che sia a livello tecnico (standard SPARQL) che tecnologico (Middleware) non si è ancora raggiunti la piena maturità:

- le specifiche affrontano comunque in modo parziale il tema della federazione e più in generale il tema dell'aggregazione di più fonti dato;
- i middleware offrono implementazioni minime di SPARQL e le feature aggiuntive che vengono offerte non sono legate a degli standard, per cui è possibile avere situazioni di lock-in tecnologico.

Esiste quindi ancora ampio spazio di lavoro per sviluppare lo standard, ma già oggi vi è la possibilità reale di implementare e realizzare delle soluzioni di compromesso accettabili come ampiamente dimostrato dalla nostra sperimentazione.

Conclusioni

L'ipotesi iniziale «*il Semantic Web risulta essere una tecnologia abilitante per l'Enterprise Architecture*» è stata ampiamente discussa e confermata dalle sperimentazioni fatte e dai risultati ottenuti.
Sono però emersi anche alcuni fattori critici che riguardano lo standard SPARQL, il cui gruppo di lavoro non sembra molto attivo, riscontrato da due soli rilasci ufficiali delle specifiche in più di cinque anni, e questo contribuisce ad una evoluzione disordinata dei middleware a supporto.
Unendo queste due conclusioni, ovvero che il Semantic Web risulta abilitante per l'Enterprise Architecture e che non c'è un movimento attivo sul linguaggio SPARQL, si possono aprire due scenari:

1) Aspettare che emerga una grande azienda che faccia da traino sul tema, eventualmente imponendo specifiche e linguaggi non standard;
2) Contribuire fornendo "dal basso" idee e ricerche che possano migliorare e sviluppare tecniche e tecnologie.

Ovviamente, per quanto riguarda la ricerca IT pura, la seconda opzione risulta essere più interessante, perché permette un confronto nell'ambito accademico e scientifico e consente di sperimentare e ingegnerizzare delle soluzioni non dettate da logiche di prodotto, cosa che in parte sta già accadendo.

Le «federated query» sono state introdotte in SPARQL grazie a una spinta innovativa arrivata dalla relativa community e non perché volute dal gruppo di lavoro ufficiale.
Uno scenario, quello delle innovazioni che arrivano dalle varie community IT, che oramai è un trend consolidato e che non sembra destinato ad esaurirsi nel breve periodo.
Inoltre è significativo, per questo lavoro di tesi, che sia stata la community stessa a voler introdurre il concetto di federazione
Dalle riflessioni fatte sugli scenari appena enunciati, i ricercatori e la ricerca IT si dovrebbero concentrare maggiormente sulla parte di sviluppo di tecniche e

tecnologie in grado di interrogare fonti dato che risultano essere sparse ed eterogenee.
Questa tipologia di ricerca potrebbe essere ricondotta alla ricerca su un trend attale, quale quello dei Big Data, ovvero quella disciplina che studia come gestire grandi volumi di dati, di diverse tipologie e formato, con la caratteristica di cambiamento rapido nel tempo.
In questo tipo di scenario diventa difficile pensare di poter replicare su di un proprio singolo sistema informativo grandi moli di dato provenienti dall'esterno della propria organizzazione.
Diventa quindi importante poter interrogare on-line le singole fonti dato, e una possibile soluzione a questa esigenza potrebbe essere fornita dai servizi web chiamati endpoint SPARQL, motivo cardine che ha spinto la sperimentazione ad indagare nella direzione di questi middleware.
Purtroppo la loro maturità non sembra ancora quella che ci si aspetta da questo tipo di tecnologie.

La federazione è un archetipo molto comune, per cui è pensabile la sua adozione in diverse situazioni a valore aggiunto:

- Pubblica Amministrazione (PA);
- Fusione di aziende e quindi di sistemi informativi.

Nella PA, per esempio, i dati risultano essere sparsi su più enti e inoltre presentano una molteplicità di relazioni con soggetti privati.
Uno scenario che ben si presta sia ad erogare servizi integrati che ad essere analizzato nel suo complesso.
Allo stesso modo, durante una fusione di più sistemi informativi, oppure nella scelta di integrare servizi in full outsourcing, risulta importante poter governare questi processi fornendo un supporto attivo in ottica di analisi dei dati per proiezioni sul futuro, piuttosto che per un'analisi analitica di quanto successo nel passato.
Un limite emerso dall'utilizzo della federazione di fonti dato è la gestione del modello dei dati, oltre che alla necessità di integrarle secondo una specifica.

A rigor di logica, questi limiti sono presenti anche nel caso tradizionale di Business Intelligence, ma nel caso della federazione risultano essere più onerosi nella loro gestione, questo perché si ha a che fare con fonti dato indipendenti anche dal punto di vista organizzativo.

Vi è comunque la possibilità di procedere verso un modello federato per piccoli step evolutivi, per cui è possibile aggirare l'ostacolo a patto di mantenere una funzione di Enterprise Architecture ben presidiata all'interno dell'organizzazione che intende gestire la federazione e che trae i reali benefici dall'analisi dei dati estratti.

L'Enterprise Architecture risulta difficilmente applicabile attraverso una sequenza di progetti non correlati tra loro e quindi, dal punto di vista squisitamente finanziario, è da considerarsi un costo fisso che può partire con budget anche molto piccoli, ma che non può essere interrotto per lunghi periodi senza perdere i vantaggi dell'aver costruito un modello comune.

Oltre a questo ha anche un impatto organizzativo rilevante, perché si pone come funzione complementare alla Business Intelligence in quanto si concentra sul supporto delle decisioni per il futuro piuttosto che sull'analisi e la misurazione delle performance passate.

A questo va aggiunto che l'introduzione di un modello dati comune, anche se in modo graduale, ha un impatto su tutta la comunicazione tra i vari gruppi di lavoro o tra le organizzazioni, in quanto dovranno sostituire i micro linguaggi, che normalmente si creano negli ambienti chiusi, con un linguaggio più aperto e comprensibile anche all'esterno delle singole organizzazioni.

Questo è uno sforzo molto significativo, confermato anche dal personale dell'azienda in cui è stato svolto il lavoro di tesi.

La definizione dei concetti e delle loro caratteristiche è l'aspetto fondante delle loro consulenze in ambito Enterprise Architecture e hanno trovato moltissime resistenze nell'adozione di uno standard comune anche all'interno della singola azienda, per cui questo problema non potrà che amplificarsi in scenari più grandi ed eterogenei.

Quindi in conclusione si può affermare che gli aspetti esaminati nella tesi sono quelli dove c'è maggior spazio per fare ricerca IT a valore aggiunto, ma anche che la situazione è ancora molto fluida sia per quanto riguarda i metodi che le tecnologie.
Questo induce a suggerire di adottare una funzione di Enterprise Architecture in quei contesti in cui il costo non sposti in modo significativo il budget dell'organizzazione e che il Semantic Web sia comunque abilitante per tale funzione e che debba essere adottato insieme ad essa.
Tuttavia è altrettanto ragionevole suggerire di non legarsi a singoli fornitori con soluzioni poco flessibili, perché al momento non è facile prevedere in quale direzione evolveranno le migliori tecniche e tecnologie.

Dal punto di vista delle possibili prospettive future, sicuramente la non piena maturità a livello tecnico, ovvero a livello di linguaggio SPARQL, e la non piena maturità a livello tecnologico, ovvero i middleware sul mercato, fornisce una buona motivazione per effettuare ampia ricerca IT, e in particolare negli ambiti del Semantic Web e dell'Enterprise Architecture.
Dal punto di vista della soluzione implementata, sicuramente vi è ampio spazio di miglioramento. In particolare la possibilità di identificare e realizzare strategie nel che permettano la gestione degli endpoint, in particolare gestire casi in in cui uno o più endpoint risultino essere irraggiungibili, piuttosto che uno o più endpoint risultino essere off-line, piuttosto che uno o più endpoint restituiscano risultati errati.
Allo stesso modo si potrebbero identificare e realizzare dei pattern di aggregazione e analisi dei dati, come ad esempio il pattern di UNION, piuttosto che il pattern di intersezione, con l'obiettivo di fornire un'ampia scelta di soluzioni per la gestione dei risultati.
Dal punto di vista del linguaggio SPARQL si potrebbe fornire una possibile estensione del linguaggio, tramite la quale federare endpoint SPARQL. Questa estensione, dai noi realizzata tramite il costrutto foreach, prevede in ingresso i componenti della federazione, dopo di che esegue la query su tutti i membri.
Questa soluzione a livello di linguaggio permette di utilizzare tutti i costrutti forniti da SPARQL.

Infine, ma non meno importante, si potrebbe introdurre a livello di vocabolario standard e condivisibile, come ampiamente citato il caso del vocabolario standard foaf, una ontologia che descriva la federazione, la cui utilità è individuabile nella possibilità di gestire in modo standard e trasparente qualsiasi tipologia di federazione in ambito Semantic Web.

Appendice

A. Rappresentazione delle triple nei contesti Semantic Web

I principali linguaggi definiti da specifiche W3C per il Web Semantico sono RDF, RDF-S e OWL.

A.1 Linguaggio RDF e notazioni N-Triple, RDF/XML, Turtle e N3

Resource Description Framework (RDF) è una specifica del World Wide Web Consortium (W3C) rilasciata nel 1999. Progettato per l'elaborazione di metadati web, RDF fornisce una piattaforma di interoperabilità tra applicazioni sul web che si scambiano informazioni comprensibili dalle macchine. Il framework facilita l'elaborazione automatica delle informazioni e può essere utilizzato in diverse aree di applicazione: nella ricerca di risorse (resourcediscovery), per migliorare le capacità dei motori di ricerca, per descrivere il contenuto e le relazioni disponibili in un particolare sito web, pagina o digital library, per facilitare condivisione e scambio di conoscenza tramite gli agenti intelligenti, nella valutazione del contenuto, nella descrizione di collezioni di pagine che rappresentano un singolo documento logico, nella descrizione del copyright di pagine web, per esprimere le preferenze di un utente, per la gestione della privacy di un sito.
Rdf è basato su tre differenti tipi di dati:

- *Uniform Resource Identifier (URI):* sono riferimenti utilizzati per identificare le risorse come ad esempio http://www.w3.org/2000/01/rdf-schema#Class. La parte che segue il simbolo "#" è nota come FragmentIdentifier (identificatore di frammento). Dal momento che gli URI possono condividere prefissi comuni, la notazione QName (QualifiedName) è spesso utilizzata per rendere più leggibile il documento e per brevità. Questa consiste nel definire un elemento prefisso che sostituisce un namespace, ovvero la parte radice dell'URI (la parte precedente al simbolo "#") e sostituita con una stringa, in genere più breve, seguita da ":". Nel caso dell'URI mostrato in precedenza la

parte a sinistra del "#" può essere sostituita con "rdfs:". In questo modo lo stesso URI può essere ottenuto per sostituzione con "rdfs:Class" È anche possibile definire una stringa vuota per identificare un namespace. In quest'ultimo caso la notazione QName risulterebbe essere ":Class";

- *Letterale (Literal):* si tratta di una semplice stringa di caratteri e numeri delimitata da apici, rappresentate un valore concreto, e seguita da una opzionale specifica del tipo di dato contenuto, come ad esempio stringa, intero o data;
- *Nodi vuoti (BlankNodes):* sono utilizzati per rappresentare concetti che non sono noti o non specificati. Possono definire risorse anonime che non sono identificate da un URI. Dal momento che non è specificato un URI per referenziare il concetto il prefisso è espresso tramite la notazione "_:". Un nodo vuoto non può essere usato per identificare globalmente una risorsa ma solo all'interno del documento RDF in cui è specificato.

Il modello dai dati, ovvero la struttura logica dei dati, utilizzati da RDF è basato sull'idea di fare affermazioni (o statement) sui dati. Queste affermazioni sono espresse tutte nella forma "soggetto predicato oggetto", e per questo le affermazioni prendono il nome di triple. Una di queste triple rappresenta la più piccola unità di informazione presente in un documento RDF.

Le tre parti di uno statement RDF sono espressa da:

- *Risorse*: tutte le cose descritte da espressioni RDF sono chiamate risorse. Una risorsa può essere un'intera pagina web, o una parte, o un particolare elemento all'interno di un documento HTML o XML. Una risorsa può anche essere una collezione di pagine come un intero sito web. Può ancora essere un oggetto che non è direttamente accessibile via web come un libro. Le risorse sono sempre identificate tramite un URI (ogni cosa può avere un URI) o un nodo vuoto. Le risorse possono essere soggetto od oggetto di uno statement RDF.
- *Proprietà*: è uno specifico aspetto, caratteristica, attributo o relazione utilizzata per descrivere una risorsa. Ogni proprietà ha un significato specifico, definisce i valori permessi, i tipi di risorse che può descrivere e le

relazioni con altre proprietà. Le proprietà sono sempre identificate come URI e costituiscono i predicati all'interno di uno statement RDF.

L'oggetto di un'affermazione RDF può anche essere un dato di tipo letterale. Poiché il modello dei dati RDF è utilizzato per rappresentare i Linked Data ogni affermazione RDF può essere rappresentato come un grafo, in cui soggetto e oggetto costituiscono i nodi del grafo, mentre il predicato costituisce l'arco che li collega.

Esistono diversi formati di serializzazione per produrre un documento descrittivo di uno specifico dataset a partire dal modello dei dati RDF. Di seguito verranno descritti i principali:

- ***N-Triples***: è il formato più semplice. Gli URI vengono espressi delimitandolo con parentesi angolari e le stringhe dei letterali sono delimitate da doppi apici. Ogni tripla occupa una singola riga e termina con un punto. La sua semplicità è quello che ha reso N-Triples molto popolare per insegnare alle persone ad usare RDF, e alcuni parser sono molto veloci ad analizzare questo formato perché hanno minor lavoro da svolgere, ma la verbosità del formato lo rende meno popolare di altri formati più compatti.
- ***RDF/XML (Resource Description Framework / Estensible Mark-Up Language)***: è il format più vecchio e fu parte della specifica originale di RDF nel 1999. In questo caso il grafo RDF è costruito utilizzando i tag del formato XML. I nodi del grafo possono essere costituiti sia da tag XML che da stringhe. Questo formato è il più prolisso e il più difficile da leggere e non è mai diventato molto popolare per varie ragioni, tra cui complicazioni nell'integrarsi con documenti XML o per la difficoltà con cui poteva essere processato da popolari strumenti per il formato XML.
- ***Turtle (Terse RDF Triple Language):*** è un format simile al N-Triples. Le differenze principali consistono nel poter definire un prefisso comune per gli URI e nella capacità di esprimere in un'unica riga più una tripla. In questo caso, però, le triple devono avere un soggetto comune, oppure soggetto e predicato comuni. In particolare:

- notazione separata da virgola: unisce le triple che hanno in comune gli stessi soggetto e predicato;
- notazione separata da punto e virgola: unisce le triple che hanno in comune lo stesso soggetto;
- notazioneQName: tramite la parola "@prefix" sostituisce un namespace URI con la specifica parola indicata.

Questo formato è sicuramente uno dei più usati per via della sua compattezza e facilità di lettura. Il documento mostrato in figura 2.1 risulta essere in formato Turtle;

- ***N3 (Notation 3):*** è un altro formato popolare per serializzare dati RDF. Fu un progetto personale di Tim Berners-Lee ed è caratterizzato da essere molto simile al formato Turtle. Infatti tutti i parser in gradi di analizzare N3 sono in grado di analizzare anche Turtle. Questo perché Turtle è una sottoparte di N3, infatti N3 offre in più alcune funzionalità e alcune scorciatoie per descrivere più fatti aventi lo stesso soggetto. In realtà queste funzionalità non sono molto usate ed N3 non è mai diventato uno standard.

A.2 RDF-Schema e OWL

RDF Schema (RDFS, noto anche come RDF vocabulary description language) è una estensione semantica per RDF introdotta con lo scopo di descrivere le ontologie. Fornisce un meccanismo di base per descrivere gruppi di risorse collegate tra loro e le relazioni tra le stesse risorse. È in grado di definire concetti e relazioni, e di definire vocabolari di termini per modellare specifici domini attraverso il linguaggio RDF stesso [18].

RDFS è basato su un set di classi e proprietà di default con cui è possibile specificare altre classi e proprietà, e specificare, così, un vocabolario completo. Il vocabolario costruito deve definire tutti i possibili concetti e tutte le relazioni tra i concetti, con lo scopo di poter costruire una struttura adatta a descrivere correttamente l'ambito informativo che si vuole modellare.

Il sistema di proprietà e classi di RDFS è simile a quello di un normale linguaggio ad oggetti, la differenza sta nel fatto che, invece di definire una classe in base alle proprietà che possono avere le istanze, descrive le proprietà in base alle classi di risorse a cui si applicano. Un beneficio di questo approccio focalizzato sulle proprietà è di permettere di estendere la descrizione di risorse esistenti molto più facilmente senza la necessità di ridefinire la descrizione originale, uno dei principi architetturali del web.

Il Web Ontology Language OWL è un linguaggio di markup semantico per la pubblicazione e la condivisione di ontologie sul World Wide Web . OWL è sviluppato come estensione del vocabolario di RDF (Resource Description Framework) e deriva dal DAML + OIL Web Ontology Language . Lo scopo è quindi quello di descrivere il significato semantico dei dati, presenta strumenti più espressivi di RDFS per esprimere le informazioni semantiche dei dati di cui ne costituisce una estensione.

La maggiore espressività, rispetto ad RDFS è dovuta alla capacità di descrivere:

- vincoli di cardinalità (es . una persona ha una sola madre e un solo padre);
- possibilità di indicare che due classi definite in schemi differenti rappresentano lo stesso concetto. Proprietà molto utile all'interno di un ambito distribuito come il Web, per collegare risorse definite in due schemi indipendenti. È il caso di "owl:sameAs";
- possibilità di indicare che due istanze, definite separatamente, rappresentano lo stesso concetto;
- possibilità di indicare che una proprietà è transitiva;
- la possibilità di definire una nuova classe come combinazione di più classi esistenti.

La definizione e la realizzazione di queste (e altre) capacità sono lo scopo del gruppo di lavoro sui linguaggi per la definizione di ontologie.

B. Open Data e Big Data

Il progetto Open Definition di Open Knowledge Foundation utilizza la seguente frase per definire dati (e contenuti) aperti: «un contenuto o un dato si definisce aperto se chiunque è in grado di utilizzarlo, ri-utilizzarlo e ridistribuirlo, soggetto, al massimo, alla richiesta di attribuzione e condivisione allo stesso modo».

I dati aperti (open data anche nel contesto italiano), sono alcune tipologie di dati liberamente accessibili a tutti, privi di brevetti o altre forme di controllo che ne limitino la riproduzione e le cui restrizioni di copyright eventualmente si limitano ad obbligare di citare la fonte o al rilascio delle modifiche allo stesso modo. L'Open Data si richiama alla più ampia disciplina dell'open government, cioè una dottrina in base alla quale la pubblica amministrazione dovrebbe essere aperta ai cittadini, tanto in termini di trasparenza quanto di partecipazione diretta al processo decisionale, anche attraverso il ricorso alle nuove tecnologie dell'informazione e della comunicazione; ha alla base un'etica simile ad altri movimenti e comunità di sviluppo "open", come l'open source, l'open access e l'open content. Nonostante la pratica e l'ideologia che caratterizzano i dati aperti siano da anni ben consolidate, con la locuzione "open data" si identifica una nuova accezione piuttosto recente e maggiormente legata a Internet come canale principale di diffusione dei dati stessi.

Gli Open Data fanno di frequente riferimento a informazioni rappresentate in forma di database e riferite alla tematiche più disparate, ad esempio: cartografia, genetica, composti chimici, formule matematiche e scientifiche, dati medici e pratica, delle bioscienze, datia nagrafici, dati governativi della pubblica amministrazione, dati che derivano da soggetti pubblici acquisiti mediante ricerche o rilevazioni le più diverse, ecc.

Vi sono alcune difficoltà oggettive che impediscono alla pratica dei dati aperti una larga diffusione. Uno dei problemi principali spesso riguarda il valore commerciale che gli stessi dati, visti sia in forma puntuale che aggregata. I dati sono di frequente controllati da organizzazioni, sia pubbliche che private, che spesso mostrano reticenza di fronte alla possibilità di diffondere il proprio patrimonio informativo. Il

controllo sui dati può avvenire attraverso limitazioni all'accesso, alle licenze con cui vengono rilasciati, ai diritti d'autore, brevetti e diritti di riutilizzo.

Di fronte a queste forme di controllo sui dati, e più in generale sulla conoscenza, i sostenitori dell'Open Data affermano che tali restrizioni siano un limite al bene della comunità e che i dati dovrebbero essere resi disponibili senza alcuna restrizione o forma di pagamento. Inoltre, è importante che i dati, dopo essere stati pubblicati, siano riutilizzabili senza necessità di ulteriore autorizzazione, anche se determinate forme di riutilizzo (come la creazione di opere derivate) può essere controllato attraverso specifiche licenze (ad esempio Creative Commons, GFDL).

Inoltre accade spesso che gli stessi creatori di dati sottovalutino l'importanza degli stessi e non considerino la necessità di precisare lec ondizioni della proprietà intellettuale, delle licenze e del loro riutilizzo. Ad esempio, molti enti (siano essi di natura scientifica o governativa) per mancanza di consapevolezza dell'importanza dei propri dati non prendono in considerazione l'ipotesi di rilasciarli con licenze aperte. La mancanza di una determinata licenza che certifichi la possibilità di riutilizzare i dati rende difficile determinare lo stato di un insieme di dati e ne limita l'uso.

Secondo i sostenitori del movimento Open data, i dati andrebbero trattati come beni comuni; di seguito alcune argomentazioni a sostegno di questa tesi:

- I dati appartengono al genere umano. Esempi tipici sono i genomi, i dati sugli organismi per la scienza medica, dati ambientali e meteorologici, ecc.;
- I dati prodotti dalla pubblica amministrazione, in quanto finanziati da denaro pubblico, devono ritornare ai contribuenti, e alla comunità ingenerale, sotto forma di dati aperti e universalmente disponibili;
- Restrizioni sui dati e sul loro riutilizzo limitano lo sviluppo della comunità;
- I dati sono necessari per agevolare l'esecuzione di comuni attività umane(ad es. i dati cartografici, i dati delle istituzioni pubbliche, ecc.);
- In campo scientifico il tasso di scoperta è accelerato da un migliore accesso ai dati;
- È essenziale che i dati scientifici siano resi aperti per fare in modo che la scienza sia più efficace e la società ottenga il massimo beneficio dalle ricerche scientifiche.

In estrema sintesi si possono individuare aspetti che caratterizzano un insieme di dati come "aperti":

- I dati aperti devono essere indicizzati dal motori di ricerca;
- I dati aperti devono essere disponibili in un formato aperto, standardizzato e leggibile da un'applicazione informatica per facilitare la loro consultazione ed incentivare il loro riutilizzo anche in modo creativo;
- I dati aperti devono essere rilasciati attraverso licenze libere che non impediscano la diffusione e il riutilizzo da parte di tutti i soggetti interessati.

Big Data è il termine per descrivere una raccolta di dataset così grande e complessa da richiedere strumenti differenti da quelli tradizionali, in tutte le fasi del processo: dall'acquisizione, alla *curation*, passando per condivisione, analisi e visualizzazione.
I Big Data possono essere definiti come "una raccolta di dati così grande e complessa da richiedere strumenti differenti da quelli tradizionali per essere analizzati e visualizzati".
Nell'ambito dei Big Data i dati possono provenire potenzialmente da fonti eterogenee, come dati strutturati e non, immagini, dati GPS, informazioni provenienti da social network, immagini, video e suoni.
Sulle caratteristiche dei BigData tutti sono d'accordo sulle 3 V:

- volume: capacità di acquisire, memorizzare ed accedere a grandi volumi di dati;
- velocità: capacità di effettuare analisi dei dati in tempo reale o quasi;
- varietà: riferita alle varie tipologie di dati, provenienti da fonti diverse.

Ed alcuni parlano di una 4' V: veridicità: ossia la qualità dei dati intesa come il valore informativo che si riesce ad estrarre.
Insomma il mercato dei BigData richiede fondamentalmente:

- Grandi sistemi di storage dei dati, in termini di exabyte (10^{18}), zettabyte (10^{21}) e yottabyte (10^{24});
- Grande capacità di calcolo parallelo;

- Personale qualificato (Data Analyst, Data Scientist) capace di “fiutare” dei risultati interessanti analizzando grosse quantità di dati apparentemente slegati tra loro;
- SW di acquisizione dei dati continua, SW di analisi dei dati e SW di raffigurazione visuale dei dati.

Come è stato detto in precedenza, il volume di dati dei Big Data e l'ampio uso di dati non strutturati non permette l'utilizzo dei tradizionali RDBMS, che non rendono possibile archiviazione e velocità di analisi. Gli operatori di mercato invece utilizzano sistemi con elevata scalabilità e soluzioni basate sulla NoSQL.

Nell'ambito della Business Analytics nascono nuovi modelli di rappresentazione in grado di gestire tale mole di dati con elaborazioni in parallelo dei database. Architetture di elaborazione distribuita di grandi insiemi di dati sono offerte da MapReduce di Google, e dalla controparte open source Apache Hadoop. Con questo sistema le applicazioni sono separate e distribuite con nodi in parallelo, e quindi eseguite in parallelo (funzione map). I risultati vengono poi raccolti e restituiti (funzione reduce).

I Big Data sono una grandissima opportunità per le grandi imprese di hardware e software IT (IBM, HP, EMC, Oracle,ecc.), infatti cresce l’esigenza di SW semplice, dedicato e personalizzato alle Data Analisi. Ma mantenere petabyte o zettabyte in Cloud costa veramente tanto può addirittura convenire mantenersi una propria infrastruttura.

Diverso è se abbiamo alcuni terabyte di dati su cui vogliamo fare Data Analysis, infatti si pensa sia questo lo scenario più generale, dove i servizi di un Cloud diventano veramente concorrenziali.

Recentemente i Big dell’IT hanno fatto partite moltissime opportunità per imprese, startup e mondo della ricerca relativamente i Big Data, è una vera esplosione per l’economia IT. [ALE13]

C. Reasoner

I reasoner sono degli strumenti software, disponibili agli sviluppatori tramite API o attraverso gli application framework, che permettono, tramite un "metodo logico" di ragionare in automatico sulla conoscenza ed effettuare deduzioni.
Si distinguono principalmente per l'espressività logica e il linguaggio ontologico che supportano: dai costrutti di base di RDFS e OWL Light, che permetto di modellare gerarchie di classi, istanze e semplici relazioni, a OWL DL, che possiede l'espressività delle logiche descrittive, fino al completo supporto di FOL usando OWL Full.
La maggior parte dei reasoner per il Semantic Web mira a supportare pienamente l'espressività delle logiche descrittive, ance se le più recenti evoluzioni dei reasoner, contemplano l'utilizzo di regole compatibili con il linguaggio SWRL e il supporto ad alcuni tipi di dato per eseguire operazioni e confronti tra numeri e tipi di dato complessi come date e stringhe.
Alcuni reasoner si sono evoluti per gestire in modo ottimizzato grandi quantità di dati, mentre altri si sono focalizzati sul controllo delle complesse relazioni tra classi, questo perché vi è sempre di più la necessità di avere reasoner scalabili e performanti dettata dall'aumento della dimensione e della complessità delle ontologie.
Tra i più famosi reasoner in commercio citiamo RacerPro, considerato uno dei migliori attualmente, il quale si focalizza sulle logiche descrittive supportando ontologie OWL Light e OWL DL, compatibile con SWRL.
Pellet è un altro reasoner per OWL DL, molto simile alle prestazioni di RacePro, con la funzionalità aggiuntiva dell'utilizzo tramite Jena o DIG.
Altri reasoner da citare sono OWLIM, reasoner più veloce per OWL, e KAON2, che a differenza degli altri, offre un miglior supporto per SWRL.

D. Simulazione

Simulare significa creare una "copia" di un sistema, farlo funzionare per un periodo di tempo adeguato, allo scopo di valutarne il comportamento sotto diverse condizioni di esercizio.

La simulazione quindi permette di osservare il comportamento di un sistema senza doverlo necessariamente costruire o di comprendere come le varie componenti interagiscono tra loro e incidano sulle performance globali.

La simulazione è un potente strumento di descrizione e analisi di sistemi complessi nell'ambito del manufacturing, della logistica, della distribuzione, dei servizio, ma soprattutto nell'ambito della disciplina dell'Enterprise Architecture.

Generalmente quando si stabiliscono gli obiettivi in ambito aziendalesi utilizzano degli indicatori, chiamati KPI (Key performance indicator), per misurare le performance degli obiettivi prefissati.

Questi indicatori in una struttura aziendale agiscono su diverse unità di business e di IT, cercando di evidenziare eventuali relazioni e performance.

I KPI sono generalmente utilizzati, anche nell'ambito dell'Enterprise Architecture, per la simulazione di possibili scenari futuri in relazione ad attività e processi aziendali, flusso di esecuzione, obiettivi a medio e lungo termine, risultati ottenibili.

I modelli di simulazione sono molto importanti, diventando un supporto per la comprensione approfondita del sistema, alla valutazione di lungo termine, come l'analisi costi/benefici relativa a decisione che coinvolgono aspetti strutturali dell'impresa, alla valutazione tattica anche chiamata "what-if", alle decisioni a breve termine.

Riallacciandoci all'ambito universitario, un possibile esempio di simulazione potrebbe analizzare i dati relativi ai professori in un determinato anno accademico, verificare se qualcuno di loro l'anno accademico successivo andrà in pensione, capendo quindi che se assumere o meno personale.

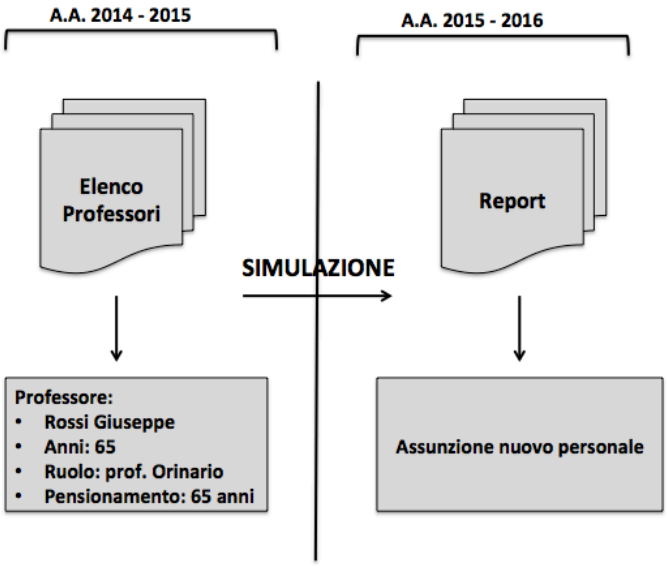

Figura 40 - Simulazione.

In definitiva la simulazione è una tecnica a basso costo, che non implica grandi rischi, ma che rappresenta una valido supporto al management nel processo del decision-macking.

E. High Availability

Per High Availabilitysi intende la capacità di rispettare le specifiche di funzionamento nel tempo. In sintesi, l'alta affidabilità di un sistema comunque complesso o di un semplice componente (server, servizio, applicazione, etc.) è la misura della probabilità che il sistema (od il componente)considerato non si guasti (ovvero non presenti deviazioni dal comportamento descritto nella specifica) in un determinato lasso di tempo.

L' High Availability è costituita da un insieme di aspetti quali ad esempio:

- *la tolleranza ai guasti (fault-tolerance):* è la capacità di un sistema di non subire fallimenti(cioè interruzioni di servizio) anche in presenza di guasti. É importante notare che la tolleranza ai guasti non garantisce l'immunità da tutti i guasti, ma solo che i guasti per cui è stata progettata una protezione non causino fallimenti;
- *La garanzia sui servizi erogati.* I servizi devono continuare ad essere disponibili anche in caso di guasto alle macchine su cui girano;
- *Garanzia e sicurezza dei dati memorizzati:* Deve essere garantita l'integrità dei dati memorizzati sui supporti di memorizzazione di massa, e deve essere garantita la loro raggiungibilità futura anche in caso di guasti solamente alle entità (persone o processi)autorizzate a farlo.

I cluster High Availability, o Cluster HA, sono una tipologia di cluster disegnata per garantire continuità dei servizi informatici erogati. L'impiego di componenti ridondanti, algoritmi e protocolli di monitoraggio dell'attività dei singoli nodi e l'analisi del flusso delle informazioni consente di offrire elevatissime garanzie di funzionamento.

F. ACID (Atomicità, Consistenza, Isolamento, Durabilità)

Nell'ambito dei Database, una transazione può essere definita come un insieme di istruzioni di lettura e scrittura sulla base di dati, eventualmente immerse in un linguaggio di programmazione, per cui siano garantite determinate proprietà per la loro corretta esecuzione in ambiente concorrente.
Queste proprietà, dette ACID (acronimo di Atomicità, Consistenza, Isolamento e Durabilità) sono quattro proprietà fondamentali che le transazioni devono godere per eseguire operazioni sulla base di dati in maniera corretta e sicura.

Atomicità

L'atomicità di una transazione è la sua proprietà di essere eseguita in modo atomico, cioè con un approccio "tutto o niente". I suoi effetti sulla base di dati devono essere resi visibili del tutto (cioè al termine della transazione) oppure nessun effetto deve essere mostrato. Questa proprietà è garantita dal DBMS attraverso la sua capacità di annullare (operazione di UNDO) una transazione che ha subito un ABORT prima del commit, e la sua capacità di rifare (operazione di REDO) una transazione che è incorsa in un errore dopo il commit.
L'ABORT può essere causato da:

- omicidio: la transazione viene terminata dal sistema per vari motivi (violazione di vincoli, guasto del sistema, ecc.);
- suicidio: la transazione raggiunge un'istruzione ROLLBACK, quindi richiede l'annullamento degli effetti prodotti da se stessa.

In questo modo l'istruzione COMMIT fissa in modo definitivo e indivisibile il momento in cui la transazione ha avuto successo, quindi se essa avrà o meno effetto sulla base di dati.

Consistenza

La consistenza di una transazione è la sua capacità di non violare i vincoli referenziali e di integrità della base di dati. Una transazione che violi questi vincoli deve essere abortita e i suoi effetti annullati. Questo controllo può essere eseguito per

via immediata, se fatto direttamente sulla singola istruzione della transazione, oppure per via differita se effettuato al termine della transazione (cioè al raggiungimento del comando di COMMIT).
La consistenza deve essere inoltre garantita dopo l'esecuzione di trigger e altri eventi immediatamente successivi al commit della transazione (altrimenti l'intera transazione deve essere annullata).

Isolamento

L'isolamento è la garanzia che il DBMS esegua le transazioni in maniera indipendente ed isolata, in particolare che non interferiscano tra di loro se l'esecuzione è contemporanea. Questa proprietà deve essere garantita anche in caso qualche transazione effettui ROLLBACK, senza che le altre ne siano influenzate.
L'isolamento è garantito se l'esecuzione di transazioni in contemporanea genera lo stesso effetto sulla base di dati che avrebbero se venissero eseguite in maniera sequenziale. Il componente del DBMS che si occupa di garantire questa proprietà è il controllore di concorrenza.

Persistenza

La persistenza si ottiene garantendo che l'effetto delle transazioni che hanno eseguito con successo l'istruzione COMMIT sia mantenuta per sempre nel database, anche in caso di guasti e malfunzionamenti. Idealmente la persistenza deve essere assicurata anche in caso di eventi catastrofici, come incendi o allagamenti, oltre che a mancanze di alimentazione. Ovviamente questo goal può essere raggiunto solo distribuendo la base di dati in più server farm (e il DBMS deve essere in grado di gestirle).

Bibliografia

[ALE12] Alessandro Rezzani:"Business Intelligence", Apogeo, 2012

[ALE13] Alessandro Rezzani:"Big Data. Architettura, tecnologie e metodi per l'utilizzo di grandi basi di dati", Maggioli by PDE, 2013

[ALL14] AllegroGraph:"AllegroGraphRDFStore Benchmark Result", Franz Inc, 2014
http://franz.com/agraph/allegrograph/agraph_benchmarks.lhtml

[ALL15] AllegroGraph:"AllegroGraph", Franz Inc, 2015
http://franz.com/agraph/allegrograph/

[APA14] Apache CXF: "Apache CXF: An Open-Source Services Framework", The Apache Software Foundation, 2014
http://cxf.apache.org/

[APJ14] Apache Jena: "Apache Jena Fuseki", The Apache Software Foundation, 2014
https://jena.apache.org/

[APA15] Apache Jena: "Apache Jena: a free and open source Java Framework for building Semantic Web and Linked Data application.", The Apache Software Foundation, 2015
https://jena.apache.org/

[APAL15] Apache Logging: "Apache LOG4J2", The Apache Software Foundation, 2015
http://logging.apache.org/log4j/2.x/

[AUE08] Auerbach: “Enterprise Architecture A to Z” - Frameworks, Business Process Modeling, SOA, and Infrastructure Technology - Jun 2008.

[BOB11] Bob DuCharme: “Learning SPARQL”, O’Reilly, 2011

[CHR08] Christian Bizer, Andreas Schultz: “Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints.”, Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems, 2008
http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/BizerSchulz-BerlinSPARQLBenchmark.pdf

[CHR09] Christian Bizer, Andreas Schultz:”Berlin Sparql Benchmark (BSPM)”, International Journal on Semantic Web & Information System, Vol5, 2009
http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/Bizer-Schultz-Berlin-SPARQL-Benchmark-IJSWIS.pdf

[CHR15] Christian Bizer, Andreas Schultz:”Berlin Sparql Benchmark (BSPM)”, univeristà of manneheim, 2015
http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

[DIZ14] Dizionario Italiano:”Federazione”, dizionarioitaliano.it, 2014
http://dizionarioitaliano.it/federazione/

[EMA09] Emanuele Della Valle – Irene Celino – Dario Cerizza: “Semantic Web – Dai fondamenti alla realizzazione di un’applicazione”, Pearson – Addison Wesley, 2009.

[FEL13] Felice Pescatore: “Enterprise Architecture and Enterprise IT Architecture”, felicepescatore.it, 2013
http://www.felicepescatore.it/blog/30-blog/thinkagile/68-enterprise-architecture-and-enterprise-it-architecture

[FRA13] Francesco Bocola, Manager HSPI: “Enterprise Architecture: gli obiettivi, i framework, i possibili approcci, gli strumenti, le relazioni con altre pratiche”, HSPI, 2013
http://www.hspi.it/files/enterprise_architecture.pdf

[FRF13] Francesco Ficetola: “DB NoSQL – Analisi Potenziale”, francescoficetola.it, 2013
http://www.francescoficetola.it/wp-content/uploads/2013/02/Database-NoSQL-Comparazione-tecnologie.pdf

[FOF14] Dan Brickley and Libby Miller: “FOAF Vocabulary Specification 0.99”, XMLSN.com, 2014
http://xmlns.com/foaf/spec/

[JNA13] Jnan Dash: “RDBMS vs. NoSQL: How do you pick?”, zdnet.com, 2013
http://www.zdnet.com/article/rdbms-vs-nosql-how-do-you-pick/

[JUI14] JUnit: “JUnit”, junit.org, 2014
http://junit.org/

[MAR06] Marco Bozzetti: “IT Governance, che cos’è?”, Mondo digitale, 2006
http://www.malaboadvisoring.it/index.php?option=com_docman&task=doc_download&gid=8

[MAR12] Margaret Rose:”Data Governance Policy”, Techtarget.com, 2012

http://searchcompliance.techtarget.com/definition/Data-governance-policy

[MAT06] MatteoBallarin: "SKOS: Simple Knowledge Organisation System", ISKO Italia, 2006
http://www.iskoi.org/doc/skos.htm

[MED14] Media.ttgtmedia.com:"Data Ownership", ttgtmedia, 2014
http://cdn.ttgtmedia.com/searchDataManagement/downloads/LoshinEnterprisech2.pdf

[MOH11] Mohamed Morsey, Jens Lehmann, Soren Auer, and Axel-CyrilleNgongaNgomo:"DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data", Department of Computer Science Leipzig Univertisty, Germany, 2011
http://svn.aksw.org/papers/2011/VLDB_AKSWBenchmark/public.pdf

[NOR05] Noreen Kendle: "The Enterprise Data Model", TDAN, 2005
http://www.tdan.com/view-articles/5205

[PAT13] P. Atzeni, S. Ceri, P.Fraternali, S. Paraboshi, R. Torlone: "Basi di Dati: modelli e linguaggio di interrogazione", McGraw-Hill Italia, 2013

[PHI10] Philipp Diefenthaler, Bernhard Bauer: "Gap Analysis in Enterprise Architecture using Semantiv Web Tecnologies", Softplant GmbH, Munich, Germany, 2010
http://www.softplant.de/fileadmin/data/publications/ICEIS-Paper-130-Diefenthaler.pdf

[PRI09] Privitera Stefano, Dameri Renata P.:"IT Governance, Concetti teorici e implementazione", Franco Angeli, 2009

[SAS13] Sas Data Management: ”Tutto quello che è necessario sapere sulla gestione strategia dei dati”, Sas Institute, 2013
http://www.sas.com/offices/europe/italy/tecnologie/dataintegration/download/quartino-data-management.pdf

[SES15] Sesame: “Sesame is a powerful Java Framework for processing and handling RDF data”, 2015
http://rdf4j.org/

[SGU14] S. Guarracino:”Federazione e confederazione”, Dizionario di storia moderna e contemporanea, 2014
http://www.pbmstoria.it/dizionari/storia_mod/f/f024.htm

[STE07] Stefania Costantini, Antonella Dorati: “Approccio al Web Semantico”, Websemantico.org, 2007.
http://www.websemantico.org/articoli/approcciwebsemantico.php

[STE11] Stefano Rossini: “Panoramica sulla IT Governance”, MokaByte, 2011
http://www2.mokabyte.it/cms/article.run?permalink=mb162_itgovernance-1

[SUI08] Suise Stephens: “The Enterprise Semantic Web: Technologie and Application for the Real World”, Oracle 2008
http://www.springer.com/cda/content/document/cda_downloaddocument/9780387485300-c2.pdf?SGWID=0-0-45-439602-p173700012

[THE08] The Smart Data Advantage: “SPARQL vs. SQL”, Cambridge Semantic, 2008
http://www.cambridgesemantics.com/semantic-university/sparql-vs-sql-intro

[TOM13] Tommaso di Noia, Roberto De Virgilio, Eugenio Di Sciascio, Francesco M.Donini: “Semantic Web – Tra ontologie e Open Data”, Apogeo, 2013.

[VIR14] Virtuoso Openlink:”LUBM and Virtuoso”, Openlink Software, 2014 http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSArticleLUBMBenchmark

[VIR15] Virtuoso Openlink:”Virtuoso Universal Server”, Openlink Software, 2015 http://virtuoso.openlinksw.com/

[W3C04] W3C Recommendation: “OWL Web Ontology Languange”, W3C, 2004 http://www.w3.org/TR/owl-ref/

[W3C08] W3C Recommendation: “SPARQL Query Language for RDF”, W3C, 2008 http://www.w3.org/TR/rdf-sparql-query/

[W3C12] W3C Recommendation: “SPARQL 1.1 Query Language”, W3C, 2012 http://www.w3.org/TR/2012/WD-sparql11-query-20120105/

[W3C13] W3C Recommendation: “SPARQL 1.1 Federated Query”, W3C, 2013 http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/

[W3C15] W3C Recommendation: “Semantic Web”, W3C, 2015 http://www.w3.org/2001/sw/