Scuola universitaria professionale
della Svizzera italiana

SUPSI

Corso di Laurea in Ingegneria Informatica

Progetto di Semestre 2011-2012

LOD Viz
Visualizzazione di open data

C07302

Studenti: Omar Davide Molteni
Stefano Mondini

Relatore: Lorenzo Sommaruga
Correlatore: Nadia Catenazzi

Data: 4 maggio 2012

Il Web è più un'innovazione sociale che un'innovazione tecnica.
L'ho progettato perché avesse una ricaduta sociale, perché aiutasse le persone a collaborare, e non come un giocattolo tecnologico.
Il fine ultimo del Web è migliorare la nostra esistenza reticolare nel mondo.

Tim Berners-Lee, *L'architettura del nuovo Web*
(Co-inventore del World Wide Web assieme a Robert Cailliau)

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Riassunto/Abstract

LOD Viz è un visualizzatore grafico di open data specificatamente realizzato per i dati del Laboratorio di Sistemi Semantici e Multimediali del Dipartimento Tecnologie Innovative della SUPSI.

Questo lavoro si inserisce nel contesto del più ampio progetto "Brick - Web di dati", fornendo un nuova modalità di accesso e navigazione dei dati basata su una visualizzazione grafica degli stessi, più intuitiva e di maggior impatto visivo.

Gli open data, rappresentati come XML-RDF, vengono caricati in formato JSON e interpretati dinamicamente dall'applicativo che è così in grado di mostrare le relazioni con le risorse selezionate. Visivamente tramite colori e frecce è facilmente distinguibile il tipo di relazione che intercorre tra di esse.

*****

LOD Viz is a graphic open data viewer specifically designed for the Semantic and Multimedia Systems Laboratory of the Department of Innovative Technologies at the SUPSI.

This work is part of the broader "Brick - Web of data" project, providing a new means of data access and navigation based on a graphical display of the data in a more intuitive way with a strong visual impact.

The open data, represented as XML-RDF, are dynamically loaded in JSON format and interpreted by the application to show the relationships with the selected resources. In a visual way through colors and arrows is easily distinguishable the type of relationship that exists among them.

Progetto assegnato

Descrizione

Il Laboratorio Sistemi Semantici e Multimediali (DTI-ISIN-LSMS) sta lavorando su un progetto di divulgazione delle tecnologie web semantico ed in particolare del "Web of Data".

Questo progetto mira a visualizzare i dati esistenti esposti precedentemente in RDF come open data - LOD (http://linkeddata.org).

Questo lavoro si focalizza su dati SUPSI (p.e. corsi, persone, competenze) che sono pubblicati sul web, collegati fra loro e da visualizzare con modalità grafiche appropriate in pagine interattive HTML5 e CSS3.

Compiti

- Analisi dei dati LOD esistenti
- Progettazione e sviluppo di front-end per presentazione e navigazione dati
  - Definizione di modalità di visualizzazione grafica
  - Sviluppo di pagine interattive corrispondenti
- Documentazione e presentazione del progetto

Obiettivi

- Acquisire competenze di sviluppo di applicazioni basate su tecnologie web e semantic web
- Completare lo sviluppo di un dimostratore delle potenzialità di Linked Data
- Portare a termine un progetto di sviluppo completo

Tecnologie

- HTML5 + JavaScript + CSS3
- XML/RDF

Capitolo 1

Introduzione

Il web, per come lo si conosce e per come è stato concepito fin dalla sua nascita, è un enorme insieme di informazioni e documenti. Negli anni si è espanso in maniera esponenziale e oggi è un mastodontico sistema che ingloba informazioni di ogni tipo in una miriade di formati diversi. Il problema è che i dati in quanto tali, essendo mascherati e presentati in base alle specifiche necessità dei vari siti web non sono disponibili direttamente. Se questo dal punto di vista di un singolo essere umano che legge una pagina web è un bene, per una macchina che esegue una scansione del web (un *crawler*) non è il massimo della semplicità.

In pratica il vero passo in avanti del web sarà la liberazione dei dati dai formati e dalle formattazioni. I dati nella loro forma più pura (*raw data*) possono essere connessi (*linked data*) e strutturati. Il significato di tutto questo è la possibilità di cercare le informazioni e di effettuare ricerche mirate che abbiano come obiettivo quello di rispondere a domande specifiche in un linguaggio il più naturale possibile per un essere umano, ma anche nel modo più semplice e interpretabile per una macchina. Tutto questo rientra in quello che viene descritto come web semantico.

1.1 Linked Data

L'interoperabilità è uno dei vantaggi più importanti del modello *Open Data*[1]. I dati, se isolati, hanno poco valore; viceversa, il loro valore aumenta sensibilmente quando *data set* differenti, prodotti e pubblicati in modo indipendente da diversi soggetti, possono essere incrociati liberamente da terze parti. Questo è alla base del processo di creazione di valore aggiunto sui dati: le applicazioni. Le applicazioni, di valore sociale e/o economico, sfruttano quello che può essere visto come un grande database aperto e distribuito per offrire viste e servizi. L'interoperabilità è dunque un elemento chiave di uno degli aspetti più innovativi offerti dagli open data: l'uso dei dati in modi e per scopi "inattesi", nuovi in quanto non previsti dai singoli enti e soggetti che pubblicano i "dati grezzi".

Per consentire il riuso dei dati occorre poter combinare e mescolare liberamente i data set. Occorre cioè collegare i dati tra loro, stabilendo un link diretto quando i dati (possibilmente provenienti da diverse sorgenti) si riferiscono a oggetti identici o comunque relazionati tra loro. Tale collegamento diretto si manifesta come la possibilità di "saltare" da un dataset all'altro, ad esempio quando si vuole accedere a dati (come i dettagli su una particolare entità) che non si posseggono all'interno.

[1] Open Data: http://it.wikipedia.org/wiki/Dati_aperti

Per fare questo, se i dati non sono "collegati" occorre in qualche modo creare questi link, processando i dati a mano o attraverso algoritmi ad hoc.

1.2 Il Web Semantico

Nel 2001 Tim Berners-Lee[2] e i suoi collaboratori pubblicarono un articolo chiamato *"The Semantic Web"*, in cui presentavano la loro idea di "una nuova forma di contenuto Web che fosse significativo per i computer scatenando così una rivoluzione di nuove possibilità". Negli ultimi anni quest'idea ha preso piede e anche nuove tecnologie sono diventate disponibili per costruire parte di questa visione. Purtroppo, per iniziare non è così facile, perché ci sono molti concetti con nomi e significati leggermente diversi e tecnologie con diversi nomi criptici, quindi cominceremo con alcune definizioni.

Il primo termine è appunto *Web Semantico*. Il web semantico descrive la visione che le macchine un giorno saranno in grado di comprendere il significato ("semantica") delle informazioni su Internet, ed essere in grado di "svolgere compiti automaticamente e individuare le relative informazioni sul conto dell'utente" (cfr. Wikipedia[3]). Ciò che è importante da capire, è che questo termine descrive un amalgama di concetti e tecnologie (simile al "Web 2.0") e non una singola tecnologia.

Un concetto tecnologico che fa parte della visione del semantic web è il collegamento dei dati che descrive "un metodo di pubblicazione dei dati strutturati, in modo che possano essere interconnessi e diventare così più utili".

I Linked Data di per sé non devono per forza essere dati pubblicamente disponibili, ma possono benissimo essere usati in privato, e per abbiamo bisogno di una maggiore definizione: Open Data . Esso descrive "una filosofia e la pratica richiedendo che alcuni dati siano liberamente accessibili a tutti, senza restrizioni di copyright, brevetti o altri meccanismi di controllo".

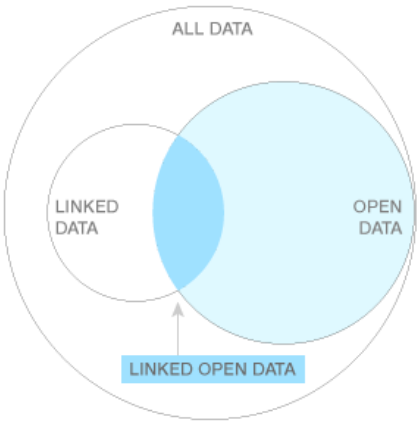


Figura 1.1: Connessione tra Linked Data e Open Data.

Il grafico qui sotto mostra la dimensione del web Linked Open Data, alla fine del 2010: ogni bolla è un sito a cui è possibile accedere tramite tecnologie Linked Open Data in modi simili, come si farebbe normalmente accedendo a un database.

[2]Tim Berners-Lee: inventore del World Wide Web.
[3]Web semantico: http://it.wikipedia.org/wiki/Web_semantico

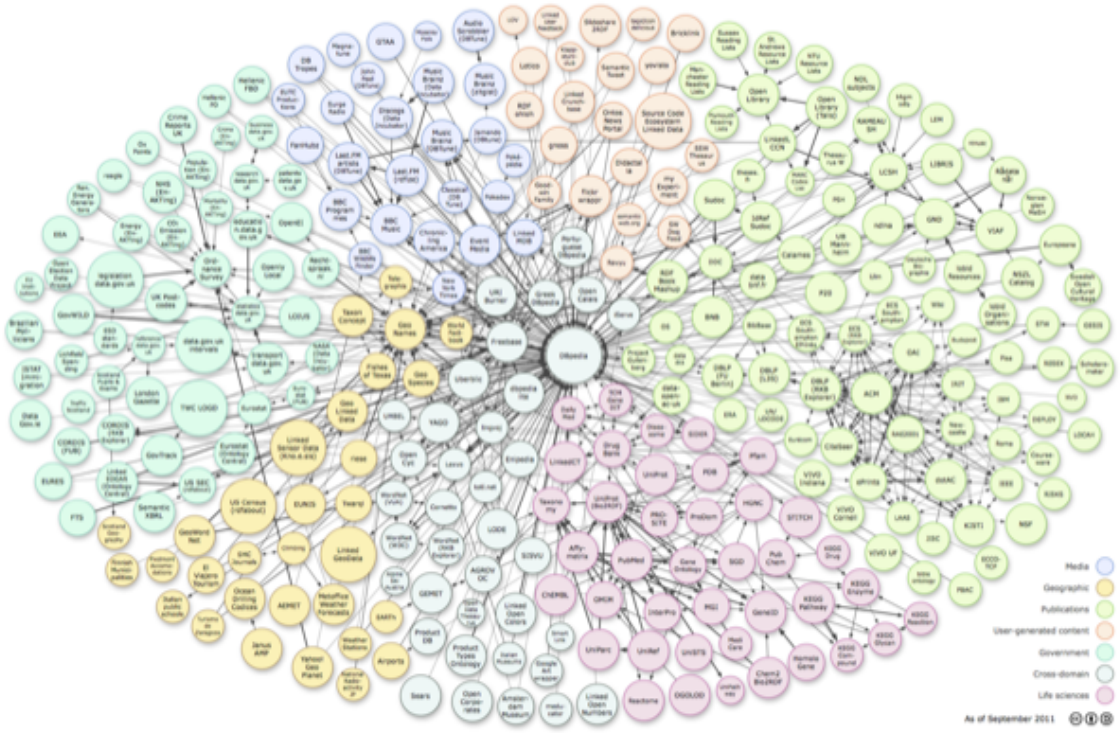


Figura 1.2: Set di dati pubblicati nel formato dei Linked Data e interconnessi con altri dataset.

Capitolo 2

Requisiti e Specifiche

Questo progetto richiede conoscenze di sviluppo di applicazioni basate su tecnologie web e *semantic web* come l'analisi del progetto *LOD* (Linked Open Data) e lo studio delle tecnologie associate. Si è reso inoltre necessario lo studio e documentazione dei linguaggi e degli strumenti da utilizzare, come HTML5, JavaScript e relative librerie.

Bisognava quindi sviluppare un visualizzatore grafico di open data con dati pubblici formattati secondo un'ontologia, già definita in un progetto precedente, e quindi identificarli e rappresentarli in forma visuale con le relative relazioni.

2.1 Obiettivi

- Acquisire competenze di sviluppo di applicazioni basate su tecnologie web e semantic web
- Studio e ricerca documentazione dei linguaggi e tecnologie da utilizzare (HTML5, JavaScript, jQuery, CSS3, JSON)
- Studio dello stato dell'arte: utilizzo di mappe visuali in altri siti (es. LinkedIn InMaps)
- Studio del progetto attuale Brick - Web di dati (link: `http://isin06.dti.supsi.ch/brick_web_di_dati`) e di un progetto simile EasyOnto sviluppato in Flash (link: `http://isin06.dti.supsi.ch:8500/EasyOnto/main.html`)
- Sviluppo di una pagina web interattiva che permetta di visualizzare e interagire con dati in formato JSON presenti nel progetto Brick
  - Possibile rappresentazione tramite bolle collegate con al centro la risorsa richiesta
  - Rappresentazione tramite riquadri suddivisi in sotto-riquadri con colori e dimensioni calcolati tramite media pesata

2.2 Tecnologie e software utilizzati

2.2.1 Tecnologie

- JSON

- HTML5
- CSS3
- JavaScript
- jQuery

Per una più completa descrizione delle suddette tecnologie si rimanda alla Sezione 3.4.

2.2.2 Software

- *JetBrains WebStorm*[1], IDE commerciale per JavaScript, CSS e HTML costruito sulla piattaforma IntelliJ IDEA.
- Browser: Mozilla Firefox, Google Chrome, Apple Safari, Opera
- Firebug, add-on di Firefox che permette il debug, la modifica e il monitoraggio di tutti gli aspetti di una pagina web

Per maggiori informazioni sulla compatibilità dei browser con il progetto LOD Viz e della fase di test si rimanda alla Sezione 5.4.

[1] JetBrains WebStorm: http://www.jetbrains.com/webstorm/

Capitolo 3

Studio delle soluzioni

Di fondamentale importanza è stata la ricerca dello stato dell'arte, ovvero il "livello delle conoscenze raggiunte in un determinato ambito professionale" (Sabatini Coletti 2006), che nel nostro caso riguardava l'attuale utilizzo di open data e la loro rappresentazione grafica.

Sono state quindi analizzate diverse soluzioni evidenziando vantaggi e svantaggi che hanno portato alla scelta finale per l'implementazione del progetto. Si è infatti appurato che nessuna delle attuali librerie JavaScript esistenti facevano completamente al caso in oggetto e si è così deciso di implementare da zero il codice che potesse rispondere alle esigenze del committente.

3.1 Progetti esistenti

Come base di partenza del progetto si è partiti da un lavoro preesistente e attualmente usato, ovvero *Brick - Web di dati* (§ 3.1.2), nonché da un progetto simile chiamato *EasyOnto* (§ 3.1.1) e sviluppato in *Flash*.

3.1.1 EasyOnto

EasyOnto[1] è un progetto di diploma di Alessandro Ruggeri (A.A. 2008/2009) il cui obiettivo principale era quello di offrire un'applicazione web semplice ed interattiva. L'interfaccia utente doveva essere accessibile da diverse categorie di autori e pertanto essere intuitiva da usare. Strumenti di navigazione e diversi tipi di visualizzazioni consentono all'utente di ottenere una esperienza completamente interattiva (Figura 3.1).

Lo scopo di EasyOnto era quindi quello di semplificare le strutture semantiche più facilmente rispetto ad altre applicazioni analoghe.

EasyOnto è stato sviluppato utilizzando la tecnologia *Adobe Flex* e il suo livello logico è basato su server *ColdFusion*.

Pur essendo un buon lavoro questo progetto soffre di alcuni problemi: tra tutti la lentezza di caricamento dell'applicazione, probabilmente dovuta alla sua mole ma anche al server ospitante, e anche alla sua, ormai vecchia, concezione di utilizzare Flash per creare qualcosa di dinamico. Ormai con l'avvento di nuove tecnologie e standard, come HTML5 e affini, questo progetto è caduto in disuso soppiantato da una versione più semplice e quindi più fluida come Brick.

[1] EasyOnto: http://isin06.dti.supsi.ch:8500/EasyOnto/main.html

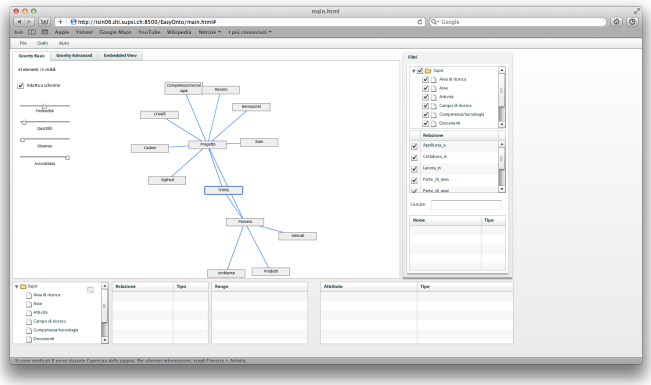
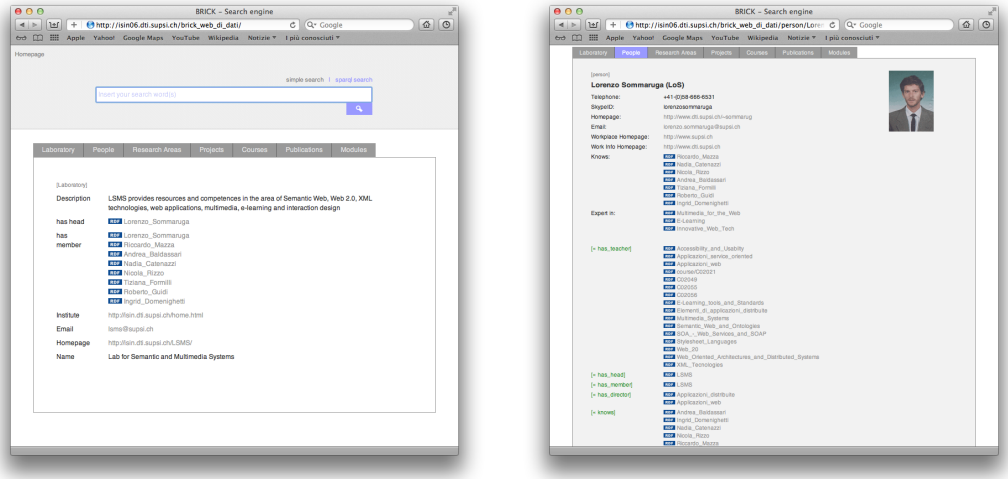


Figura 3.1: EasyOnto: visualizzatore di ontologie.

3.1.2 Brick - Web di dati

Il progetto *Brick - Web di dati*[2] di Nicolas Urech (A.A. 2010/2011) prevedeva di sviluppare e realizzare un'applicazione in grado di esporre e utilizzare pagine web come dati (Figura 3.2). Questo ha permesso di valorizzare i propri dati collegandoli con altri domini che contengono numerose fonti d'informazione già esistenti e utilizzabili, come ad esempio il progetto Linked Open Data.

L'applicazione adopera dati pubblici SUPSI (corsi, persone, competenze, progetti) per pubblicarle nel web. Essi sono collegati tra loro secondo un'ontologia definita dal LSMS[3] e visualizzati in pagine web per una loro navigazione oppure ricerca.



(a) Pagina principale di Brick.

(b) Visualizzazione di una risorsa (Lorenzo Sommaruga).

Figura 3.2: Screenshot dell'utilizzo di Brick - Web di dati.

[2]Brick - Web di dati: http://isin06.dti.supsi.ch/brick_web_di_dati
[3]LSMS: Laboratorio Sistemi Semantici e Multimediali

3.2 LinkedIn InMaps

LinkedIn[4] è una piattaforma di social network che ha lo scopo di mettere in contatto i propri iscritti al fine di creare opportunità professionali. Quest'ultimo, presso i propri laboratori, ha concepito un nuovo strumento, ancora in fase sperimentale, chiamato *InMaps*[5] è un nuovo strumento, ancora in fase sperimentale, concepito dai laboratori di LinkedIn. Tramite questo servizio è possibile visualizzare una mappa delle proprie connessioni lavorative attraverso la quale vedere le aree di concentrazione dei propri contatti e dove questi si intersecano fra loro (Figura 3.3).

L'idea è interessante anche se non davvero pratica: LinkedIn InMaps crea delle aree di diverso colore assemblando le connessioni fra loro attraverso il tipo di azienda, scuola o altri punti in comune ma non assegna in automatico delle *label* per ciascun gruppo. Il che significa che attivando questa funzionalità si vedranno 3 o 4 macchie colorate alle quali bisognerà dare un'identità.

Al seguente link http://www.youtube.com/watch?v=PC99Nw2JX8w ("Visualize your LinkedIn network with InMaps") è possibile vedere un video di presentazione per farsi un'idea del servizio.

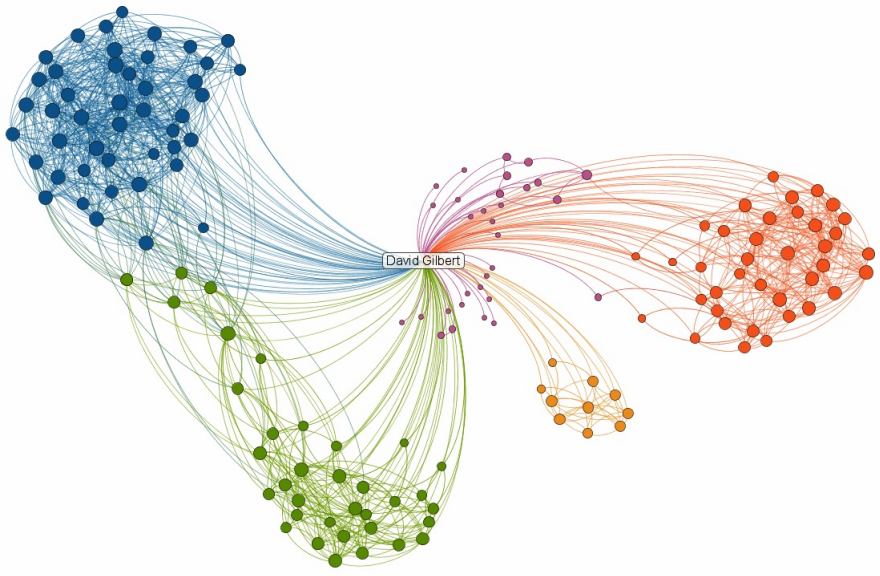


Figura 3.3: Esempio di LinkedIn InMaps.

InMaps è stato uno degli esempi grafici di visualizzazione di dati semantici che è possibile reperire in rete. Seppur l'idea sia buona abbiamo cercato di far tesoro di alcune peculiarità - ma anche note negative - di questo strumento. Per quanto riguarda le caratteristiche positive abbiamo tenuto conto dell'utilizzo di diversi colori per visualizzare tipi di risorse diverse così come la possibilità di visualizzare i dettagli semplicemente posandosi sopra le bolle. Per quanto riguarda gli aspetti negativi, ovvero i caotici e dispersivi collegamenti far risorse, abbiamo cercato di porvi rimedio dando una quanto più possibile visione pulita e ordinata.

[4]LinkedIn: http://www.linkedin.com
[5]LinkedIn InMaps: http://inmaps.linkedinlabs.com

3.3 Librerie grafiche

3.3.1 D3.js

D3.js[6] è una libreria JavaScript per manipolare documenti sulla base di dati (D3 è sinonimo di *Data-Driven Documents*).

D3 consente di associare dati arbitrari in un Document Object Model (DOM), e quindi applicare trasformazioni al documento basate sui dati. Come esempio banale, è possibile utilizzare D3 per generare una tabella HTML di base da una serie di numeri o in alternativa, utilizzare gli stessi dati per creare un grafico interattivo a barre con transizioni dinamiche.

D3 non è un framework di visualizzazione tradizionale. Piuttosto che fornire un sistema monolitico, D3 risolve solo il nocciolo del problema: la manipolazione efficiente di documenti sulla base di dati. Questo dà a D3 una straordinaria flessibilità, esponendo tutte le funzionalità delle tecnologie di base come CSS3, HTML5 e SVG. Con un *overhead* minimo, D3 è estremamente veloce, supporta grandi serie di dati e comportamenti dinamici per l'interazione ed animazione oltre a permettere il riutilizzo del codice attraverso una variegata collezione di moduli opzionali (Figura 3.4).

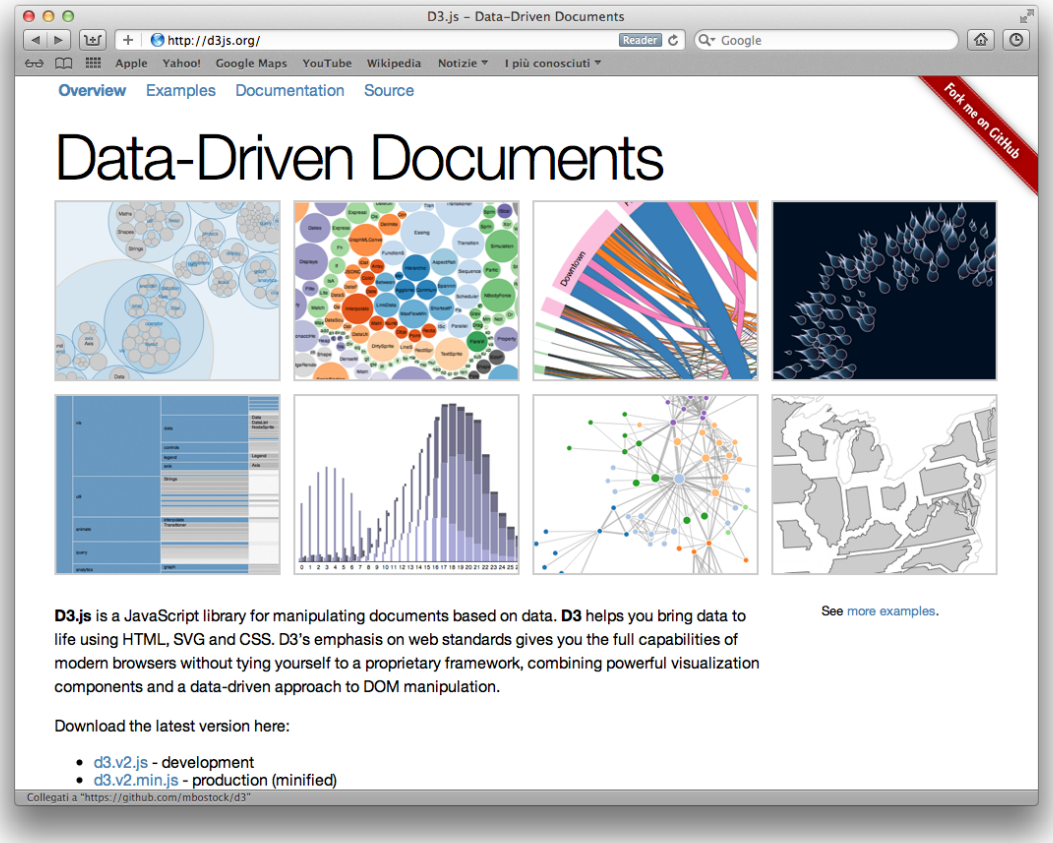


Figura 3.4: Alcuni esempi delle potenzialità di D3.js.

[6] D3.js: http://www.d3js.org

Abbiamo dunque analizzato le potenzialità di D3, ma seppur una libreria molto potente, dopo diversi tentativi abbiamo deciso di non utilizzarla perché limitata alla manipolazione di dati in un formato che sia solo numerico o breve testo. Nel nostro caso invece abbiamo diversi tipi di dati: URI, literal, numeri che vanno trattati in modo differente.
Inoltre una volta estratti i dati questi vengono convertiti in SVG e quindi non più modificabili o selezionabili.

3.3.2 Raphaël

Raphaël[7] è una piccola libreria JavaScript che si propone di semplificare il lavoro con la grafica vettoriale sul web. Se si desidera creare un grafico specifico o ritagliare l'immagine e ruotare widget, per esempio, è possibile ottenerlo in modo semplice e facilmente con questa libreria.

Raphaël utilizza la raccomandazione W3C SVG[8] e VML[9] come base per la creazione di grafica. Questo significa che ogni oggetto grafico creato è anche un oggetto DOM, in modo da potergli anche collegare i gestori di eventi JavaScript o modificarli in un secondo momento. L'obiettivo di questa libreria è quindi quello di fornire un adattatore che renda il disegno vettoriale facile e compatibile con pagine *cross-browser*.

Abbiamo dunque analizzato le potenzialità di Raphaël e pure quest'opzione è stata scartata perché non ricopriva appieno le nostre richieste.

3.4 Tecnologie esistenti

Di seguito sono descritte le tecnologie esistenti e utilizzate nel progetto.

3.4.1 JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web. Fu originariamente sviluppato da Brendan Eich della Netscape Communications con il nome di Mocha e successivamente di LiveScript, ma in seguito è stato rinominato "JavaScript" ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Oracle. JavaScript è stato standardizzato per la prima volta tra il 1997 e il 1999 dalla ECMA con il nome ECMAScript.

La caratteristica principale di JavaScript è quella di essere un linguaggio dinamico e interpretato: il codice non viene compilato, ma interpretato (in JavaScript lato client, l'interprete è incluso nel browser che si sta utilizzando). La sintassi è relativamente simile a quella del C, del C++ e di Java.
Il linguaggio definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented tramite prototipi. È inoltre un linguaggio debolmente tipizzato.

Altri aspetti di interesse: in JavaScript lato client, il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il server non viene sovraccaricato a causa delle richieste dei client. Di contro, nel caso di script che presentino un sorgente particolarmente grande,

[7] Raphaël: http://www.raphaeljs.com
[8] W3C Scalable Vector Graphics (SVG): http://www.w3.org/Graphics/SVG/
[9] Vector Markup Language (VML): linguaggio XML aperto, destinato alla creazione di grafica vettoriale elaborate in 2D o 3D (statiche o animate) sulle pagine web.

il tempo per lo scaricamento può diventare abbastanza lungo. Un altro svantaggio è il seguente: ogni informazione che presuppone un accesso a dati memorizzati in un database remoto deve essere rimandata ad un linguaggio che effettui esplicitamente la transazione, per poi restituire i risultati ad una o più variabili JavaScript; operazioni del genere richiedono il caricamento della pagina stessa. Con l'avvento di AJAX[10] tutti questi limiti sono stati superati.

3.4.2 jQuery

jQuery è una libreria di funzioni (framework) JavaScript, cross-browser per applicazioni web, che si propone come obiettivo quello di semplificare la programmazione lato client delle pagine HTML.
Tramite l'uso della libreria jQuery è possibile, con poche righe di codice, effettuare svariate operazioni, come ad esempio ottenere l'altezza di un elemento o farlo scomparire con effetto dissolvenza.

Anche la gestione degli eventi è completamente standardizzata, automatica; stessa cosa per quanto riguarda l'utilizzo di AJAX, in quanto sono presenti alcune funzioni molto utili e veloci che si occupano di istanziare i giusti oggetti ed effettuare la connessione e l'invio dei dati.

Figura 3.5: Logo di jQuery.

3.4.2.1 Compatibilità tra browser

Uno dei più grandi vantaggi di jQuery consiste nel fatto che gestisce molti dei problemi più critici di compatibilità tra browser. Utilizzare quindi questa libreria ha permesso di scrivere del codice che possa essere eseguito nello stesso modo in tutti i browser principali (compreso, per alcune cose, Internet Explorer 6).

3.4.2.2 CDN di Google

Un metodo alternativo per includere la libreria jQuery è tramite la rete CDN (*Content Delivery Network*) di Google. Una CDN è una rete di computer progettati in modo specifico per fornire contenuti agli utenti in modo rapido e scalabile. Questi server vengono spesso distribuiti geograficamente, con ogni richiesta gestita dal server più vicino della rete.
Google ospita nella propria CDN numerose librerie open source note, incluso jQuery. Quindi, invece di ospitare i file di jQuery sul proprio server rischiando di avere versioni non aggiornate, si può utilizzare quella ospitata su Google godendo anche della velocità e affidabilità della sua infrastruttura. Un altro vantaggio derivante dall'utilizzo della CDN di Google consiste nel fatto che molti utenti avranno già scaricato jQuery da Google durante la visualizzazione di un altro sito. Di conseguenza, verrà caricato dalla cache quando

[10] AJAX: http://it.wikipedia.org/wiki/AJAX

visiteranno il nostro sito, dato che l'URL al file di JavaScript sarà lo stesso e i tempi di caricamento saranno notevolmente più rapidi.

3.4.3 Mustache

Mustache[11] è un semplice sistema di template web disponibile per molti linguaggi, tra cui JavaScript.
Il supporto in JavaScript comprende sia la programmazione lato client con molte popolari librerie e framework come jQuery, Dojo e YUI, così come sul lato server utilizzando Node.js CommonJS.

Mustache viene descritto come un sistema "logic-less" perché manca di tutte le dichiarazioni esplicite di controllo del flusso, clausole *else* o cicli. Esistono invece solo dei tag che vengono sostituiti con uno o più valori.

Nota curiosa: si chiama "Mustache" (*baffi* in inglese) a causa del pesante uso di parentesi graffe che assomigliano, appunto, a dei baffi.

Mustache può essere usato per HTML, file di configurazione e codici sorgenti. Funziona espandendo i tag in un modello utilizzando i valori forniti in un hash o un oggetto.

Nello specifico questa libreria si è resa utile nella formattazione del contenuto (testuale e non) del blocco principale, contenente cioè le informazioni della risorsa selezionata.

3.4.4 Twitter Bootstrap

Bootstrap[12] è un toolkit front-end open-source per aiutare i designer e gli sviluppatori a creare rapidamente ed in maniera efficace pagine e componenti online.
Creato e utilizzato dapprima internamente da Twitter, è stato poi rilasciato a tutta la comunità. L'obiettivo è quello di fornire una libreria raffinata, ben documentata ed estesa di componenti di design flessibili, realizzati in HTML, CSS e JavaScript, così che altri possano usarli come base ed innovare. Ad oggi è cresciuto fino ad includere dozzine di componenti ed è diventato il progetto più popolare su GitHub, con più di 13000 watchers e 2000 forks.

Twitter Bootstrap 2.0 include una più completa documentazione di tutti i componenti, un nuovo sistema a griglia, nuove media query per responsive web design, nuovi stili di base, nuovi componenti e 12 ottimi plugin jQuery pronti per l'uso.

Per il progetto si è quindi usufruito di alcune di queste *feature* come la creazione del menu in cima alla pagina, alcune icone e soprattutto la possibilità di visualizzare in modo semplice e veloce del testo all'interno di finestre popover.

3.5 Risk management

Una corretta gestione del rischio è indispensabile per portare a termine in modo corretto un progetto. Grazie ad esso è possibile valutare ed anticipare i possibili rischi, applicando eventualmente delle azioni correttive. A tal proposito è stato effettuato un primo elenco di possibili rischi.

- **Ritardo nelle consegne**
  Per garantire la consegna del progetto e scongiurare quindi ritardi nelle scadenze

[11] Mustache: http://mustache.github.com
[12] Twitter Bootstrap: http://twitter.github.com/bootstrap/

(sia intermedie che nella consegna finale, compresa la documentazione), sono state decise delle *deliverable* interne. In pratica si raggiungevano man mano i vari obiettivi richiesti, ma garantendo comunque la funzionalità del programma. Grazie a questo è stato possibile garantire la consegna finale di un programma funzionante, anche nel caso peggiore dove sarebbero potute mancare parti di obiettivi.

- **Raggiungimento degli obiettivi**
  Uno dei rischi più evidenti era quello di non riuscire a raggiungere gli obiettivi prefissati nel tempo prestabilito. Inoltre una mancata consegna (o comunque incompleta) poteva portare a una valutazione insufficiente, e quindi era importante stabilire in anticipo i rischi e le possibili ripercussioni.

- **Perdita o danneggiamento dei dati**
  I rischi come la perdita di dati non sono contemplati, visto che si è provveduto ad utilizzare un server di gestione dati (SVN) con copie distribuite tra almeno 3 computer. Seppur il rischio di danneggiamento dei dati è comunque esistente, esso è stato giudicato ininfluente rispetto al resto.

- **Problemi organizzativi e di comunicazione**
  I problemi nel server di produzione hanno impedito di effettuare test completi fino all'ultimo. L'aver programmato in modo corretto la gestione dei dati ha però permesso un passaggio senza particolari problemi dall'utilizzo di dati locali a quelli remoti.

Dopo questa prima valutazione, abbiamo poi realizzato un'analisi più approfondita dei rischi, riepilogata nella Tabella 3.1.

Tabella 3.1: Tabella della gestione del rischio

#	**Tipo**	**Probabilità [0-1]**	**Impatto [0-1]**	**Prob.×Imp [0-1]**	**Rischio**	**Impatto**	**Azione correttiva**
1	Organizzativo	0.80	1	0.80	Ritardo nella consegna del progetto nei termini prestabiliti	Fallimento del progetto, valutazione insufficiente	Mettere traguardi intermedi completi
2	Visivo	0.60	0.80	0.48	Risoluzione insufficiente	Visione non completa dei dati, navigazione scomoda	Valutare l'impatto nella riduzione della dimensione degli elementi
3	Tecnico	0.40	0.80	0.32	Compatibilità del browser	Il sito non funziona correttamente	Messaggio di avviso o *workaround*
4	Dati	0.20	1	0.20	Problemi nel formato dei dati ricevuti	Errore nel programma	Non previsto
5	Tecnico	0.50	0.50	0.25	Server dei dati non disponibile	Mancanza di prove finali, bug imprevisti a causa del cambio di collegamento	Non previsto

Capitolo 4

Design e concezione

Il progetto LOD Viz consisteva principalmente nel rappresentare i dati provenienti dall'esistente progetto Brick in una maniera organizzata secondo una gerarchia di relazioni (Figura 4.1).

Già durante le prime riunioni è stata ricevuta un'idea abbastanza dettagliata di quello che doveva essere il risultato finale. Si sarebbero potute analizzare e proporre altre soluzioni, ma in questo particolare progetto la fase iniziale di studio e design era già stata effettuata.

In ogni caso è stata studiata la parte di estetica e presentazione, in modo da essere facilmente utilizzabile dal target di utenti che adopereranno questa applicazione. Il target di riferimento sono infatti le persone che attualmente usano e lavorano sul progetto Brick.

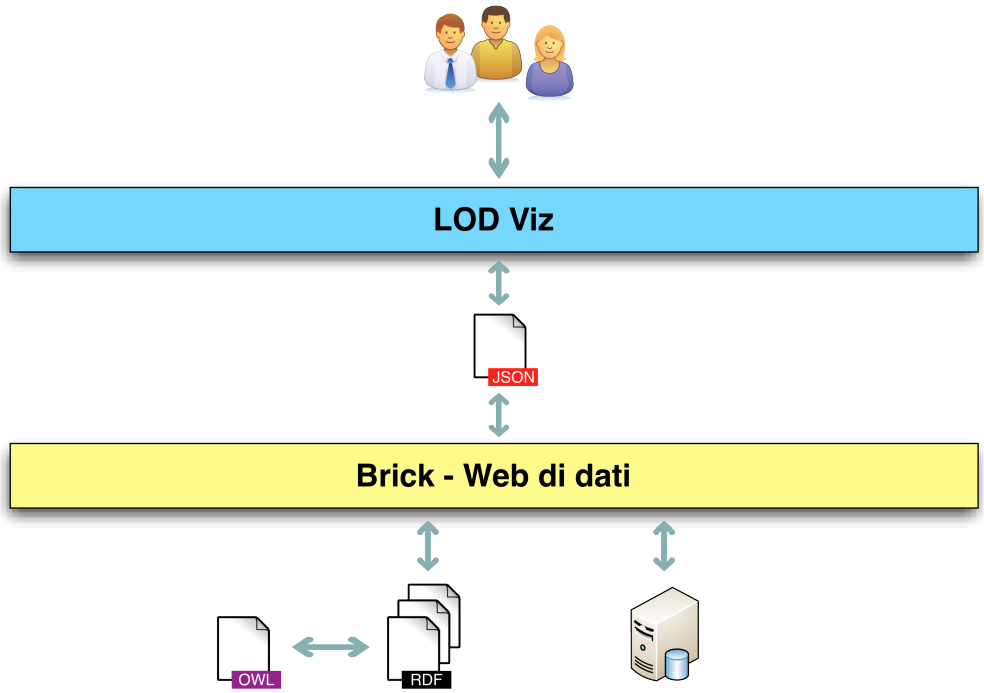


Figura 4.1: Schema di funzionamento di LOD Viz.

Si è deciso, infine, di non effettuare nessuna valutazione tramite l'ICF[1] essendo già il contesto e l'idea di partenza ben definiti.

4.1 Accessibilità

Non è stato effettuato nessuno studio per persone con disabilità, come ad esempio ipovedenti o persone con deficit motorie. Il nostro contesto si basa su un modello di persona senza particolari problemi. Inoltre per una questione di termini non è stato valutato e inserito un supporto all'input tattile.

È necessario uno studio sulle tecnologie esistenti e l'applicabilità sul sito web, nonché l'hardware necessario per effettuare le prove. Altre forme di input naturali non sono state considerate.

4.2 Valutazione NUI

Non sono state fatte particolari valutazioni sulle NUI[2] anche se, nel caso di un touchscreen, dovrebbe permettere una navigazione senza problemi, a patto che il sistema sia configurato in modo corretto.

4.3 Interfaccia

4.3.1 Colore e forme

Quello che rimaneva da studiare è come presentare queste informazioni all'utente, valutando le scelte di colore, forme e interattività.
Riguardo al colore sono stati semplicemente usato colori chiari ma contrastati, anche se però in caso di daltonismo non è possibile distinguere in maniera sufficiente le relazioni. Pertanto oltre al colore era necessario valutare le tipologie di forme: siccome si hanno relazioni entranti e uscenti, è stato deciso di utilizzare delle frecce direzionali per indicarne il verso. In ogni caso esistono pure dei riferimenti testuali che indicano il tipo di relazione.

A livello di forme sono presenti elementi comuni, racchiusi tra insiemi anch'essi con una forma comune. Le relazioni sono rappresentate in ingresso e in uscita tramite frecce, collegate ad un elemento centrale. Quest'ultimo contiene l'oggetto preso in considerazione attualmente, con al suo interno le informazioni di dettaglio. Il blocco centrale è rappresentato in modo distinto rispetto al resto, sia in termini di colore che di forma (rettangolo con bordi arrotondati e sfondo grigio).

Se l'utente passa sopra un elemento, appare una finestra indicante il tipo e il contenuto della risorsa. Un possibile problema per chi non ha mai usato il programma può essere la mancanza di immediatezza nella navigazione: anche se gli elementi visuali mostrano delle indicazioni al passaggio del mouse, non è ancora garantito che l'utente riesca a capire che cliccando sull'elemento potrà accedere a maggiori informazioni. Per mitigare questo rischio è stato aggiunto il fatto che il colore dell'elemento cambi durante il passaggio del puntatore del mouse e che appaia un cursore diverso: lo stesso è utilizzato per i collegamenti

[1] International Classification of Functioning, Disability and Health (ICF): http://www.who.int/classifications/icf/en/

[2] Natural User Interface (NUI): http://it.wikipedia.org/wiki/Interfaccia_utente_naturale

ipertestuali (una mano con l'indice, ma è comunque dipendente dal sistema operativo utilizzato).

4.3.2 Evoluzione dell'interfaccia grafica

Di seguito è riportata l'evoluzione dello sviluppo dell'interfaccia grafica.

Dapprima ci si è concentrati sulla parte di codifica e quindi l'obiettivo principale è stato quello della manipolazione e visualizzazione testuale dei dati. Si può infatti notare come in Figura 4.2 vengano mostrati i puri dati grezzi con una prima suddivisione tra quelli facenti parte del blocco principale (riquadro verde) e quelli inerenti ai collegamenti con le risorse (blocchi blu).

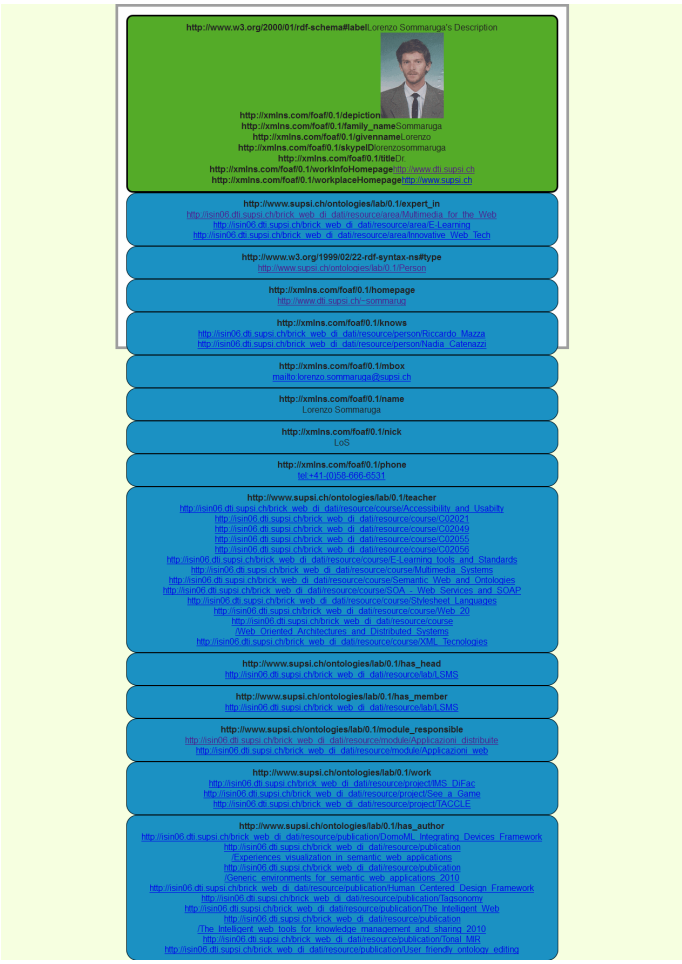


Figura 4.2: Evoluzione dello sviluppo dell'interfaccia grafica (1): prove di estrazione e visualizzazione testuale dei dati.

Il passo successivo è stato quello del posizionamento dei singoli blocchi attorno al principale. Parte che ha richiesto non poche fatiche soprattutto per la gestione della matematica al fine di disporre in maniera corretta, senza quindi sovrapposizioni, i box delle relazioni (Figura 4.3).
I dati estratti dai file JSON e posizionati nei box blu sono ancora in forma grezza, ovvero semplici URI ad altre risorse puntate.

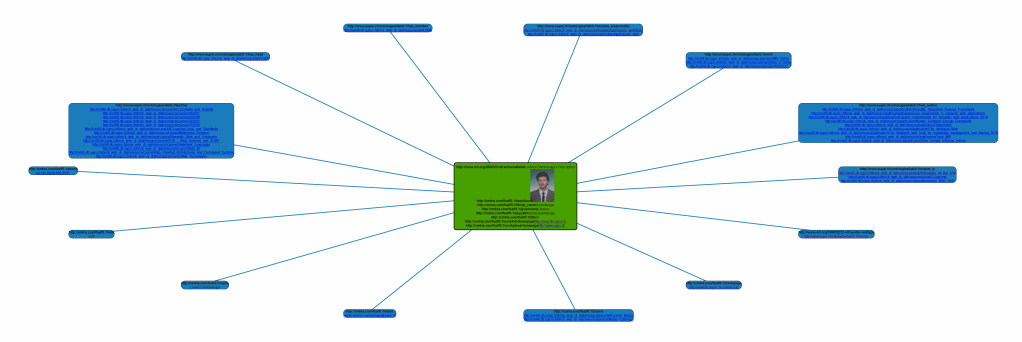


Figura 4.3: Evoluzione dello sviluppo dell'interfaccia grafica (2): prima rappresentazione grezza dei dati in box disposti radialmente a quello centrale.

Giunti a questo punto si è cominciato a pensare di più alla grafica e all'interazione che questa dovesse avere con l'utente.
I blocchi sono quindi diventati delle bolle come richiestoci dal committente e al cui interno compare un numero indicante la quantità delle risorse; inoltre anche la dimensione è dipendente da essa. Sotto le bolle compare il tipo di relazione che intercorre con la risorsa selezionata. Cliccando poi con il mouse sopra alle bolle appare un popup contenente le informazioni delle risorse (Figura 4.4).

In questa fase i dati sono già formattati nei vari blocchi, utilizzando sia Mustache (per il template) e Bootstrap (per icone e popup).

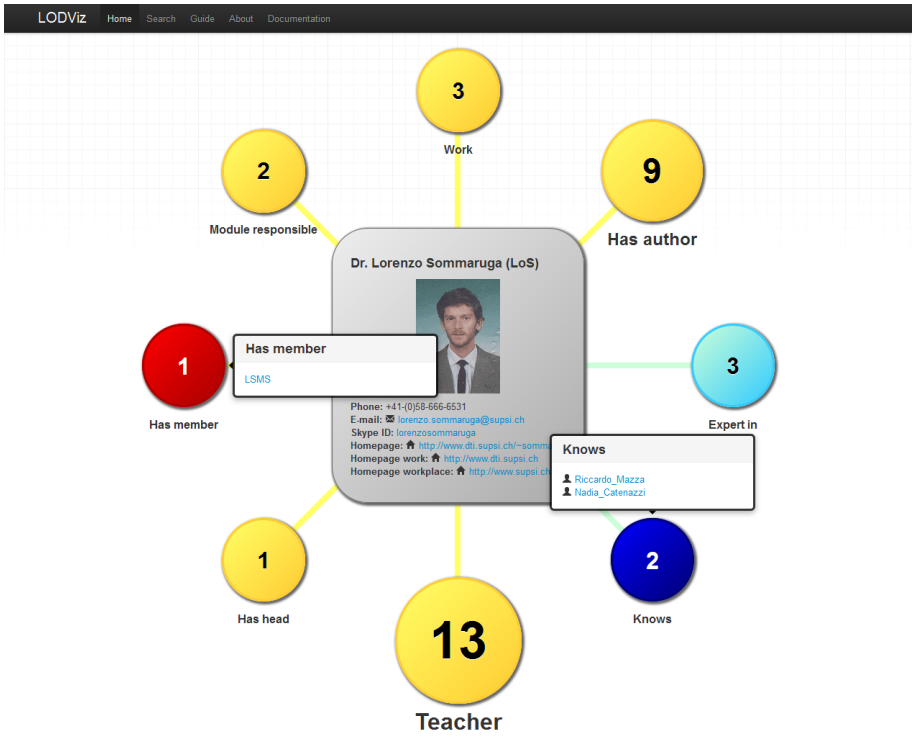


Figura 4.4: Evoluzione dello sviluppo dell'interfaccia grafica (3): dati formattati correttamente.

Dopo aver sottoposto e discusso questo risultato col relatore si è giunti alla conclusione che la dimensione variabile di bolle e numero interno potesse travisarne il significato, dando all'utente la sensazione che a maggior grandezza corrispondesse una maggior importanza. Si è così deciso di cambiare in parte la grafica ottenendo infine il risultato finale mostrato in Figura 4.5.
Le bolle esterne sono diventate dei contenitori di altre piccole bolle ognuna rappresentante una singola risorsa del tipo indicato esternamente. Cliccando uno di questi piccoli cerchi il grafo viene automaticamente ridisegnano ponendo al centro la risorsa scelta. Inoltre anche i dati vengono ora caricati dinamicamente avendo collegato il codice al server centrale che fornisce gli open data.
Altra cosa importante è l'introduzione di frecce orientate poste sulle linee di collegamento tra risorsa centrale e bolle periferiche. Queste sono essenziali perché, insieme ai due tipi di colore, indicano se la risorsa è di tipo entrante o uscente. Le risorse, infatti, sono di colore giallo (*in*) se dall'esterno hanno un qualche tipo di connessione verso il dato centrale (es. Laboratorio ha come membro una Persona - *Laboratory has member Person*); mentre sono azzurre (*out*) se i dati puntano verso risorse esterne.

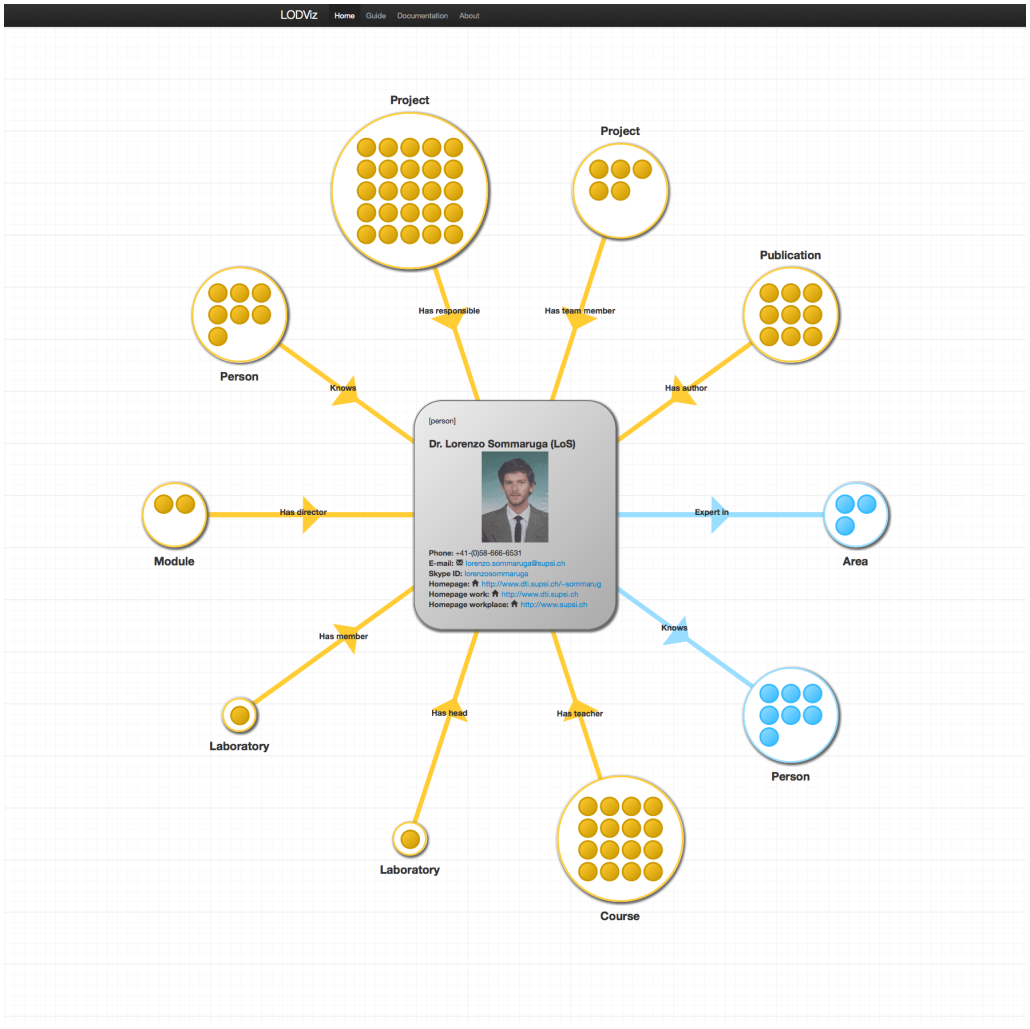


Figura 4.5: Evoluzione dello sviluppo dell'interfaccia grafica (4): risultato finale.

Capitolo 5

Realizzazione e Test

Questo capitolo contiene tutte le informazioni sulla realizzazione del progetto e sui test effettuati, sia a livello di singole parti (tramite debugger), sia a livello di integrazione col server, sia a livello lato utente (compatibilità con i vari browser e test d'utilizzo).

5.1 Realizzazione

Per la realizzazione abbiamo proceduto per passi, interessandoci dapprima allo studio del linguaggio Javascript ed eseguendo piccolo test d'utilizzo. Nella prima fase di analisi abbiamo anche testato alcune librerie che apparentemente avrebbero potuto fare al caso nostro, ovvero D3.js (§ 3.3.1) e Raphaël (§ 3.3.2) ma che poi si sono rivelate essere non adatte per il nostro scopo.

5.1.1 Struttura del progetto

Di seguito una breve descrizione dei file e della struttura del progetto LOD Viz.

- **about.html** Alcune informazioni sui creatori e sulle persone che hanno seguito ed aiutato durante il lavoro
- **documentation.html** Dov'è possibile reperire la documentazione del progetto, ovvero questa relazione e la relativa presentazione
- **guide.html** Una breve guida all'utilizzo dell'applicazione
- **index.html** Rappresenta la pagina principale del sito in cui verranno visualizzati graficamente i dati
- /**css** Contiene i fogli di stile
- /**js** Directory contenente tutti i file JavaScript

  lodviz.js Rappresenta il *core* dell'applicazione

  mustache.js libreria di Mustache (§ 3.4.3) utilizzata per creare il template per la visualizzazione delle informazioni nel blocco principale

  - /**js**/**graphics** Cartella con gli script per la gestione grafica di LOD Viz

block.js Questo script gestisce la creazione e gestione degli oggetti 'blocco' e delle linee di collegamento fra essi

layout.js Qui viene gestita tutta la parte di visualizzazione dei dati nel blocco centrale e nelle bolle attorno ad esso

5.2 Architettura

Qui di seguito viene descritta l'architettura del software.
Il progetto si compone di moduli separati, ognuno con una proprio scopo, rappresentati di seguito:

- Visualizzazione (HTML/CSS)
- Lettura ed elaborazione dati
- Formattazione dei dati e posizionamento degli oggetti

5.2.1 Visualizzazione

La parte più importante del progetto è la parte di visualizzazione. Essa si compone di un template HTML, comprendente gli script principali caricati con un ordine ben definito, e alcuni tag necessari per caricare al loro interno i dati di LOD Viz.

5.2.1.1 Pagine HTML

I file HTML presenti nel progetto sono utilizzati sia come template (e quindi modificati da file JavaScript dedicati) o come documentazione.

Come framework ci si è basati su Bootstrap, prendendo quindi il suo stile e i suoi componenti per presentare in modo gradevole le informazioni.

5.2.1.2 Fogli di stile

A livello di CSS invece abbiamo i seguenti file (all'interno della cartella CSS):

- **lodviz.css** Comprende l'intera formattazione del progetto
- **animate-custom.css** Foglio di stile generato dal plug-in *Animate.css*[1], che contiene varie animazioni utilizzate nella parte visuale

Una parte molto importante è la divisione tra il blocco centrale e i blocchi posizionati intorno ad esso. Il blocco principale contiene le informazioni dell'elemento scelto dall'utente, come ad esempio un nome e una descrizione. I blocchi legati ad esso e posizionati attorno sono invece elementi che sono collegati tramite una relazione all'oggetto stesso. Ogni blocco esterno è collegato al blocco centrale tramite una linea e una freccia orientata, indicante la tipologia (*in* o *out*). Il blocco principale viene definito anche come blocco prioritario (*priority*) all'interno della configurazione.

[1] Animate.css: http://daneden.me/animate/

5.2.2 Lettura ed elaborazione dati

Questa fase contiene la parte importante della lettura dei dati dal server e della successiva trasformazione degli stessi in una forma utilizzabile dal resto del programma. L'intera logica di questa fase si trova nel file `lodviz.js`. Esso si compone di un oggetto *Core*, che in pratica racchiude l'intero programma. Al suo interno sono presenti diversi oggetti e proprietà necessari per il corretto funzionamento.

Il programma permette inoltre di leggere dati locali o di collegarsi in modo remoto.

Un altro importante oggetto è quello denominato *configuration*. Esso contiene le seguenti informazioni:

- Modifica dei tipi degli URI (da URI generici a tipi ben definiti, come `mbox` o `image`)
- URI degli elementi necessari per descrivere un elemento (contenuti nel blocco centrale)
- Traduzione dei nomi
- Blacklist degli elementi da non visualizzare
- Traduzione delle categorie

In combinazione con la parte di gestione del layout presente in un altro modulo (`layout.js`) è possibile rappresentare in maniera convincente e chiara le informazioni desiderate.

Da notare che in mancanza di informazioni specifiche, il programma tenterà un'elaborazione dei dati generica, risultando quindi possibile caricare dati non conosciuti. Rimane però il vincolo di avere i dati categorizzati in *in* e *out*, nonché una certa struttura del file JSON.

5.2.2.1 Struttura file JSON

Per una maggiore chiarezza su come il programma elabora i file JSON provenienti dal progetto Brick, qui di seguito viene mostrata una struttura generica. Se viene rispettata questa struttura, il programma funzionerà in modo corretto, altrimenti i dati verranno ignorati.

```
{ "results":{
    "bindings":[
        {
            "type_in":{
                "type":"uri",
                "value":"http://www.supsi.ch/ontologies/lab/0.1/teacher"
            },
            "class_in":{
                "type":"uri",
                "value":"http://isin06.dti.supsi.ch/brick_web_di_dati/
                    resource/course/Accessibility_and_Usabilty"
            }
        },
```

Listato 5.1: Esempio di struttura di un file JSON

In pratica il JSON è formato da un oggetto al quale è associata una proprietà `results`. Essa contiene un altro oggetto `bindings`, che a sua volta contiene un oggetto formato da due proprietà `type_in` (o `type_out`) e `class_in` (o `class_out`), a dipendenza se il tipo di relazione dei dati è *in* o *out* (definito dal progetto Brick).
La proprietà `type_...` indica l'URI del dato (utilizzato poi nell'oggetto di configurazione per le varie elaborazioni). Invece `class_...` contiene i dati dell'URI definito in `type_...`. Pertanto è possibile avere oggetti con lo stesso URI in `type_...` ma diverso contenuto in `class_...` (come nel caso di array di elementi). Il programma si occupa poi di leggere e trasformare questi oggetti in una forma adatta per successive elaborazioni.

5.2.2.2 Caricamento dati in locale

Per il caricamento dei dati in locale è necessario rispettare i seguenti requisiti:

- I file devono essere in formato JSON;
- I file devono essere di tipo *in* o *out* (non è necessario avere entrambi). Ad esempio, se si ha un file contenente i dati per un laboratorio (es. nome: *laboratorio1*), si possono avere sia `laboratorio1_in.json` che `laboratorio1_out.json`;
- La proprietà `baseUri` dev'essere corretta e serve per verificare che i dati vengano caricati dallo stesso dominio;
- Il flag`loadLocal` deve essere impostato a *true* (si trova all'interno della funzione `configure`);
- All'interno della funzione `decodeUriSource` devono essere inseriti i mapping tra gli URI e i relativi file locali. Tutti gli URI devono avere come parte iniziale comune `baseUri`.

Per effettuare il mapping degli URI, si modifica l'oggetto `localMap`, all'interno della funzione `decodeUriSource`.

Come si nota dall'esempio seguente (Listato 5.2), si tratta di un oggetto contenente gli URI sotto forma di proprietà, e i relativi file come valore. Ci penserà poi il codice ad aggiungere la parte `_in.json` o `_out.json` a dipendenza del tipo.

```
localMap = {
'uri_completo':'file',
'http://isin06.dti.supsi.ch/brick_web_di_dati/resource/person/
    Lorenzo_Sommaruga':'json/lorenzo',
'http://isin06.dti.supsi.ch/brick_web_di_dati/resource/course/
    Lorenzo_Sommaruga':'json/Applicazioni_web_course',
...
};
```

Listato 5.2: Esempio di mapping degli URI

5.2.2.3 Caricamento dati in remoto

In questo caso è necessario solo impostare il flag `loadLocal` a *false*, che si trova nella funzione `configure` dell'oggetto *Core*. I parametri sono preconfigurati all'interno della funzione `loadBrickData`.

Per aggirare il problema del caricamento dei dati da domini diversi (cross-domain AJAX) è stato utilizzato uno script PHP (`proxy.php`) che si occupa del caricamento dei dati. Attualmente richiede un file dedicato (`test2.php`) per l'interrogazione dei dati Brick e la loro pre-elaborazione.

5.2.2.4 Conversione dei dati JSON in formato per uso interno

Una volta che i dati sono stati letti, essi vengono trasformati in una forma utilizzabile dal resto del programma per mezzo della funzione `filterBrickData`. La forma del nuovo oggetto è quella mostrata nel Listato 5.3.

```
{
  in: {
      'uri1' :  {
            data : [],
            source : '',
            type : '',
            value : ''
          },
      ...
    }
  out: ...
  priority: ...
}
```

Listato 5.3: Conversione dei dati JSON in formato per uso interno

Le proprietà *in*, *out* e *priority* sono opzionali. Ognuna può contenerne di altre (riferite dai loro URI) con all'interno le seguenti informazioni:

- `data`: i dati dell'URI sotto forma di array
- `source`: la categoria dell'URI, utilizzato solo per distinguere il tipo dei blocchi
- `type`: il tipo di dato (URI, literal, mbox e cosi via)
- `value`: l'eventuale traduzione in sostituzione dell'URI stesso

Questi dati sono poi utilizzati dai moduli di formattazione (layout) e posizionamento (Figura 5.1).

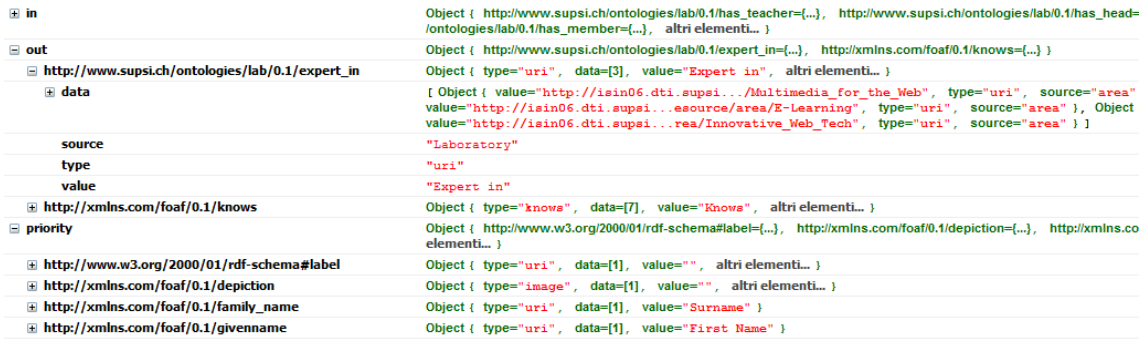


Figura 5.1: Esempio del contenuto dei dati trasformati dalla funzione `filterBrickData`.

5.2.2.5 Flowchart di `lodviz.js`

In Figura 5.2 è rappresentato il flowchart del funzionamento dello script principale `lodviz.js`. Per maggiori informazioni sulle funzioni utilizzare si rimanda alla documentazione contenuta all'interno del codice sorgente.

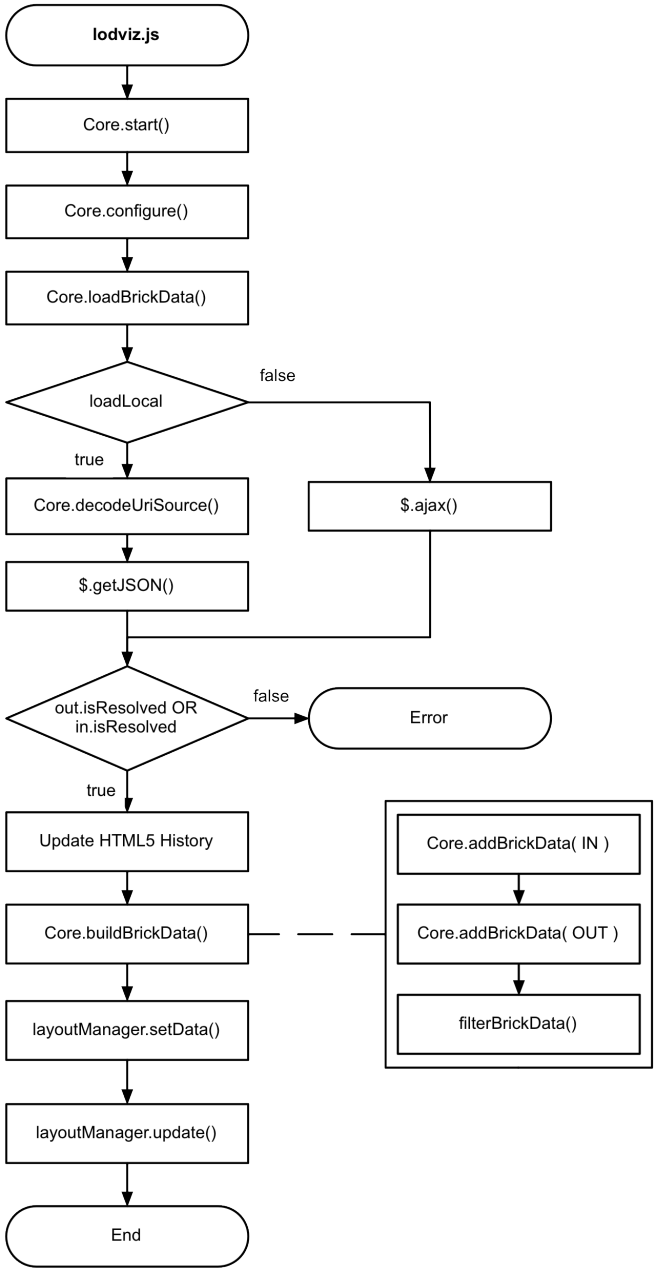


Figura 5.2: Diagramma di flusso del funzionamento dello script principale `lodviz.js` per il caricamento e la trasformazione dei dati.

5.2.3 Formattazione dei dati e posizionamento degli oggetti

Una volta letti i dati e converti in una forma più comoda da gestire, l'oggetto *layoutManager* si occupa di ricevere i dati, creare gli oggetti associati (blocchi) ed eseguire il posizionamento e la visualizzazione a video. L'intero meccanismo è messo in funzione dalle due chiamate `layoutManager.setData(data)` e `layoutManager.update()`.

5.2.3.1 Prima fase (setData)

Nella prima fase (funzione `setData`) vengono cancellati gli oggetti esistenti e vengono ricreati dai dati appena ricevuti. Le proprietà *priorityBlock* (il blocco centrale) e *blocks* (array di elementi) vengono quindi riempite con dati.

La fase di generazione dei blocchi si occupa di prendere i dati e di generare gli oggetti *Blocks* necessari (sia quello centrale che quelli esterni). In questa fase è importante notare che non vengono ancora creati visivamente e posizionati gli elementi (Figura 5.3).

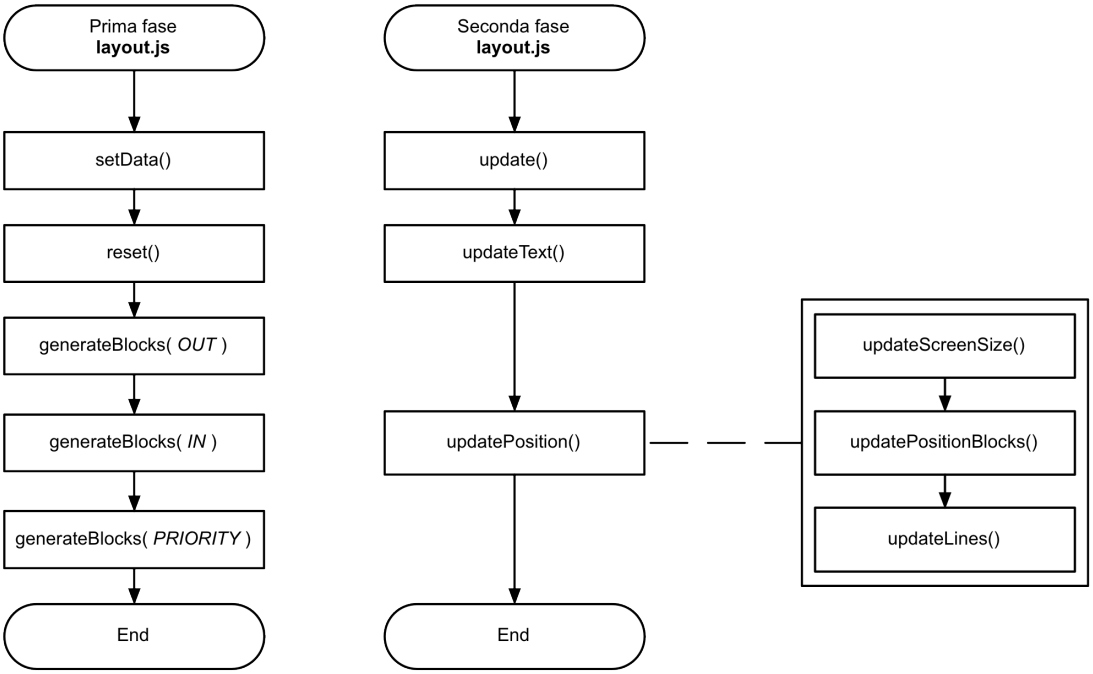


Figura 5.3: Diagramma di flusso del funzionamento dello script `layout.js` per la formattazione dei dati e il posizionamento degli oggetti.

5.2.3.2 Seconda fase (update)

Nella seconda fase (funzione `update`) ci si occupa di aggiornare il contenuto dei blocchi con i relativi dati, e il posizionamento di tutti gli elementi.

Per prima cosa si inizia a cercare l'elemento con dimensione più grande, per ottenere un valore (raggio, *radius*) utilizzato poi in tutte le altre funzioni per il posizionamento.

Successivamente con la funzione `updateText` si prepara il testo contenuto nel blocco centrale. Per i casi comuni esistenti nel progetto Brick (come ad esempio le categorie di dati

Person, *Lab*, *Courses* e così via) sono stati preparati dei modelli utilizzati con la libreria Mustache.

Per effettuare la selezione, si verifica il contenuto di `data.sourceType` (`data` è riferito ai dati impostati tramite la funzione `setData`) e se viene verificato che esiste un caso dove è gestito, vengono letti i dati e formattati in un modo preciso (Listato 5.4).

```
var bdata = priorityBlock.getComplexElements();
...
if (data.sourceType.toUpperCase() === "LAB") {
var ce_name = bdata["http://xmlns.com/foaf/0.1/name"];
var ce_email = bdata["http://xmlns.com/foaf/0.1/mbox"];

var data, template;

data = {
  name:function () {
    if (typeof ce_name !== "undefined") {
      return ce_name.data[0].value;
    }
  },

  email:function () {
    if (typeof ce_email !== "undefined") {
      return ce_email.data[0].value.substring(7);
    }
  }
};

template = '{{#name}}<h3>{{name}}</h3>{{/name}}' + '<br/><p>' +
    '{{#email}}<strong>E-mail:</strong> <i class="icon-envelope"></i>' +
    '<a href="mailto:{{email}}">{{email}}</a><br/>{{/email}}' +
    '</p>';

str = str + Mustache.to_html(template, data);
}

...

$(priorityBlock.node).html(str);
```

Listato 5.4: Verifica dei dati e applicazione del template tramite Mustache

Nel caso descritto si verifica il caso che la categoria (*sourceType*) sia *LAB*. Ci si occupa quindi di accedere ai dati (*bdata*) necessari per la nostra maschera. Mustache richiede due oggetti per funzionare correttamente: i dati (denominati *data* nel nostro esempio) e il template HTML (denominato *template*).
All'interno dell'oggetto *data* si verifica per ogni elemento che esista e che sia utilizzabile, poi in *template* si formattano i vari tag e posizione degli elementi.

Da notare che Mustache offre varie possibilità. Ad esempio usando `{{name}}` si accede al valore di *name*. Se invece si racchiude del testo tra `{{#name}}...{{name}}` viene effettuato un controllo del contenuto di *name*. Se è vuoto (o non esiste) l'intero contenuto non viene visualizzato. Infine tramite il triplo "baffo" `{{{...}}}` è possibile mostrare il contenuto di una variabile/funzione senza trasformazioni da parte di Mustache (tipo caratteri di escape).

Successivamente viene centrato il blocco centrale e ridimensionata l'area di lavoro (tag `<div>` identificato dall'id "base") Il posizionamento dei blocchi periferici viene effettuato utilizzando un raggio e posizionando in modo equidistante ogni blocco. Una volta posizionato tutto, vengono create le linee che collegano i blocchi periferici al blocco centrale.

5.2.3.3 Oggetti

Nel file `blocks.js` sono presenti gli oggetti utilizzati per la parte visuale, ovvero blocchi e linee. I blocchi sono oggetti che possono contenere array di elementi (lo stesso oggetto viene usato sia per il blocco centrale e i blocchi periferici). Gli elementi per il blocco centrale necessitano di più informazioni, pertanto sono riferiti come oggetti complessi (*complexElements*) mentre nel caso dei blocchi periferici gli oggetti sono contenuti in un semplice array denominato *elements*.
I blocchi offrono diverse funzioni come ad esempio il centraggio tramite coordinate (`setCenter`) o la creazione e la visualizzazione dei propri elementi in forma di cerchi (`updateElements`).
La linea invece si occupa di visualizzare una relazione tra il blocco centrale e i blocchi periferici. Viene sempre creata una freccia direzionale, e i metodi utilizzati sono `setPosition` per il posizionamento delle coordinate (x1, y1, x2, y2) e `drawLine` per i calcoli necessari a creare la freccia, ruotarla (tramite CSS3) e posizionarla alla distanza corretta.

5.3 Utilizzo

Gli open data vengono caricati dinamicamente e per default la scheda principale è quella di Lorenzo Sommaruga.

Nel blocco centrale sono riportare tutte le informazioni personali (nel caso la risorsa sia di tipo *Person*) oppure le altre informazioni di descrizione presenti nei file JSON. Le bolle che compaiono attorno al riquadro principale sono di due tipi: di colore azzurro corrispondono a tutte quelle relazioni di tipo *in*, ovvero relazioni entranti mentre le gialle sono relazioni di tipo *out*. Relazioni entranti e uscenti sono quindi quelle risorse che puntano o sono puntate dall'elemento selezionato attualmente (contenuto cioè nella scheda centrale). Le frecce orientate, poste sulle linee di collegamento tra risorsa centrale e bolle periferiche, indicano se la risorsa è di tipo entrante o uscente.

Per visualizzare gli elementi contenuti nella bolla basta posizionarsi sopra con il mouse. Si aprirà quindi una popup contenenti la risorsa. Cliccando uno di questi piccoli cerchi il grafo viene automaticamente ridisegnano ponendo al centro la risorsa scelta. Inoltre anche i dati vengono ricaricati dinamicamente e la visuale cambia in base ad esso.

5.4 Test

Per un'applicazione di questo tipo è difficile creare dei test specifici se non navigando attraverso il sito e osservando se tutto funziona correttamente.
Sicuramente in fase di progettazione è stato di fondamentale importanza il debug tramite strumenti come *Firebug*[2]. Esso è un'estensione di Mozilla Firefox che permette il debug, la modifica e il monitoraggio di tutti gli aspetti di una pagina web, come il codice HTML, i fogli di stile, la struttura DOM e il codice JavaScript. Fornisce anche altri strumenti per

[2]Firebug: http://getfirebug.com

lo sviluppo web come una console JavaScript e una funzione chiamata "Net" che permette di monitorare il tempo di caricamento in millisecondi di immagini e script e relative performance (Figura 5.4).

È possibile quindi monitorare le chiamate fatte dall'applicazione al server e vedere quale risposta restituisce. Oppure tramite la console di Firebug controllare i dati contenuti in quel momento da un oggetto.

5.4.1 Compatibilità

Data la nostra tipologia di applicazione, è necessario tenere conto di cosa potrebbe utilizzare l'utente finale. Analizzando il nostro contesto, dove è possibile avere una versione dell'applicazione che gira prevalentemente su una piattaforma decisa in precedenza (e ridotta in termini di interattività e funzionalità negli altri casi) abbiamo deciso di orientarci prevalentemente sul browser *Mozilla Firefox*.

L'utilizzo di elementi moderni come HTML5 (*canvas* e *history*) e CSS3 (trasformazioni ed effetti) su browser che non implementano ancora tutte le possibili funzionalità rendono lo sviluppo di applicazioni web una sfida complessa, con tante variabili da tenere conto. Oltre a questo HTML5 non è stato ancora ratificato come standard in vigore, lasciando ancora dei buchi implementativi.
Nonostante questo è stato deciso di effettuare lo sviluppo su un browser singolo, cercando però di utilizzare nel possibile metodi comuni a tutti i browser. Ad esempio nelle trasformazioni è stato deciso di implementare le varianti per ogni browser.

Inoltre tramite siti come *CanIUse.com*[3] e *HTML5Please*[4] o ancora http://www.findmebyip.com/litmus/ abbiamo potuto verificare le compatibilità dei vari browser con le nuove direttive imposte dall'HTML5.

5.4.1.1 Browser testati

I browser presi in considerazione per i test sono i seguenti:

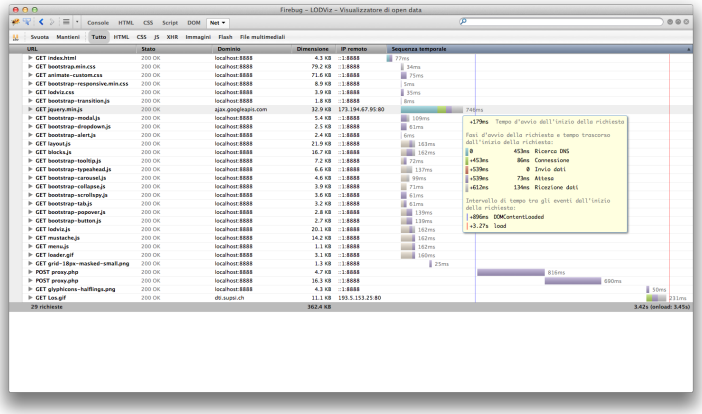
Windows 7

- Internet Explorer 9
- Firefox 12
- Opera 11.62
- Chrome 18 (Stable), 19 (Beta), 20 (Canary), 20 (Chromium)
- Safari 5.1.7

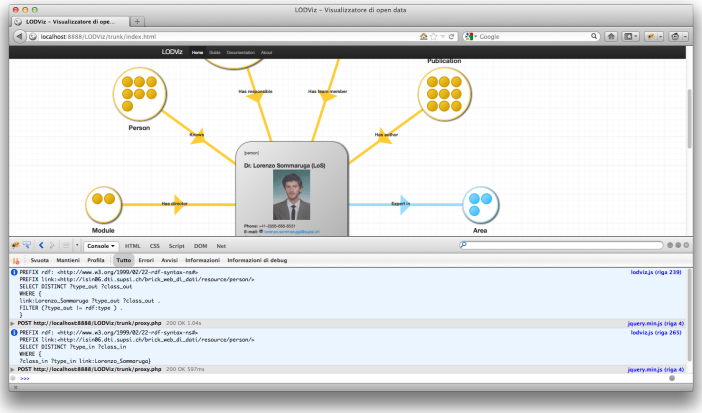
Mac OS X Lion (10.7)

- Firefox 12
- Chrome 18
- Opera 11
- Safari 5.1.7

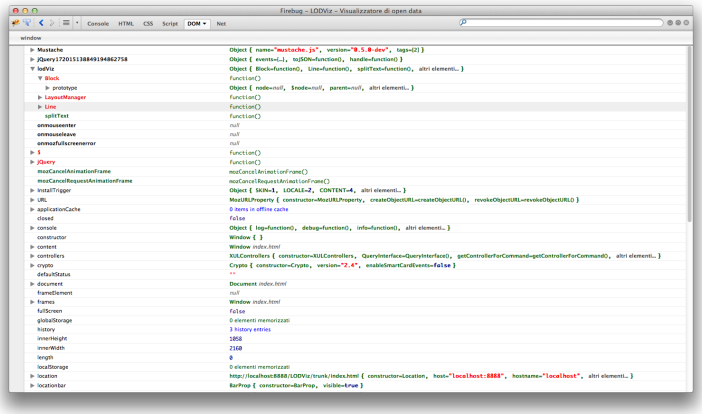
[3]Can I use: http://caniuse.com
[4]HTML5 Please: http://html5please.com



(a) Tempo di caricamento dei singoli elementi della pagina.



(b) Utilizzo della console.



(c) Esplorazione del contenuto degli oggetti.

Figura 5.4: Utilizzo di Firebug durante il debugging del progetto.

5.4.1.2 Note sulla compatibilità

Sono stati fatti degli sforzi nel cercare di rendere l'esperienza d'uso uguale su tutti i browser, e in caso di mancanze (ad esempio l'*HTML5 History* in IE9) è stato inserito del codice per evitare problemi.

Di seguito viene mostrato lo stato di compatibilità per ogni browser.

Firefox 12

Firefox si comporta in modo corretto anche perché è stato scelto come piattaforma principale.

Chrome (su Windows 7)

Chrome sotto Windows 7 ha una gestione non corretta della rotazione dei *canvas*. Pertanto durante il disegno della freccia e la sua rotazione, si nota la mancanza degli stessi o comunque una grafica non corretta (Figura 5.5).

La versione *Beta*, *Canary* e addirittura la versione *Bleeding Edge (Chromium)* soffrono degli stessi problemi, anche se quest'ultimo disegna correttamente le frecce nel caso che abbiamo una rotazione di 45°.

Figura 5.5: Problemi di visualizzazione di Chrome su Windows 7.

Per valutare se la rotazione veniva effettuata correttamente, abbiamo utilizzato la modalità di debug (sotto `about:flags`) per il disegno degli elementi; vedi Figura 5.6).

Come si nota dalla Figura 5.7, si notano gli elementi orientati in modo corretto, ma il loro contenuto è assente o comunque incompleto.
Sotto Chromium si notano dei leggeri miglioramenti, ma anche qui il risultato è insoddisfacente.

Chrome (su Mac OS X)

Chrome si comporta in modo corretto.

Opera

Opera si comporta in modo corretto su entrambi i sistemi operativi.

Safari

Safari si comporta in modo corretto su entrambi i sistemi operativi.

Internet Explorer 9

Si notano alcuni problemi, anche se essi non influiscono nelle funzionalità di base. In

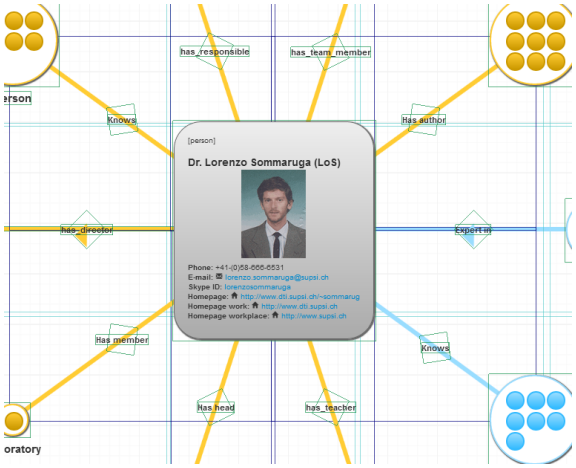


Figura 5.6: Modalità debug di Chrome su Windows 7.

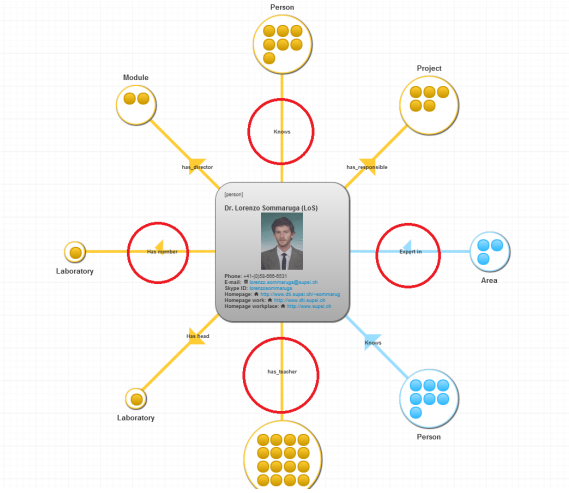


Figura 5.7: Problemi di visualizzazione di Chrome su Windows 7.

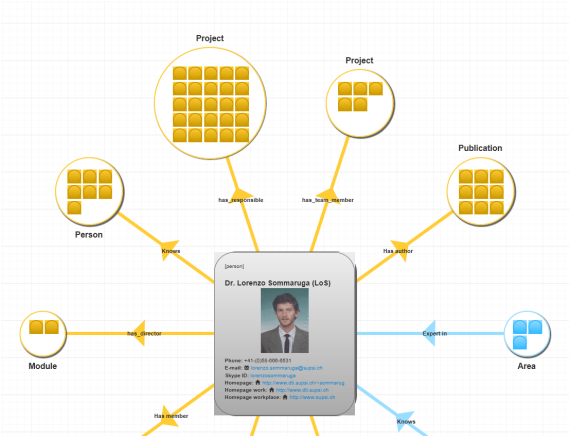


Figura 5.8: Problemi di visualizzazione di Internet Explorer 9 su Windows 7.

particolare i gradienti non sono mostrati in modo gradevole, mostrando rettangoli al posto di cerchi (Figura 5.8).

L'*HTML5 History* non è supportata, pertanto non è possibile muoversi nella cronologia di navigazione per le chiamate AJAX.
Manca completamente la parte di effetti e animazioni CSS.
Infine anche l'indicatore di caricamento dei dati non è visibile e tutto questo peggiora l'esperienza d'uso.

Capitolo 6

Piani di Lavoro

Di seguito è riportata la pianificazione del tempo e analisi dei lavori del progetto. Decisamente importante al fine di portare a termine il lavoro è stato un iniziale diagramma di Gantt, redatto nella prima settimana quando si sono definiti compiti ed obiettivi, così come una corretta gestione del tempo e relative priorità.

6.1 Diagrammi di Gantt

Come si può notare dal piano iniziale (Figura 6.1a) erano previste scadenze settimanali, ovvero incontri e discussioni con i relatori per aggiornarli sullo stato del progetto.
Nel diagramma finale (Figura 6.1b) è invece rappresentata la gestione effettiva del tempo e si nota come la parte iniziale di studio e test del linguaggio JavaScript abbia ricoperto una parte preponderante della pianificazione.

I tempi della progettazione hanno richiesto un tempo non indifferente a causa della rivalutazione e modifica della struttura dati usata dal programma e si sono quindi dilatati. Il codice inoltre è stato più volte riscritto e riadattato conseguentemente all'avanzare delle nostre competenze e delle richieste da parte del committente.
Per lo sviluppo il tempo rimanente era relativamente poco, pertanto si è deciso di effettuare una prima soluzione (quella con i numeri dentro le bolle) per ottenere perlomeno un risultato intermedio. Acquisendo esperienza si è riusciti verso la fine ad aumentare la produttività e a raggiungere invece la soluzione prevista inizialmente con il committente.

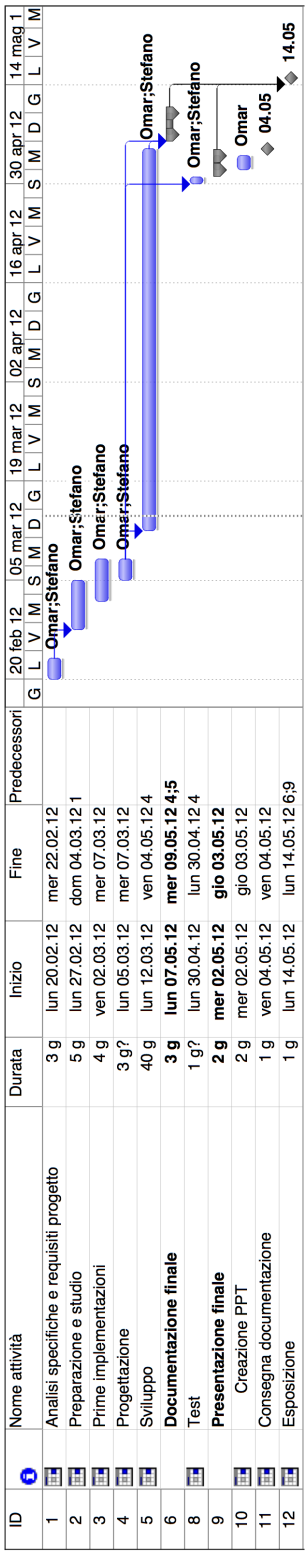
Riguardo a *milestone*, essi erano perlopiù riferiti ai meeting. Non abbiamo avuto necessità di effettuare delle *deliverable*, anche se durante gli incontri di aggiornamento sullo stato del progetto si mostrava quanto sviluppato fino a quel punto.

Da notare anche l'aggiunta della reale data di presentazione (21 maggio 2012) mentre nella pianificazione iniziale quest'informazione era fittizia perché non era ancora definita.

6.1.1 Principio di Pareto

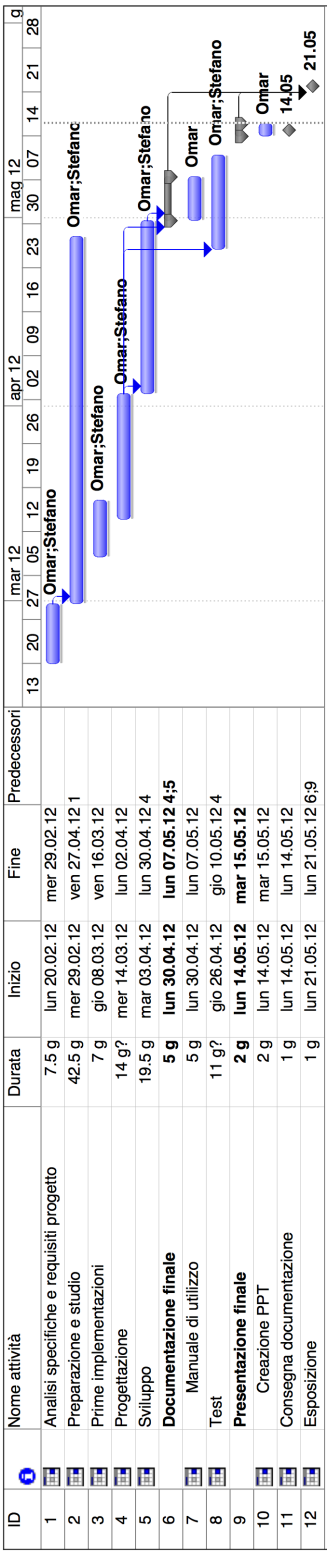
Riguardo al principio di Pareto[1], abbiamo potuto constatare che la parte di visualizzazione richiedeva comunque un impegno molto elevato (sia in termini di complessità del codice che di elaborazione). Si poteva semplificare questo processo e raggiungere un risultato simile con meno complessità, ma avrebbe comportato l'impossibilità di arrivare al traguardo prefissato originariamente. In ogni caso gli ultimi passi per arrivare alla fine del progetto

[1]Principio di Pareto: http://it.wikipedia.org/wiki/Principio_di_Pareto



ID	Nome attività	Durata	Inizio	Fine	Predecessori
1	Analisi specifiche e requisiti progetto	3 g	lun 20.02.12	mer 22.02.12	
2	Preparazione e studio	5 g	lun 27.02.12	dom 04.03.12	1
3	Prime implementazioni	4 g	ven 02.03.12	mer 07.03.12	
4	Progettazione	3 g?	lun 05.03.12	mer 07.03.12	
5	Sviluppo	40 g	lun 12.03.12	ven 04.05.12	4
6	**Documentazione finale**	**3 g**	**lun 07.05.12**	**mer 09.05.12**	**4;5**
8	Test	1 g?	lun 30.04.12	lun 30.04.12	4
9	**Presentazione finale**	**2 g**	**mer 02.05.12**	**gio 03.05.12**	
10	Creazione PPT	2 g	mer 02.05.12	gio 03.05.12	
11	Consegna documentazione	1 g	ven 04.05.12	ven 04.05.12	
12	Esposizione	1 g	lun 14.05.12	lun 14.05.12	6;9

(a) Piano di lavoro iniziale



ID	Nome attività	Durata	Inizio	Fine	Predecessori
1	Analisi specifiche e requisiti progetto	7.5 g	lun 20.02.12	mer 29.02.12	
2	Preparazione e studio	42.5 g	mer 29.02.12	ven 27.04.12	1
3	Prime implementazioni	7 g	gio 08.03.12	ven 16.03.12	
4	Progettazione	14 g?	mer 14.03.12	lun 02.04.12	
5	Sviluppo	19.5 g	mar 03.04.12	lun 30.04.12	4
6	**Documentazione finale**	**5 g**	**lun 30.04.12**	**lun 07.05.12**	**4;5**
7	Manuale di utilizzo	5 g	lun 30.04.12	lun 07.05.12	
8	Test	11 g?	gio 26.04.12	gio 10.05.12	4
9	**Presentazione finale**	**2 g**	**lun 14.05.12**	**mar 15.05.12**	
10	Creazione PPT	2 g	lun 14.05.12	mar 15.05.12	
11	Consegna documentazione	1 g	lun 14.05.12	lun 14.05.12	
12	Esposizione	1 g	lun 21.05.12	lun 21.05.12	6;9

(b) Piano di lavoro finale

Figura 6.1: Diagramma di Gantt con i piani di lavoro

sono stati molto impegnativi, quindi si può dire che le percentuali previste da Pareto sono state realistiche.

6.2 Meeting

Per l'esecuzione del lavoro era importante trovarci regolarmente con i nostri committenti in modo da poter valutare lo stato di avanzamento del lavoro o ottenere informazioni e consigli sull'uso di certe tecnologie.
Tutti i nostri meeting sono stati fondamentali per l'avanzamento e la riuscita del progetto, sia che essi fossero formativi, spontanei o decisionali.

L'uso di protocolli è stato effettuato principalmente agli inizi, quando bisognava impostare il progetto e valutare i dettagli principali, come ad esempio lo schema grafico.

I meeting creativi sono stati effettuati solo agli inizi, ma data la natura del progetto e la volontà di ottenere un risultato già abbastanza definito, non sono state sviluppate in modo approfondito altre idee, sia a causa del tempo mancante sia a causa dei vincoli imposti. Il margine di creatività è stato quindi molto limitato.

I nostri meeting erano principalmente di tipo spontaneo (misto), anche se spesso c'era un preavviso, ci si trovava comunque in un momento considerato libero da entrambe parti.

Il numero di meeting è stato poi ridotto da frequenza settimanale a frequenza stabilita di volta in volta, permettendo un certo margine di sviluppo indipendente, ma con la certezza che era possibile avere un incontro in tempi molto brevi se ci fossero stati problemi.

Verso la fine dei termini di consegna abbiamo effettuato dei meeting decisionali per valutare lo stato attuale del progetto e capire cos'era ancora possibile modificare per ottenere un risultato soddisfacente. Il nostro metodo di sviluppo agile ha permesso di effettuare diverse modifiche in un tempo ridotto, consentendo quindi il raggiungimento degli obiettivi principali.

Queste riunioni richiedevano comunque un tempo non indifferente per essere effettuate (in media un paio d'ore).
Anche se i meeting erano di diverse tipologie, non c'è stata una preparazione approfondita e quindi gli incontri non avevano una direzione netta, aumentando quindi i tempi dello stesso e causando la necessità di ulteriori contatti per dettagli non discussi in modo approfondito negli stessi.

6.3 Time management

Il progetto si compone principalmente di una parte visuale e di una parte di codice per lettura, interpretazione e visualizzazione dei dati. A livello di distinzione tra le varie attività si è scelto di privilegiare principalmente la parte di codice, relegando quella puramente visuale ad un passo successivo.

Le macro aree sono state quindi quelle di analisi e studio, progettazione (parte visuale e codice vero e proprio), documentazione ed infine consegna ed esposizione. Queste aree chiave, con le relative priorità e rilevanze, sono ben schematizzate nella Tabella 6.1.

L'attività veniva considerata conclusa dal momento che il progetto rispettava i requisiti minimi richiesti, ma nasceva comunque un problema di dipendenza tra le varie aree. L'area visuale è quella che stabiliva se il progetto era terminato o no. Il codice e la documentazione seguivano di pari passo l'evoluzione della parte grafica.

Data la natura del progetto, venivano spesso effettuati test su aree specifiche del progetto, sviluppando il codice fino ad arrivare ad un punto tale da poter gestire i casi più comuni (ed eventuali errori).
Il *testing* vero e proprio è stato effettuato dapprima da noi, e poi successivamente dal committente. Le prove venivano effettuato per ogni tipologia di categoria caricata (come ad esempio *Person*, *Laboratory*, *Courses* e cosi via).

Riguardo al *monitoring* del tempo, anche se la pianificazione iniziale poteva essere una buona strada, alla fine ci si è resi conto che il tempo a disposizione era molto inferiore. I problemi sono stati la valutazione del tempo di apprendimento delle nuove tecnologie e l'applicazione delle stesse, mitigate comunque in parte effettuando meeting formativi con esperti del settore (a nostra disposizione durante l'intero corso del progetto).
Verso la fine abbiamo stilato anche una tabella più dettagliata di compiti e scadenze cercando quindi di rispettarle il più possibile così da permettere al progetto di avanzare e di essere consolidato ad ogni step intermedio.
Grazie anche a successivi incontri eseguiti in tempi brevi, e all'approccio di sviluppo agile adottato, abbiamo potuto implementare in tempi brevi tutte le parti necessarie per soddisfare i requisiti originali.

Tabella 6.1: Tabella di gestione del tempo e relative priorità.

Priorità	**Rilevanza**	**Area chiave**	**Descrizione**	**Responsabile**	**Scadenza**
A	B	Analisi	Analisi specifiche e requisiti progetto	ODM[1], SM[2], LS[3]	22/02/2012
B	B	Analisi	Riunioni intermedie	ODM, SM, LS	settimanali
B	B	Studio	Preparazione e studio	ODM, SM	04/03/2012
A	B	Studio	Prime implementazioni	ODM, SM	07/03/2012
A	A	Progettazione	Progettazione	ODM, SM	07/03/2012
A	A	Progettazione	Sviluppo	ODM, SM	30/04/2012
B	B	Progettazione	Test globali e finali	ODM, SM, LS	30/04/2012
C	A	Progettazione	Consegna progetto al committente	ODM, SM, LS	04/05/2012
C	A	Documentazione	Presentazione finale	ODM, SM	03/05/2012
C	A	Documentazione	Rapporto finale	ODM, SM	03/05/2012
C	A	Consegna	Consegna documentazione	ODM, SM	04/05/2012
C	A	Esposizione	Esposizione del progetto	ODM, SM	21/05/2012

[1]ODM: Omar Davide Molteni
[2]SM: Stefano Mondini
[3]LS: Lorenzo Sommaruga

Capitolo 7

Conclusioni

Il progetto si è concluso positivamente e tramite esso è possibile dimostrare, in formato grafico, le potenzialità del Linked Data. Infatti, l'intero lavoro potrebbe sembrare a prima vista un banale sito web ma in realtà è un mondo, come lo rappresenta bene la Figura 4.5 a pag. 25, composto da possibili infiniti collegamenti.

Durante lo sviluppo di questo progetto abbiamo imparato nuove tecnologie come JavaScript e pure le varie librerie che lo completano, jQuery *in primis*. Abbiamo inoltre approfondito i concetti di web semantico e del suo utilizzo pratico senza tralasciare le nozioni sicuramente utili acquisite durante il corso di Ingegneria del Software 3.

7.1 Risultati

Il progetto si è concluso in modo positivo, ed è stato possibile implementare le richieste principali del committente in una forma da subito utilizzabile. Il lavoro di elaborazione dati e visualizzazione è risultato molto più impegnativo del previsto, sia a causa delle nuove tecnologie da imparare e utilizzare, sia a causa dell'evoluzione del programma stesso.

Grazie a questo progetto si intravedere le potenzialità del Linked Data, anche se attualmente ci basiamo su un sottoinsieme di dati del progetto Brick. È quindi possibile navigare ed esplorare una moltitudine di collegamenti, ottenendo informazioni sia sull'elemento scelto, sia sulle varie relazioni. Questo è importante per ottenere un web semantico, dove ogni collegamento è riferito ad una sua ontologia e a un suo significato. Il fatto di avere delle relazioni visive rispetto ad una forma tabellare, permette di osservare rapidamente le relazioni e le loro implicazioni.

Questo progetto è stato molto importante anche a livello personale, grazie al fatto di aver dovuto vedere più tecnologie complementari tra loro in maniera abbastanza approfondita (soprattutto JavaScript), e applicato ad una situazione reale con tecnologie moderne (HTML5, CSS3). Abbiamo anche sfruttato librerie esterne come jQuery, Mustache e Bootstrap, cercando di ottenere un software coerente e funzionale.

A livello di sviluppo abbiamo sfruttato appieno la gestione delle versioni tramite Subversion, permettendoci di collaborare e di effettuare modifiche in tranquillità.

Grazie anche alle lezioni di Ingegneria del Software 3, abbiamo potuto applicare la moltitudine di argomenti affrontati, come ad esempio la gestione del rischio (valutazione ed eventuale applicazione di eventi correttivi), il modo di affrontare i meeting nonché l'analisi e la critica sull'interfaccia grafica, la sua interattività e le implicazioni causate dall'uso di

persone diverse. Abbiamo anche cercato di provvedere a descrivere nella documentazione le parti più importanti riguardo a questi argomenti.

7.2 Problemi riscontrati

Come prima cosa il fattore tempo è qualcosa che incide in maniera molto determinante sulla riuscita di un lavoro. Ci sono tempi da rispettare e un certo tempo per capire, sviluppare e testare il tutto. Pertanto è stato necessario fare una prima pianificazione di massima. A causa però dei tempi stretti e del ritardo nell'arrivare ad un livello sufficiente di conoscenze e pratica, è stato necessario affrontare lo sviluppo in maniera agile e con iterazioni molto rapide.

Oltre a questo lo sviluppo dell'applicazione stessa è cresciuta in maniera decisiva verso la fine, creando però del codice che bisognerà rivedere e ristrutturare secondo i canoni dell'ingegneria del software, applicando eventualmente i design pattern più idonei.

Una fase critica è stata lo sviluppo della parte visiva ed interattiva. Grazie ai relatori siamo riusciti ad arrivare ad un risultato simile a quanto si era pianificato inizialmente, ma era necessario effettuare riunioni in tempi brevi per correggere errori e valutazioni errate nella presentazione grafica.

Un altro problema rilevante è la compatibilità tra piattaforme e browser diversi. Anche se il codice è già in parte indipendente dal browser (grazie anche al fatto di usare librerie che fanno già astrazione, come jQuery) abbiamo notato diversi problemi nella visualizzazione di certe combinazioni di browser e sistemi operativi. Non è stato possibile in certi casi avere una correzione o soluzione temporanea per arginare il problema.

7.3 Sviluppi futuri

A livello di codice si può migliorare e rifattorizzare la struttura, eliminando le eventuali ridondanze, correggendo i bug rimanenti ed eventualmente implementare meccanismi più sofisticati per la gestione dei dati e il posizionamento dei blocchi.

È anche necessario definire bene i dati di ingresso e i loro tipi, in modo da ottenere più versatilità nell'elaborazione degli stessi (ad esempio non avendo i dati con solo tipi "URI" e "literal").

Oltre alla questione di ottenere un codice corretto e funzionale, è importante eseguire test su più piattaforme e browser in modo da ridurre gli errori e ottenere qualcosa di realmente indipendente da essi (*cross-browser*). In ogni caso è necessario stabilire una serie di requisiti minimi, come ad esempio il supporto a certe funzionalità HTML5, ma è importante avere meccanismi alternativi in caso di mancanza degli stessi.

Bibliografia

Riferimenti bibliografici

Castledine, E., C. Sharkie e M. Perotti

2011 *JQuery. Guida completa*, Guida completa, Apogeo, ISBN: 9788850330348.

Dadzie, Aba-Sah e Matthew Rowe

2011 "Approaches to visualising Linked Data: A survey.", *Semantic Web*, 2, 2, pp. 89–124, http://dblp.uni-trier.de/db/journals/semweb/semweb2.html#DadzieR11.

Freeman, E. e E. Robson

2011 *Head First HTML5 Programming: Building Web Apps with JavaScript*, Head First, O'Reilly Media, ISBN: 9781449390549.

Gigliotti, Gabriele

2011 *HTML 5 e CSS 3*, Apogeo, ISBN: 9788850330119.

Pilgrim, M. e B. Sansone

2011 *HTML 5. Guida operativa*, Tecniche Nuove, ISBN: 9788848125048.

Steele, J. e N. Iliinsky

2011 *Designing Data Visualizations: Representing Informational Relationships*, Oreilly and Associate Series, O'Reilly Media, ISBN: 9781449312282.

Yank, K. e C. Adams

2008 *JavaScript*, Guida completa, Apogeo, ISBN: 9788850327577.

Sitografia

Mozilla Developer Network - JavaScript [2012], https://developer.mozilla.org/en/JavaScript.

Wikipedia - L'enciclopedia libera [2011], http://www.wikipedia.org.

Allegati

CD-ROM

- Codice sorgente
- Documentazione
- Presentazione
- Poster
- Scheda catalogo