ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria con sede a Cesena

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

(d.m. 270)

Interfaccia web per un sistema di condivisione semantica dell'informazione: studio e implementazione

Elaborata nel corso di: Calcolatori Elettronici

TESI DI LAUREA DI:

Tampellini Luca

RELATORE:

Chiar.mo Prof. Luca Roffia

CORRELATORE:

Ing. Francesco Morandi

ANNO ACCADEMICO 2011-2012

SESSIONE II

PAROLE CHIAVE

Web Semantico

RDF

SPARQL

OWL

Javascript

Php

SMART-M3

INDICE

Sommario

SOMMARIO

Questa tesi progettuale nasce per integrare gli sforzi attuali sullo sviluppo del web semantico.

Dopo essere venuto a conoscenza della tematica del nuovo web e dei relativi progetti in cantiere dell'Università di Bologna, sviluppati all'interno di progetti europei, di cui il progetto SOFIA rappresenta il capostipite, ho accettato questa tematica come occasione per lo sviluppo della tesi.

La piattaforma di riferimento sulla quale è stato svolto il presente lavoro è SMART-M3. Questa piattaforma mette a disposizione uno spazio condiviso di informazioni, rappresentate e accessibili secondo le tecnologie del web semantico.

In questo scenario, nasce la necessità di disporre di un'interfaccia web capace di interagire con la piattaforma ed in grado di la complessità intrinseca dei dati semantici, e per averne un completo controllo; ricerche precedenti a questo proposito hanno dato come frutto una libreria PHP che mi è stata consegnata come strumento per lo sviluppo dell'interfaccia.

La tesi si è articolata in 3 fasi principali: una fase iniziale di documentazione sull'argomento, eseguita principalmente sul libro *"A developer's guide to the semantic web"* di Liyang Yu e sulla tesi *"Ontologie per il web semantico: un'analisi comparativa."* di Indrit Beqiri; una seconda fase, quella principale, di sviluppo del progetto informatico; una terza fase, infine, di sviluppo di questo elaborato di tesi, da considerarsi come la trattazione di tutto il percorso soprascritto, dall'inizio alla fine, secondo l'ordine cronologico in cui si svolto l'intero processo della tesi.

In particolare, la tesi è stata organizzata nei seguenti capitoli:

Cap.1 Introduzione

Questo capitolo contiene una breve panoramica su cosa sia il Web semantico: che cos'è, da dove è nato, perchè è nato e cosa ci si aspetta che diventi.
Vengono quindi citati per sottocapitoli le tecnologie e i linguaggi su cui esso si basa.

Cap.2 Scenario di riferimento

Viene descritto in quale scenario di partenza si è inserito lo sviluppo della tesi.

In particolare viene data una rappresentazione schematica di cosa sia SMART-M3 e delle librerie PHP già sviluppate, per concludere con una sintesi degli strumenti esterni che sono stati utilizzati per sviluppare il progetto.

Cap. 3 Analisi delle specifiche

Si descrive cosa si voleva ottenere dal progetto, esponendo ogni specifica per singolo sottocapitolo.

Cap.4 Progettazione

Lo sviluppo del progetto viene descritto con una trattazione organizzata per passi: ogni passo corrisponde ad una fase temporale del progetto, e i passi si susseguono secondo l'ordine cronologico.

I primi passi corrispondono alla fase di documentazione e alle fasi di test del materiale di partenza; ogni passo consecutivo, quindi, corrisponde al rilascio di una nuova versione dell'applicativo, seguendo un processo di ingegnerizzazione per prototipi.

Cap.5 Futuro del progetto

Si prospetta in che modo il progetto può essere integrato nello scenario iniziale trattato nel cap.2, facendo una particolare menzione ai GRDDL.

Cap.6 Conclusioni

Viene data una stima dell'efficenza del sistema di ingegnerizzazione del progetto.

Si descrivono le performance dell'applicazione sviluppata, pregi e mancanze.

L'ultimo sottocapitolo riguarda i ringraziamenti.

Cap.7 Appendici

Si fornisce in appendice la documentazione necessaria all'utilizzo dell'applicativo.

Cap.8 Bibliografia & Riferimenti

Il primo sottocapitolo fornisce la bibliografia dei testi cartacei utilizzati.

Il secondo sottocapitolo definisce i riferimenti alle fonti elettroniche delle ulteriori conoscenze utilizzate.

CAP.1 INTRODUZIONE

1.1 Il Web Semantico

Per capire cosa vogliamo dire parlando di “Web Semantico” bisogna prima avere ben chiaro cosa sia la semantica e cosa sia il suo complementare, ossia la sintassi. Quando formuliamo una frase creiamo una sintassi, ossia l'aspetto della frase, e vi associamo un ben determinato significato: il significato degli elementi di una frase è di fatto la semantica.

Dall'altra parte, se riceviamo una frase come -per esempio- “Ciao Papà”, il nostro cervello comprende automaticamente, per un meccanismo naturale sviluppato in qualche lontano e indefinito punto della nostra evoluzione, che siamo stati salutati da nostro figlio.

Elevando la cosa, possiamo definire:

- Sintassi: come si manifesta un insieme di elementi di un linguaggio (fenomeno linguistico)
- Semantica: che cosa significa un insieme di elementi di un linguaggio (noumeno linguistico)

Il Web, come lo conosciamo oggi, è un insieme di dati raccolti in pagine che vengono lette dalla macchina e interpretate secondo un approccio “sintattico”.

Quando apriamo una richiesta http, il nostro browser processa un insieme di costrutti linguistici che determinano il formato con cui i dati vengono passati all'utente, la natura di questi dati non è però d'interesse per la macchina. Di fatto essa applica una trasformazione lineare al contenuto sulla base della sua sintassi, e ne da il contenuto all'utente.

In un certo senso il browser si comporta come una macchina combinatoria... non ha cioè una “memoria”.

Ritornando al nostro paradigma iniziale di frase umana, possiamo disegnare, idealmente, cosa vorrebbe dire trasformare sintatticamente sul web le loro due frasi (Fig. 1.1).

Fig. 1.1 : Trasformatore sintattico

Di fatto la trasformazione sintattica si basa semplicemente nel ripetere all'utente il messaggio iniziale in una forma diversa.

Sempre a livello ideologico, possiamo pensare che se il punto intermedio fosse capace di comprendere che cosa le due persone si stanno dicendo, agirebbe come una intelligenza vera e propria, capace di fare proprio (memorizzarlo) il contenuto del messaggio ed eventualmente modificare il messaggio stesso (Fig.1.2).

Fig. 1.2: Trasformatore semantico... un'IA?

Possiamo quindi pensare che applicazioni capaci di lavorare con informazioni semantizzate, agiscano in modo intelligente? Questo argomento verrà elaborato al sottocapitolo 1.4.

Analizziamo ora la storia del Web semantico.

Il concetto di Web semantico nasce da una visione di Tim Berners-Lee[1], l'inventore del Web stesso, espressa in queste frase:

1 http://www.w3.org/People/Berners-Lee/

“I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize.”

-Tradotto-

“Io ho un sogno per il Web [in cui i computers] diventano capaci di analizzare tutti i dati sul Web- contenuti, collegamenti e passaggio di informazioni fra esseri umani e computers. Un “Web Semantico”, che renderebbe questo possibile, deve ancora emergere, ma quando lo farà, i meccanismi quotidiani di commercio, burocrazia e le nostre vite giornaliere saranno gestite da macchine parlanti con altre macchine. Gli “agenti intelligenti” che le persone hanno meditato per anni finalmente prenderanno vita.”

E' da notare come per mezzo del nuovo approccio semantico, l'usanza di racchiudere i dati in pagine diventa obsoleta (o comunque non più necessaria): tutto il web viene riconcepito come un enorme database di dati, si parla quindi di ***www3*** !

Per dare un'idea, si potrebbe ipotizzare che un motore di ricerca non ritorni più pagine come risultati, ma dati estratti intelligentemente da varie pagine internet e poi ricombinati in un documento completamente nuovo.

E' importante capire che con il nuovo ***www3*** si intende informatizzare qualsiasi tipo di risorsa che possa essere concepita dall'essere umano, non solo risorse presenti fisicamente sulla rete (vedi sottocapitolo 1.2).

Le tecnologie per la definizione dei dati semantici si chiamano RDF, RDFS e OWL (vedi sottocapitolo 1.3) e sono estensioni di un caratteristica delle vecchie pagine web: i cosiddetti “metadati”.

Una pagina html contiene normalmente un insieme di descrizioni racchiuse nei tag <meta></meta>, che permettono di avere informazioni molto generali del tipo: titolo, autore, parole chiave, tipo di

contenuto, etc. Questi tag html, decisamente sotto-utilizzati nel vecchio www, sono un esempio primitivo di semantizzazione!

1.2 Elementi di fondo

L'elemento di fondo del web semantico è la risorsa. Ogni risorsa viene identificata da un URI, per esteso: “Uniform Resource Identifier”, e può essere direttamente accessibile tramite internet, come una pagina web, oppure non esserlo: si può per esempio associare un URI ad una persona fisica o -peggio- ad un'idea.

I vecchi url con cui si usavano le pagine sono, di fatto, un sotto-insieme degli URI. Per questo un URL è sempre un URI, ma non tutti gli URI sono URL... questo da garanzie di integrabilità col vecchio sistema.

Nell'appendice B si da una descrizione più approfondita della differenza fra URI e URL, per adesso ci limitiamo a darne la rappresentazione schematica da noi usata: nella fattispecie, gli URI del progetto seguono una codifica chiamata “HASH”, e appaiono come segue:

URL+ **#** + **fragment identifier**

Ad esempio: http://www.mydomain.it#LucaTampellini potrebbe essere l'URI che mi identifica.

Data la difficile leggibilità per l'essere umano, si usa abbreviare rappresentare gli URI in URN, che appaiono come segue:

dominio dei nomi + **:** + **nome della risorsa**

In cui il dominio dei nomi, meglio detto “qName” o “namespace”, viene associato all'URL, mentre il nome della risorsa è il nome che essa assume nel dominio, ossia il fragment identifier.

Ad esempio: myDomain:LucaTampellini sarebbe l'URN rispettivo dell'esempio di URI soprascritto.

Come verrà spiegato più tardi (si veda sotto-capitolo 1.3), un insieme di nomi e le relazioni tra essi viene anche definito “ontologia”.

Le nuove tecniche di assegnamento dei significati (o “semantizzazione”) si basano su triple di informazioni chiamate “statement”.

Lo statement si compone di una forma sintattica (vedi Fig. 1.3) simile a quella di una comune frase umana: soggetto, predicato ed oggetto [nota: proprio per questo si è ricorso al paradigma di una frase nell'introduzione]. Si osservi bene questo evento curioso: le informazioni semantiche vengono

descritte per mezzo di forme sintattiche, concetto da tenere bene in considerazione per comprendere quanto spiegato al sottocapitolo 1.4 .

Fig. 1.3: La forma sintattica dello statement.

Di fatto con uno statement, si crea un collegamento fra il soggetto e l'oggetto in cui il predicato definisce in che modo il soggetto interagisce con l'oggetto.

Per fare un esempio: myDomain:LucaTampellini Foaf:knows hisDomain:LucaRoffia

Oltre agli URI, gli elementi dello statement possono assumere altri 2 tipi di valori:

- letterali, ossia semplici stringhe di caratteri o numeri. Possono essere tipizzati (integer, float, string, etc.) oppure non avere tipo.
- blank node, ossia “elementi anonimi”, assimilabili ai costrutti linguistici usati dagli umani quando si vuole fare riferimento ad entità che non si conosce: la parola che rispecchia meglio questo concetto è “qualcosa”; tipicamente i blank node vengono usati per creare statement più complessi.

Per esempio:

myDomain:LucaTampellini Foaf:knows _b:000000001

_b:000000001 myDomain:canBe hisDomain:LucaRoffia

_b:000000001 myDomain:canBe “Francesco Morandi”

Significa “io conosco qualcuno, quel qualcuno può essere Luca Roffia o Francesco Morandi.” o per essere più brevi “io conosco sia Luca Roffia sia Francesco Morandi”; _b:000000001 è il blank node e “Francesco Morandi” è un valore literal.

Riassiumiamo:

1. Il Soggetto: rappresenta la risorsa iniziale (start node) del collegamento, deve essere un URI o un blank node.
2. Il Predicato: rappresenta in che modo la risorsa iniziale e collegata a quella finale, deve essere un URI.
3. L' Oggetto: rappresenta la risorsa finale (end node) del collegamento, può essere URI, literal o blank node.

Nel caso in cui l'oggetto sia un letterale, si usa definire la tripla anche con questi nomi: risorsa (soggetto), proprietà (predicato), valore (oggetto).

Un gruppo di statement viene comunemente chiamato “Graph”.

1.3 Tecnologie per la semantizzazione: RDF, OWL, SPARQL

I disegnatori della W3C che hanno schematizzato gli statement, li hanno concepiti per essere il blocco di fondo attorno a cui hanno sviluppato la tecnologia cardine del web Semantico: l'RDF.

RDF significa Resource Description Framework (Framework per la descrizione delle risorse) ed è stato disegnato dal World Wide Web Consortium[2] (W3C). I suoi disegnatori volevano che avesse queste caratteristiche:

1. RDF è un linguaggio per rappresentare risorse nel World Wide Web.
2. RDF è un Framework.
3. RDF deve essere general-purpose.
4. RDF si basa sugli statement, rappresentati per mezzo di vocaboli.
5. RDF, a sua volta, è la tecnologia di base su cui si sviluppano altre tecnologie per effettuare una semantizzazione di più alto livello.
6. RDF è per il Web semantico ciò che HTML era per il Web sintattico.
7. RDF deve essere comprensibile alle macchine. Una pagina html rimane identica per l'utente che possieda definizioni RDF o meno.

2 http://www.w3c.org

RDF è quindi un Framework per esprimere gli statement: mette a disposizione un insieme di URI (o vocaboli) da utilizzare indirettamente per strutturare lo statement, indicando alla macchina “qui sta il soggetto, qui il predicato e qui l'oggetto” oppure da usare direttamente come soggetti, predicati od oggetti dello statement stesso.

Abbiamo ora bisogna di capire come scrivere gli RDF direttamente dentro a documenti del web, trasformandolo in un linguaggio di programmazione. Per fare questo dobbiamo quindi “serializzare” gli RDF, ossia dargli un tipo di annotazione.

Esistono vari tipi di annotazioni, ma i principali sono:

1. RDF/XML
2. Notation3
3. Turtle
4. N-Triples

Allo scopo del progetto, verrà trattata solo la prima che è di fatto la più utilizzata e che permette di serializzare gli RDF come espressioni XML.[3]

Un file RDF/XML deve apparire più o meno in questo modo:

```
<?xml version=”1.0”?>                                          → tag di apertura xml
<rdf:RDF                                                       → inizio Graph
        xmlns:rdf= “http://www.w3.org/1999/02/22-rdf-syntax-ns#”   → dichiarazione namespace
        xmlns:foaf= “http://xmlns.com/foaf/0.1#”>              → dichiarazione namespace
<rdf:Description                                               → inizio statement
        rdf:about= “http://www.myDomain.it#LucaTampellini”>    → soggetto
    <foaf: knows                                               → predicato
        rdf:resource= “http://www.hisDomain.it#LucaRoffia”>    → oggetto
</rdf:Description>                                             → fine statement
</rdf:RDF>                                                     → fine Graph
```

3 http://www.w3.org/1999/02/22-rdf-syntax-ns

(legenda: il simbolo → commenta l'elemento xml)

Il progetto da noi utilizzato mostrerà all'utente -comunque- solo statement non serializzati.

Per le stesse ragioni mi limito a citare tecnologie alternative che permettono di implementare gli RDF direttamente dentro l'html, senza ricorrere a file esterni: microformati ed RDFa, che possono poi essere estratti per mezzo delle GRDDL.

D'interesse per il progetto è invece l'RDFS [4] (o RDF Schema), che permette di estendere il vocabolario di base della sintassi RDF.

Si estende RDF avendo la possibilità di definire:

1. Classi: nuovi tipi di risorsa. Possono ereditare in Sotto-Classi.
2. Proprietà: nuovi tipi di predicati, modellabili secondo i loro possibili valori.
3. Datatypes: nuovi tipi di valori letterali.

Inoltre usando le Utilities si possono estendere le classi in orizzontale, garantendo legami di somiglianza, e fare retro-ereditarietà.

L'RDFS, di fatto, mette insieme delle tipologie (schemi, appunto).

Queste tipologie sono estremamente potenti, ma hanno come limite che sono vincolate all'uso degli RDF... cosa fare se si volessero dei vocaboli “liberi” da usare per definire le nostre risorse?

A questo scopo entra in gioco il concetto di OWL, Web Ontolgy Language.

Nel sotto-capitolo 1.2 si era già accennato cosa fosse il concetto di “ontologia”, ora verrà spiegato in questi termini:

> *“Un'ontologia definisce formalmente un* insieme *di termini che vengono utilizzati per definire un dominio di concetti... Un area di sapere è caratterizzata per mezzo di un'ontologia.”*
>
> -OWL Web Ontology Language Use Cases and Requirements[5]

4 http://www.w3.org/2000/01/rdf-schema

5 http://www.w3.org/TR/webont-req/

Un esempio di ontologia può essere un set di termini utile a dare descrizioni di creazioni artistiche: tale ontologia esiste già e si chiama “Dublin Core”[6]

Queste ontologie possono quindi essere definite per mezzo dello standard OWL.

Le direttive OWL possono essere serializzate direttamente in XML[7] in modo parzialmente o completamente indipendente dalle parti definite con RDF/RDFS: in effetti OWL non si basa sulle triple o statement che usa RDF, ma sugli “axiom” (assiomi) che possono avere da due a un numero variabile di elementi, sono quindi potenzialmente più semplici, espressivi e dinamici di uno statement. Ognuno dei singoli elementi di un axiom viene definito “entity” (entità).

Ai fini del progetto, gli assiomi OWL vengono comunque combinati con statement RDF e, per rispettare le direttive definite al capitolo 3, vengono quindi assimilati alle sue triple.

Si sono date, in definitiva, le descrizioni delle principali tecnologie per la costruzione di dati semantici. E' ora opportuno presentare una tecnologia sviluppata per l'interrogazione di questi dati: tale è lo SPARQL. (vedi Fig.: 1.4)

Fig.1.4: il significato di SPARQL (acronimo ricorsivo)

Con la semantizzazione dei dati e la possibilità di estrarli dalle singole pagine, tutto il web viene riconcepito come un enorme database su cui si possono effettuare interrogazioni in SPARQL, similimente a ciò che fa SQL.[8]

6 http://purl.org/dc/elements/1.1/

7 http://www.w3.org/2002/07/owl

8 http://www.w3.org/2009/sparql/wiki/Main_Page

SPARQL permette di fare query a insiemi di dati RDF. Per farlo dobbiamo usare un protocollo apposito, che ci fa lavorare in remoto.

Nonostante il www3, stesso, virtualmente diventi un database, rimane il bisogno di utilizzare un punto fisico di passaggio: dobbiamo, cioè, salvare prima i dati su un database fisico. Per questo si usano i cosiddetti RDF data store; ne esistono varie realizzazioni: nel nostro progetto viene utilizzato “Redland”.[9]

Gli RDF data store sono database dedicati esclusivamente a dati RDF e sono per questo modellati appositamente: i record sono esclusivamente triple Soggetto-Predicato-Oggetto come gli statement RDF.

A questi data store si associano degli “Endpoint” che permettono a SPARQL di interfacciarsi col database.

Il nostro progetto, però, usa un'interfaccia più complessa e potente di un semplice “Endpoint”, chiamata SMART-M3 (si veda il capitolo 2).

Le query messe a disposizione da SPARQL sono di 4 tipi: SELECT, ASK, DESCRIBE e CONSTRUCT.

Ognuna di queste query, invia una descrizione delle informazioni richieste sfruttando due tipi diversi di schemi: “triple pattern” e “graph pattern”. Quando si richiede un triple pattern, si richiede di fatto una singola tripla RDF: soggetto predicato ed oggetto della tripla possono essere stringhe costanti (se noti a priori) oppure essere sosituiti da variabili; per ogni costante, l'Endpoint cercherà le corrispondenze con i record attualmente presenti nel data store; per ogni valore variabile, verrà effettuato il “binding” (“aggancio”) a tutti i valori possibili; l'intersezione degli insiemi di valori ottenuti, determina i record che sono effettivamente restituiti. I graph pattern sono invece richieste più complesse, ottenute come combinazioni di più triple patterns; la selezione dei record effettivamente restituiti rispetta la stessa metodologia usata per i triple pattern, sebbene valori prefissati e variabili sono molti di più.

9 http://librdf.org

1.4 Intelligenze artificiali sul Web

E' possibile pensare che una applicazione possa capire veramente il significato di un gruppo di dati e reagire con intelligenza?

Prima di trovare la riposta a questo, bisogna interrogarsi sulla natura della comprensione umana, dato che stiamo cercando -di fatto- di emularla. La digressione filosofica è quindi necessaria.

Nel primo sottocapitolo si sono associati i termini “fenomeno” e “noumeno” - rispettivamente - alla sintassi e alla semantica. Questi due termini vennero coniati da Immanuel Kant, il filosofo, nel suo studio sulla natura della ragione[10]. Stando a quanto affermato da Kant, forma e significato sono due concetti rigidamente separati: il fenomeno è comprensibile per mezzo dei sensi, mentre il noumeno è conoscibile solo per mezzo della ragione pura. Questo dualismo viene chiamata “dualismo gnoseologico” e non nasce con Kant, ma affonda la proprie radici fino in Platone: egli scrisse un mito piuttosto famoso chiamato “Mito della Caverna”[11] che studia proprio la differenza fra mondo sensibile e mondo del significato. (Fig.1.5)

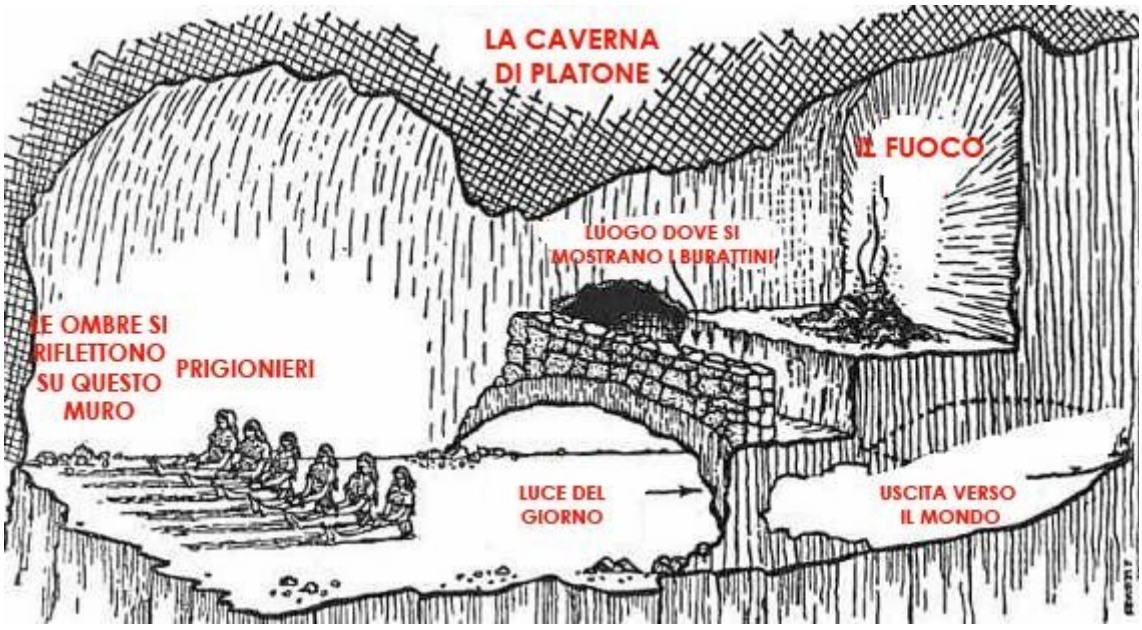


Fig. 1.5: Mito della Caverna

Secondo il mito, un gruppo di prigionieri sono vincolati a osservare, da sempre, un muro, senza aver mai visto la luce del giorno. Dietro di essi si trova un fuoco che proietta le ombre (mondo sensibile) di un gruppo di statue (mondo del significato – semantico) sul muro. Per i prigionieri esistono solo le ombre sul muro: essi non sono a conoscenza delle statue che le proiettano da dietro. Fuori dalla caverna vi sono, invece, delle persone libere che vedono tutto e sanno bene che ogni ombra è

10 “Kritik der reinen Vernunft” – Immanuel Kant
11 “La Repubblica” – Platone

associata ad una statua. Questo mito calza perfettamente con il nostro caso di studio se si associano i prigionieri alle macchine, le persone libere agli utenti delle macchine e le persone che trasportano le statue agli sviluppatori delle applicazioni che girano sulle macchine. Secondo Platone il mondo del significato è raggiungibile per il prigioniero nel momento in cui esso dovesse liberarsi, ma se lo facesse non potrebbe mai ragionare come una delle persone libere (le quali sanno che le ombre derivano dalle statue) dato che egli è nato vedendo ombre e quindi ragiona con le ombre: vedendo le statue crederebbe che siano le ombre a proiettare luce per mezzo delle statue sul fuoco... ragionerebbe al contrario!

Platone non dissocia definitivamente aspetto e significato, tant'è questa dissociazione avvenne in un momento storico intermedio: è con la nascita del pensiero scientifico, il processo a Galileo, la rivoluzione copernicana e più generalmente l'avvento dell'epoca moderna che nasce questa separazione netta fra “come” e “perchè”; in questo contesto Kant ne è stato il maggior e principale teorico.

Probabilmente è necessario superare questa separazione netta per ottenere il risultato che vogliamo? Tradotto: è probabile che sia necessario abbandonare il metodo scientifico stesso perchè si possa sviluppare una vera intelligenza artificiale.

Verrà quindi fatta una brevissima e piuttosto cruda dimostrazione filosofica per dimostrare temporaneamente al lettore l'assurdità di questa rigida separazione, operata da Galileo e ratificata in Kant. Poniamo Immanuel Kant come ascoltatore di una frase qualsiasi, per facilità riutilizziamo quella d'esempio del sottocapitolo 1.1: all'arrivo all'orecchio di Kant della frase “Ciao papà!” egli inizierà nella sua analisi dividendo il suono dal significato, passando ignaro di fronte all'evidenza: nel momento in cui egli stesso riceve la frase e parte alla ricerca del significato è in atto nel suo cervello un'associazione automatica di suono e significato. Quindi nel cervello umano la separazione fra sintassi e semantica non esiste, bensì sintassi e semantica sono due aspetti diversi di un tutt'uno!

Dove sta allora la difficoltà nel traslare un'intelligenza umana in un'intelligenza artificiale? Nel fatto che questi “tutt'uno” (fenomeno+noumeno) sono entità continue che possono essere “salvate” nella memoria umana costituita da un supporto cerebrale che è di fatto esso stesso un “tutt'uno”, ossia si hanno informazioni continue salvate su supporti continui mentre una macchina usa informazioni discrete e le salva su supporti discreti.

Esiste un errore intrinseco, irremovibile, nel campionamento dei dati continui (la realtà) in dati discreti (i dati che metteremo nel nostro web semantico) che può però essere limitato aumentando la quantità di dati discreti utilizzati per rappresentare una singola informazione continua: i termini di questo arrotondamento sono normalmente calcolati, in ingegneria, per mezzo del teorema di

Shannon.

Dato che come abbiamo detto nel sottocapitolo 1.2, il nostro web semantico viene definito per mezzo di informazioni sintattiche (soggetto-predicato-oggetto), non si può creare una vera e propria intelligenza artificiale, a causa dell'errore intrinseco, ma si può, utilizzando un numero abbastanza elevato di statement, simulare comportamenti abbastanza complessi da sembrare dettati da un'intelligenza vero e propria.

Queste simulazioni di intelligenze umane, possono quindi offrire un parziale pre-raziocinio di quelle che sono le volontà dell'utente, anticipando le sue mosse nel web.

Se utilizzate per un motore di ricerca, una intelligenza artificiale si occuperà da sola di andare a comporre le varie informazioni estratte dal web per creare dinamicamente un proprio documento da ritornare all'utente come prodotto della ricerca.

Si rimanda il lettore alla lettura del capitolo 5.3 per capire come queste intelligenze potrebbero integrarsi con Google.

CAP.2 SCENARIO DI RIFERIMENTO

2.1 Smart-M3

Smart-M3[12] è un progetto open-source, avente alle spalle come finanziatori la NOKIA, che provvede una piattaforma di scambio di dati semantici fra applicazioni e dispositivi. Attualmente è utilizzato all'interno del progetto SOFIA.

L'idea chiave di Smart-M3 è che dispositivi e applicazioni possono mettere a disposizione le informazioni contenute in loro per altri dispositivi e applicazioni attraverso componenti chiamate "SIB", ossia Shared Information Brokers, con cui si può dialogare per agenti capaci di processare queste informazioni: "KP agent" (Knowledge Processor agent). Il sistema è rappresentato in Fig 2.1:

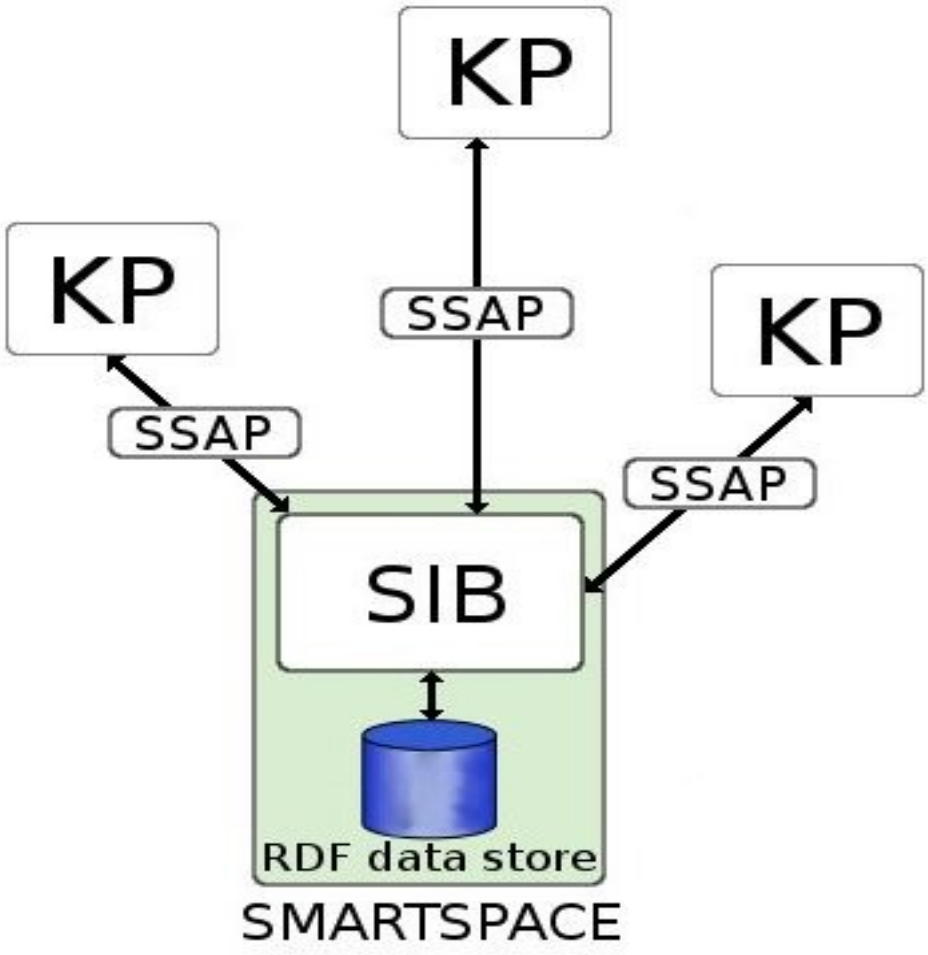


Fig. 2.1: Sistema Smart-M3: schema e componenti

12 http://sourceforge.net/projects/smart-m3/

Le SIB sono le componenti che si interfacciano con il data store e si occupano di organizzare le triple in esso contenuto per metterle a disposizione dei KP.

A seconda del data store utilizzato esistono vari tipi di implementazioni della SIB, la versione utilizzata ai fini di questo progetto è stata sviluppata con RedLand e il suo autore è il correlatore della tesi: Francesco Morandi.[13]

Di fatto è l'access point tramite cui si possono effettuare semplici manipolazioni: join, leave, insert, remove, update, query, subscribe, unsuscribe; è anche l'End point tramite cui si possono fare interrogazioni SPARQL (vedi sotto-capitolo 1.3), che permettono manipolazioni più complesse.

Le comunicazioni con la SIB possono essere fatte sfruttando vari tipi di meccanismi di trasporto, fra cui TCP/IP, Http e BlueTooth. Ai fini del nostro progetto, il dialogo con la SIB viene fatto tramite richieste Http.

Un insieme di una o più SIB interconnesse e i relativi data store formano uno SMART SPACE, che si può vedere come il punto d'accesso fisico ad un insieme ontologico.

Il Knowledge Processor è l'agente applicativo che si occupa di dialogare direttamente con la SIB. Il dialogo avviene tramite un protocollo chiamato SSAP: “Smart Space Access Protocol”, che consiste di formati XML definiti dalla W3C, difficili da leggere direttamente per un utente umano.

In questo tabella possiamo vedere tutti i tipi di manipolazioni della SIB permessi in SSAP.

Tab. 2.1: Comandi SSAP

Join	Connette ad uno Smart Space
Leave	Disconnete da uno Smart Space
Insert	Inserimento di un Graph RDF nello Smart Space
Remove	Rimozione di un Graph RDF dallo Smart Space
Update	Aggiornamento di un vecchio Graph RDF con uno nuovo all'interno dello Smart Space: di fatto la SIB effettua prima una Remove e poi una Insert
Query	Effettua una query allo Smart Space usando come formato della domanda un Graph RDF oppure una forma SPARQL.
Subscribe	Configura una sottoscrizione allo Smart Space: di fatto il KP effettua permanentemente delle Query alla SIB e viene notificato quando il contenuto della SIB si modifica
Unsubscribe	Cancella una sottoscrizione allo Smart Space

13 http://sourceforge.net/projects/smart-m3/files/Smart-M3_B_v0.3.1-alpha/Smart-M3_B_v0.3.11.tar.gz/download

Così come i meccanismi di dialogo avvengono usando diversi tipi di supporto, il KP può essere realizzato in diversi tipi di linguaggio. Il nostro progetto, utilizzando richieste http, usa un insieme di librerie PHP prodotte da ricerche precedenti dell'università ALMA MATER.

Queste librerie si chiamano SmartM3-PHP-KPI e verranno descritte nel prossimo capitolo.

2.2 PHP-KPI

Le librerie SmartM3-PHP-KPI[14] del dipartimento sono state sviluppate allo scopo di provvedere uno strumento per connettersi e dialogare con uno Smart Space in remoto per mezzo di richieste Http.

Esistevano in precedenza già varie implementazioni di KP, di cui le principali erano in C e Java: quella in Java è stata il modello per svilupparne una versione PHP (data la somiglianza dell'approccio di programmazione ad oggetti).

La logica di fondo è di strutturare tutta la libreria in due classi (così come era in Java): KPICore e SSAP_XMLTools.

- KPICore è la classe che implementa la parte di interfaccamento diretto con la SIB. Fornisce la possibilità di inviare e ricevere messaggi dalla SIB all'applicazione sovrastante. I messaggi non vengono però scritti o letti dalla stessa.
- SSAP_XMLTools è la classe che fornisce e legge il codice XML dei messaggi SSAP usati dal KPICore.

Nella Fig. 2.2 possiamo vedere per esteso lo schema di funzionamento:

14 http://sourceforge.net/projects/sm3-php-kpi-lib/

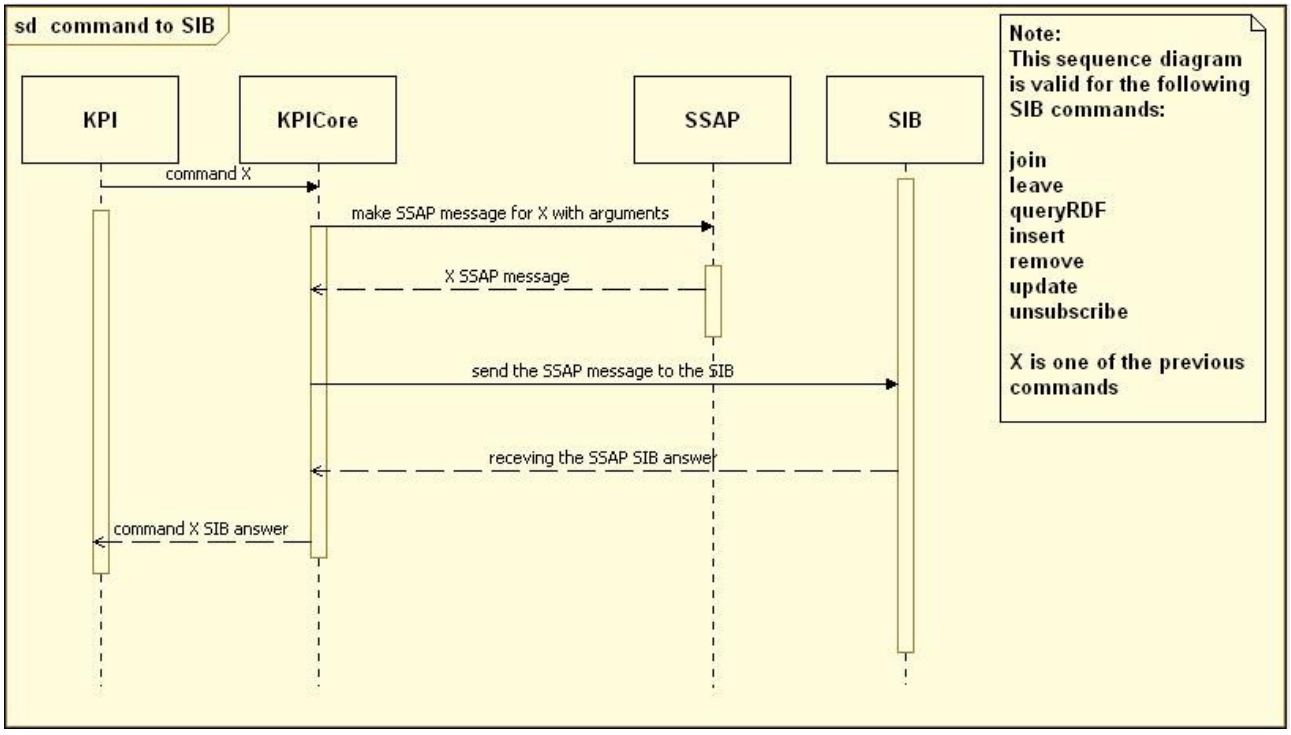


Fig. 2.2: schema di funzionamento libreria SmartM3-PHP-KPI nella versione 1.0

Da notare come il set di istruzioni della libreria, al momento della consegna al sottoscritto (ossia nella sua versione 1.0) sia limitata rispetto alla totalità delle possibili operazioni con la SIB citate al sottocapitolo 2.1: non esiste la possibilità di fare Query in SPARQL o di fare delle Subscribe; questo perchè le Query SPARQL sono state lasciate a future implementazioni, dato che gli standard W3C su SPARQL non sono tuttora completate, nemmeno al momento della stesura di questa tesi; le Subscribe sono state invece abbandonate completamente a causa della deficienza implicita di PHP: essendo PHP processato una sola volta per richiesta, era impossibile garantire la connessione permanente necessaria per effettuare una sottoscrizione.

Fra i compiti assegnati al sottoscritto, rientrava quindi necessariamente la reingegnerizzazione della libreria stessa, allo scopo di:

1. Implementare la funzionalità delle query SPARQL.
2. Testare la libreria ed effettuare un eventuale debug.
3. Ripulire la libreria da vecchie funzionalità deprecate (chiamate WQL).
4. Ottimizzare le performance della libreria.

2.3 Tecnologie a disposizione

Gli strumenti ulteriori di cui il sottoscritto si è avvalso sono stati reperiti per mezzo delle conoscenze personali in materia di sviluppo Web.

Dovendo sviluppare una Web Application basata su libreria PHP, ad una stima iniziale è risultato opportuno crearla con i più basilari e tipici linguaggi di sviluppo web: Javascript, PHP, Html e CSS.

Ad essi sono state aggiunte queste librerie Javascript:

- **jQuery**[15] : jQuery è una libreria Javascript potentissima, probabilmente la più popolare, attualmente, presso tutti gli sviluppatori Web. Essa permette un controllo completo tramite Javascript sul DOM della pagina. Facilita la gestione di eventi, animazioni e rendering grafico tramite CSS.

- **Backbone.js**[16] : Backbone.js è un Framework Javascript che permette di creare la pagina sfruttando un pattern MVC, ovvero “Model-View-Controller”. La pagina viene strutturate in 3 sezioni separate: il modello contiene i dati d'interesse per l'utente, la vista li rappresenta nella pagina allo stesso e il controller si occupa di ascoltare le interazioni dello stesso (Fig. 2.3). Ognuna delle componenti può lavorare in modo asincrono, garantendo flessibilità nello sviluppo dell'applicazione. Inoltre Backbone fornisce la classe Collection, per raggruppare modelli simili e inviarli o scaricarli in formato JSON al server.

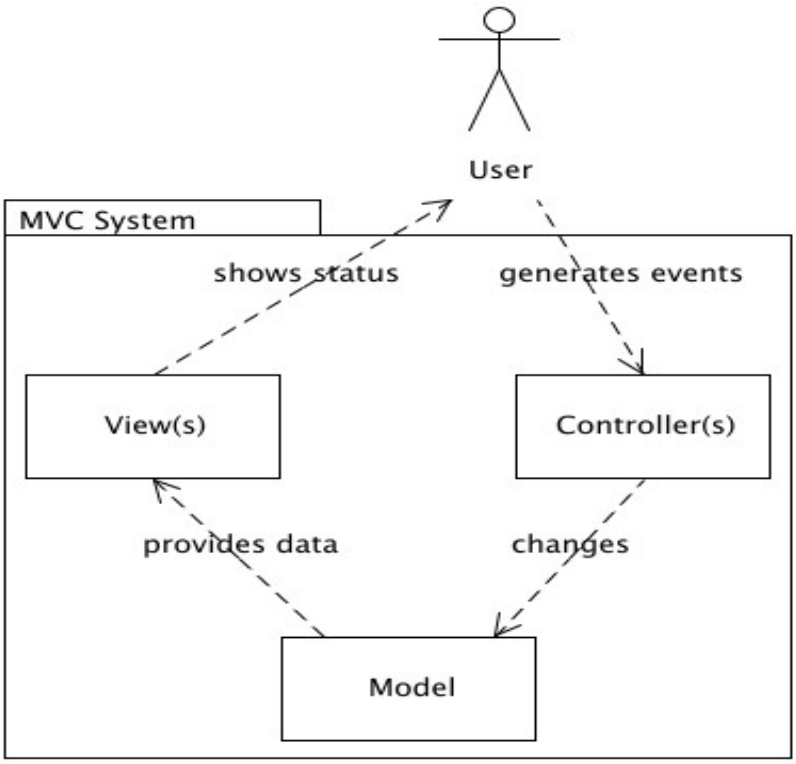


Fig.2.3 : MVC system

15 http://docs.jquery.com/
16 http://www.backbonejs.org

- **Underscore.js**[17] : libreria che fornisce una serie di utility per la programmazione in Javascript, inoltre provvede un meccanismo per la compilazione di template html. Quest'ultima funzionalità è stata la parte principale che la libreria ha giocato nel progetto. Il funzionamento è piuttosto semplice: si realizza un template html di una compenente grafica della pagina, con un insieme di parti variabili che vengono disegnate (sempre in linguaggio html) dal Client dell'utente stesso (non sul server come farebbe invece PHP).

Inoltre, abbiamo sfruttato nel progetto le nuove caratteristiche da poco rilasciate dalla W3C per i linguaggi Html (che aggiorna, di fatto, anche lo scripting in Javascript aggiornando l'ECMAscript) e CSS:

- **Html5**[18] : strutturazione dell'Html in gruppi tematici, che forniscono una semantizzazione di base al contenuto della pagina. Nuovi tipologie di renderizzazione grafica: svg e canvas. Sistema di connessione con Web Socket. Multithreading Javascript con Web Workers. Salvataggio dei dati di sessione e cookies all'interno del documento html. IndexedDB.
- **CSS3**[19] : renderizzazione grafica avanzata per mezzo di nuove animazioni, trasformazioni delle immagini, effetti visivi sul testo, ombreggiatura, font-face, keyframes, etc.

Come applicazioni di sviluppo, il progetto si è avvalso di:

- **Bluefish**[20] : Ambiente di sviluppo web open source per piattaforma linux. Garantisce: supporto di controllo per html, php, javascript e altri linguaggi grazie ad opportuni plugin; fra gli aiuti che esso fornisce si citano: colorazione degli elementi del linguaggio, controllo degli errori, rifattorizzazione automatica del codice. E' paragonabile a ciò che è Eclipse per lo sviluppo in Java.
- **GIMP**[21] : Tool open source per lo sviluppo di grafica in 2d, probabilmente la più potente e la più popolare.
- **Apache**[22] : Web server. Utile per il test dell'applicazione in localhost.

---

17 http://www.underscorejs.org/
18 http://www.w3.org/TR/2011/WD-html5-20110525/
19 http://www.w3.org/TR/css3-selectors/
20 http://bfwiki.tellefsen.net/index.php/Main_Page
21 http://docs.gimp.org/2.8/it/
22 http://www.apache.org/

CAP. 3 ANALISI DELLE SPECIFICHE

3.1 Capacità estese di interfacciamento con la SIB

Quando mi venne consegnata la richiesta della tesi nelle mani era già presente nelle mani del prof. Roffia (e quindi dell'università) una applicazione scritta, insieme al KP, in C# : mi fu quindi evidenziata, dallo stesso, l'impellente necessità di una nuova applicazione che permettesse di interfacciarsi con le SIB di Smart-M3, a causa della pesantezza di utilizzo del KP scritto in C# e la presenza presso il dipartimento di efficienti librerie PHP non ancora utilizzate.

L'interfaccia avrebbe dovuto garantire, secondo il disegno iniziale, il controllo completo della SIB, utilizzando tutti i comandi descritti nella Tab. 2.1, e sopra di questi si sarebbero poi realizzate complesse funzionalità di più alto livello.

Come spiegato nel sotto-capitolo 2.3, per ottenere l'accesso a tutti i comandi messa a disposizione dalla SIB, è necessario prima estendere le librerie KPI in modo da implementarvi le richieste SPARQL; inoltre al fine dell'implementazione del comando Subscribe è necessario lavorare a più alto livello giacchè PHP non può realizzarlo direttamente a causa del suo meccanismo di esecuzione one-time: basandosi sul fatto che la Subscribe è di fatto una Query permanente al sito, in cui le informazioni vengono inviate all'utente solo in caso di cambiamento del contenuto della SIB, si può realizzare la stessa utilizzando una componente dell'applicazione che, lavorando in modo asincrono rispetto al resto, periodicamente effettui una chiamata alla funzione di Query della libreria PHP e aggiorni l'applicazione in caso di cambiamento del messaggio ricevuto rispetto a quello precedente.

Ulteriori funzioni di alto livello comprendono:

1. organizzazione del contenuto della SIB sulla base di ordinamenti lessicali di soggetto, predicato od oggetto da usare come indici;
2. dato che normalmente una SIB contiene migliaia di informazioni, deve essere fornito un meccanismo di ricerca rapida all'interno dei risultati per mezzo di parole chiave;
3. numerare le triple e fornire informazioni generali sui Graph RDF presenti nella SIB;
4. connessione simultanea a più SIB contemporaneamente, che permetta il confronto e il dialogo fra più SIB appartenenti a Smart Space diversi;

5. meccanismo di trasforformazione degli URL in URN e viceversa.

3.2 Studio dell'usabilità

Nonostante i dati provenienti dall'RDF data store e quindi dalla SIB, siano statement RDF nudi e crudi, la difficoltà di lettura implicita rimane, soprattutto a causa dell'incomprensibilità intrinseca degli URI.

Gli RDF sono concetti che, al momento della stesura di questa tesi, non hanno raccolto ancora il dovuto successo presso il pubblico degli utenti di internet, né presso quello degli sviluppatori.

E' necessario quindi fornire all'utente un interfaccia facile da utilizzare e da comprendere, che non dia per scontate le capacità dello stesso, così da ampliare l'eventuale raggio di utenze.

Per come ce lo siamo disegnato, l'utente dovrebbe essere capace di utilizzare la SIB senza neanche sapere cosa sia un RDF!

Lo scenario che abbiamo visto è la rappresentazione di informazioni RDF rappresentate all'utente come strutture molto simili a frasi umane, come negli esempi citati nel Cap. 1 .

L'inserimento di Statement deve permettere all'utente una creazione di descrizioni complesse, secondo ontologie inserite dallo stesso utente o reperite altrove. Questo tipo di descrizioni possono essere fatte solo se si ha una comprensione intuitiva dei termini linguistici con cui si sta lavorando. Si rimanda il lettore al Cap. 5 per comprendere come questo tipo di descrizioni complesse permetterebbe di indicizzare facilmente il nuovo ***www3*** .

Un'altro aspetto cardine dell'applicazione web da sviluppare deve essere quello di garantire una connessione virtualmente permanente con la SIB e un dialogo continuo, nonostante le librerie sottostanti impongono di doversi connettere e disconnettere ad ogni singola operazione. Si vuole quindi dare all'utente l'impressione di interagire in real-time.

3.3 Studio dell'interfaccia grafica

Per quanto non lo si voglia ammettere, l'estetica fa il suo gioco. Il bello è spesso più importante

dell'utile. Le masse sono sempre state più attratte da ciò che è scenografico, rispetto a ciò che è veramente utile: gli antichi usavano chiamare questo fenomenologia sociale “Panem et Circenses”.

Dato che fra le intenzioni principali di questa web application, c'è proprio quello di aprire l'uso degli RDF ad un largo raggio di utenze, questo aspetto risulta per noi fondamentale.

Si passi, poi, che il sottoscritto è un convinto sostenitore della teoria platonica “Kalos Kai Agathòs”: ciò che è bello è anche buono.

L'applicazione web deve presentarsi all'utente come una pagina “smooth” e dinamica, associato ad un aspetto artisticamente elevato.

Per realizzare l'effetto smooth e dinamico ci si è potuti ben appogiare alle potenzialità di jQuery, mentre il nocciolo duro della grafica poteva poggiare sulle nuove componenti di CSS3 e Html5 insieme ad un piccolo sforzo creativo -personale- di renderizzazione 2d con il programma di manipolazione di immagini.

Infine, dato che il progetto verrà distribuito su Sourceforge, una interfaccia accattivante garantirebbe al sottoscritto l'acquisizione di fama come Web Designer.

CAP.4 SVILUPPO DEL PROGETTO

4.1 Fase di documentazione

La fase iniziale, precedente la fase di progettazione vero e propria, consisteva nel reperimento di tutte le informazioni necessarie allo sviluppo.

La lettura di questo sottocapitolo, quindi, presenta al lettore un percorso completo di studio che fornisce tutto il bagaglio conoscitivo di base necessario per iniziare a sviluppare sul web semantico, creando applicazioni le cui potenzialità vanno ben al di la di quelle di cui dispone effettivamente il progetto di questa tesi.

La prima tappa è -doverosamente- l'incontro con il pensiero del creatore stesso del web semantico, Tim Berners Lee. A questo titolo la fonte migliore è un video[23] presente sul portale “TED: Ideas Worth Spreading”, che consiste di 16 minuti di esposizione da parte del medesimo autore sulla storia del vecchio web, dei meccanismi che hanno portato alla nascita del web semantico e il riconcepimento del web come un enorme database di dati collegati, non più come un insieme di pagine; è presente anche l'esposizione del sistema “Linked Data”, che non è però utile ai fini del nostro progetto.

Il passo successivo consiste nella lettura di un'altra tesi, fornitami in precedenza dal prof. Roffia: si tratta della tesi di Indrit Beqiri “Ontologie per il Web Semantico”[24] . La tesi fornisce una trattazione completa delle tecnologie per la semantizzazione, da noi citate al sottocapitolo 1.3; essa è organizzata in 3 capitoli per noi interessanti:

1. La prima tratta in modo molto esteso cosa sono gli RDF, gli Statement, microformati e RDFa, come vengono serializzati, Linked data, uri, blank node e typed data. Vi si trova anche la spiegazione dei GRDDL.
2. La seconda tratta le tecnologie per la creazione delle ontologie: RDFS, SKOS[25], OWL e il metodo per interrogarle con SPARQL.

23 http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html
24 “Ontologie per il Web Semantico” - Indrit Beqiri, 2012
25 http://www.w3.org/2004/02/skos/

3. La terza elenca in modo esteso le principali ontologie attualmente utilizzate sul web: Dublin Core , WordNet, Cyc, UMBEL , Basic Formal Ontology, DOLCE , OntoSpace , IDEAS, General Formal Ontology, SUMO, COSMO , ABC ,YSO , DBpedia Ontology (ognuna di queste ontologie può risultare utile per la creazione di statement da usare nel nostro progetto). Da infine una classificazione delle principali categorie ontologiche.

Il cuore vero e proprio delle informazioni utili è però stato estratto da un libro dedicato agli sviluppatori del web semantico. Il libro è stato scelto in seguito ad una ricerca, condotta dal sottoscritto, fra i principali testi pubblicati sull'argomento: il prodotto della ricerca è stato il libro di un autore cinese -tale “Lyang Yu”- che reca il titolo “A Developer’s Guide to the Semantic Web”[26] (la versione utilizzata era quella inglese); fra tutti quelli ricercati, esso risulta essere il più chiaro, dato che l'autore spiega le cose in modo netto e seguendo un filo logico ben delineato.

Il libro è piuttosto cospicuo (circa 600 pagine) per cui si presenteranno al lettore solo i primi 6 capitoli, i cui argomenti vertono direttamente verso l'interesse della tesi:

1. Il Cap. 1 tratta in modo generale il nuovo Web, delineando la rivoluzione concettuale che esso porta: da un web di pagine ad un web di dati. Vengono proposti -frequentemente- casi di utilizzo del nuovo web.
2. Il Cap. 2 tratta esclusivamente gli RDF: ne definisce il modello astratto, citando le definizioni ufficiali della W3C, la struttura a statement, i modi di serializzazione; vengono poi spiegati tutti gli elementi del vocabolario con frequenti esempi; la parte più originale è quella finale in cui si spiega come creare RDF usando alcune regole di base. Si spiega come inserire un file RDF/XML in una pagina e come usare un RDF validator per estrarli.
3. Il Cap. 3 tratta le tecnologie legate ad RDF per la semantizzazione direttamente in html: microformati, RDFa e GRDDL.
4. Il Cap. 4 spiega come si estende il linguaggio RDF con RDFS. Come per RDF, ogni vocabolo di RDFS viene riportato. Dagli schemi RDFS viene poi presentato al lettore il concetto più esteso di ontologia, definendo come il sapere semantico viene organizzato.
5. Il Cap. 5 descrive come creare le ontologie con OWL nelle sue due versioni (OWL1 e OWL2). Viene spiegato cosa siano gli assiomi e come siano relazionati con gli statement. Di particolare interesse sono gli esempi.

26 “A Developer's Guide to the Semantic Web” - Lyang Yu, 2011

6. Il Cap. 6 tratta il linguaggio SPARQL. Vengono citati i concetti di RDF data store e gli Endpoint. La trattazione si articola nei quattro tipi di Query: SELECT, CONSTRUCT, ASK e DESCRIBE; per ognuno dei quattro tipi viene descritta blocco per blocco, la struttura sintattica della Query.

Per quanto riguarda Smart-M3, dato che il progetto non possiede un vero e proprio sito ma solo una serie di versioni pubblicato su sourceforge e un wiki, il sottoscritto si è rifatto a documenti procurati personalmente dal prof. Roffia; in particolare è stata chiarificante la lettura del documento “Smart-M3 Information Sharing Platform ”[27] .

Tutti gli approfondimenti ulteriori -specialmente riguardo i vocaboli e i costrutti sintatti di RDF[28], RDFS e OWL[29]- sono stati fatti consultando direttamente le specifiche dal sito della W3C.

A causa della difficoltà di lettura di queste specifiche, si consiglia comunque al lettore la lettura del libro di Lyang Yu, che risulterà più che sufficiente.

4.2 Progettazione

Date le tecnologie di sviluppo spiegate al Cap.2 e le brevi specifiche iniziali spiegate al Cap.3, si è pensato di organizzare lo sviluppo del progetto seguendo un metodo “per prototipazione”: esso consiste in uno sviluppo progressivo del prodotto finale, partendo da un'applicazione molto semplice e raffinandola ad ogni passo con nuove funzionalità; si basa quindi sulla possibilità per cliente (nel nostro caso il prof. Roffia) e sviluppatore (nel nostro caso il sottoscritto) di mantenersi sempre in contatto per ridefinire ogni volta nuove specifiche da aggiungere al progetto. Di fatto, ad ogni singolo prototipo è associata una versione dell'applicazione e -normalmente- al momento del rilascio ufficiale dell'applicazione la versione dovrebbe essere la 1.0. Questo tipo di sviluppo è tipicamente il più elastico ed efficiente, dato che la continua comunicazione col cliente garantisce di non uscire dalle specifiche ; inoltre, lo sviluppo di oggetti semplici da estendere mano a mano ben si sposa con il meccanismo di programmazione ad oggetti di Javascript “per prototipazione” e mette al

27 “Smart-M3 Information Sharing Platform ”
28 http://www.w3.org/RDF/
29 http://www.w3.org/TR/owl2-overview/

riparo dai bug: un buon codice si sviluppa testandolo ogni volta che vengono fatte delle modifiche.

Come pattern di partenza sono stati utilizzati dei “boilerplate” ottenuti da vecchi progetti di pagine e applicazioni Web sviluppate dal sottoscritto: un “boilerplate” è un insieme strutturato di file standard da usare come punto di partenza per lo sviluppo di un'applicazione web; tipicamente è costituita dalla pagina iniziale “index.php” con uno scheletro html della pagina che verrà creata, una cartella per i file CSS con alcune definizioni iniziali (body, font-size, padding, font-family, etc.), una cartella per il Javascript con le librerie già incluse (jQuery, Backbone e Underscore), una cartella per le immagini con icone, textures e logo utilizzati in varie occasioni e una cartella PHP contenente pagine-scheletro per le chiamate Ajax e librerie di utilities.

Per come si prospettavano le specifiche del progetto, era chiaro che questa applicazione avrebbe dovuto lavorare principalmente dal lato Client, e questo lasciava poco spazio al PHP mentre favoriva il lavoro in Javascript. L'uso di elementi Java da collegare con Live connect alla componente Javascript dell'applicazione sembrava, inoltre, superflua.

La prospettiva era quindi lo sviluppo di un applicazione Client-Heavy, scritta per il 90% in Javascript.

All'inizio le potenzialità di un Framework MVC per questo tipo di progetti non mi era così chiaro, né ero abituato a organizzare il Javascript secondo uno schema esclusivamente orientato agli oggetti, per cui in una prima versione il codice sarebbe dovuto essere steso in modo procedurale, con un lungo file “core.js” da includere nella pagina.

Da un punto di vista grafico, la visione, che poi si è confermata vincente, era quella di una pagina strutturata in “Box”, ossia caselle separate sulla base della tematica del loro contenuto: un box per la connessione alla SIB, un box per i comandi dell'utente (Insert, query, remove, etc.), un box per il risultato delle interrogazioni e così via.

I box sarebbero poi dovuti esser posti su diversi piani grazie alla proprietà “Z-index”. Lo Z-index permette di trasformare la pagina a 2d in un pseudo 3d con visuale perpendicolare al piano XY, in cui la coordinata Z è appunto definita dalla proprietà Z-index dell'elemento, che di default è pari 0.

Un primo schizzo che feci con GIMP portò come risultato il logo dell'applicazione (Fig. 4.1) che sarebbe poi stata l'allegoria del design che la pagina avrebbe seguito.

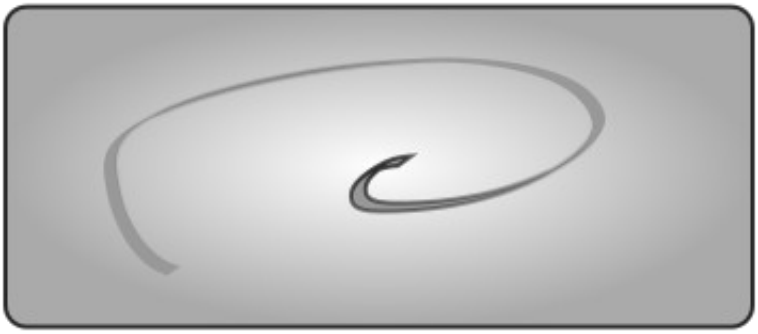


Fig. 4.1: Il logo dell'applicazione

In una prima idea questi box sarebbero dovuti essere disegnati per mezzo delle nuove tecniche dell'Html5: gli SVG.

Si è visto in seguito che un banale box costituito da un DIV a cui siano associate una serie di proprietà (border, color, opacity) del CSS, specialmente usando le nuove proprietà della versione CSS3 come ombreggiature e colori a gradiente, producevano risultati migliori in modo molto più semplice.

Per il dimensionamento degli elementi della pagina e dei font ci si è avvalsi della nuova unità di misura apportata da CSS3 e html5: gli “em”. Gli “em” permettono di definire le dimensioni di un elemento secondo un layout elastico che si ridimensiona in base alla dimensione del testo dell’utente. Più precisamente, un’interfaccia elastica si ridimensiona in base alla dimensione del testo del browser—solitamente di 16px—che l’utente può cambiare a piacimento.

Ritornando alla questione dello sviluppo del codice del Client vero e proprio, il primo problema da affrontare era l'incapacità di PHP a garantire una connessione e un'interazione in real-time. Per sopperire alla deficienza del linguaggio la risposta più funzionale era l'uso di AJAX.

4.3 Interfacciamento AJAX

L’acronimo AJAX significa esattamente Asynchronous JavaScript And XML[30] (JavaScript asincrono ed XML). Non si tratta di una nuova tecnologia né di un’invenzione bensì di un concetto utilizzato per sviluppare applicativi avanzati: il concetto è in parte espresso nell’acronimo scelto, un

30 http://www.w3.org/TR/XMLHttpRequest/

utilizzo asincrono di Javascript che attraverso l'interfacciamento con XML, può permettere ad un client di richiamare informazioni lato server in modo veloce e trasparente, allargando gli orizzonti delle rich internet applications.

Dato che è proprio lato server che lavora il PHP, con cui si utilizzano le funzioni messe a disposizione dalle librerie del dipartimento per dialogare con le SIB, la soluzione nostra era quella di creare un pseudo-real-time definendo per ogni singola iterazione con la SIB una singola pagina PHP, la quale verrà -poi- chiamata per mezzo di richieste AJAX.

Lo schema di funzionamento di un ipotetico dialogo con la SIB viene spiegato nelle seguenti figure:

FASE 1) CONNESSIONE CON LA SIB (comando di Join)

Ogni dialogo con una SIB deve iniziare necessariamente con la fase di connessione.

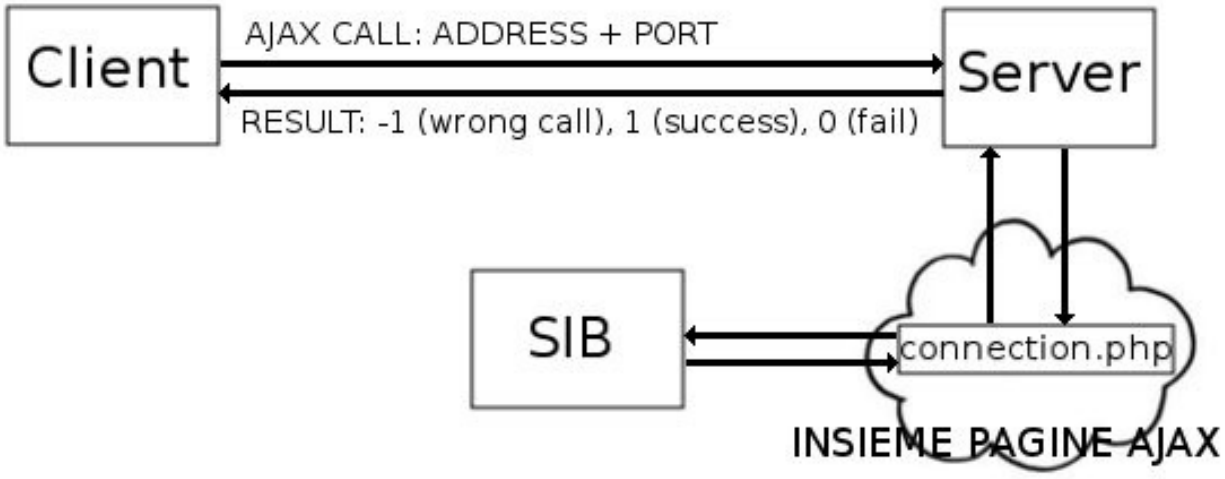


Fig 4.2: iterazione AJAX per effettuare il comando Join

Il Client riceve in input indirizzo e porta della SIB a cui connettersi dall'utente e li invia tramite una richiesta AJAX al server, richiedendo l'apertura della pagina appositamente creata per l'operazione di JOIN con la SIB.

Il codice che permette questo tipo di operazione è il seguente:

```
<?
    //CONSTANTS                    → costanti da usare come risultati
    define("WRONG_CALL" ,   -1);
```

```
define("ERROR"          ,   0 );
define("SUCCESS"        ,   1 );

//PHP LIBRARIES                          → inclusione delle librerie del dipartimento
require_once "../lib/KPICore.php";
require_once "../lib/SSAP_XMLTools.php";
                                         → parsing dei parametri
$loadAddress = $_GET['addr'] ;           → indirizzo della SIB
if (($loadAddress=="null")||($loadAddress=="*")) die(WRONG_CALL);

$loadPort = $_GET['port'] ;                    → porta della SIB
if (($loadPort=="null")||($loadPort=="*")) die(WRONG_CALL);
                                         → viene istanziata la classe KPICore
$KPAgent = new KPICore($loadAddress,$loadPort,"",true,true);
$result = $KPAgent->join();              → viene effettuato il comando Join
//if ($result==null) echo "0";
                                         → viene scritto il risultato al Client
if (substr_count($result, "Success") >= 1) echo SUCCESS;
else{
    echo ERROR;
}
$KPAgent->leave();                       → si chiude la connessione con la SIB
?>
```

Se il Server ritorna al Client il messaggio di successo, il Client salva indirizzo e porta dentro apposite variabili globali, altrimenti se il messaggio è di errore o “wrong_call” il Client imporrà a queste variabili il valore NULL. La presenza di valori o meno in queste variabili globali determinerà se il Client è attualmente connesso o no virtualmente, mentre in realtà, a livello macchina, si devono aprire e chiudere connessioni dalle pagine PHP ad ogni iterazione usando i dati di queste variabili globali. (quindi si fanno Join e Leave in serie)

FASE 2) DIALOGO CON LA SIB: QUERY, INSERT, REMOVE O UPDATE

In seguito alla connessione, si possono effettuare Query, Insert, Remove e Update. In questa illustrazione cartacea del progetto ci limitiamo a mostrare il funzionamento di una Query RDF, ma il meccanismo illustrato non è molto diverso nel caso si vogliano fare Insert, Remove o Update.

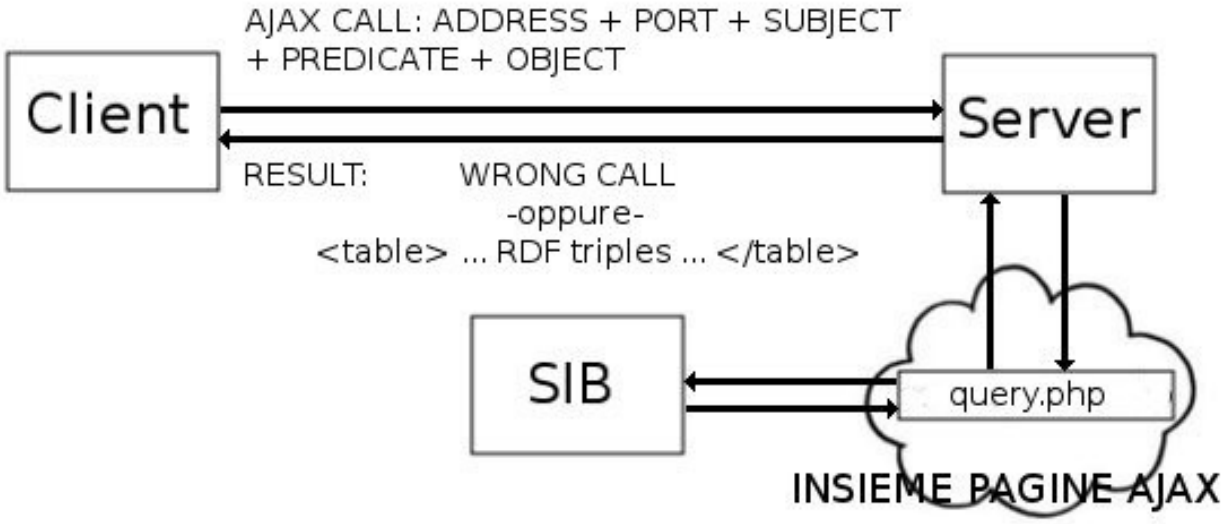


Fig. 4.3: iterazione AJAX per una Query

Il Client riceve in input soggetto, predicato ed oggetto della (o delle) triple desiderate e le invia alla SIB, associandovi indirizzo e porta salvati nella FASE 1, tramite una richiesta AJAX al server, richiedendo l'apertura della pagina “query.php”, appositamente creata per l'operazione di Query RDF.

Soggetto, predicato od oggetto possono essere definiti dall'utente se si cerca una singola tripla oppure arbitrariamente tralasciati (lasciando uno o più degli spazi di input bianco o usando la wildcard “*”) se si vuole ottenere tutte le triple possibili.

Il codice che permette questo tipo di operazione è il seguente:

```
<?                                                    → inclusione librerie del dipartimento
   require_once "../lib/KPICore.php";
   require_once "../lib/SSAP_XMLTools.php";
                                                      → parsing dei parametri
```

```
$loadAddress = $_POST['addr'] ;                    → indirizzo salvato nella fase 1
if (($loadAddress=="null")||($loadAddress=="*")) die("wrong call!");

$loadPort = $_POST['port'] ;                       → porta salvata nella fase 1
if (($loadPort=="null")||($loadPort=="*")) die("wrong call!");

$subject = $_POST['subj'];                         → soggetto
if (($subject=="null")||($subject=="*")||($subject=="")) $subject = null ;

$predicate = $_POST['pred'];                       → predicato
if (($predicate=="null")||($predicate=="*")||($predicate=="")) $predicate = null;

$object = $_POST['obj'];                           → oggetto
if (($object=="null")||($object=="*")||($object=="")) $object = null;

$obj_type = null;

                                                   → istanziazione KPICore
$KPAgent = new KPICore($loadAddress,$loadPort,"",true,true);
$KPAgent->join();                                  → connessione alla SIB

$xmlTools = new SSAP_XMLTools();                   → istanziazione SSAP_XMLTools

                                                   → riconoscimento del tipo dell'oggetto
if (substr_count($object, "http://") >= 1) $obj_type="uri";
else $obj_type = "literal";

                                                   → viene effettuata il comando Query RDF
$result=$KPAgent->queryRDF($subject,$predicate,$object,"uri",$obj_type); //NOTA: uso la query con il tipo di soggetto fissato a URI
```

→ *parsing del risultato*

```
$result=$xmlTools->getQueryTriple($result);
```

→ *se il risultato il risultato è "No triples found!"*

```
if (sizeof($result) == 0) {
        $message="No triples found!";
}
else{                                   → altrimenti viene ritornata una tabella html
    $table = "<table id=\"table\">
                        <thead>
                        <tr>
                            <th>#</th>
                        <th>SUBJECT</th>
                        <th>PREDICATE</th>
                        <th>OBJECT</th>
                        <th>OBJ TYPE</th>
                        </tr>
                        </thead>
                        <tbody>";

    $counter=1;                         → in ogni riga vengono inseriti gli Statement
    foreach ($result as $triple) {
                    $table .= "<tr>";
      $table .= "<td> $counter </td>";
      $subject = $xmlTools->triple_getSubject($triple);
      $table .= "<td> $subject </td>";
      $predicate = $xmlTools->triple_getPredicate($triple);
      $table .= "<td> $predicate </td>";
```

```
        $object = $xmlTools->triple_getObject($triple);
        $table .= "<td> $object </td>";
        $obj_type = $xmlTools->triple_getObjectType($triple);
        $table .= "<td> $obj_type </td>";
                                        $table .= "</tr>";
        $counter+=1;
    }
    $table .="      </tbody>
                                        </table>" ;
    $message = $table;
}

echo "$message";                    → il messaggio di ritorno viene inviato

$KPAgent->leave();                  → chiusura della connessione con la SIB
?>
```

Se il Server non ritorna “Wrong call”, il messaggio di risultato conterrà la stringa “no triples found!” oppure una tabella contenente le triple richieste dalla SIB e verrà disegnato sulla pagina dell'utente.

Le operazioni eseguite di fatto dal Server con la SIB sono 3: Join, Query (RDF) e Leave.

FASE 3) DISCONNESSIONE

La fase finale di disconnessione con la SIB non necessita di chiamate AJAX, giacchè il software sottostante, come spiegato in precedenza, non ha nessuna connessione aperta! L'unica operazione effettuata è l'assegnamento delle due variabili globali “address” e “port” della SIB a NULL.

Identificando con “FASE 0” lo stato di riposo dell'applicazione (ossia quando non ci sono tentativi di dialogo), possiamo definire il seguente automa a stati finiti: (Fig. 4.4)

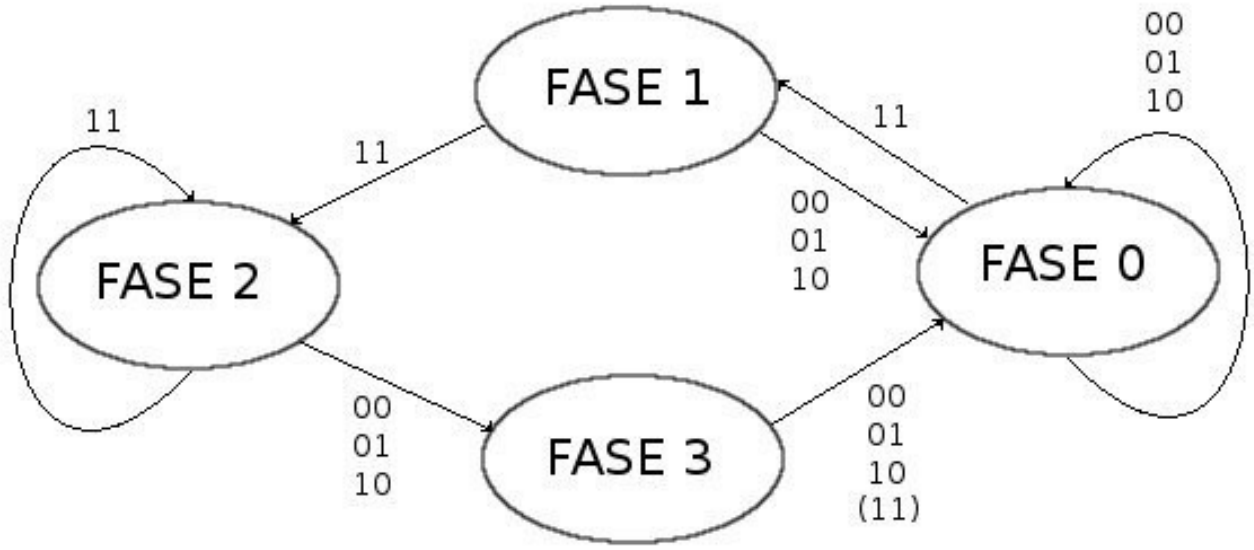


Fig. 4.4: Automa a Stati Finiti corrispondente all'interfacciamento AJAX

Si osservi che l'Automa è in equilibrio bistabile: Fase 0 (sleep) e Fase 2 (dialogo) sono stabili, mentre Fase 1 (connessione) e Fase 3 (disconnessione) sono fasi di transito.

Finalmente, si vuole specificare che nonostante le richieste AJAX normalmente siano asincrone, per questioni di solidità del codice il Client è stato forzato ad attendere la fine di queste chiamate prima di poter proseguire.

4.4 Primi tentativi di dialogo con la SIB

Una volta sviluppato il sistema di pagine PHP da chiamare con AJAX, è venuto il momento di fare i primi test.

I test vengono normalmente eseguiti in localhost, almeno in una fase iniziale: per questo si è dovuto

installare in locale sia il server che avrebbe ospitato l'applicazione web, sia un sistema Smart-M3 con una SIB da interrogare.

Il progetto, come spiegato nel Cap.2, è preposto per l'utilizzo di Apache come Web Server: la versione migliore è quella integrata nel pacchetto AMP (“Apache MySQL PHP”) che installa sulla macchina il succittato server, il database MySQL e il preprocessore PHP tutti insieme. In particolare dato che la macchina di sviluppo gira su sistema operativo Linux, la versione installata si chiamerebbe LAMP. Utilizzando Ubuntu dalla versione 12 in poi, tutto il pacchetto può essere scaricato in un colpo utilizzando l'applicazione “tasksel”.

Il server apache non necessita di particolare configurazioni per fare questo tipo di test in localhost, è sufficiente copiare tutto il progetto all'interno della cartella */var/www.*

Per quanto riguarda Smart-M3, verrà utilizzata una versione sviluppata dal correlatore Francesco Morandi e basata su data store Redland (si veda Cap.2).

L'installazione della SIB è piuttosto semplice su sistemi linux: è sufficiente eseguire da terminale il file “*install.sh*”; a questo punto il sistema va avviato per mezzo di due comandi:

sudo redsibd

e

sudo sib-tcp

da avviare in due terminali separati. A questo punto la SIB è pronta per comunicare all'indirizzo:porta “localhost:10010”.

E' già stato spiegato quali problemi avevano le librerie KPI-PHP al momento della consegna nelle mie mani, pertanto, essendo questo il momento migliore, si è intervenuti sul codice seguendo le linee indicate nel sottocapitolo 2.2, tranne l'inserimento delle query SPARQL che verrà illustrato più avanti. Per l'esecuzione dei test si è fatto ricorso ad un insieme di pagine che mi sono state fornite insieme alle librerie.

Ad una prima vista la libreria non riportava errori e così si trattava solo di ripulire il codice dalle funzionalità WQL e aggiungere la funzione per il comando Update.

Più tardi si è visto, purtroppo, che i meccanismi di connessione alla SIB possono presentare errori a causa del timeout troppo anticipato.

Finito il debug delle librerie, si è passati al test delle pagine AJAX che hanno funzionato come richiesto, salvo qualche errore con la consegna dei parametri: le prime versioni delle pagine AJAX da usare nella FASE 2 accettavano parametri usando il metodo GET, ma questo creava dei seri problemi dato che uno statement può avere fino a 3 URI e un URI completo tipicamente è piuttosto lungo: con il metodo GET la lunghezza delle informazioni che si possono inviare è limitata dai 256 caratteri che possono costituire al massimo un URL; per questo la soluzione era il passaggio di parametri con metodo POST.

Una volta assicurato il funzionamento della parte in PHP del progetto, è venuto il momento di abbozzare un primo esempio di interfaccia grafica; essendo un primo tentativo l'interfaccia risultava piuttosto semplice: priva di qualsiasi tipo di immagini, design e stilizzazione CSS, presentava solo una piccola form con cui immettere indirizzo e porta di connessione, soggetto, predicato ed oggetto per le interazioni della FASE 2 e tutti i relativi bottoni (Join, Insert, Query, Remove e Update). Vista la semplicità del codice in Javascript questa parte dello sviluppo è costata molto poco tempo.

4.5 La prima versione v.0.5

La prima versione del progetto rilasciato (v.0.5) era piuttosto semplice: consisteva di un insieme di file piuttosto ristretto: un'unico file Javascript contenente tutta la logica dell'applicativo, chiamato “core.js”, un singolo file CSS, una pagina html contenente solamente il logo dell'applicazione nell'header e 3 box (realizzati con CSS) disposti uno sopra l'altro: il primo conteneva la form per la connessione, il secondo la form per il dialogo e il terzo i messaggi di risultato.

Nonostante la pesantezza e la complicatezza dell'applicazione contenuta in “core.js”, era possibile eseguire dialoghi molto elementari con la SIB: Join, Query (RDF), Insert, Remove venivano eseguite senza errori; inoltre le sessioni di dialogo con la SIB sopravvivevano al refresh e alla chiusura del browser grazie ad un sistema di cookies; essendo ad una fase iniziale, le funzionalità erano effettivamente limitate.

Il file “core.js” era scritto in modo procedurale e privo di classi (e quindi di prototipazione). L'organizzazione era la seguente:

- Una prima parte contenente le variabili globali address e port della SIB

- Una seconda parte contenente le definizioni delle funzioni relative all'uso dei cookies e le linee di codice che processano i cookies all'apertura della pagina, garantendo il mantenimento della sessione di dialogo.
- Una terza parte contenente le definizioni delle funzioni deputate ad eseguire i comandi di Join, Query (RDF), Insert, Remove e Update.

Le funzioni, chiamate in seguito al click del relativo bottone, prendevano in ingresso i parametri necessari al comando, prendendoli dalle form, e l'elemento DOM da usare per inserire il risultato con jQuery.

Nella Fig. 4.5 si può vedere l'aspetto della pagina durante la FASE 2:

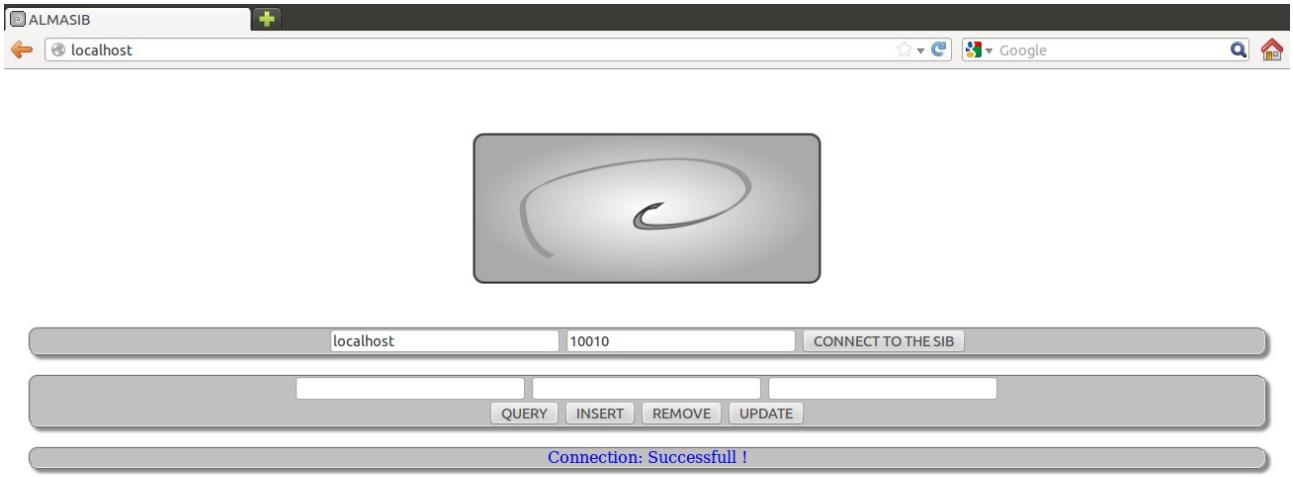


Fig. 4.5: ALMASIB v.0.5

Nota: il codice sottostante permetteva anche di eseguire comandi di Update, ma l'effettiva implementazione sulla pagina richiedeva 3 ulteriori campi di input che sono stati inseriti solo con il rilascio di una sotto-versione successiva (v.0.5.5).

Per essere agli esordi, il progetto funzionava già in modo egregio.

Il problema che angustiava di più e che doveva essere risolto per la versione successiva rimaneva quello del pessimo aspetto grafico; il problema si articolava in due punti principali:

1. Come rappresentare gli Statement ritornati dalla SIB all'utente in modo leggibile, facilmente vagliabile e ordinato?

2. Come creare un tema grafico generale per la pagina che risulti attraente?

4.6 Tecnologie d'ausilio → dataTables

Per rispondere al problema della rappresentazione dei dati ritornati, si è fatto ricorso ad una plugin di jQuery che appartiene al bagaglio di precedenti conoscenze del sottoscritto: il nome del plugin è “dataTables”.[31]

Partendo da comunissime tabelle HTML, dataTables le trasforma automaticamente in tabelle dall'aspetto gradevole e customizzabile (per mezzo dei temi), dinamiche e capaci di interagire con l'utente. Le caratteristiche chiave sono qui riassunte:

1. Paginazione dei risultati. I risultati possono essere raggruppati in quantità variabili.
2. Ricalcolo automatico della larghezza delle celle sulla base di contenuto e spazio disponibile.
3. Riassortimento automatico sulla base di uno qualsiasi degli indici scelto dall'utente con semplice click. L'ordinamento può essere ascendente o discendente.
4. Salvataggio dello stato fra i dati di sessione della pagina.
5. Calcolo automatico della quantità di dati presenti nella tabella.
6. Barra di ricerca per filtrare sottoinsiemi di dati dall'insieme della tabella sottostante. Quest'ultima è probabilmente la caratteristica più utile per il progetto!

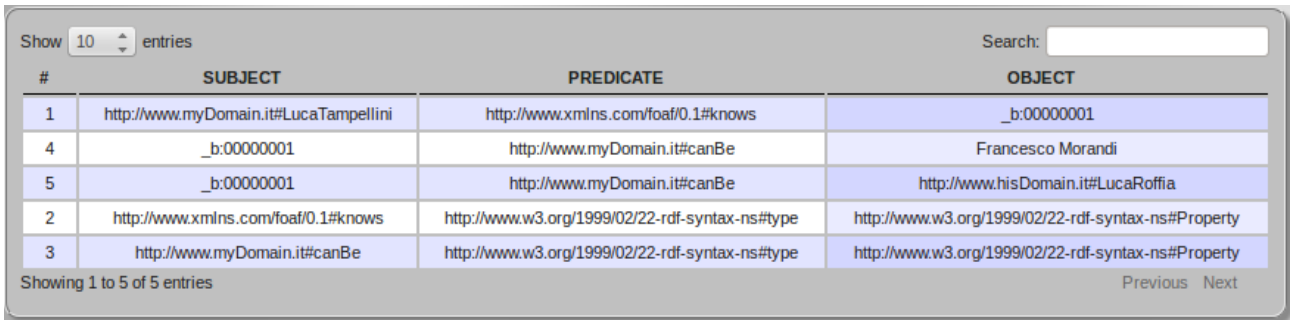
L'implementazione all'interno del progetto è stata piuttosto semplice: basta aggiungere fra le librerie JS il plugin e fra i CSS un file contenente la definizione dell'aspetto finalle delle tabelle.

L'utilizzo è ancora più semplice: come si vede al sotto-capitolo 4.3, la pagina query.php ritorna al Client una tabella con tutti gli Statement distribuiti per riga e con 4 colonne per soggetto, predicato, oggetto e tipo di oggetto. Una volta inserita la tabella all'interno della pagina, è sufficiente chiamare sopra di essa la chiamata jQuery:

$(“#nomeTabella”).dataTables();

31 http://datatables.net/

Il risultato potrebbe essere questa tabella: (Fig. 4.6)


Show 10 entries Search:

#	SUBJECT	PREDICATE	OBJECT
1	http://www.myDomain.it#LucaTampellini	http://www.xmlns.com/foaf/0.1#knows	_b:00000001
4	_b:00000001	http://www.myDomain.it#canBe	Francesco Morandi
5	_b:00000001	http://www.myDomain.it#canBe	http://www.hisDomain.it#LucaRoffia
2	http://www.xmlns.com/foaf/0.1#knows	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
3	http://www.myDomain.it#canBe	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property

Showing 1 to 5 of 5 entries Previous Next

Fig. 4.6: possibile risultato con dataTables

Fra i futuri sviluppi dell'applicazione, vi è quello di permettere all'utente di modificare manualmente i dati della tabella, inviando poi le modifiche al server. (si veda Cap.5)

4.7 La seconda versione v.0.6 →jQuery-UI

Nella prospettiva del rilascio della versione v.0.6 le questioni da affrontare ruotavano principalmente intorno alla creazione del tema grafico e al multiprofiling – ossia la possibilità di dialogare con più SIB contemporaneamente, avendo un profilo per ognuna e più profili per pagina.

Riguardo al tema grafico, già vagamente tracciato nello schema a box, si volevano due cose:

- Un insieme di componenti con effetti grafici avanzati da integrare a blocchi dentro al sito, eventualmente costituite da terze parti.
- Template contenenti pezzi di pagina, sviluppati apposta dal sottoscritto, che sarebbero stati riempiti a run-time dall'applicazione e aggiunti alla pagina.

Il primo punto è stato soddisfatto dall'utilizzo di un altro plugin per jQuery, ossia jQueri-UI[32] (User Interface).

JQuery-UI fornisce nuove interazioni ed animazioni, effetti avanzati e widgets, da integrare dentro al sito con chiamate jQuery.

Nella versione 1.8 (quella da noi usata) le caratteristiche si possono suddividere in 4 macro aree:

- Interactions : drag&drop, ridimensionamento, ordinamento e selezione.
- Widgets : componenti grafiche per l'interfaccia utente.

32 http://jqueryui.com/

- Effects : transizioni animate.
- Utilities : utilità di basso livello da usare per le interazioni, i widgets e gli effetti .

Vediamo nel dettaglio gli elementi di ogni macro-area:

Interactions

- Draggable - Rende gli elementi *draggable*
- Droppable - Control dove gli elementi *dragged* possono essere *droppati*
- Resizable - Rende gli elementi ridimensionabili
- Selectable - Avanzate caratteristiche di selezione per liste di elementi
- Sortable - Liste di elementi facilmente ordinabili

Widgets

- Accordion - classico container Accordion
- Autocomplete - Auto-completamento basato su ciò che l'utente digita
- Button - Pulsante avanzato
- Datepicker - Date-picker avanzato
- Dialog - Mostra dialog box sopra altri contenuti
- Progressbar - Progress bar, animate e non
- Slider - Slider completamente customizzabile con varie features
- Tabs - Tabbed user interface handling, con content sia inline che ondemand

Effects

- Color Animation - Animazione attraverso il passaggio da un colore ad un altro
- Toggle Class, Add Class, Remove Class, Switch Class - Animazione basata sul passaggio da uno style ad un altro
- Effect - Varietà di effetti (appear, slide-down, explode, fade-in, etc.)
- Toggle - Switcha un effetto On oppure Off
- Hide, Show - Simile all'effetto di cui sopra

Utilities

- Position - Imposta la posizione relativa di un elemento rispetto ad un altro (alignment)

Allo scopo del nostro sviluppo, le parti di principale interesse erano:

1. Tutte le iterazioni, specialmente la proprietà “Resizable” che rende i campi di input ridimensionabili dall'utente.
2. I Tabs, tramite i quali si sarebbe realizzata il sistema di multiprofiling, associando ad ogni Tab un singolo profilo e quindi una singola sessione di dialogo con una SIB.
3. Dialog per comunicare immediate informazioni all'utente con finestre simili a PopUp, ed evitare di usare la funzione Js alert() che risulta piuttosto brutta.
4. Button per ottenere bottoni più attraenti.

Tutti gli elementi di jQuery-UI vengono poi disegnati seguendo dei temi definiti in appositi file CSS, similmente a quelli di dataTables. Il tema utilizzato nel nostro progetto è stato ideato da zero mantenendo l'assonanza con l'ideogramma del logo dell'applicazione (tonalità di grigio a gradienti, ombreggiature ed effetti “a specchio”, contorni tridimensionali).

Infine si voleva fare in modo che il contenuto di ogni profilo fosse scritto riempiendo template html precostituiti, con i 3 box della v.0.5.

Il meccanismo per la compilazione del template html viene fornito dalla libreria Underscore.js (si riveda Cap. 2, sotto-capitolo 2.3):

- PASSO 1) si scrive un file contenente lo scheletro del pezzo di pagina, includendolo all'interno di particolare tag <script> che reca come type “text/template”, che può essere costituito di parti statiche in normale Html, parti variabili con normalissimo codice Javascript oppure con i dati che verrano inseriti al momento della compilazione – le posizioni in cui inserire i parametri di compilazione si definiscono così <%= nomeParametro %>.
- PASSO 2) si compila il template con una chiamata di questo tipo:

  var template = _.template(“nomeTemplate”, { … parametri in formato JSON... });
- PASSO 3) si inserisce il codice Html prodotto dalla compilazione all'interno dell'elemento selezionato (nel nostro caso uno dei Tab) con una chiamata jQuery:

  $(“#numeroTab”).html(template);

Si notifica al lettore che il template è stato semantizzato utilizzando i tag strutturali dell'Html5 (article, header, aside e section).

A causa della quantità enorme di componenti da includere tramite Javascript nella v.0.6, specialmente il multiprofiling, si erano manifestati errori e incoerenze a Run-Time in gran quantità, a tal punto da rendere piuttosto arduo il debug.

4.8 La terza versione v.0.7 → Backbone.js

La soluzione più pratica quando ci si trova con problemi così complicati come quelli ereditati dalla v.0.6, è riscrivere tutto il codice da capo prendendo -uno alla volta- pezzi dal vecchio codice e testandoli mano a mano. Data questa evenienza e dato che si voleva riorganizzare tutto il codice dell'applicazione secondo un pulito approccio ad oggetti, questa versione -la v.0.7- avrebbe visto la reingegnerizzazione completa del software sottostante.

Il nuovo “core.js” doveva essere sviluppato sulla base del pattern MVC fornito dalla libreria Backbone.js.

Come descritto nel Cap.2, sotto-capitolo 2.3, un pattern MVC struttura l'applicazione in 3 blocchi, funzionanti in modo asincrono, in cui ogni blocco è assegnato ad un compito specifico:

- il Model contiene i dati utili all'applicazione, i metodi per accedervi e lavorarci;
- il View visualizza i dati contenuti nel model e si occupa di manifestarli direttamente all'utente o ad altri oggetti;
- il Controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti ;

Per essere corretti, bisogna dire che il Controller fornito da Backbone non è così utile, tant'è che nell'ultima versione è stato rinominato in “Router”: per questo la parte di input dell'utente viene normalmente integrata nella View. Potremmo dire che invece di un sistema MVC, l'applicazione monta un pattern MV.

Se il lettore è interessato, una semplice ma ottima guida all'utilizzo di Backbone.js, la stessa che è stata letta dal sottoscritto, è “Backbone Tutorials. Beginner, Intermediate and Advanced.”[33]

L'applicazione avrebbe istanziato un modello chiamato “SIB” per ogni profilo e in questo modello sarebbero state mantenute tutte le informazioni riguardanti la relativa sessione di dialogo:

33 “Backbone Tutorials. Beginner, Intermediate and Advanced.”

1. Indirizzo e porta della SIB (che erano variabili globali nelle versioni precedenti).
2. I dati attualmente dell'ultima richiesta dell'utente.
3. Il numero del profilo.

Ad essi si sarebbero aggiunti un insieme di metodi, uno per ogni singolo comando alla SIB (connect, insert, query, etc.).

Ogni Model “SIB” è associato ad una View chiamata “Profile”. Essa contiene le seguenti proprietà:

1. il Model associato, avendo accesso a tutte le sue proprietà e metodi.
2. L'elemento della pagina su cui scrivere, ossia il contenuto del Tab con il numero di profilo corrispondente.
3. Una serie di eventi, con associata la funzione handler.

I suoi metodi sono invece:

1. Render : disegna lo stato di dialogo con la SIB all'interno del Tab compilando il template con Underscore.js. E' la funzione-cuore della View.
2. Le funzioni di handling collegate ai bottoni delle form della pagina, che chiamano dentro di se i metodi del modello per inviare comandi alla SIB.
3. Funzioni che contribuiscono al rendering grafico.

Infine è stato aggiunto un footer alla pagina, contenente le informazioni di chi ha contribuito a creare l'applicazione e raccomandazioni sui browser da usare: Firefox e Chrome. L'applicazione dovrebbe funzionare bene anche su Opera e Safari. Per Internet Explorer si aspetterano versioni future che sfrutteranno una libreria chiamata modernizer.js. (si veda Cap.5)

A questo punto dello sviluppo, il progetto sarebbe già potuto essere pubblicato su Sourceforge:: normalmente le applicazioni che sono in vicine alla fine nel processo di produzione possono essere pubblicate ad un numero ristretto di utenti in quelle che vengono definite versioni alpha, e ad un numero più esteso in quelle che vengono definite versioni beta, così da avere dei “tester” dell'applicazione. Questa versione v.0.7 è stata quindi ribattezzata in “versione alpha”.

4.9 Versione v.0.8 → SPARQL e qName

Nel Cap. 1, sotto-capitolo 1.3, si è discusso di cosa sia lo SPARQL.

Le potenzialità del linguaggio lo rendono uno strumento di interrogazione della SIB molto più efficiente delle normali query RDF, giacchè la quantità di dati che si vogliono ottenere non è legata alla struttura degli statement ma è dinamica, inoltre questi dati possono essere combinati arbitrariamente per costruire nuovi tipi di frasi.

Perchè lo SPARQL potesse essere implementato nell'applicazione, era prima necessario modificare le librerie del dipartimento in modo da implementare nella classe KPICore la funzione querySPARQL, che prende come unico parametro la stringa rappresentante il comando SPARQL,richiede alla classe XMLTools di generare la relativa domanda SSAP e la invia alla SIB, mentre nella classe SSAP_XMLTools andava aggiunta una funzione con lo stesso nome, deputata a generare la richiesta SSAP, e una funzione capace di fare il parsing del messaggio di ritorno e ritornare il risultato in un'array bi-dimensionale (o matrice). E' stata aggiunta anche una piccola funzione di utility.

Il passo successivo è stato quello di creare una pagina AJAX che effettuasse il comando di query in SPARQL utilizzando le nuove funzioni della libreria, per ritornare una tabella html contenente il risultato.

Sulla pagina si è aggiunto un nuovo box sottostante quello dei comandi semplici, con la maschera per l'inserimento del comando SPARQL e il bottone per l'invio. Al bottone si è associato un nuovo metodo della classe SIB (ossia il Model) e il lavoro era completato.

A questo punto l'utente avrebbe potuto ottenere tutto il contenuto della SIB inserendo nella textarea una query banale come questa:

*SELECT **

WHERE{

?subject ?predicate ?object

}

Nonostante il guadagno in termini di funzionalità portato al progetto dall'implementazione delle query SPARQL, il carico di lavoro necessario è stato più che affrontabile.

Lo stesso discorso non si è potuto fare per lo sviluppo della funzionalità successiva, ossia il sistema dei qNames.

Nel Cap. 1, sotto-capitolo 1.3, si da una spiegazione preliminare di cosa sia uno “spazio di nomi”.

Con i qNames si è voluto sollevare il lettore dal dover leggere gli URI per esteso, data la loro

incomprensibilità implicita, facendo in modo che la lettura di un Statement fosse quanto più simile alla lettura di una frase umana.

Il sistema dei qName si basa sull'associazione di un prefisso all'URL iniziale contenuto in un URI, tutti questi prefissi e i relativi URL vengono salvati in una tabella con una piccola descrizione della tipologia di informazioni che esso rappresenta. Per esempio a *http://www.w3.org/1999/02/22-rdf-syntax-ns#* viene sostituito il prefisso “rdf”. La tabella si chiama qTable e appare all'utente nell'angolo in alto a destra dell'applicazione in un box minimizzabile, fissato allo scrolling della pagina (usando la proprietà position: fixed) e sovrastante tutto il resto dell'applicazione (z-index = 3).

Si noti che tale tipo di sistema non lavora direttamente con la SIB, ma solo a livello superficiale. Tramite esso -di fatto- si fa un'astrazione dei dati che lavorano con l'applicazione sottostante.

Si deve perciò estendere l'applicazione in una struttura multi-layer, che rispecchi questo schema: (Fig. 4.7)

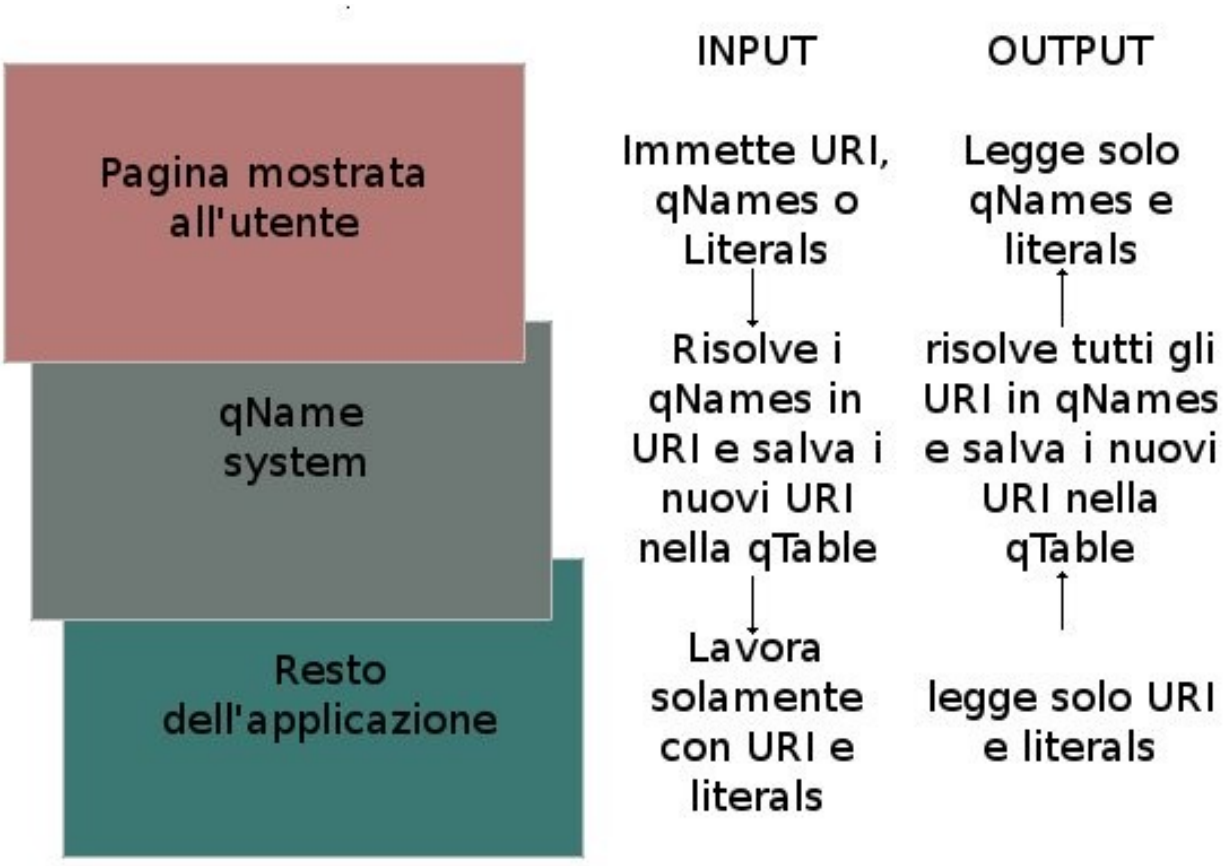


Fig. 4.7: qName System

La realizzazione fisica del sistema dei qName è stata fatta su un file separato dal “core.js” e chiamato, appunto, “qName.js”.

All'interno del file è presente un altro sistema MV, in cui il Model “Qname” contiene un array con

tutti gli elementi della qTable e due metodi:

1. checkSIB : viene chiamato all'apertura di ogni connessione con una nuova SIB e riceve come parametri indirizzo e porta della stessa. Effettua una chiamata AJAX ad una nuova pagina php chiamata “qNameChecker.php” che fa il parsing di tutte le triple contenute nella SIB per calcolare tutti i qName necessari; ritorna in formato JSON un array con prefisso, qName e descrizione per ogni entry.
2. addQName : viene chiamata in seguito all'immissione di un nuovo URI da parte dell'utente nelle form della pagina. Effettua un AJAX ad una pagina -più semplice della precedente- chiamata “qNameSingleChecker.php”.

Il funzionamento della pagina “qNameChecker.php” è piuttosto complesso, perlomeno più delle altre pagine AJAX.

A questo titolo si riporta il lavoro svolto:

```
<?
    /** checks the SIB for all available qnames, and returns an array with prefix, description and url for every qname in JSON format
    *       @author Katuiros
    *       @date 24/09/2012
    */                                          → inclusione delle librerie
  require_once "../lib/KPICore.php";
  require_once "../lib/SSAP_XMLTools.php";
  //Utilities                                   → funzioni di supporto
  function checkPresence($array2d,$string){     → controlla la presenza di una stringa
      foreach($array2d as $row){                → il controllo viene fatto su una matrice
          if($row['URL']==$string) return true;
      }
      return false;
  }
      function getPrefix($url){                 →estrae il prefisso dall'URL
      $tempArray=explode("/", $url);
```

```
$amount=0;
foreach($tempArray as $urlElem) $amount++;
for($counter=3; $counter<$amount; $counter++){
        if (preg_match('/[A-Za-z]/', $tempArray[$counter])) {
                $fragPos=strrpos($tempArray[$counter], "#");
                if ($fragPos> 0) $prefix= substr($tempArray[$counter], 0, $fragPos);
                else $prefix=$tempArray[$counter];
                return $prefix;
        }
}                                                   → il valore è l'ultimo nome prima del #
if (($prefix==null)||($prefix=="")) {               → o, in alternativa, il nome del dominio
        $fragPos=strrpos($tempArray[2], "#");
        if ($fragPos> 0) $prefix= substr($tempArray[2], 0, $fragPos);
        else $prefix=$tempArray[2];
        return $prefix;
}
}
function getTitle($Url){                            → estrae descrizione dall'URL
$str = file_get_contents($Url);
if(strlen($str)>0){
        preg_match("/\<title\>(.*)\<\/title\>/",$str,$title);
        return $title[1];
}
}
 $loadAddress = $_POST['addr'] ;                    → parsing dei parametri forniti all'AJAX
 if (($loadAddress=="null")||($loadAddress=="*")) die("wrong call!");
 $loadPort = $_POST['port'] ;
```

```
if (($loadPort=="null")||($loadPort=="*")) die("wrong call!");
$queryString= "SELECT * WHERE { ?a ?b ?c }";
                                                    → si istanzia KPICore
$KPAgent = new KPICore($loadAddress,$loadPort,"",true,true);
$KPAgent->join();
$xmlTools = new SSAP_XMLTools();
$result=$KPAgent->querySPARQL($queryString); → si ottiene contenuto SIB con query SPARQL
$array=$xmlTools->parseSPARQLQueryResult($result);
$newArrayCounter=0;
$newArray=null;
$rowCounter=0;
foreach($array as $currentRow){                    → viene parsato il contenuto alla ricerca di URI
if($rowCounter==0) {
        $rowCounter++;
        continue;
}
foreach($currentRow as $cellValue){
        trim($cellValue);
        //Se non è un URI continua il loop
        if(substr_count($cellValue, "http://")<1) continue;
        //Controlla la posizione del cancelletto
        $fragPos=strrpos($cellValue, "#");
        //Deduce il normal URI (o URL)              → se trova un URI ne deduce l'URL
        if($fragPos>0) $normalUri=substr($cellValue, 0, $fragPos);
        else continue;
        //Controlla se il normal URI è già presente nell'Array da ritornare
        if($newArray==null) {                      → caso base:se l'array è vuoto, si salva il valore
```

```
                $prefix=getPrefix($cellValue);
                $title=getTitle($cellValue);
                $newArray= array (array('PREFIX' => $prefix,'DESCRIPTION' => $title,'URL' => $normalUri));
                $newArrayCounter++;
            }                                   → si controlla sull'array la presenza del valore
            else if(!checkPresence($newArray,$normalUri)) {  → se non è presente:
                $prefix=getPrefix($cellValue);          → si estrae il prefisso
                $title=getTitle($cellValue);            → si estrae la descrizione
                $newArray[$newArrayCounter]=array('PREFIX' => $prefix,'DESCRIPTION' => $title,'URL' => $normalUri);  → si inserisce il valore nell'array
                $newArrayCounter++;
            }
        }
        $rowCounter++;
    }
    $KPAgent->leave();
    echo json_encode($newArray);                → l'array viene ritornato in formato JSON
?>
```

La View si chiama invece “QNameTabler ” e contiene pochi elementi:

1. Il modello associato e l'elemento della pagina su cui scrivere: un pannello nella parte superiore della pagina.
2. L'evento “redraw”, generato ad ogni cambiamento del modello.
3. La funzione di render che ridisegna tutta la pagina.

Model e View sono istanziati una volta sola in tutta l'applicazione e sono quindi unici per tutti i profili.

Infine, il file core.js è stato semplificato, estraendo tutte le funzioni non direttamente necessarie al

funzionamento del sistema Model-View; queste funzioni di supporto sono state poi spostate in un altro file chiamato “utilities.js”. Fra queste funzioni, sono di particolare nota:

1. trimSpaces : ripulisce i valori inserite in input dall'utente di spazi iniziali e finali;
2. getRandomColor : genera automaticamente una stringa contenente un colore casuale, da utilizzare per il sistema dei qName. Il codice della funzione è stato inventato da Paul Irish[34];
3. isHashURI : determina se un valore è un URI in codifica Hash;
4. isQName : determina se l'argomento è un qName;
5. checkURIPresence : prende come parametro una matrice (array a 2dimensioni, tipicamente rappresentante una tabella) e una stringa e controlla se essa è presente nella matrice;

In conclusione, la versione v.0.8 assume anche il nome di versione “beta”.

Gli utenti dell'applicazione vedranno questo: (Fig. 4.8)

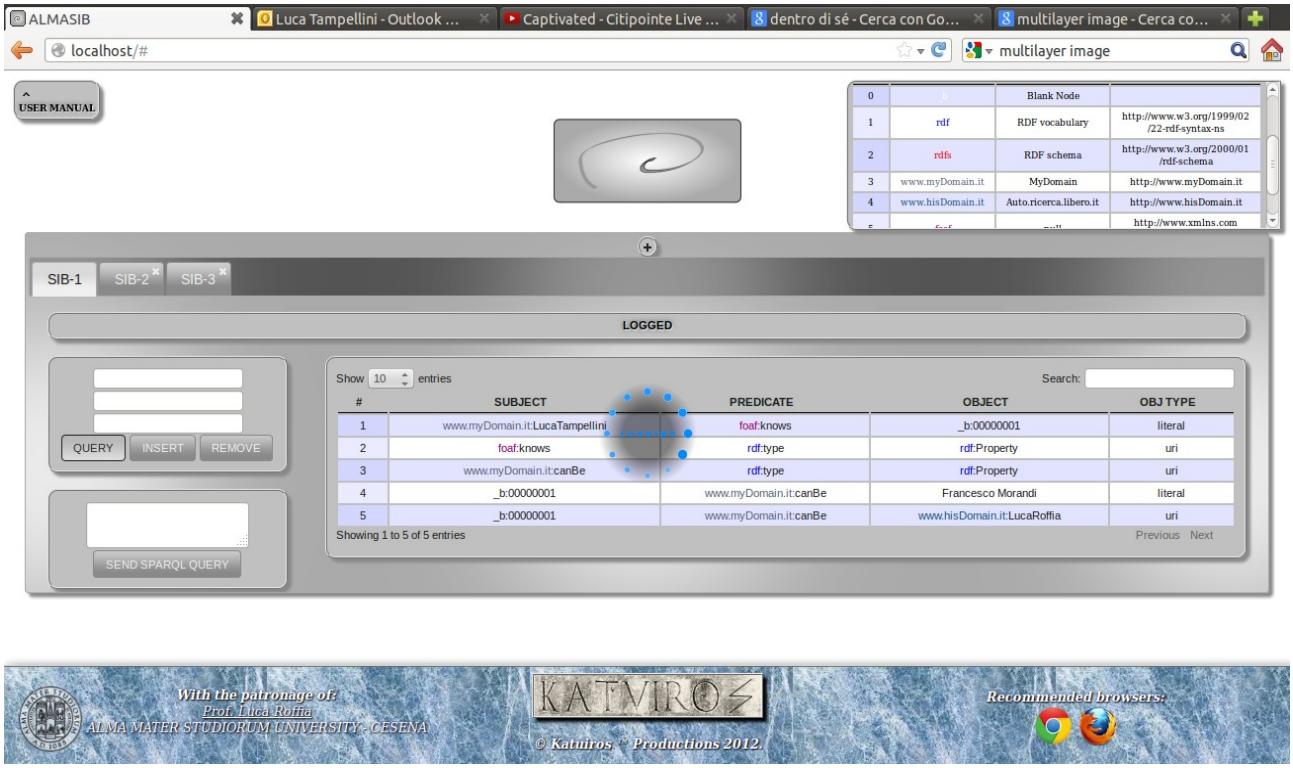


Fig. 4.8: ALMASIB v.0.8

In alto a sinistra si può notare un ulteriore pannello chiamato “User Manual”, che andrà a contenere con il rilascio della versione v.0.8.3 il contenuto dell'Appendice A di questa tesi.

CAP.5 FUTURO

34 http://paulirish.com

DEL PROGETTO

5.1 Pubblicazione su Sourceforge

Come è stato spiegato nel Cap. 3, l'ambizione finale del progetto, dichiarata fin dall'inizio, era quella di ottenere un insieme applicativo funzionante da distribuire pubblicamente tramite Sourceforge[35].

Sourceforge è una piattaforma per lo scambio e lo sviluppo in cooperazione di progetti Open-Source su larga scala, la più importante al mondo in questo campo: vanta più di 2 milioni di utenti.

Per quanto soprascritto, il progetto assumerà obbligatoriamente la licenza open-source che priva il sottoscritto di qualsiasi tipo di diritto sul software stesso: la particolarità delle licenze open source è che gli autori invece di vietare, permettono non solo di usare e copiare, ma anche di modificare, ampliare, elaborare, vendere e quant'altro.

In che modo può essere dunque utile pubblicare un progetto open source, dato che -a prima vista- non c'è nessun guadagno?

Il guadagno è tutto in quel piccolo footer in cui sono contenute le informazioni di contatto del sottoscritto: se l'applicazione raggiunge un numero considerevole di utenze, altrettanto grande è la fama che si può ottenere. La fama guadagnata può garantire, in un secondo momento, partecipazioni a progetti più importanti che garantiscano altri tipi di ritorno...

Citerò a questo titolo le parole di un grande personaggio, tale “Zarathustra”:

“ .. chi vuol essere il primo, si ponga come l'ultimo ..”

Essendo un progetto open-source, la pubblicazione in versione beta invece che in versione completa è regolare.

Ciononostante è ambizione personale del sottoscritto in accordo con quella del prof. Roffia, continuare lo sviluppo del progetto in seguito alla pubblicazione di questa stessa tesi e il completamento della laurea.

In particolare per la versione v.1.0 si prospettano due nuove funzionalitò:

1. Traslare il sistema dei qName in un “Web Worker”.
2. Implementazione del comando Subscribe.

35 http://sourceforge.net/

Un Web Worker[36] è un modo per eseguire un blocco di codice Javascript in un processo separato rispetto a quello principale usato dal browser, ovvero in un “Tread” in background.

In alcuni casi può capitare di dover gestire l'esecuzione di un processo particolarmente lungo o oneroso come, nel nostro progetto, il rintracciamento delle descrizione di ogni qName tramite l'apertura del relativo URL: l'interfaccia utente non risponde fino al termine dell'operazione e l'utente non ha alcun modo per annullarne l'esecuzione.

Questo scenario è ideale per l'uso di un Worker: il processo di elaborazione dei qName viene eseguito in un Thread separato rispetto a quello utilizzato dal browser per la visualizzazione della pagina, così da mantenere attiva l'interfaccia dell'applicazione ed eventualmente notificare all'utente lo stato di avanzamento dell'operazione, dandogli la possibilità di interromperla.

La comunicazione tra pagina e Worker avviene usando le “messaging API” : con il metodo *postMessage(...)* inviamo risultati e notifiche del Worker all'applicazione principale, la quale potrà gestire questi messaggi in una o più funzioni handlers collegate all'evento *onmessage* .

Per quanto riguarda la funzionalità di Subscribe, l'implementazione necessita a sua volta di un altro Web Worker dedicato esclusivamente a questo scopo. Il codice di questo Web Worker dovrà eseguire quanto descritto nella Tab. 2.1- del Cap. 2, sotto-capitolo 2.1 - : avvierà un loop infinito con la funzione *setInterval()* dentro al quale rieseguirà periodicamente query di tutti i dati della SIB, nel momento in cui i dati ottenuti risultassero diversi, l'applicazione principale verrebbe notificata del cambiamento e obbligata ad auto-aggiornarsi.

Infine, come accennato al Cap.4, sotto-capitolo 4.8, l'applicazione dovrebbe essere resa portabile per tutti i tipi browser, specialmente in vista di un utilizzo futuro dei Web Workers, che sono una caratteristica recentissima portata con Html5 e non sono ancora utilizzabili da tutti i browsers. Al nostro soccorso viene una libreria chiamata “Modernizr.js”[37] che determina quali delle caratteristiche Html5 e CSS3 possono funzionare sul browser dell'utente.

Modernizr testa più di 40 funzionalità di prossima generazione, poi crea un oggetto JavaScript (chiamato appunto “Modernizr”) che contiene il risultato di questi test come valori boolean. Aggiunge classi ai tag dell'output Html basandosi su quali funzionalità sono o non sono supportate sul Client.

Ecco una lista completa degli elementi di cui Modernizr si preoccupa:

---

36 http://www.w3.org/TR/workers
37 http://modernizr.com/

Tab. 5.1: Elementi valutati da Modernizr.js

•	@font-face	•	CSS 3D Transforms
•	Canvas	•	Flexible Box Model
•	Canvas Text	•	CSS Transitions
•	WebGL	•	Geolocation API
•	HTML5 Audio	•	Input Types
•	HTML5 Audio formats	•	Input Attributes
•	HTML5 Video	•	localStorage
•	HTML5 Video formats	•	sessionStorage
•	rgba()	•	Web Workers
•	hsla()	•	applicationCache
•	border-image	•	SVG
•	border-radius	•	Inline SVG
•	box-shadow	•	SVG Clip paths
•	text-shadow	•	SMIL
•	Multiple backgrounds	•	Web SQL Database
•	background-size	•	IndexedDB
•	opacity	•	Web Sockets
•	CSS Animations	•	hashchange Event
•	CSS Columns	•	History Management
•	CSS Gradients	•	Drag and Drop
•	CSS Reflections	•	Cross-window Messaging
•	CSS 2D Transforms	•	Touch Events

5.2 Possibili scenari applicativi

I probabili scenari applicativi che si delineavano al momento della consegna della versione alpha nelle mani del prof. Roffia prospettavano l'uso congiunto ad un insieme di sensori collegati tramite sistema wireless alla rete, identificati da un singolo URI, un indirizzo e una porta e capaci di montare una micro-installazione di una SIB Smart-M3. Tali SIB verrebbero rifornite in real-time, da un'altra applicazione realizzata appositamente, delle informazioni del sensore, ad esempio: temperatura, volume dell'acqua, quantità di luce, suono, ecc.

In questo scenario, l'applicazione permetterebbe all'utente di collegarsi ad ognuna delle SIB di

questi sensori e ottenerne le informazioni.

Queste informazioni potrebbero essere liberamente associate tramite opportuna richiesta SPARQL per dare all'utente vari tipi di prospetto del luogo (o dei luoghi) monitorati dai mircrosensori, tutto questo mentre l'utente è in una posizione arbitrariamente remota rispetto agli stessi.

Si osservi questo esempio:

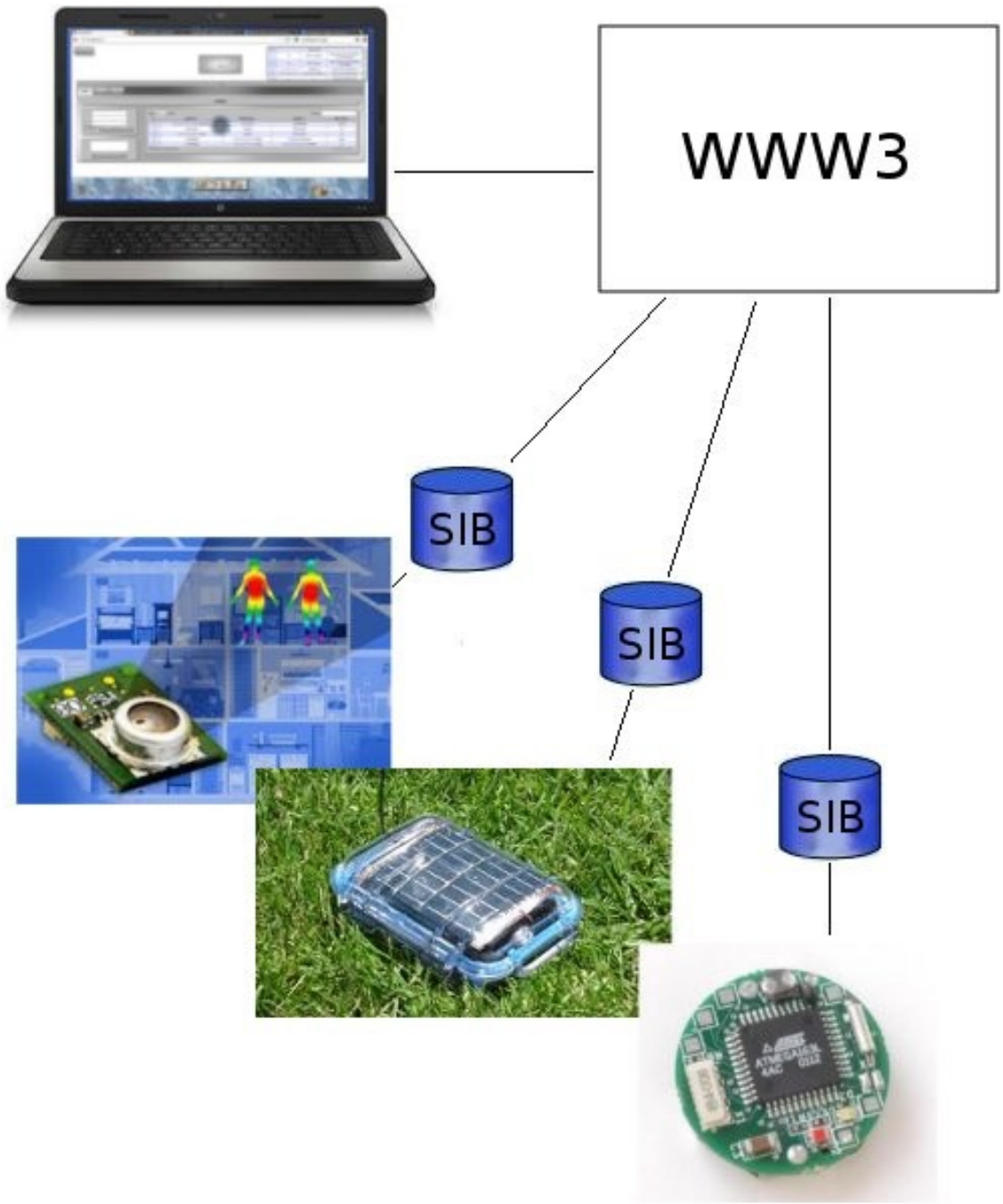


Fig. 5.1 : scenario applicativo → sistema di sensori e SIB.

Un applicazione di questo tipo potrebbe essere integrata negli sforzi di ricerca che l'Università ALMA MATER STUDIORUM sta effettuando sui Wireless Sensor Network[38]

Nella visione del sottoscritto, un progetto del genere diventerebbe alquanto più fruttuoso se utilizzato come motore di ricerca per valutare tutto il web: si immagini di poter catturare tutte le descrizione di tutte le ontologie possibili dal ***www3*** e inserirle in un'insieme di SIB collegate in un unico -enorme- SmartSpace.

Su queste SIB si potrebbe operare una ricerca per ottenere, specialmente con query SPARQL, un insieme di statement con dati estratti da varie pagine web. Per esempio: si potrebbe valutare queste SIB per sapere quale prezzo pagheremmo per comprare il modello XYZ di una macchina della società ABC in Italia e paragonarlo al prezzo a cui è venduta nel paese di produzione, sfruttando dati estratti da migliaia di pagine (quella della società ABC, delle varie concessionarie, delle riviste di automobili, etc.)

Perchè si possa realizzare uno scenario del genere è prima necessario comprendere come si possono estrapolare i dati dalle pagine del web.

5.3 GRDDL

GRDDL[39] (Gleaning Resource Descriptions from Dialects of Languages) abilita gli utenti -che nel nostro caso sarebbero applicazioni automatiche che parsano tutte le pagine del web per riempire le SIB- ad ottenere triple RDF fuori da documenti XML (chiamati dialetti XML), particolarmente da documenti XHTML[40] .

Nel campo dei GRDDL si usano queste terminologie:

- GRDDL-aware agent: un agente software capace di riconoscere le trasformazioni GRDDL ed eseguirle per estrapolare triple RDF.
- GRDDL Transformation: un algoritmo per ottenere triple RDF da un documento.

GRDDL divenne una raccomandazione W3C dall'11 settembre 2007. In questo documento standard, GRDDL è così definito:

> *“ GRDDL è un meccanismo per ottenere descrizioni di risorse da dialetti di linguaggi. GRDDL introduce un vocabolario di markups basati su standard*

38 http://middleware.unibo.it/node/694
39 http://www.w3.org/2004/01/rdxh/spec
40 http://www.w3.org/TR/xhtml1/

già esistenti per la dichiarazione di dati compatibili con RDF all'interno di un documento XML e per collegarli ad algoritmi (rappresentati tipicamente in XSLT), per estrarre questi dati dal documento."

In buona sostanza GRDDL automatizza l'estrazione degli RDF dalle pagine del Web.

Se noi sviluppassimo applicazioni che implementano l'interfaccia di GRDDL-aware agent, potremmo avviarle su server dedicati su cui siano presenti le SIB, e da li esse processerebbero tutto il Web, pagina per pagina, ricercando descrizioni RDF da inserire in Smart-M3. A questo punto l'insieme di SIB sarebbe accessibile direttamente da un'evoluzione della nostra applicazione.

5.4 I problemi di Google

La domanda di fondo che il lettore di questa tesi si sarà posto, molto probabilmente, è: perchè dovremmo utilizzare nuovi sistema di ricerca sul Web quando abbiamo già Google[41]?

Il problema di fondo di Google è che nonostante egli sia bravissimo a trovare tutti i possibili match per una selezione di parole chiave inserite nel suo motore e lo faccio in un tempo veramente esiguo, i risultati effettivamente ritornati all'utente sono -tipicamente- migliaia se non milioni di pagine. Risulta quindi impossibile per l'utente scandagliare tutti i risultati alla ricerca del contenuto che lo stesso si era immaginato.

E' necessario quindi un sistema di raffinazione dei risultati basato sulla definizione del tipo di contenuto che l'utente vuole, esattamente ciò che permette tutto l'insieme di concetti che ruotano attorno al progetto da noi sviluppato. Questo meccanismo di raffinazione dovrebbe lavorare a valle delle ricerche effettuate da Google: prima si ottengono 62 000 000 di pagine e poi delle applicazioni intelligenti, le GRDDL-aware agent del sotto-capitolo 5.3, le parsano in tempo reale estraendone i dati con le relative descrizioni RDF, ricombinandoli secondo il desiderio dell'utente.

Ricapitoliamo in 3 punti cosa si guadagnerebbe con meccanismi di ricerca basati su statement RDF lavoranti a valle del già esistente motore Google:

1. Raffinazione dei risultati ottenuti dalle ricerche sintattiche (per parole chiave).
2. Integrazione dei servizi forniti dai vari siti in nuovi servizi di più alto livello.
3. Estrapolazione dei dati dalle varie pagine senza che l'utente debba visitarle una per una.

41 http://www.google.org/

CAP.6 CONCLUSIONI

6.1 Considerazioni finali

In un analisi critica a posteriori, si potrebbe dire che risultati migliori si sarebbero ottenuti se al posto di uno sviluppo per “prototipazione” si fosse sviluppato il progetto con metodo “top-down”, ma questo avrebbe richiesto che il sottoscritto avesse una completa conoscenza dello scenario di riferimento, delle tecnologie in gioco (RDF, RDFS e OWL) e della totalità delle specifiche fin dall'inizio.

Essendo il web semantico stesso una novità per me, l'attività di sviluppo era necessariamente accompagnata ad una documentazione progressiva.

La struttura MVC non è stata realizzata come sarebbe risultata se si fosse utilizzata una metodologia di sviluppo più ortodossa; inoltre il risultato finale assomiglia molto a quello che si chiamerebbe un “Front Controller”: Il Front Controller Pattern è un modello di progettazione per il software, in particolare si applica alla progettazione di applicazioni web (come nel nostro caso), e si fonda sul concetto di convogliare tutti gli ingressi dell'utente in un punto di ingresso centralizzato per la gestione delle richieste.

In siti Web complessi ci sono procedure che possono risultare estremamente simili quando si processano gli Input dell'utente: nella nostra applicazione, infatti, la lettura delle form di ingresso per i comandi Query (RDF), Insert e Remove è praticamente uguale!

Queste procedure possono essere definite con funzioni Javascript uniche, che vengono poi prototipate opportunamente a Run-Time nel comportamento specifico desiderato. Queste procedure, una volta definite, delegano poi ad altre funzioni la gestione restante per la richiesta.

Per quanto richieda una migliore ingegnerizzazione, l'applicazione -in ogni caso- fa tutto ciò che deve fare e lo fa in tempi decenti.

Generalmente credo di poter affermare che il risultato ottenuto sia piuttosto buono visti i brevi tempi di sviluppo.

Si darà quindi all'utente un benchmark delle varie componenti dell'applicazione. (Tab. 6.1)

Si tenga in conto che, nella prospettiva futura di implementare il sistema qName nei Web Workers, la funzione di rintracciamento delle descrizioni per ogni singolo prefisso è stata disabilitata.

Tab. 6.1: Benchmark

COMPONENTE	TEMPO MEDIO DI ESECUZIONE	COMMENTI
Apertura dell'applicazione	0.5 sec	Ottimo
Connessione alla SIB	2 sec	Buono
Disconnessione	Immediata	Ottimo
Query (RDF) di tutte le triple della SIB (circa 5000)	3 sec	Buono
Insert	0.5 sec	Ottimo
Remove	0.3 sec	Ottimo
Query (SPARQL) di tutte le triple della SIB (circa 5000)	3 sec	Buono
Ritardo dovuto alla valutazione dei qNames (senza descrizione dei prefissi)	2 sec	Insufficiente
Ritardo indotto dalla presenza di dialoghi con altre SIB	Pressochè nullo	Ottimo
Riordinamento delle tabelle con 5000 entries	1 sec	Ottimo
Uso della casella di ricerca su tabelle con 5000 entries	2.5 sec	Buono

6.2 Ringraziamenti

In primo luogo vorrei ringraziare i miei genitori: mio padre Gerardo Tampellini e mia madre Sabine Kranz; per tutti i sacrifici che avete affrontato perchè avessi questa laurea, per avermi supportato moralmente e affettivamente.

Al prof. Luca Roffia, per avermi dato la possibilità di fare questa tesi, per me davvero appassionante; per essere stato così paziente e disponibile quando avevo delle domande.

Al correlatore Francesco Morandi per l'aiuto tecnico durante le fasi di sviluppo.

A mia sorella Laura Tampellini.

A tutti i famigliari che mi sono stati vicini in quesi mesi, in particolare mio cugino Dr. Ing. Mag. Mattia Tampellini, per avermi spiegato come condurre gli ultimi passi nell'università.

A mia nonna Maria Laghi, che mi ha lasciato poco prima che potessi consegnare questa Tesi. La dedico a te.

Ai volontari e ai responsabili del centro Caritas, presso cui ho svolto volontariato durante l'ultimo anno di università e nei mesi di sviluppo del progetto.

Ai miei amici, per avermi fornito la spalla.

A Don Pietro Fabbri e a Don Luigi Casamenti, le mie guide spirituali.

Al mio signore Gesù Cristo, per essere da sempre la mia unica forza, specialmente nei momenti più difficili.

A chi sta leggendo queste righe, per la pazienza che avrà avuto a capire quello che scrivo.

A tutti voi, di cuore, Grazie.

CAP.7 APPENDICI

7.1 Manuale d'utilizzo (APPENDICE A)

Introduzione:

Al momento della scrittura di questa tesi (ottobre 2012) il progetto si trova solamente su SourceForge.

Non ci sono Web server, almeno per il momento, a cui connettersi con richiesta HTTP per accedere all'applicazione... dovrete scaricare il tutto, installarlo sul vostro Web Server e accedervi in localhost.

Note: si raccomanda l'utilizzo di Firefox o Google Chrome per l'accesso all'applicazione.

Preparazione delle tool necessarie:

Se avete già un Web Server e avete PHP aggiornato, saltate questa fase e passate a “Download & installazione”.

Se avete sistema operativo linux, il pacchetto d'installazione delle componenti necessarie per il deploy dell'applicativo si chiama lAMP: aprite un terminale e digitate

sudo apt-get install tasksel
dopodichè:

sudo tasksel

Nella finestra che vi appare scorrete la lista con i tasti direzionali fino a “LAMP server”, selezionate con la barra di spazio e poi inviate con enter.

Durante l'installazione vi verrà richiesta la password di MySQL.

Riavviate il pc

Se avete sistema operativo Windows, il pacchetto che vi serve è WAMP. Trovate file e documentazione per l'installazione a: http://www.wampserver.com/en/

Download & Installazione:

Connettersi https://sourceforge.net/projects/almasib/files/latest/download?source=files e scaricare il file compresso nella posizione a voi più congeniale.

Estraete tutto dal file e copiatelo nella cartella di host di Apache (/var/www per sistemi linux).

Assicuratevi di non avere deployato in sotto-cartelle rispetto a quella di host, altrimenti dovrete attendere la v.0.9.0 a causa di un bug.

Avviate il vostro browser e digitate “localhost” nella barra degli indirizzi... et voilà!

Manuale d'utilizzo:

L'interfaccia è scomposta in una serie di pannelli (“Box”) sovrapposti. In alto a destra trovate il pannello dei qName, in alto a sinistra un pannello che contiene questo manuale d'utilizzo (dalla v.0.8.3). I due succitati pannelli sono minimizzabili se l'utente clicca sulle freccette contenute nell'header del pannello.

Al centro avrete il cuore dell'applicazione, organizzato in tab, da cui dialogare con una o più SIB in contemporanea.

Per avviare dialoghi con nuove SIB dovete aprire un nuovo profilo: cliccate sull'icona “+” cerchiata in alto e aprirete un nuovo tab.

Una volta in un tab spostate il mouse sul box “LOG-IN” e vi apparirà una maschera in cui inserire nell'ordine: indirizzo e porta della SIB a cui desiderate connettervi. Inseriteli e cliccate sul bottone “CONNECT TO THE SIB”. Se non vi appare un popup contenente un messaggio d'errore e il contenuto del Tab si srotola verso il basso, siete connessi.

Mentre vi connettete, ALMASIB va automaticamente a controllare il contenuto della SIB ed aggiorna coi dati ritrovati la tabella dei qName.

Al posto del box di “LOG-IN” ne troverete uno con scritto “LOGGED”; se volete disconnetervi spostate il mouse su di esso e cliccate sul bottone “DISCONNECT”.

Nella colonna di sinistra troverete due box per l'inserimento dei comandi: il primo contiene 3 campi, nell'ordine Soggetto – Predicato – Oggetto, e 3 bottoni di comando Query (RDF), Insert e Remove, che svolgono i relativi compiti descritti nella Tab. 2.1 del Cap.2. Se uno, o più, dei 3 campi non è settato oppure se ha come valore l'asterisco “*”, l'applicazione tratterà quell'elemento dello Statement come una wildcard, ossia vi sostituirà tutti i possibili valori che fanno match col resto dello Statement: se cliccate su Query con i campi vuoti, quindi, otterrette tutto il contenuto

della SIB; attenti a non fare la stessa cosa cliccando su Remove, o cancellerete tutti i dati della SIB. L'Update è temporaneamente disabilitato... lo troverete insieme alle Subscribe nella v.1.0. Il secondo contiene una textarea estendibile (per estendere spostate il mouse nell'angolo in basso a destra e trascinate a piacimento) e il bottone di “SEND SPARQL QUERY”.. si noti che il comando SPARQL che inserite nella textarea deve essere ben formattato, o l'applicazione vi ritornerà un popup di errore al momento della chiamata AJAX.

Nella zona principale del Tab, quella in basso a destra, c'è un box contenente il risultato della vostra ultima interrogazione alla SIB; appena vi connettete noterete scritto “waiting for commands”.

Inviando uno dei comandi di query dalla colonna di sinistra, i valori ritornati verranno visualizzati nell'area di risposta come tabelle interattive.

Queste tabelle possono essere riordinate cliccando su uno degli indici soprastanti la tabella; sono paginati (quindi potrete proseguire la lettura dei dati cliccando su next o previous) e la quantità di dati per pagina è regolabile con il menu a tendina in alto a sinistra; in basso a destra trovate informazioni sulla quantità di dati ritornati. La funzionalità più utile di queste tabelle, in ogni caso, è quella piccola barra di ricerca in alto a destra: qualsiasi valore inserito, tranne che si trattasse di un qName, farà partire automaticamente l'applicazione alla ricerca di tutte le sottostringhe che fanno match con la stringa che avete digitato; se inserite altre parole separate da spazio, la barra continuerà la propria ricerca dei match intersecando il risultato con quello della parola (o delle parole) precedenti.

Il sistema dei qName garantisce che sulla tabella appaiono solo URI leggibili nel formato *prefix:fragmentId* .

Nota: si consiglia all'utente di utilizzare, per comodità, i qName nei campi d'inserzione. Se un qName non fosse già presente, l'utente sarà obbligato ad inserirlo nella versione con URI completo una sola volta: il sistema sottostante lo riconoscerà automaticamente e lo salverà nella tabella.

Passiamo alla spiegazione della funzione principale: le query in SPARQL.

Guida SPARQL

Le query SPARQL si basano sul meccanismo del "pattern matching" e in particolare su un costrutto, il "triple pattern", che ricalca la configurazione a triple delle asserzioni RDF fornendo un modello flessibile per la ricerca di corrispondenze.

```
?titolo cd:autore ?autore.
```

In luogo del soggetto e dell'oggetto questo "triple pattern" prevede due variabili, contrassegnate con ?.

Le variabili fungono in un certo senso da incognite dell'interrogazione, *cd:autore* funge invece da costante: le triple RDF che trovano riscontro nel modello associeranno i propri termini alle variabili corrispondenti.

Per chiarire, ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM  <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
            ?titolo cd:anno ?ann .
}
```

L'analogia con SQL è lampante.

`PREFIX` dichiara prefissi e namespace.

`SELECT` definisce le variabili di ricerca da prendere in considerazione nel risultato (nell'esempio: `titolo, autore` e `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").

È inoltre possibile ricorrere a clausole FROM NAMED e alla parola chiave GRAPH per specificare più set di dati.

La clausola `WHERE` , infine, definisce il criterio di selezione specificando tra parentesi graffe uno o più "triple patterns" separati da punto fermo.

Applicando la query al set di triple del capitolo precedente, si ottiene il seguente risultato:

titolo	**autore**	**anno**
"Amber"	"Autechre"	1994

Il "binding" tra variabili e termini reperiti corrispondenti (in questo caso, un termine per ciascuna variabile) è reso in forma di tabella come un rapporto campo-valore: le righe rappresentano i singoli risultati, le intestazioni di cella rappresentano le variabili definite nella clausola SELECT , le celle i termini associati alle variabili.

La query precedente ha catturato esclusivamente le triple dotate di tutti e tre i termini richiesti (`titolo, autore, anno`), escludendo le triple che ne possedevano due soltanto (`titolo, autore`).

È possibile riformulare la query in modo più elastico, prevedendo l'eventuale assenza di alcuni

termini.

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM  <http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
            OPTIONAL {?titolo cd:anno ?anno}
}
```

Nell'esempio, il secondo pattern è dichiarato opzionale: l'informazione è aggiunta al risultato solo se disponibile, altrimenti le variabili compariranno prive di valore (unbound). Risultato della query:

titolo	autore	anno
"Permutation"	"Amon Tobin"	
"Bricolage"	"Amon Tobin"	
"Amber"	"Autechre"	1994

Le risorse sprovviste della proprietà anno sono mostrate ugualmente e le celle dei valori mancanti sono lasciate vuote.

Un altro modo per assicurare una certa elasticità nel reperimento dei dati è il seguente:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM  <http://cd.com/listacd.ttl>
WHERE{
                {?titolo cd:autore ?autore}
                UNION
                {?titolo cd:anno ?anno}
}
```

La parola chiave `UNION` esprime un OR logico: la query non si limita pertanto alle triple che soddisfano entrambi i *triple patterns*, ma cattura sia quelle che soddisfano il primo, sia quelle che soddisfano il secondo.

È possibile porre restrizioni sui valori da associare alle variabili. Ad esempio:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?anno
FROM  <http://cd.com/listacd.ttl>
WHERE {?titolo cd:anno ?anno.
            FILTER (?anno > 2000).
}
```

In questo caso, la restrizione è effettuata mediante l'operatore di confronto > : il filtro esclude i

termini che non soddisfano la condizione definita tra le parentesi tonde: il risultato in questo caso è nullo.

Nella sezione successiva, *11.4 Operators Definitions*, sono descritti gli operatori specifici del linguaggio: tra questi, mi limito a segnalare `regexp`, valido corrispettivo dei criteri di ricerca LIKE dell'SQL che permette di adoperare espressioni regolari per il matching dei letterali. Si consideri il seguente esempio:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM   <http://cd.com/listacd.ttl>
WHERE  { ?titolo cd:autore ?autore .
            FILTER regex(?autore, “^au”, “i”)
}
```

Il filtro seleziona, senza riguardo per maiuscole o minuscole, solo gli autori che iniziano per "au".

titolo	autore
"Amber"	"Autechre"

Come in SQL, è possibile escludere dal risultato i valori duplicati mediante la parola chiave DISTINCT, ad esempio:

```
SELECT DISTINCT ?titolo ?autore
```

Altri costrutti supportati da SPARQL per la manipolazione del risultato sono:

```
ORDER BY DESC(?autore)
LIMIT 10
OFFSET 10
```

L'espressione `ORDER BY` imposta l'ordine dei risultati della query: stando all'esempio, i risultati verranno presentati in ordine decrescente (`DESC`) in base alla variabile `?autore`.

`LIMIT` pone restrizioni al numero dei risultati, limitandoli, secondo quanto indicato nell'esempio, ai soli primi 10.

`OFFSET` permette di "saltare" un certo numero di risultati, escludendo, stando all'esempio, i primi 10.

7.2 URL, URI, URN (APPENDICE B)

Che cos'è un URI? (Uniform Resource Identifier)

URI è un metodo per identificare un'unità di contenuto su internet. Il contenuto può essere una pagina di testo, un'immagine, un file audio o video, o qualsiasi cosa esiste sul web. Inoltre si può associare un URI anche a risorse non presenti fisicamente su internet! Per esempio una persona o un idea possono essere identificate con un URI.

Tutti gli altri metodi (come URL e URN) sono sottoinsiemi di URI. (Fig. 7.1)

Che cos'è un URL? (Uniform Resource Locator)

URL è un tipo di URI e rappresenta l'indirizzo unico con cui si può accedere alla posizione dell'unità di contenuto sul Web. Per esempio “http://www.digitalpurview.com/online-task-manager/” è l'indirizzo di una pagina web intitolata “Online Task Manager” e usa il protollo HTTP per essere scaricata. Similmente, possiamo avere l'indirizzo per un'immagine, un file audio o video salvati in qualche Web server. L'indirizzo inizierebbe con ftp:// . Mailto è un protocollo per identificare indirizzi mail.

Che cos'è un URN? (Uniform Resource Name)

URN è l'identificatore nominale di qualsiasi entità rappresentabile informaticamente sul Web, senza che questa risorsa rappresentata debba essere necessariamente reperibile. L'idea di fondo per cui è stato creato l'URN era avere un modo per identificare risorse che si potevano spostare da un web server all'altro.

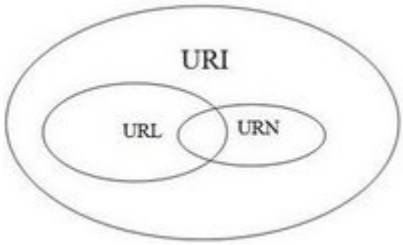


Fig. 7.1: URI, URL e URN

CAP.8 RIFERIMENTI & BIBLIOGRAFIA

[1] http://www.w3.org/People/Berners-Lee/ → Tim Berners Lee

[2] http://www.w3c.org → Homepage World Wide Web Consortium

[3] http://www.w3.org/1999/02/22-rdf-syntax-ns → RDF Syntax specification document, World Wide Web Consortium

[4] http://www.w3.org/2000/01/rdf-schema → RDF Schemx specification document, World Wide Web Consortium

[5] http://www.w3.org/TR/webont-req/ → OWL Web Ontology Language Use Cases and Requirements, World Wide Web Consortium

[6] http://purl.org/dc/elements/1.1/ → Dublin Core specification document, Dublin Core Metadata Initiative

[7] http://www.w3.org/2002/07/owl → OWL specification document, World Wide Web Consortium

[8] http://www.w3.org/2009/sparql/wiki/Main_Page → SPARQL Working Group Wiki , World Wide Web Consortium

[9] http://librdf.org → Redland RDF Libraries, Redland, Dave Beckett

[10] “Kritik der reinen Vernunft” → “Critica della Ragion pura”, Immanuel Kant, 1781 - Ed. Laterza, 2012

[11] “La Repubblica” → libro VII – Platone, 520 a.C.

[12] http://sourceforge.net/projects/smart-m3/ → Smart-M3 v.1.1 (beta) , description & download page, hannlain, ioliver, jhonkola, jpluoma, kirill krinkin, otyrkko

[13] http://sourceforge.net/projects/smart-m3/files/Smart-M3_B_v0.3.1-alpha/Smart-M3_B_v0.3.11.tar.gz/download → Smart-M3_B_v.0.3.11, Francesco Morandi

[14] http://sourceforge.net/projects/sm3-php-kpi-lib/ → Smart-M3 PHP KPI library, description & download page, davide ensini, marco piraccini, matteo bianchi

[15] http://docs.jquery.com/ → jQuery documentation page

[16] http://www.backbonejs.org → Backbone.js documentation page

[17] http://www.underscorejs.org/ → Underscore.js documentation page

[18] http://www.w3.org/TR/2011/WD-html5-20110525/ → Html5 specification document, World Wide Web Consortium

[19] http://www.w3.org/TR/css3-selectors/ → CSS3 specification document, World Wide Web Consortium

[20] http://bfwiki.tellefsen.net/index.php/Main_Page → Bluefish project wiki

[21] http://docs.gimp.org/2.8/it/ → GIMP project documentation page

[22] http://www.apache.org/ → Apache Software Foundation homepage, Apache Foundation

[23] http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html → Tim Berners Lee talk on the Semantic Web, Ted Talks

[24] “Ontologie per il Web Semantico” → Indrit Beqiri, 2012 - http://amslaurea.unibo.it/3581/

[25] http://www.w3.org/2004/02/skos/ → SKOS Simple Knowledge Organization System homepage, World Wide Web Consortium

[26] “A Developer's Guide to the Semantic Web” → Lyang Yu, 2011 - Ed. Springer – ISBN 978-3-642-15969-5

[27] “Smart-M3 Information Sharing Platform ” → “piattaforma di condivisione di informazioni Smart- M3” - Jukka Honkola, Hannu Laine, Ronald Brown, Olli Tyrkkö - *http://www.fruct.org/sites/default/files/Smart-M3_Information_Sharing_Platform.pdf*

[28] http://www.w3.org/RDF/ →Resource Description Framework wiki

[29] http://www.w3.org/TR/owl2-overview/ → OWL 2 Web Ontology Language Document Overview, World Wide Web Consortium

[30] http://www.w3.org/TR/XMLHttpRequest/ → XMLHttpRequest Level 2 specification document, World Wide Web Consortium

[31] http://datatables.net/ → dataTables project homepage, SpryMedia 2012

[32] http://jqueryui.com/ → jQuery-UI homepage, jQuery Foundation 2012

[33] “Backbone Tutorials. Beginner, Intermediate and Advanced.” → Thomas Davis, 2012 – Ed.

[34] http://paulirish.com → Paul Irish homepage

[35] http://sourceforge.net/ → sourceforge homepage

[36] http://www.w3.org/TR/workers → Web Workers specification document, World Wide Web Consortium

[37] http://modernizr.com/ → Modernizr.js homepage

[38] http://middleware.unibo.it/node/694 → Reliable communication for mobile MANET-WSN scenarios - A. Corradi, L. Foschini

[39] http://www.w3.org/2004/01/rdxh/spec → GRDDL specification document, World Wide Web Consortium

[40] http://www.w3.org/TR/xhtml1/ → XHTML specification document, World Wide Web Consortium

[41] http://www.google.org/ → Google foundation homepage