**Interfaccia visuale per l’interrogazione e l’interazione con data set semantici**

# a cura di

# Valentini Simona

# Ringraziamenti

Elaborare una tesi è difficile, per me lo è stato. Non è stato difficile però scrivere questa parte, perché posso ringraziare solo due persone. Ossia, quelle che mi hanno permesso di raggiungere questo traguardo e di essere la persona che sono. I miei genitori.

**SOMMARIO**

Indice delle figure3

**Introduzione**4

**Capitolo 1: Il Web Semantico4**

1.1 L’architettura del Web Semantico7

1.2 URI (Uniform Resource Identifier)8

1.3 I linguaggi del Web Semantico10

1.3.1 Metadati10

1.3.2 RDF11

1.3.2.1 Modello e sintassi RDF11

1.3.2.2 RDF Schema14

1.3.3 SPARQL15

1.3.4 Vocabolari e Ontologie18

1.3.5 I Linked Data 22

**Capitolo 2: Strumenti di aggregazione dei dati presenti nel Web**2**5**

2.1 Applicazione per il Web Mashup25

2.2 Applicazione per il Semantic Web Mashup26

**Capitolo 3: World Pipes28**

3.1 Compatibilità e tecnologie utilizzate28

3.1.1 SWOWS, Fuseki, Apache30

3.1.2 HTML, CSS, Javascript30

3.1.3 Librerie31

3.2 Struttura del codice32

3.3 L’applicazione34

3.3.1 L’interfaccia34

3.3.2 I componenti37

3.3.3 Costruire una pipeline40

**Conclusione**43

**Bibliografia**44

**INDICE DELLE FIGURE**

[Figura 1 – Architettura del Web Semantico 6](#_Toc365213601)

[Figura 2 - Modello generico di una descrizione RDF 11](#_Toc365213602)

[Figura 3 - Rappresentazione grafica di uno statement RDF 13](#_Toc365213603)

[Figura 4 – Tabelle relazionale di due diversi database 20](#_Toc365213604)

[Figura 5 - Modello di dati relazionale per un unico grafo 23](#_Toc365213605)

[Figura 6 – Architettura Web 29](#_Toc365213606)

[Figura 7 – Struttura del Vettore dei Componenti 32](#_Toc365213607)

[Figura 8 - Interfaccia World Pipes 34](#_Toc365213608)

[Figura 9 – World Pipes user interface 35](#_Toc365213609)

[Figura 10 - Input Graph e Input Default Graph 36](#_Toc365213610)

[Figura 11 - Output Graph e Output Default Graph 37](#_Toc365213611)

[Figura 12 - Union Graph 37](#_Toc365213612)

[Figura 13 - Construct Graph 38](#_Toc365213613)

[Figura 14 - Updatable Graph 38](#_Toc365213614)

[Figura 15 - Dataset Graph 38](#_Toc365213615)

[Figura 16 - Esempio Pipeline 39](#_Toc365213616)

[Figura 17 - Esempio Pipeline composta di due soli componenti 40](#_Toc365213617)

[Figura 18 - Tabella delle Proprietà del componente Input Graph 40](#_Toc365213618)

[Figura 19 - Esempio Tabella degli Input del componente Construct Graph 41](#_Toc365213619)

**Introduzione**

Ci sono moltissimi dati che tutti noi usiamo ogni giorno, ma che non fanno parte del Web. Possiamo vedere la nostra situazione bancaria, possiamo vedere le nostre fotografie, e possiamo vedere i nostri appuntamenti nel calendario attraverso il Web. Ma è possibile vedere le nostre foto in un calendario per vedere che cosa stavamo facendo quando sono state scattate? Oppure vedere la nostra situazione bancaria giornaliera sulle righe di un calendario?

La risposta è no, perché il Web attuale non è un Web di dati. I dati, infatti, sono controllati da applicazioni e ogni applicazione li tiene per sé.

Il Web Semantico, al contrario, è un Web di dati - di date, titoli, numeri e qualsiasi altro tipo di dato possibile. L'insieme delle tecnologie del Web Semantico (RDF, OWL, SPARQL, ecc) fornisce un ambiente in cui un'applicazione può eseguire una query sui dati, trarre conclusioni utilizzando vocabolari, ecc. Tuttavia, per rendere il Web semantico una realtà, è importante avere i dati disponibili in un formato standard, raggiungibile e gestibile da strumenti del Web Semantico. Inoltre, non solo il Web Semantico ha bisogno di accedere ai dati, ma anche di conoscerne le relazioni che devono essere rese disponibili. Questa collezione d’insiemi di dati correlati sul Web prende il nome di *Linked Data*.

Il Web dei *Linked Data* sta diventando sempre più ricco in termini di dimensioni e di complessità, ma il suo uso è ancora limitato dalla mancanza di applicazioni che usufruiscono di questi dati.

Diversi strumenti, librerie e framework sono disponibili per manipolare e visualizzare i dati del Web Semantico. Ma le applicazioni RDF costruite con questi strumenti sono in genere scritte in un linguaggio di programmazione host, utilizzando i linguaggi del Web Semantico (SPARQL, OWL, ecc) solo in singole fasi del processo.

Si vuole realizzare un’applicazione che sfrutti le tecnologie del Web Semantico, in cui un'applicazione è progettata costruendo una pipeline composta di un insieme di operatori su grafi RDF. Abbiamo quindi sviluppato un'applicazione basata sul Web che permette di costruire visivamente le pipeline, basandosi completamente sugli standard del Web Semantico, inclusi *SPARQL Construct* per la trasformazione dei dati e *SPARQL Update* per lo stato di transizione. La principale novità risiede nel supporto all'interazione, attraverso la disponibilità di interfacce utente e flussi di eventi come ingressi alla pipeline. Le pipeline inoltre, possono essere utilizzate come componenti di altre pipeline per realizzare applicazioni complesse attraverso un approccio modulare.

Si è deciso di suddividere l’elaborato in tre capitoli. Nel primo capitolo si dipinge un quadro generale di come la semantica entra a far parte del Web e della tecnologia su cui il Web dei dati si basa. Nel capitolo due si descrivono le applicazioni che attualmente sfruttano il concetto di l*inked* *data* del Web Semantico. Infine, nel terzo capitolo, il cuore dell’elaborato, si descrive l’applicazione realizzata, le tecnologie utilizzate, il suo funzionamento e i risultati ottenuti.

# Capitolo 1

IL WEB SEMANTICO

La parola semantica implica di per sé significato e comprensione. Come tale, la differenza fondamentale tra le tecnologie del Web Semantico e le altre tecnologie legate ai dati (come ad esempio database relazionali o il World Wide Web stesso) sono che il Web Semantico si occupa del significato e non della struttura dei dati.

L’idea principale è quella di far evolvere il Web attuale, non di sostituirlo, spostando il concetto di Web dal “*machine readable*” al “*machine understandable*”, in altre parole rendere le informazioni processabili e comprensibili direttamente alle macchine, le quali diventano in grado di gestire autonomamente le informazioni. Il termine Web Semantico è associato all’idea di un Web nel quale agiscono *agenti* intelligenti, applicazioni in grado di comprendere il significato dei testi presenti sulla rete e perciò in grado di guidare l’utente direttamente verso l’informazione ricercata, oppure di sostituirsi a lui nello svolgimento di alcune operazioni.

Un agente dovrebbe essere in grado di:

1. Comprendere il significato dei testi presenti sulla rete.
2. Creare percorsi in base alle informazioni richieste dall’utente, guidandolo poi verso di esse.
3. Collegare logicamente elementi diversi dall’informazione richiesta.

Ovviamente, le macchine non sono certo capaci di comprendere quello che viene chiesto loro, ma dovranno essere in grado di riconoscere quelle informazioni importanti per gli utenti. Fondamentale sarà la struttura data all’informazione, che dovrà far sì che il computer sappia quali bit d’informazione rappresentano il significato di un documento on-line.

Il primo passo per mettere le macchine in grado di operare in maniera più efficiente sono dei documenti ben strutturati corredati di *metadati*. Essi sono informazioni, strutturate per essere elaborate da una macchina, che descrivono una risorsa Web.

In definitiva, la sfida del web semantico è quella di trovare un linguaggio adatto a esprimere sia i dati sia le regole per il ragionamento automatico sui dati.

## L’ARCHITETTURA DEL WEB SEMANTICO

Come detto nel capitolo precedente, le tecnologie semantiche rappresentano delle informazioni cui sono associati significati comprensibili alle macchine. Per processare i dati correttamente il Web Semantico è stato strutturato su più livelli.

L’architettura è rappresentata dal W3C sotto forma di una pila di livelli indipendenti, come possiamo vedere in Figura 1.

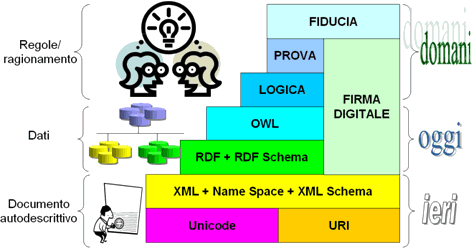


Figura 1 – Architettura del Web Semantico

Ogni livello è indipendente ma connesso a quello precedente e a quello successivo. Alla base della piramide c’è lo standard URI, utilizzato per la definizione univoca di risorse on line, come documenti, file e indirizzi Internet. Al livello immediatamente superiore troviamo il metalinguaggio XML, il quale ha un ruolo primario nell’architettura insieme con i Namespace e l’XML Schema. XML è un linguaggio che porta con sé alcune informazioni sulla semantica degli oggetti, mentre i Namespace identificati tramite gli URI garantiscono l’interoperabilità tra i dizionari di metadati.

Salendo ancora troviamo RDF, RDF Schema e OWL (per descrivere ontologie): essi sono i linguaggi, basati su XML, che permettono di rappresentare le risorse Web sotto forma di dati e di renderle disponibili ai livelli successivi. RDF serve solo per la definizione della semantica; per eseguire ricerche all’interno delle proprietà dei documenti si ha bisogno di un altro linguaggio, SPARQL (*Simple Protocol And RDF*). L’utilizzo congiunto di questi due linguaggi consente di realizzare delle *semantics pipes*, le quali sono semplicemente una sequenza memorizzata di query SPARQL su una o più fonti RDF.

I livelli appena descritti, indicati con il nome di livelli di base, sono già definiti e pubblicati con raccomandazioni ufficiali; non lo sono invece le tecnologie dei livelli più alti. Il livello della logica dovrebbe consentire alle macchine di acquisire nuove conoscenze da quelle già rappresentate, ad esempio con RDF e OWL. Il livello di prova sarà utilizzato per verificare che le nuove conoscenze acquisite con il livello logico siano dimostrabili. Infine, il livello della fiducia assicura che l’intero processo di rappresentazione, elaborazione e produzione di nuove informazioni sia affidabile, con l’uso di firme digitali, e possa essere utilizzato con sicurezza in applicazioni o servizi Web.

## URI (Uniform Resource Identifier)

L’architettura del Web Semantico si basa su una pila di tecnologie, a fondamento della quale è il concetto che in rete sia possibile assegnare un riferimento univoco – URI - a qualsiasi cosa. Anche una persona può essere rappresentata tramite l’indirizzo della sua pagina web o quello della sua casella di posta elettronica. Gli URI costituiscono la tecnologia di base ideale con la quale costruire un Web globale. Possiamo definire un URI per un qualsiasi oggetto, e qualsiasi cosa che ha un URI può essere considerata sul Web. Essi sono il fondamento del Web: mentre ogni parte del Web stesso può essere rimpiazzata, gli URI no.

Un URI non è un insieme di direttive che indicano al computer dove trovare un file specifico nel Web (sebbene lo faccia anche), ma è un nome per una risorsa, accessibile o meno attraverso Internet. Non dobbiamo assumere che un URI faccia qualcosa di più che fornire un identificatore per una risorsa.

Un URI può essere classificato come *URL* o come *URN*.

* Un *URL ([Uniform Resource Locator](http://it.wikipedia.org/wiki/Uniform_Resource_Locator" \o "Uniform Resource Locator))* è un URI che, oltre a identificare una risorsa, fornisce mezzi per agire o per ottenere una rappresentazione della risorsa descrivendo il suo meccanismo di accesso primario o la sua ubicazione (location) in una rete.
* Un *URN ([Uniform Resource Name](http://it.wikipedia.org/wiki/Uniform_Resource_Name" \o "Uniform Resource Name))* è un URI che identifica una risorsa mediante un *nome* in un particolare dominio di nomi ([namespace](http://it.wikipedia.org/wiki/Namespace" \o "Namespace)). Un URN può essere usato per parlare di una risorsa senza lasciar intendere la sua ubicazione o come ottenerne una rappresentazione.
  1. LINGUAGGI DEL WEB SEMANTICO
     1. Metadati

Nel navigare sul Web si seguono dei link, che portano a quella che formalmente è detta *risorsa*, identificata univocamente da un URI. Affinché le risorse siano elaborate da computer e software è necessario che siano accompagnate da informazioni riguardanti le risorse stesse, che ne specifichino il contesto e ne descrivano i contenuti. Tali informazioni prendono il nome di *metadati*. Le meta-informazioni permettono agli utenti di specificare sui documenti informazioni che non sono soltanto leggibili, ma anche interpretabili dalle applicazioni di rielaborazione e dai motori di ricerca.

Esistono tre modi per acquisire i metadati:

1. I metadati sono contenuti nella risorsa stessa, come per esempio nella sezione HEAD di un documento HTML;

2. Al momento del trasferimento della risorsa, le informazioni sono trasferite dal server al client (GET) o dal client al server (PUT o POST);

3. I metadati sono estratti da un’altra risorsa.

Pertanto, i metadati riguardanti un documento possono essere estratti dalla risorsa stessa, o da un’altra risorsa, o possono essere trasferiti con il documento.

Tim Berners-Lee definisce formalmente i metadati come “*Informazioni sulle risorse Web comprensibili ai calcolatori*”. Essi quindi svolgono una funzione fondamentale nei processi di reperimento e di elaborazione delle informazioni.

1.3.2 RDF

In ottica di Web Semantico, l’inserimento di metadati consentirebbe di dare un senso logico alle risorse online e offrirebbe alle macchine l’opportunità di ragionare sui dati e interpretarli. Essi apportano dunque un contributo fondamentale al miglioramento dell’accesso all’informazione. L’uso efficace dei metadati, tuttavia, richiede che siano stabilite delle convenzioni per la semantica, la sintassi e la struttura dei documenti e delle risorse online. Lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati – e dunque per introdurre la semantica all’interno dei documenti del Web, RDF (*Uniform Description Framework*) - è un linguaggio progettato con l’obiettivo di aggiungere meta-informazioni ai documenti Web e in generale alle risorse online. RDF permette l’aggiunta di semantica ai contenuti e ai documenti del Web, senza fare alcuna assunzione sulla sua struttura. La caratteristica principale di questo linguaggio è di consentire l’interoperabilità semantica tra applicazioni che si scambiano informazioni sul Web. Esso consente, dunque, la costruzione della struttura semantica all’interno dei documenti. RDF è costituito da due componenti:

* *Modello e sintassi RDF;*
* *RDF Schema*.

1.3.2.1 Modello e sintassi di RDF

RDF fornisce un modello per descrivere le risorse, definendole come un qualsiasi oggetto che sia identificabile univocamente mediante un Uniform Resource Identifier (URI).

Il modello dei dati di RDF è molto semplice, ed è basato su tre tipi di oggetti:

1. *Risorse* (resources): qualunque cosa descritta da un’espressione RDF è detta risorsa. Una risorsa può essere una pagina Web, o una sua parte, o un elemento XML all’interno del documento sorgente. Una risorsa può essere un’intera collezione di pagine, o anche un oggetto non direttamente accessibile via Web (per es. un libro, un dipinto, etc.). Le risorse sono sempre individuate da un URI.
2. *Proprietà* (properties): Una proprietà è un aspetto specifico, una caratteristica, un attributo, o una relazione utilizzata per descrivere una risorsa. Ogni proprietà ha un significato specifico, definisce i valori ammissibili, i tipi di risorse che può descrivere, e le sue relazioni con altre proprietà. Le proprietà associate alle risorse sono identificate da un nome, e assumono dei valori.
3. *Asserzioni* (statements): Una risorsa, con una proprietà distinta da un nome, e un valore della proprietà per la specifica risorsa, costituisce un’asserzione RDF. Un’asserzione è una tripla composta di un *soggetto* (risorsa), un *predicato* (proprietà) e un *oggetto* (valore). L’oggetto di un’asserzione (cioè il *property value*) può essere un’espressione oppure un’altra risorsa.

Graficamente, le relazioni tra risorse, proprietà e oggetto sono rappresentate mediante *grafi etichettati orientati*, in cui le risorse sono identificate come nodi (graficamente delle ellissi), le proprietà come archi orientati etichettati (graficamente delle frecce), e i valori corrispondenti a sequenze di caratteri sono rappresentati come rettangoli. La Figura 2 mostra una rappresentazione grafica di una generica descrizione RDF.

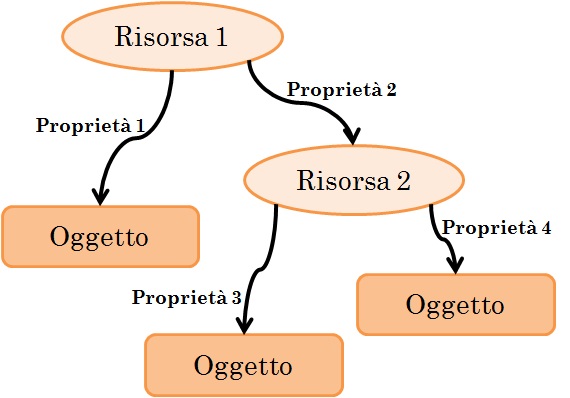


Figura 2 - Modello generico di una descrizione RDF

L’utilizzo di RDF può essere chiarito con qualche semplice esempio.

Consideriamo queste due espressioni:

1. “Simona Valentini è l’autore di DocumentoX”

2. “L’autore di DocumentoX è Simona Valentini”

Per un essere umano, le due espressioni sono del tutto equivalenti, poiché comunicano la stessa informazione; una macchina invece le vede come due espressioni diverse. RDF, mediante il suo semplice modello basato su risorsa, proprietà e valore intende fornire un metodo per esprimere la semantica con una codifica comprensibile alla macchina. Per dare un valore alle proprietà occorre che sia dichiarata la risorsa. Per prima cosa quindi, va dichiarata una risorsa che rappresenti il DocumentoX. Si ha pertanto la tripla:

*Resource* – http://www.tesi.it/ValentiniSimona/DocX

*Property* – Author

*Value* - Simona Valentini

L’asserzione dell’esempio sarebbe rappresentata come:

La risorsa http://www.tesi.it/ValentiniSimona/DocX has Author Simona Valentini

Graficamente è rappresentata come mostrato in Figura 3.

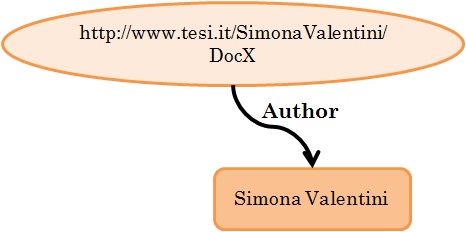


Figura 3 - Rappresentazione grafica di uno statement RDF

RDF fa una cosa semplice: rappresenta significati associando alcune risorse identificabili sul Web con dei valori. Permette di collegare una risorsa Web a un’altra e specificare che tipo di relazione hanno. Tali relazioni si possono creare non solo tra pagine Web, ma tra singole parti di esse, tra elementi (come gli indirizzi e-mail), tra una pagina e un’entità numerica (ad esempio una data), tra dati non raggiungibili direttamente dal web e anche tra concetti astratti. Tutte le relazioni formano una rete di significati, che può essere estratta da un software e rielaborato.

1.3.2.2 RDF schema

Il data model RDF da solo può soltanto stabilire relazioni fra le risorse ma non consente di definire un vocabolario esteso e condiviso di termini, di dedurre nuove proprietà dai concetti spiegati oppure di stabilire delle *regole* per la definizione delle risorse. A questo pensa RDF Schema che consente di definire dei vocabolari, quindi l’insieme delle proprietà semantiche individuate da una particolare comunità. RDF Schema permette di definire il significato, le caratteristiche e le relazioni di un insieme di proprietà, compresi eventuali vincoli sul dominio e sui valori delle singole proprietà. Inoltre, implementando il concetto di classe e sottoclasse, consente di definire gerarchie di classi, con il conseguente vantaggio che agenti intelligenti possono utilizzare queste relazioni per svolgere i loro compiti.

1.3.3 SPARQL

RDF serve a definire la semantica, ma da solo non può fare molto. Si devono poter effettuare delle ricerche all’interno delle proprietà di tutti i documenti indicizzati, si ha bisogno quindi di un linguaggio apposito. RDF diventa standard nel 1998, negli anni a seguire vari linguaggi hanno fornito soluzioni diverse per l’interrogazione di repository RDF. Dalle ceneri di diverse proposte, è emerso SPARQL (Simple Protocol and RDF Query Language), divenuto standard il 15 gennaio 2008.

SPARQL è un elemento chiave del web semantico e consente di estrarre informazioni dalle basi di conoscenza distribuite sul Web. Esso è un linguaggio d’interrogazione e un protocollo per l’accesso a RDF. Come un linguaggio di query, è *data-oriented* poiché interroga solo le informazioni contenute nei modelli, non vi è alcuna inferenza nel linguaggio della query stessa. SPARQL non fa altro che prendere la descrizione di ciò che l'applicazione desidera, nella forma di una query e restituisce tali informazioni, sotto forma di un insieme di associazioni o di un grafo RDF.

Il modello RDF è indipendente dalla particolare sintassi usata per esprimerlo ed esistono molti tools e librerie per convertire in modo semplice da una sintassi a un’altra. Un grafo RDF è rappresentato fisicamente mediante una serializzazione. Le principali serializzazioni adottabili per un grafo RDF sono:

* *XML*: RDF è serializzato in un file XML. È stato definito principalmente per il trasferimento macchina – macchina perché per l’uomo è di difficile lettura.
* *N-Triple*: si serializza un grafo come un insieme soggetto – predicato – oggetto (come visto nel capitolo precedente)
* *N3*: si serializza il grafo descrivendo, uno per volta, una risorsa e tutte le sue proprietà.

SPARQL adotta la sintassi *Turtle*, un’estensione di N-Triple, scelta estremamente sintetica e intuitiva a XML/RDF. Si considerino le seguenti triple RDF:

@prefix cd: < <http://example.org/cd> >

@prefix: < <http://example.org/esempio> >

:Buona domenica cd:autore “Antonello Venditti”.

:Amo cd:autore “Renato Zero”.

:Zerolandia cd:autore “Renato Zero”.

:Zerolandia cd:anno 1978.

Le asserzioni sono espresse in sequenze soggetto – predicato – oggetto e delimitate da un punto. @prefix introduce prefissi e namespace; i due punti senza prefisso (seconda riga) definiscono il namespace di default. Gli URI sono inclusi tra parentesi angolari. I numeri e le stringhe tra virgolette prendono il nome di letterali.

L‘idea fondamentale sottostante alla definizione di SPARQL è l‘uso di *graph pattern*. Il pattern contiene simboli (?x, ?y…) non legati, attraverso il loro collegamento si selezionano sottografi del grafo RDF. Se una tale selezione esiste, allora la query restituisce le risorse che si sono legate ai simboli. Le query si basano in particolare sul costrutto *triple pattern*, che riprende la configurazione a triple delle asserzioni RDF, fornendo così un modello flessibile per la ricerca di corrispondenze. Il punto focale è che soggetto, predicato e oggetto possono essere delle variabili, come nell’esempio seguente:

?titolo cd:autore ?autore.

Le variabili sono precedute da un punto interrogativo. In questo triple pattern le variabili sono ?titolo (soggetto) e ?autore (oggetto). Esse rappresentano in un certo senso le incognite dell’interrogazione, mentre cd:autore si comporta come una costante. Le triple RDF che trovano riscontro nel modello, assoceranno i propri termini alle variabili corrispondenti. Il seguente esempio mostra una semplice query di selezione SPARQL:

**PREFIX** cd: < http://exsample.org/cd/ >

**SELECT** ?titolo ?autore ?anno

**FROM** < http://cd.com/listacd.ttl >

**WHERE** {?titolo cd:autore ?autore.

?titolo cd:anno ?anno.

}

PREFIX dichiara prefissi e namespace. SELECT definisce le variabili di ricerca da prendere in considerazione nel risultato (nell’esempio sono titolo, autore, anno). FROM specifica il set di dati su cui dovrà operare la query (si suppone che le triple siano immagazzinate presso l’indirizzo fittizio http://cd.com/listacd.ttl). Infine, la clausola WHERE definisce il criterio di selezione specificando tra parentesi graffe uno o più triple patterns separati da un punto. Applicando la query al set di triple di cui sopra, si ottiene il seguente risultato:

|  |  |  |
| --- | --- | --- |
| **Titolo** | **Autore** | **Anno** |
| Zerolandia | Renato Zero | 1978 |

Il risultato è dato in forma di tabella. Le righe rappresentano i singoli risultati, le intestazioni di cella rappresentano le variabili definite nella clausola SELECT; le celle, i termini associati alle variabili.

Questo linguaggio consente di interrogare modelli RDF sfruttando le relazioni semantiche rappresentate nel modello. È stato scelto da W3C per la sua completezza, espressività e semplicità d’uso, e per l’esecuzione di query remote come Web Service o richieste/risposte http.

1.3.4 Vocabolari e Ontologie

Un file RDF può, da solo, stabilire relazioni fra risorse ma non consente, se non tramite RDF Schema, di definire un vocabolario esteso e condiviso di termini, di dedurre nuove proprietà dai concetti spiegati oppure di stabilire delle *regole* per la definizione delle risorse. A questo pensano i *vocabolari.*

I *vocabolari* definiscono i concetti e le relazioni usati per descrivere e rappresentare l'area d’interesse, sono adoperati per classificare i termini che possono essere utilizzati in una particolare applicazione, caratterizzare possibili relazioni, e definire possibili limitazioni sull'uso di questi termini. In pratica, i vocabolari possono essere molto complessi (con diverse migliaia di termini) o molto semplici (descrivono solo uno o due concetti).

Non c'è una chiara divisione tra quello che viene definito come *vocabolario* e quella che viene definita come *ontologia*. La tendenza è di utilizzare il termine *ontologia* per indicare una raccolta di termini in cui sono definite relazioni formali tra essi, mentre *vocabolario* viene utilizzato quando non è necessario tale formalismo. I vocabolari sono gli elementi di base per le tecniche di *inference* sul Web Semantico. L’inferenza nel Web Semantico è uno degli strumenti utilizzati per migliorare la qualità dell'integrazione dei dati sul Web, scoprendo nuove relazioni e analizzando automaticamente il contenuto dei dati. Tecniche di base di *inference* sono importanti anche a scoprire eventuali incongruenze nei dati.

La fonte di tali informazioni in più può essere definita, come detto inizialmente, tramite i *vocabolari* o tramite un insieme di regole. Entrambi questi approcci s’ispirano alle tecniche di rappresentazione della conoscenza. In generale, le *ontologie* si concentrano sui metodi di classificazione, mettendo l'accento sulla definizione di *classi* e *sottoclassi*, su come risorse individuali possono essere associate a tali classi, e su come caratterizzare le relazioni tra le *classi* e le loro *istanze*. Le regole, invece, si concentrano sulla definizione di un meccanismo generale per scoprire e generare nuovi rapporti oltre a quelli esistenti.

Un esempio può aiutare a comprendere meglio l’importanza dei vocabolari. Un libraio può decidere di integrare i dati provenienti da diversi editori. I dati possono essere importati in un modello comune RDF, ad esempio, utilizzando convertitori nelle banche dati degli editori. Tuttavia, un database potrebbe utilizzare il termine *autore*, mentre l'altro può usare il termine *creatore*. Per rendere completa l’integrazione, la nuova definizione deve essere aggiunta ai dati RDF, indicando che, il rapporto descritto come *autore* è lo stesso di *creatore*. Questa informazione supplementare è un *vocabolario* (o un'*ontologia*), anche se estremamente semplice.

In un caso più complesso l'applicazione può avere bisogno di un’*ontologia* più dettagliata. Questo può includere la descrizione formale di come gli autori devono essere identificati in modo univoco, come i termini utilizzati in quest’applicazione possano fare riferimento ad altri insiemi di dati sul Web (ad esempio Wikipedia), come il termine *autore* (o *creatore*) può essere correlato a termini come *redattore*, ecc.

Quindi, il ruolo dei vocabolari sul Web Semantico è di aiutare l'integrazione dei dati, quando, per esempio, possono esistere ambiguità sui termini utilizzati nei diversi insiemi di dati, o quando una maggiore conoscenza può portare alla scoperta di nuove relazioni, utilizzando formalismi standard, e sfruttando la potenza dei *linked data*.

* 1. I LINKED DATA

I *Linked Data* sono il cuore del Web Semantico: l'integrazione su larga scala, e il ragionamento sui dati nel web. Quasi tutte le applicazioni del Web Semantico si basano essenzialmente sull’accessibilità e l'integrazione dei dati collegati a vario livello di complessità.

Il modello dietro il Web potrebbe essere grossolanamente riassunto come un sistema per pubblicare i documenti rappresentati in modo standard (HTML), che contiene collegamenti ad altri documenti e accessibili tramite Internet utilizzando protocolli standard (TCP/IP e HTTP). Il risultato potrebbe essere visto come un file system mondiale di documenti interconnessi che gli esseri umani possono leggere, scambiare e discutere.

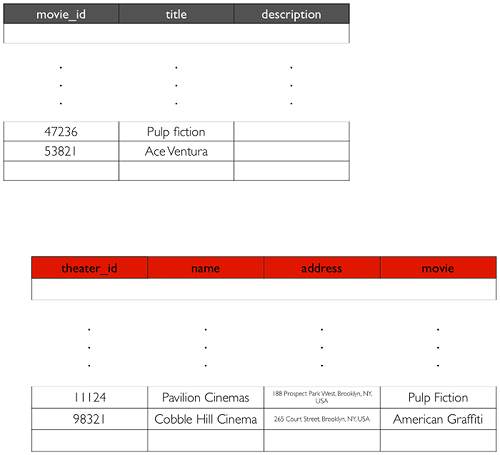


Figura 4 – Tabelle relazionale di due diversi database

Proviamo attraverso un esempio pratico a illustrare l’importanza dei *linked data*.

La Figura 4 mostra due tabelle relazionali di due diversi database. Il primo contiene dati sui film, con titoli, attori e nome del regista. Il secondo è una raccolta di sale in una città, dove questi film sono in programmazione. È subito evidente che esiste una relazione implicita tra le file di queste due tabelle.

Immaginiamo si voglia pubblicare queste informazioni in modo che possano essere facilmente utilizzate da altri siti Web o programmi informatici ad esempio, un’applicazione potrebbe interrogare automaticamente il nostro sito Web e qualsiasi altro sito che ha le informazioni sulla programmazione dei film al fine di mostrare una visione completa in un unico luogo.

Qui è dove il Web paga il suo più grave debito con la tecnologia. Non esiste un modo standard per rendere tali tabelle disponibili sul Web e facilmente accessibili da un'applicazione.

Ma è qui che la potenza dei *Linked Data* entra in gioco.

L'obiettivo dei *Linked Data,* è di pubblicare dati strutturati in modo tale da poter essere facilmente utilizzati e combinati con altri dati collegati. Possiamo dire che, i *Linked Data,* in un certo senso, rappresentano il Web Semantico realizzato attraverso quattro principi di base:

1. Utilizzare gli URI come nomi per ogni oggetto.
2. Utilizzare gli URI/HTTP in modo che gli utenti possano cercare tali nomi.
3. Quando qualcuno cerca un URI, occorre fornire informazioni utili, utilizzando gli standard, come RDF e SPARQL.
4. Includere link ad altri URI in modo che possano scoprire più cose.

Applichiamo le regole appena elencate. Immaginiamo di voler pubblicare le tabelle illustrate nella Figura 4:

1. Invece di utilizzare specifiche chiavi d’identificazione dei database che li associa a un insieme di URI, ogni identificatore deve essere associato a un singolo URI. Ad esempio, ciascuna fila delle due tabelle deve poter essere identificata mediante un suo URI univoco.
2. L’URI deve essere deferenziabile, ossia deve essere accessibile via HTTP come una pagina web. Questo è un aspetto chiave dei *Linked Data*: ogni singola riga delle nostre tabelle è identificata univocamente e reperibile ovunque sul web.
3. Ora occorre modellare i dati con RDF.
4. Una volta che tutte le righe della nostra tabella sono state identificate in modo univoco, deferenziate tramite HTTP, e descritte con RDF, l'ultimo passo è fornire collegamenti con le righe di altre tabelle. L'obiettivo principale è quello di rendere espliciti quei link che prima erano impliciti.

Nel nostro esempio, i film sono ora collegati alle sale in cui sono in programmazione.

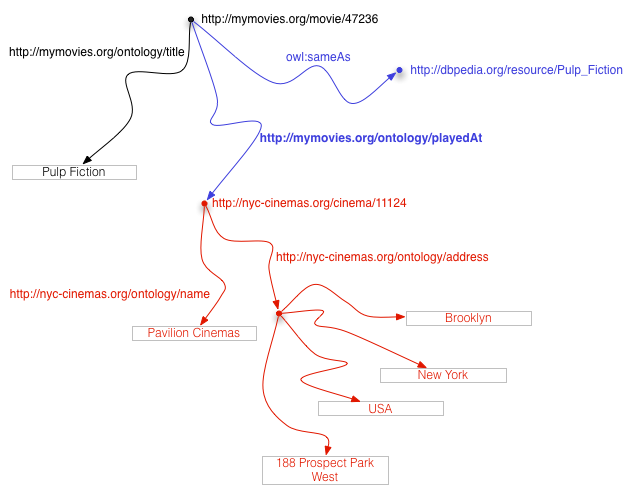


Figura 5 - Modello di dati relazionale per un unico grafo

Una volta che le nostre tabelle sono state così pubblicate, grazie alle regole dei *Linked Data* gli utenti in tutto il Web possono iniziare a farvi riferimento e a consumare i dati nel nostro file.

Con i *Linked Data* non si espongono soltanto i dati utilizzando tecnologie web. Né sono semplicemente un modo elegante per risolvere i problemi d’interoperabilità. Essi sono una parte fondamentale per la costruzione di un Web di dati.

Immaginate centinaia di differenti set di dati pubblicati sul Web in base ai principi dei *Linked Data*: migliaia e migliaia d’identificatori diversi possono farvi affidamento per utilizzare dati su libri, film, attori, città, o qualsiasi cosa siamo grado di immaginare. In poche parole, questi insiemi di dati formano un gigantesco database che si potrebbe incorporare nelle applicazioni e farvi riferimento ogni volta che ce ne sia necessità.

**Capitolo 2**

STRUMENTI DI AGGREGAZIONE DEI DATI PRESENTI NEL WEB

* 1. APPLICAZIONI PER IL WEB MASHUP

Lo sviluppo delle moderne applicazioni web punta verso la realizzazione di *Web mashup*, applicazioni web ottenute dalla combinazione di contenuti e servizi disponibili sul web sotto forma di API (*Application Programming Interface*). Un’API è un programma software che facilita l'interazione con altri programmi software. Permette ai programmatori di interagire con un'applicazione utilizzando un insieme di funzioni richiamabili. Lo scopo è ottenere un'[astrazione](http://it.wikipedia.org/wiki/Astrazione_(informatica)) a più alto livello, di solito tra l'[hardware](http://it.wikipedia.org/wiki/Hardware) e il programmatore o tra [software](http://it.wikipedia.org/wiki/Software) a basso livello e quello ad alto livello semplificando così il lavoro di [programmazione](http://it.wikipedia.org/wiki/Programmazione_(informatica)). Le API permettono infatti di evitare ai [programmatori](http://it.wikipedia.org/wiki/Programmatore) di riscrivere ogni volta tutte le [funzioni](http://it.wikipedia.org/wiki/Funzione_(informatica)) necessarie al programma dal nulla, ovvero dal basso livello.

Il passo fondamentale per lo sviluppo dei *mashup* è la produzione di *servizi aperti*, pubblicati sul web e quindi facilmente accessibili e riutilizzabili. Tali servizi, possono essere API remote, servizi basati sullo scambio di messaggi (per esempio, servizi web), API basate sull’interrogazione programmatica di codice, o anche feed RSS/Atom, o contenuti estratti dai siti web più disparati. Tali servizi possono inoltre essere dotati di un’interfaccia utente propria, che diventa parte integrante del mashup finale. I mashup più popolari integrano API pubbliche che permettono a utenti e sviluppatori generici di invocare servizi accessibili tramite Web. Esistono diverse proposte di strumenti per lo sviluppo dei mashup, che in genere mettono a disposizione ambienti di sviluppo visuali per poter facilmente comporre i mashup da una selezione di servizi. Yahoo, uno dei maggiori produttori dell’economia odierna su Internet ha realizzato un’applicazione che offre questo tipo di servizio, chiamata, *Yahoo Pipes*. È un servizio gratuito online che consente di creare mashup di dati utilizzando un’editor visuale. È possibile utilizzare *Yahoo Pipes* per eseguire i propri progetti web, o pubblicare e condividere i propri servizi web senza mai dover scrivere una riga di codice. Esso si concentra sull’integrazione di dati (RSS o Atom feed) e offre un linguaggio tramite cui comporre le pipes. Le pipes si creano per via grafica, inserendo vari blocchi (funzioni) in uno schema, e collegando gli ingressi dell’uno con le uscite dell’altro.

* 1. APPLICAZIONI PER IL SEMANTIC WEB MASHUP

Il Semantic Web, come ampiamente detto nei capitoli precedenti, è un Web di dati. Pertanto per realizzare il *Semantic Web Mushap,* ci si deve basare essenzialmente sull’accessibilità e l'integrazione dei *Linked Data*. Uno strumento open source che permette di lavorare con i dati del Web Semantico è *Deri Pipes*,un’applicazione ispirata da *Yahoo pipes*, che ha come obiettivo quello di trasformare il web in un ambiente accessibile a tutti, rendendo l’aggregazione delle fonti di dati semplice anche per gli utenti meno esperti.

*Deri Pipes* ha un motore di esecuzione, responsabile del trattamento dei dati definiti in un linguaggio XML ad hoc. Le pipe eseguibili con sintassi XML sono memorizzate in una banca dati. Quando sono invocate, il motore di esecuzione recupera i dati da remoto in buffer di elaborazione e quindi esegue l'albero degli operatori.

L'editor drag and drop consente agli utenti di visualizzare e costruire le loro pipeline, ispezionando i dati in ogni fase del processo, inoltre qualsiasi pipeline esistente può essere copiata e integrata nella pipeline modificata dall'utente. Le pipeline realizzate sono poi eseguite nel contesto di un *aplication batch* o un'applicazione web (all'interno di un sito Web o servizio Web), in risposta a richieste GET o POST.

*Deri Pipes* non può però essere utilizzato direttamente per applicazioni interattive basate su eventi.

**Capitolo 3**

WORLD PIPES

È stato fin qui spiegato cos’è il Web Semantico, come viene realizzato, quali sono i linguaggi di programmazione da esso utilizzati, e quali strumenti attualmente ne sfruttano le potenzialità. Ora, si vogliono mettere in pratica le conoscenze fin qui acquisite, costruendo un’applicazione interattiva, basata su eventi, che lavora con i dati del Web Semantico.

*World Pipes* è una piattaforma che fornisce un’interfaccia web-based e gli strumenti per la creazione visuale di applicazioni interattive, specificate tramite un linguaggio dichiarativo basato su RDF. Il sistema include un’interfaccia interattiva, grazie alla quale le pipeline sono costruite per via grafica, strumenti di costruzione drag & drop e la possibilità di visualizzare il codice creato in formato RDF.

* 1. COMPATIBILITA’ E TECNOLOGIE UTILIZZATE

L’idea iniziale era di rendere l’applicazione compatibile con tutti i principali browser attuali, ma si è dovuta abbandonare questa idea perché in Interner Explore molte delle librerie utilizzate non sono supportate. Quindi l’applicazione può essere utilizzata solamente da utenti che abbiano una versione recente dei browser Firefox e Google Chrome.

* + 1. SWOWS, FUSEKI, APACHE

Il linguaggio utilizzato per definire le pipeline è già in uso in SWOWS, un sistema prototipale che attualmente funziona come applicazione Java stand-alone per lo sviluppo di applicazioni web basate su tecnologie per il Web semantico (quindi tutto viene trattato come RDF). Esso è basato su un linguaggio RDF-based. Le pipeline sono costruite utilizzando alcuni operatori di base che compongono singoli grafi RDF o settati per creare nuovi grafi o serie di dati. L'implementazione è basata su Java e permette di creare applicazioni costruite su eventi per diversi tipi d’interfaccia.

Per dotare l’applicazione della capacità di salvare e caricare pipeline si ricorre a una tecnologia lato server; in particolare per salvare le pipeline sotto forma di documenti RDF si utilizza il server *Fuseki*, che comunica direttamente con lo SWOWS Engine il quale passa i dati della pipeline in formato RDF alla SWOWS Web App che ha il compito di mostrare il risultato della pipeline realizzata in World Pipes.

*Fuseki* è uno SPARQL server che fornisce REST-style SPARQL HTTP Update, SPARQL Query, e SPARQL Update, utilizzando il protocollo SPARQL su http. Riceve il codice RDF della pipeline creata, ne verifica la correttezza e lo invia allo SWOWS Engine. Attualmente la comunicazione si svolge in un’unica direzione, da Fuseki a SWOWS.

Uno dei principali problemi riscontrati durante la realizzazione dell’applicazione è stato il *cross-domain*, che impedisce a script scaricati dalla rete di accedere, tramite qualsiasi tipo di richiesta HTTP, a risorse che si trovano su server diversi rispetto a quello iniziale che ha inviato lo script. Per risolvere tale problema si fa uso dei servizi offerti da *Apache*. In particolare si utilizzano *Apache2 Web Server* e *Apache Tomcat 7*; nel primo viene eseguita l’applicazione insieme all’*Apache Proxy Module* che si occupa di passare i dati al server esterno, *Fuseki*. Mentre in *Apache Tomcat 7* nella directory *deploy* viene inserito il file WAR (abbreviazione di *Web Archive*) che gira sulla piattaforma SWOWS sopra descritta.

La Figura 6 mostra l’architettura web appena delineata:

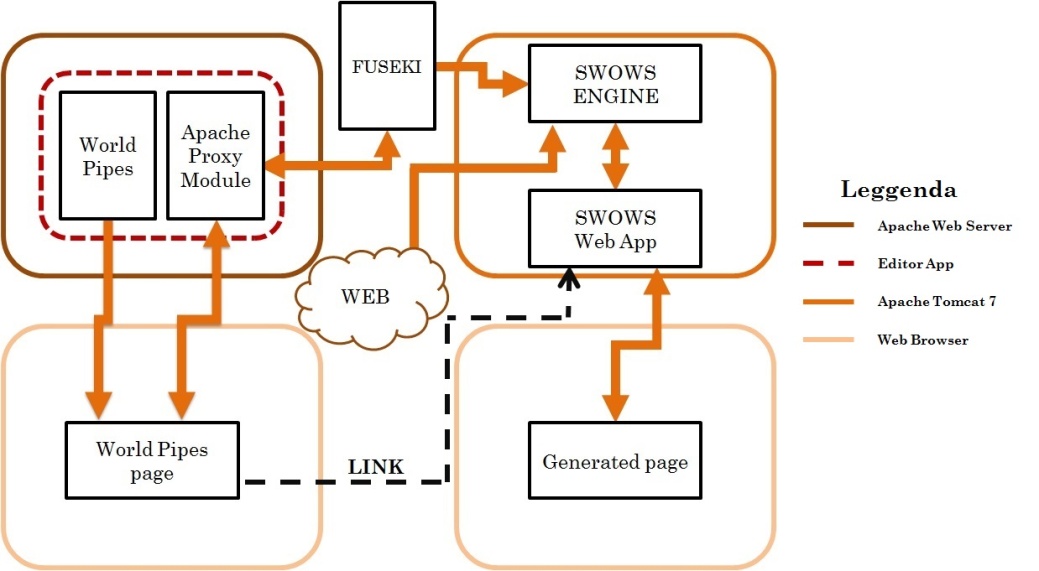


Figura 6 – Architettura Web

* + 1. HTML, CSS, JAVASCRIPT

La realizzazione di un’interfaccia semplice e intuitiva ha portato alla decisione di suddividere il codice in tre livelli, che corrispondono ai tre tipi di codice utilizzati, ossia HTML, CSS e JavaScript.

Tali livelli si sono affrontati dal basso verso l'alto:

1. Si comincia producendo il contenuto in formato HTML.
2. Si continua aggiungendo un livello di presentazione tramite i fogli di stile CSS.
3. Infine si utilizza Javascript per introdurre un altro livello d’interattività e di comportamento dinamico.
   * 1. LIBRERIE

Ogni script conta su una piccola libreria di metodi contenuta nel file *Core.js*. Questa libreria racchiude tutte le sue funzioni in un oggetto javascript, trasformandole in metodi di tale oggetto.

I metodi più utilizzati sono:

* *Core.addEventListener(target,type,listener):* questo metodo assegna la funzione listener come un listener di eventi di tipo type all'oggetto target.
* *Core.start(runnable):* questo metodo esegue il metodo init dell'oggetto script (runnable) non appena termina il caricamento del documento.

Per poter creare i diagrammi delle pipeline in *World Pipes*, si è utilizzata la libreria *JsPlumb*. Essa offre la possibilità di collegare visivamente elementi DOM presenti nell’area di lavoro, utilizzando quelle che sono chiamate *connessioni*. Un collegamento appare come una linea - il percorso che segue dipende dal tipo di connessione in uso. jsPlumb fornisce due tipi di attacchi - una linea retta, e una curva di Bezier; si è scelto di utilizzare quest’ultima in *World Pipes*.

Ogni connessione è stabilita tra due *endpoint*. Un endpoint è il punto sull’elemento cui il collegamento si unisce. *JsPlumb* fornisce tre diversi tipi di endpoint: un punto, un rettangolo e un’immagine.

Infine ogni endpoint è associato a un *anchor*. Essi definiscono dove un endpoint deve essere posizionato su un elemento. Non hanno alcuna rappresentazione visiva, ma sono semplicemente utilizzati per il posizionamento.

L'applicazione generata da una pipeline come detto è anche web-based e risponde agli eventi dell’interfaccia utente, grazie alla gestione degli eventi in javascript e all’interazione client/server realizzata tramite AJAX (*Asynchronous JavaScript And XML*). AJAX è una tecnica di sviluppo [software](http://it.wikipedia.org/wiki/Software) per la realizzazione di [applicazioni web](http://it.wikipedia.org/wiki/Applicazione_web) interattive. Ci consente di comunicare con il server e di inviargli i dati. Questa libreria ci permette di chiedere al browser di gestire piccole porzioni di documento invece che grossi frammenti d’informazioni; invece di una pagina si può richiedere di inviare al server anche un unico paragrafo.

* 1. STRUTTURA DEL CODICE

La struttura dell’interfaccia dell’applicazione è definita nel file *WorldPipesNew.html*.

Il suo stile (o la sua presentazione) viene completamente descritto nei file CSS di seguito elencati:

* *actionListenerStyle.css*
* *drag&dropstyle.css*
* *guidastyle.css*
* *pagestyle.css*
* *tablestyle.css*

Infine, per rendere l’interfaccia interattiva si utilizzano i seguenti script:

* *actionListenerImg.js*: In questo script si gestiscono gli eventi scaturiti dall'azione del mouse, come il click del mouse sui pulsanti New, Save, Play, Load pipeline e Update Pipeline e l’evento *drag & drop*.
* *actionListenerResizeX.js*: Gestisce il ridimensionamento verticale dell'area di lavoro.
* *actionListenerResizeY.js*: Gestisce il ridimensionamento orizzontale dell'area di lavoro.
* *activelink.js*: Rende interattivi i tab presenti nell’interfaccia.
* *areahelp.js*: in questo script viene gestito l’evento click su un componente.
* *Component.js*: Ogni componente trascinato nell’area di lavoro è contenuto nel vettore dei Componenti. Tale vettore è un array di classi. In questo script si gestiscono tutti i metodi get e set per lavorare su tale vettore.

Il vettore dei Componenti è strutturato come descritto in Figura 7.

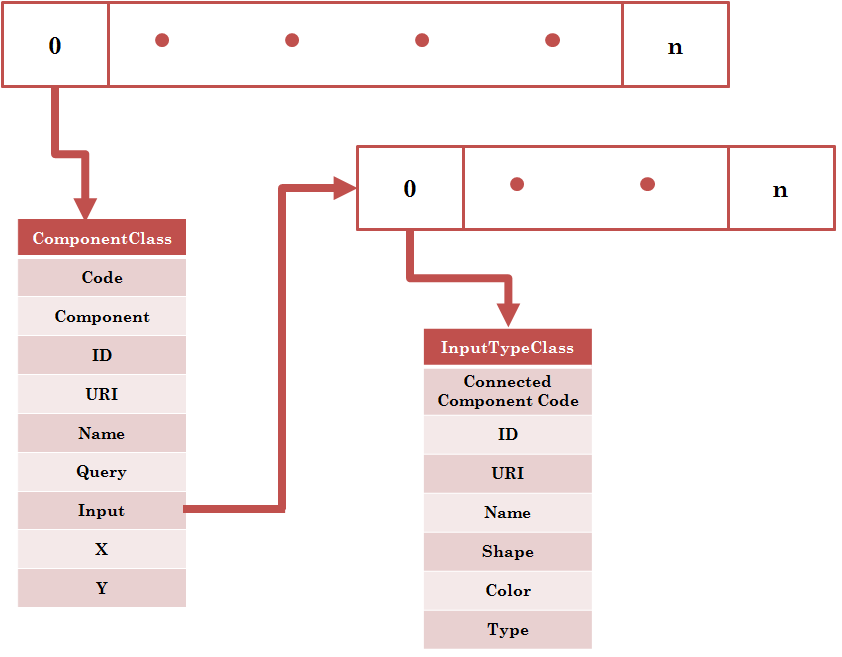


Figura 7 – Struttura del Vettore dei Componenti

Nella classe *ComponentClass* troviamo il campo *Code* contenente il codice del componente e il campo *Component* contiene il nome del grafo che il componente rappresenta. Nei campi *ID*, *URI* e *Name* troviamo rispettivamente l’Id, l’Uri e il nome del componente. Il campo *Query* contiene la query scritta dall’utente. Il campo *Input* è un array di classi contenente la lista dei componenti sorgente connessi al componente destinazione. In particolare, per ogni sorgente la classe *InputTypeClass* contiene: nel campo *ConnectedComponentCode* il codice del componente connesso, l’Id, l’Uri e il Nome nei rispettivi campi e infine la forma, campo *Shape*, il colore, campo *Colore* dell’endpoint d’input e, nel campo *Tapy* il tipo d’input, si è scelto di fissare il tipo di ogni input a *From/From Named*. Infine, di ogni componente viene mantenuta la posizione che esso occupa nell’area di lavoro, attraverso i campi *X* ed *Y* della classe *ComponentClass*.

* *Endpoint.js*: In questo script si gestiscono i metodi della libreria *jsPlumb* per gestire i collegamenti tra i componenti
* *Form.js*: In questo script si realizzano i form che permettono all'utente di inserire le proprietà di ogni componente.
* *InputType.js*: Questo script permette di inserire l'oggetto *inputType* nel campo *Input* della classe *ComponentClass*.
* *Sourcecode.js*: In questo script viene realizzato il codice RDF della pipeline creata dall'utente. Inoltre si gestisce anche l'invio del codice stesso al server *Fuseki*.
  1. L’APPLICAZIONE
     1. L’INTERFACCIA

*World pipes* è uno strumento di authoring JavaScript che consente di creare e modificare le pipeline in un'interfaccia grafica intuitiva suddivisa per aree di lavoro, come mostrato dalla Figura 8.

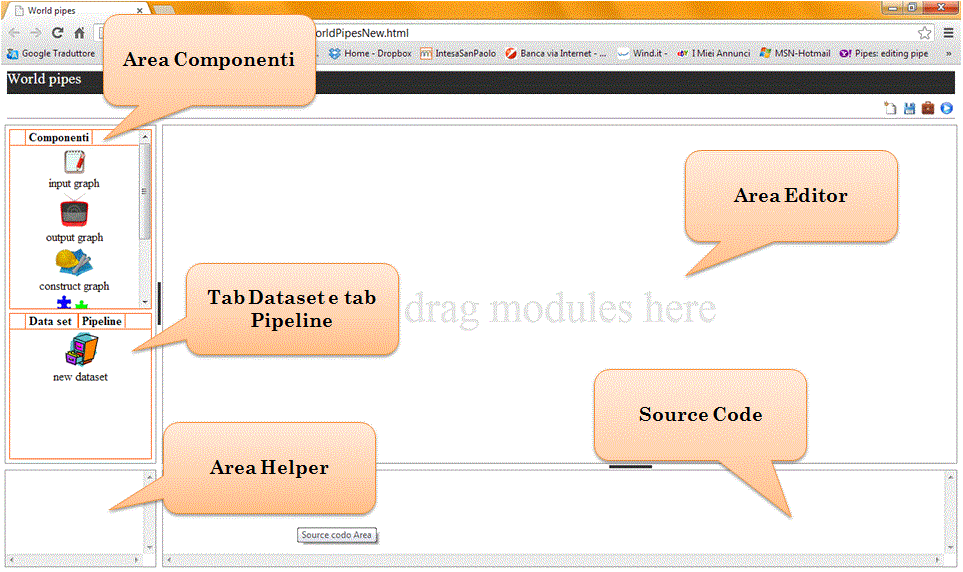


Figura 8 - Interfaccia World Pipes

* *Pannello dei comandi*: posto in alto a destra, contiene alcuni pulsanti per le operazioni relative all’intera pipeline, come ad esempio salvarla, eseguirla o ricaricarla.
* *Area componenti e Tab Dataset e Pipeline*:poste sulla sinistra. Nell’Area Componenti si ha la lista dei componenti da utilizzare per creare le nostre pipeline. Nel Tab Dataset troviamo la lista dei Dataset. Nel Tab Pipeline si ha l’elenco delle Pipeline realizzate.
* *Area editor*: è l’area di lavoro in cui è possibile sviluppare le pipeline.
* *Area helper*: cliccando su un componente, in quest’area ne compare una breve descrizione, e un link che permette di accedere al manuale d’uso per avere una spiegazione più approfondita sull’utilizzo del componente.
* *Source code*: man mano che la pipeline prende forma in quest’area, compare il codice RDF corrispondente.

La Figura 9 mostra in dettaglio l’ambiente di sviluppo.

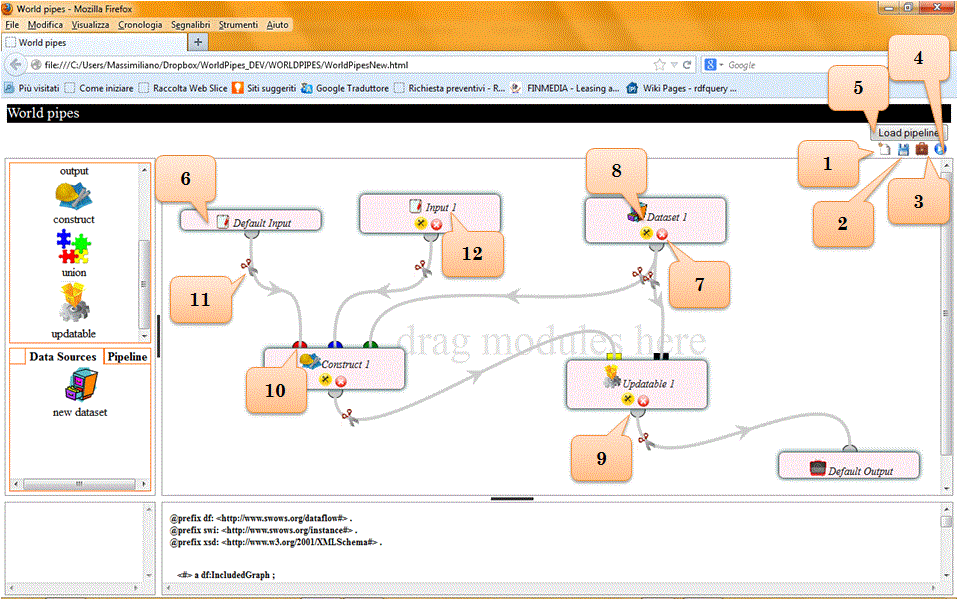


Figura 9 – World Pipes user interface

1. *Pulsante New*: ricarica la pagina, così da poter costruire una nuova pipeline.
2. *Pulsante Save*: permette di salvare la pipeline creata.
3. *Pulsante Properties*: attraverso una form permette di attribuire un nome e una descrizione della pipeline.
4. *Pulsante play*: cliccando su questo pulsante sarà possibile visualizzare l’output della pipeline.
5. *Pulsante Load*: permette di ricaricare le pipeline realizzate dall’utente.
6. *Component*: componente della pipeline.
7. *Pulsante delete*: permette di eliminare un componente dalla pipeline
8. *Pulsante property*: permette di inserire i dati riguardanti il componente, Nome, ID, URI e Query (ove richiesto).
9. *Endpoint output*: Ogni componente ha delle porte in uscita, poste in basso, da dove partono i collegamenti verso un altro nodo.
   1. *Endpoint input*: Alcuni componenti hanno porte in entrata, poste in alto, dove sono connessi i collegamenti provenienti da un altro componente.
   2. *Pulsante scissors*: permette di recidere un collegamento.
   3. *Nome del componente*: a ogni componente viene assegnato un nome, tale nome è visualizzato sul componente stesso.
      1. I COMPONENTI

Tutti i componenti sono identificati da un Name (visualizzato nell'interfaccia) e un ID (utilizzato per identificare il componente).

I componenti disponibili sono:

Input graph: Ogni pipeline per dare un risultato ha bisogno di uno o più input. Tale componente ha solamente una porta di uscita e può essere collegato con qualsiasi componente che possieda porte in entrata. Per funzionare correttamente occorre impostargli un ID e un Nome (essi non necessariamente devono differire) e un URI.

Si ha inoltre un input graph di default già presente nell’editor che non possiede URI e non può essere eliminato.



Figura 10 - Input Graph e Input Default Graph

Output graph: Una pipeline non può produrre risultati senza questo componente. Ha un'unica porta di entrata e può essere collegato con qualsiasi componente. I parametri da impostare sono l’URI e il nome.

Si ha inoltre un output graph di default già presente nell’editor che non possiede URI e non può essere eliminato.



Figura 11 - Output Graph e Output Default Graph

Union graph: È utilizzato per unire triple RDF provenienti da più grafi RDF in input. Possiede un’unica porta di output e una serie di porte d’input. Le proprietà da impostare sono l’URI e il nome. Alle porte di entrata è possibile connettere un input graph, construct graph, updatable graph ed anche un altro union graph.



Figura 12 - Union Graph

Construct graph: È utilizzato per creare triple con query SPARQL-CONSTRUCT su determinate fonti di RDF. L'uscita di quest’operatore è il risultato di una query SPARQL-CONSTRUCT eseguita su uno o più grafi RDF in input. La query è eseguita considerando tutti i grafi in input associati ai nomi che hanno nella pipeline e un grafo di default definito dall'unione di tutti i grafi in input. Si ha un’unica porta di uscita, mentre le porte di entrata vengono create dinamicamente attraverso la *Tabella degli input*, che permette di definire Nome e Id dell’input, forma e colore delle porte di entrata.

****

Figura 13 - Construct Graph

Updatable graph: È un operatore speciale che definisce una porzione dello stato dell'applicazione. Quest’operatore definisce un grafo RDF nel seguente modo: quando uno qualunque dei grafi in input si modifica viene eseguita l'operazione di update definita nel campo query, che può essere costituita anche da più query di update (separate da “;”). Si ha un’unica porta di uscita, mentre le porte di entrata vengono create dinamicamente attraverso la *Tabella degli input*, che permette di definire Nome e Id dell’input, forma e colore delle porte di entrata. Ognuna di queste porte d’input rappresenta un *trigger*.



Figura 14 - Updatable Graph

Dataset graph: Questo componente rappresenta un file RDF. Si ha una sola porta di uscita e può essere collegato con qualsiasi componente che possieda porte in entrata.



Figura 15 - Dataset Graph

Pipeline graph: rappresenta una pipeline esistente che può essere utilizzata come componente delle pipeline corrente. Non si hanno immagini di questo componente perché attualmente nell’applicazione questa parte non è trattata.

* + 1. COSTRUIRE UNA PIPELINE

Le pipeline si costruiscono trascinando i componenti nell’area Editor e assemblandoli insieme con il mouse.

Per rendere effettivo il lavoro, è necessario collegare i componenti insieme. È possibile eseguire quest’operazione, facendo clic sulla porta di uscita (in basso al centro) di un operatore alla porta di entrata (in alto) del componente che si desidera alimentare. Quando si crea una connessione, le porte di input sono evidenziate per indicare quali collegamenti sono permessi. Una pipeline si può ritenere completa quando tutti i nodi a lei necessari sono stati aggiunti.

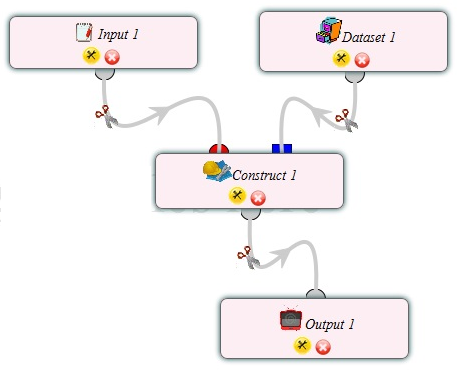


Figura 16 - Esempio Pipeline

Tutti gli elementi devono essere collegati almeno a un altro componente. La pipeline più semplice sarà composta di due nodi (ad esempio input -> output).

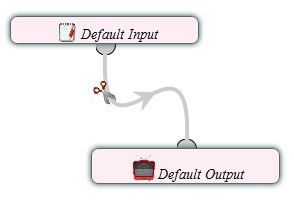


Figura 17 - Esempio Pipeline composta di due soli componenti

Oltre hai componenti è possibile trascinare nell’area *Editor* anche i Dataset presenti nel Tab DataSet e le pipeline presenti nel Tab Pipeline. Una pipeline è un subDataflow che trascinato nell’area *Editor* si comporta come un componente regolare. Come già detto, questa parte non viene trattata.

Ogni operatore, come descritto nel paragrafo 3.3.2, possiede un pulsante Property che permette di impostarne le proprietà, come mostrato in Figura 18.

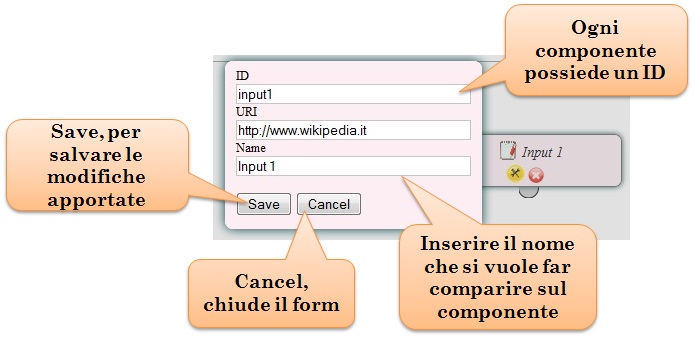


Figura 18 - Tabella delle Proprietà del componente Input Graph

Nella tabella delle proprietà occorre inserire ID, Name e se richiesto l’URI. Questo vale per tutti i componenti. Per i componenti Construct e Updatable si ha qualche campo in più da riempire, nella tabella delle proprietà troviamo in più il campo Query e la *Tabella degli input*, una tabella dinamica in cui l’utente ha la possibilità di aggiungere le porte di input dando loro un Nome e un Id e impostandone colore e forma, come mostrato in Figura 19.

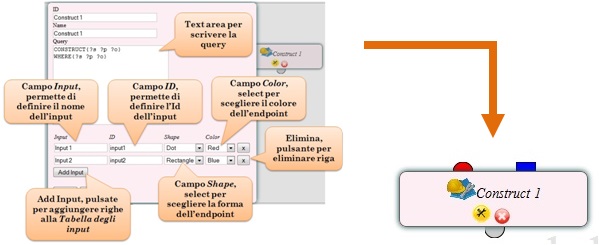


Figura 19 - Esempio Tabella degli Input del componente Construct Graph

Dare un nome ai singoli input rende la query indipendente da come viene utilizzata nella pipeline e, non obbliga l’utente a scrivere la query prima di collegare gli input.

**Conclusioni**

Dopo mesi di duro lavoro l’applicazione ha finalmente preso forma. È stato realizzato un prodotto funzionante che permette di costruire pipeline che specificano trasformazioni di grafi RDF per costruire applicazioni di visualizzazione dei dati.

È stato necessario uno studio più o meno approfondito di cosa sia il Web Semantico e dei linguaggi RDF e SPARQL in quanto base di partenza per la realizzazione dell’applicazione. Lo studio approfondito della sintassi HTML, CSS, Javascript e Ajax è stato invece necessario per poter concretamente realizzare l’interfaccia interattiva e la comunicazione client/server; in pratica, l’intera applicazione.

I risultati conseguiti sono piuttosto soddisfacenti, anche se il prodotto realizzato risulta essere incompleto e migliorabile sotto molti aspetti. L’usabilità potrebbe essere aumentata semplificando ulteriormente l’interfaccia e nuove funzionalità potrebbero essere implementate per aumentare le potenzialità, per permetterne l'utilizzo da parte di utenti meno esperti e per renderlo compatibile con tutti i browser. Al momento, i più adatti per utilizzare il sistema sono Firefox e Gooogle Chrome.

Anche se l’applicazione proposta probabilmente non sarà presentata per utilizzi reali, il lavoro svolto può offrire spunti interessanti a chi vuole progettare sistemi simili, pensati per il Web Semantico e per garantire un elevato livello di interoperabilità.

**Bibliografia**

Paolo Bottoni, Miguel Ceriani: *A user interface to built interactive visualization for the semantic web*

Sandro Hawake, Ivan Herman, Phil Archer, Eric Prud’hommeaux: *Semantic Web Activity*, www.w3.org/2001/sw

*Linked data*, www.w3.org/standards/semanticweb/data

*Links*, www.w3.org/TR/html401/struct/links.html

Paolo Cercavolo: *Che cos’è e a cosa serve il Web SemanticoI,* 20 marzo 2006 <http://www.html.it/articoli/cos-e-a-cosa-serve-il-web-semantico-1>

Antonella Dorati, Stefania Costantini: *Approcci al Web Semantico*, <http://www.websemantico.org/articoli/approcciwebsemantico.php#uno>

Irene Celino: *La semantica e il web dei dati; quando uno più uno non fa due*, 2 aprile 2009, <http://www.slideshare.net/iricelino/la-semantica-e-il-web-dei-dati>

Prof. Stefano Epifani: *Dispensa n. 5 – Il web semantico*, <http://www.slideshare.net/stefanoepifani/dispensa-5-il-web-semantico>

Massimo Martinelli, *Massimo Martinelli: il web semantico*, <http://www.tecnoteca.it/interviste/martinelli>

Frandesco Caccavella: *Alla scoperta di … Il Web Semantico …*, Maggio 2009, <http://www.pcprofessionale.it/stappro61/uploads/2009/10/pc-professionale-maggio-2009-il-web-semantico.pdf>

Armando Stellato: *Linguaggi di interrogazione per il Web Semantico*, Giugno 2010, http://art.uniroma2.it/teaching/gdc/part\_II/stuff/Semantic%20Web\_-Linguaggi%20di%20Interrogazione.pdf

Wikipedia: *Uniform Resource Identifire*, <http://it.wikipedia.org/wiki/Uniform_Resource_Identifier>

Ernesto Erra: *SPARQL: Simple Protocol and RDF Query Language*, A.A. 2011/2012

Marco Regando: *Traduzione automatica di interrogazioni Continuous-SPARQL su stream RDF in EPL e SPARQL*, A.A. 2009/2010

Oreste Signore: *RDF per la rappresentazione della conoscenza*, <http://www.w3c.it/papers/RDF.pdf>

Tutorial Apache Jena: *SPARQL Tutorial*, <http://jena.apache.org/tutorials/sparql.html>

Prof.ssa Linda Giuva, Prof. Maurizio Caminito: *L’informatica per gli Archivi e le Biblioteche*, A.A. 2012-2013, <http://www.slideshare.net/iricelino/la-semantica-e-il-web-dei-dati>

W3C: *Vocabolaries*, <http://www.w3.org/standards/semanticweb/ontology>

W3C: *Inference*, <http://www.w3.org/standards/semanticweb/inference>

Christian Mastrodonato: *Web Semantico, ontologie ed RDF/XML*, 24 febbraio 2009, <http://www.html.it/articoli/web-semantico-ontologie-ed-rdfxml-1/>

Matteo Busanello: *Semantic Web. Esplorazione/visualizzazione di ontologie*, ottobre 2009,http://www2.mokabyte.it/cms/article.run?articleId=NL6-SQG-9XK-788\_7f000001\_18359738\_4a1ff785

Linda Giuva, Maurizio Caminito: *Web Semantico: ontologie e RDF*, A.A. 2012 – 2013 [http://www.slideshare.net](http://www.slideshare.net/iricelino/la-semantica-e-il-web-dei-dati)

W3C Recommendation: *OWL Web Ontology Language Overview*, <http://www.w3.org/TR/2004/REC-owl-features-20040210/>

Yahoo Pipes: *Pipes Overview*, <http://pipes.yahoo.com/pipes/docs?doc=overview>

Danh Le Phuoc, Axel Polleres, Giovanni Tummarello, Christian Morbidoni:

*DERI Pipes: visual tool for wiring Web data sources*, <http://handbook5.com/d/deri-pipes-visual-tool-for-wiring-web-data-sources-w10438.html>

Florian Daniel, Maristella Matera: [*Quando*](http://www.miniscript.it/articoli/54/yahoo_pipes_aggregare_e_riordinare_pi_feed.html) *l’utente guida l’innovazione – Il web mashup*

W3C: *Linked Data*, <http://www.w3.org/standards/semanticweb/data>

Davide Palmisano: *Introduction to Linked Data*, Semantic University, <http://www.cambridgesemantics.com/semantic-university/introduction-to-linked-data>

Mauro Guerrini, Tiziana Possemato: *Linked Data: un nuovo alfabeto del web semanticoI,* Aprile 2012,Biblioteche oggi

ApacheJana: *Fuseki serving RDF data over HTTP*, http://jena.apache.org/documentation/serving data

SWOWS: *What?*, http://www.swows.org

Kevin Yank, Cameron Adams: *JavaScript*, Apogeo