Alma Mater Studiorum · Università di Bologna

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Triennale in Informatica

BIRD-A

Una piattaforma per la creazione di interfacce Web basate su ontologie per l'authoring di dati RDF

Relatore:
Chiar.mo Prof.
Fabio Vitali

Correlatore:
Dott.
Silvio Peroni

Presentata da:
Francesco Caliumi

Sessione III
Anno Accademico 2014/2015

Contents

1 Introduzione **7**

1.1 Un web di dati semantici 8

1.2 La creazione di dati semantici 9

1.3 Un tool general purpose distribuito: Bird-A 11

1.4 Struttura della dissertazione 13

2 Semantic web e RDF data authoring **15**

2.1 World Wide Web: La situazione attuale 15

2.1.1 I fattori che hanno decretato il successo del Web 1.0 16

2.1.2 I fattori che hanno permesso l'evoluzione verso il Web 2.0 17

2.2 World Wide Web: Verso il Semantic Web 17

2.3 Rappresentare la conoscenza: Ontologie e RDF 18

2.3.1 Lo standard RDF 18

2.4 Attuale natura dei Dataset RDF 19

2.5 RDF data authoring 20

2.5.1 Caratteristiche desiderabili 21

2.6 Panorama attuale dell'RDF Data Authoring 22

2.6.1 Haystack 22

2.6.2 Mangrove 23

2.6.3 Loomp 23

2.6.4 SPARQL Web Pages 23

2.6.5 TopBraid Platform 23

2.6.6 Semantic Wiki 24

2.6.7 Altri progetti . . . 25
2.7 Gaffe . . . 25
2.8 Considerazioni . . . 26

3 Il progetto Bird-A **27**
3.1 L'approccio di Bird-A . . . 27
3.1.1 I Widget . . . 29
3.1.2 La separazione dei ruoli . . . 29
3.2 L'architettura di Bird-A . . . 31
3.2.1 Backend . . . 31
3.2.2 Frontend . . . 32
3.3 Interfaccia Utente . . . 33
3.4 La Bird-A Ontology . . . 38
3.5 Le Bird-A API . . . 39
3.6 Aderenza agli obiettivi progettuali . . . 40

4 Bird-A: Dettagli implementativi **43**
4.1 Lo stack tecnologico . . . 43
4.2 La Bird-A Ontology . . . 45
4.2.1 Entità . . . 45
4.2.2 Proprietà . . . 46
4.2.3 Esempio di Bird-A Ontology . . . 47
4.3 Le Bird-A API . . . 50
4.3.1 Formati JSON . . . 50
4.3.2 Servizi . . . 55
4.4 Viste Frontend . . . 57
4.4.1 Home e altre pagine . . . 57
4.4.2 Lista dei form . . . 57
4.4.3 Lista delle istanze . . . 57
4.4.4 Edit di un'istanza . . . 58
4.5 Struttura directory di Bird-A . . . 58
4.5.1 Struttura directory generale . . . 58

4.5.2 Struttura e funzionamento Backend 59
4.5.3 Struttura e funzionamento Frontend 63
4.6 Build e Run 67
4.7 Conclusione 68

5 Bird-A: Valutazioni **69**
5.1 Soddisfacimento dei requisiti 69
5.1.1 General purpose 69
5.1.2 Collaborativo 70
5.1.3 Distribuito 71
5.1.4 Accessibile 71
5.1.5 User friendly 72
5.2 Valutazioni implementative 72
5.2.1 Affidabilità 73
5.2.2 Efficienza 73
5.2.3 Facilità di adozione e sviluppo 74
5.3 Sviluppi futuri 75
5.3.1 Bug noti 75
5.3.2 Sviluppi necessari 76
5.3.3 Migliorie apportabili 77
5.3.4 Feature auspicabili 78
5.4 Punti aperti 80
5.5 Conclusione 82

6 Conclusioni **83**

7 Bibliografia **85**

1. Introduzione

L'evoluzione che sta portando il Web dall'essere un veicolo di documenti verso l'essere un veicolo di dati semantici vede alla sua base l'espressione delle informazioni sotto forma di asserzioni (oggetto dello standard W3C RDF [W3C14] e RDFS [W3C14]) e la strutturazione di queste asserzioni in ontologie (oggetto dello standard W3C OWL [W3C12]), ovverosia concettualizzazioni specializzate, formali ed esplicite che, dato un dominio di interesse come ad esempio le risorse bibliografiche, ne determinano le classi, le proprietà e gli individui.

Lo standard RFD (Resource Description Framework) indica di considerare ogni individuo, classe o proprietà come una risorsa identificata da un URI (Uniform Resource Identifier, definita dall' apposito standard [BFM05]) e che ogni asserzione associ ad una risorsa (il soggetto) un predicato ed un valore (eventualmente un'altra risorsa). Questa capacità di collegare risorse tramite asserzioni (o triple) rende possibile pubblicare dati fra loro collegati e quindi fruibili attraverso interrogazioni semantiche: i Linked Data [BER06].

Da quando questa evoluzione è stata proposta da Tim Berners-Lee nel 2001 [BHL01], sono nati molti progetti per consentire a questa visione di diventare realtà.

Fra i più importanti vi sono i porting degli attuali database, come ad esempio per i dati del governo degli Stati Uniti [DAT16], e le integrazioni con le tecnologie wiki per consentire la metadatazione dei documenti o la creazione di dati RDF standalone, come ad esempio Wikidata [VK14].

Fino ad ora però, pochi progetti hanno provato a creare uno strumento di inserimento

di dati semantici general purpose, non legato cioè ad alcuna ontologia specifica, e che non richiedesse sviluppi ad hoc per creare nuovi form di inserimento, come nel caso di una nuova ontologia o semplicemente per inserire un set di informazioni differenti rispetto ad una già gestita. Oltre a ciò, gli strumenti esistenti si basano per lo più sulla gestione di un solo dataset RDF e non prevedono l'integrazione di un insieme arbitrario di dataset.

Per offrire un tool di questo tipo in quei contesti dove le soluzioni fino ad ora sviluppate o non sono applicabili (oppure sono applicabili con forti difficoltà), è nato il progetto Bird-A.

Bird-A mira ad essere uno strumento di authoring di dati RDF configurabile, flessibile e distribuito volto a fornire all'amministratore di dataset la più completa libertà d'azione, presentando agli utenti form di inserimento semplici ed intuitivi che astraggano le logiche ontologiche sottostanti e con la possibilità di accedere a dati provenienti da più fonti.

In questa dissertazione si esporrà quali siano i requisiti auspicabili per un progetto di questo tipo e come si sia arrivati alla loro definizione, si vedranno quali sono nel panorama tecnologico le migliori soluzioni realizzate per far fronte a problemi simili ed infine si affronterà l'architettura Bird-A, come questa ambisce a soddisfare i requisiti attuali e come sia stata realizzata la prima implementazione delle funzionalità core.

1.1 Un web di dati semantici

Come già accennato, la maggior parte delle informazioni oggi presenti sul web è composto da documenti, un formato concepito per un utilizzo pressoché unicamente umano. Il continuo crescere del numero di informazioni presenti, rende tuttavia molto appetibile la possibilità di processare queste informazioni in modo automatico al fine di effettuare lavori di sintesi e di deduzione su larga scala.

Per consentire però ad una macchina di processare queste informazioni, esse devono trovarsi in un formato adeguato, semplice e chiaro. Devono cioè essere decomposte nei dati basilari che le compongono.

Il formato adottato per questa rappresentazione è come già accennato RDF (Resource Description Framework). RDF si basa sul principio "the least power" ovvero di utilizzare il linguaggio meno espressivo per descrivere qualunque cosa. In RDF ogni "cosa" è una risorsa a sua volta identificata da un'URI. Ogni informazione in questo formato è espressa come una o più affermazioni, composte da soggetto (una risorsa), predicato (o proprietà) e oggetto (o valore, che può essere sia di tipo primitivo come un intero sia un'altra risorsa) dette anche triple.

Il fatto che le risorse oltre ad essere soggetti possano essere anche oggetti di una tripla porta ad una struttura a grafo di dati tra loro collegati: i Linked Data [BER06].

Questi dati insieme al significato che essi assumono nell'ambito delle concettualizzazioni semantiche che li descrivono, le ontologie, sono alla base dell'evoluzione verso il Semantic Web.

L'espansione dell'utilizzo del web come uno strumento di definizione di dati semantici necessita di tool che facilitino da una parte chi definisce le formalizzazioni della conoscenza (ontologie), dall'altro chi crea i dati in modo che siano conformi a queste formalizzazioni.

1.2 La creazione di dati semantici

Gli ultimi dieci anni hanno visto la nascita di svariati progetti per mettere a disposizione sempre più "dati connessi" e per facilitarne la fruizione. Alcuni di questi hanno avuto un discreto successo e sono attualmente utilizzati, mentre molti altri sono purtroppo naufragati. Nel capitolo 2 verranno analizzati con maggiore dettaglio.

Tra questi progetti si possono identificare alcune diverse categorie in base all'ambito su cui si concentrano:

- Conversione di database esistenti: sono i progetti più diffusi e solitamente con la minore probabilità di insuccesso. Il loro obiettivo è di esporre informazioni già presenti in qualche forma strutturata, come ad esempio i database relazionali.

Esempi a riguardo sono i dati del censo degli Stati Uniti o i vari database biologici, farmaceutici, ecc.

- Deduzione di dati: partendo da informazioni non originariamente pensate per l'elaborazione, questi progetti provano ad estrarne i dati che le compongono. Un esempio eccellente in questo ambito è DBPedia [DBP16], che sfrutta i riquadri laterali delle pagine di Wikipedia a questo scopo.

- Metadatazione di documenti: altra categoria di progetti abbastanza diffusa sono quelli che hanno come obiettivo l'estensione dei tool attuali di editing di documenti (wiki in primis, ma anche Microsoft Word, ecc.) per prevedere l'inserimento di specifici metadati in formato RDF.

- Creazione di dati RDF "standalone": dell'ultima categoria fan parte i progetti che consentono di specificare asserzioni RDF su generiche risorse RDF anziché su documenti. Un esempio è WikiData, in cui gli utenti stessi possono editare il "datamodel" (una sorta di ontologia non formalizzata) ed editare i dati RDF ad esso riferiti.

Sebbene questi progetti siano assolutamente validi ed in grado di raggiungere gli obiettivi che si prefiggono, nessuno tra quelli attualmente sviluppati permette l'editing di dati RDF su un'ontologia target configurabile (con la possibilità di gestirne un insieme non limitato).

Altra limitazione di molti di questi progetti è il focus su un solo dataset RDF per ogni installazione, con l'impossibilità di presentare (ed editare) dati provenienti da più dataset.

Per fornire uno strumento open source laddove vi sia l'esigenza di non avere queste limitazioni, nasce il progetto Bird-A.

1.3 Un tool general purpose distribuito: Bird-A

Bird-A mira a fornire uno strumento attraverso il quale un amministratore di dataset RDF, possa presentare in modo comodo e conveniente, ai fruitori da lui designati, i dati provenienti dai propri dataset (eventualmente integrati con dati esterni) e permetterne la modifica secondo opportuni form completamente configurabili.

Punti cardine di questo progetto sono quindi:

- La flessibilità dell'interfaccia e la separazione dai dati sottostanti.
- La piena configurabilità dei form.
- La divisione dei ruoli fra chi definisce il form, chi amministra i dati su cui esso opera e chi edita i dati.
- La possibilità di interazione con diversi dataset.
- La capacità di decidere chi e come può accedere ai vari dataset.
- La messa a disposizione, per l'utente finale, di uno strumento comodo e facile da usare.

Per garantire la flessibilità dell'interfaccia e la separazione dai dati sottostanti, Bird-A attinge ai principi enunciati da Gaffe (Generator of Automatic Form - Final Edition) [BID09]. Tra questi il più importante è l'applicazione del diffuso pattern MVC (Model-View-Controller [RE79]) alla progettazione concettuale di un editor di dati semantici. Gaffe mutua infatti da questo pattern netta separazione fra tre dimensioni:

- "Come il dato è modellato" (model): specifica la gestione dei dati effettivi in conformità ad una qualche ontologia di riferimento.
- "Come il modello è presentato in output" (view): stabilisce come questi dati debbano essere presentati. Un model può ovviamente avere più view e il cambiamento dell'organizzazione interna del model non influenza le view ad esso associate.
- "Come l'input condizioni lo stato interno del modello" (controller): gestisce l'interazione con l'utente, la validazione degli input, ecc.

La configurabilità dei form si basa invece sul principio di "the least power" che guida lo standard RDF stesso, fornendo un'ontologia puramente presentazionale, senza alcuna implicazione o assunzione sul significato semantico dei campi. I form permettono l'inserimento di dati relativi ad una specifica risorsa RDF e si strutturano in widget, ciascuno dei quali si occupa di gestire le triple associate ad una proprietà RDF. Questa ontologia, che prende il nome di Bird-A Ontology, non fa alcuna assunzione né sul significato della risorsa, né su quello delle proprietà, dandole così il potere espressivo di creare triple per qualsiasi ontologia (si vedano i capitoli 3 e 4 per una spiegazione più dettagliata)

Il rispetto di questi principi porta alla possibilità di separare nettamente i ruoli operanti nella generazione di dati RDF, ciascuno dei quali con delle specifiche responsabilità e con diversi bagagli tecnologici:

- Ingegnere della conoscenza (Knowledge Engineer): colui che progetta e formalizza le ontologie.
- Progettista di form: colui che date le ontologie a disposizione configura i form in grado di generare e presentare dati che rispettano dette ontologie.
- Amministratore di dataset: colui che sfrutta le ontologie e i form a disposizione per queste ontologie per creare un proprio dataset di cui abbia il pieno controllo, anche sotto il profilo degli accessi.
- Fruitore dei dati: colui che consulta ed edita i dati del dataset.

I form sono quindi istanziabili da figure in possesso di conoscenza di tecnologie semantiche (ma che non hanno necessariamente competenze di sviluppo software) e fruibili da una qualunque figura in possesso della sola conoscenza del dominio di interesse (e tendenzialmente nessuna conoscenza in ambito ontologico o di sviluppo software).

Per garantire la possibilità di interagire con più dataset, ciascuno dei quali eventualmente gestito da un'autorità a sé stante, Bird-A adotta un'architettura distribuita che vede una forte separazione di ruoli fra i backend, che servono i form e i dati tramite API, e il frontend, che accede a tali API per presentare i form, permettere la ricerca di dati ed il loro editing.

Nell'architettura Bird-A (spiegata più approfonditamente nel capitolo 3), i backend possono essere gestiti da più entità, ciascuna delle quali ha autorità su uno o più dataset RDF, mentre il frontend può essere, in linea di principio, unico ed avvalersi dei servizi offerti dai backend.

1.4 Struttura della dissertazione

Prima di scendere nel dettaglio della struttura del progetto Bird-A, è utile stilare una lista di requisiti desiderati che un sistema di RDF data authoring dovrebbe soddisfare. Fatto ciò, è opportuno osservare sinteticamente quale sia il panorama attuale dei progetti simili, analizzando perché le soluzioni attualmente implementate non siano adeguate a risolvere i problemi in esame e cercando di cogliere le migliori intuizioni da ciascun progetto. Quanto detto sarà oggetto del capitolo 2.

Nel capitolo 3 si presenterà invece l'architettura di Bird-A, la divisione di ruoli fra backend e frontend, i servizi esposti dal primo e l'interfaccia utente del secondo. Si descriverà infine ad alto livello la Bird-A Ontology.

Il capitolo 4 vedrà una trattazione tecnica più dettagliata delle API e della Bird-A Ontology, si analizzerà lo stack tecnologico utilizzato per servire i due sottosistemi e si analizzerà la struttura dei sorgenti dando un'indicazione del ruolo dei file più importanti.

Una volta in possesso di una conoscenza sufficientemente approfondita di Bird-A, sarà possibile riprendere i requisiti esposti nel capitolo 2 e valutare effettivamente come questo progetto cerchi di soddisfarli. Dato che quanto fino ad ora implementato è, sebbene funzionante, solamente un prototipo, occorrerà elencare gli sviluppi ancora necessari a renderlo un sistema adottabile in un contesto di produzione. Queste valutazioni saranno esposte nel capitolo 5.

Il capitolo 6 si occuperà di ricapitolare quanto detto portando la conclusione di questa dissertazione.

2. Semantic web e RDF data authoring

Per comprendere le esigenze che hanno portato alla nascita di Bird-A e alle considerazioni dietro la sua architettura, conviene soffermarsi brevemente sull'evoluzione in atto nel World Wide Web dall'attuale collezione di contenuti pensati ad una fruizione principalmente umana, alla visione proposta dal suo creatore Tim Berners-Lee [BER09] verso una collezione di dati e informazioni portatori di significato semantico, pensati sia per l'uso umano quanto per l'elaborazione da parte di macchine.

Vedremo i fattori che hanno sancito il successo del Web 2.0 e come questi fattori debbano essere tenuti in considerazione perché avvenga effettivamente una transizione verso il Web 3.0, o Semantic Web. Sempre questi fattori saranno tenuti in considerazione per elaborare i requisiti desiderabili di un sistema di data authoring, vero oggetto di questa dissertazione.

Si concluderà infine con un'analisi dei tentativi fatti in passato in questa direzione e quali siano ad oggi le soluzioni disponibili.

2.1 World Wide Web: La situazione attuale

Il profondo rinnovamento che il web ha subito negli ultimi anni da strumento orientato al consumo di informazioni a strumento per la produzione e la condivisione di informazioni ha le sue radici nell'introduzione di strumenti collaborativi di creazione di contenuti tra

cui, tra le principali, si possono ricordare:

- Piattaforme di blogging, quali Wordpress, Drupal e Blogger, che hanno dato modo al grande pubblico di produrre contenuti facilmente accessibili e commentabili da chiunque.
- Piattaforme wiki, come Wikimedia, che hanno permesso l'elaborazione collettiva di basi di conoscenza, tra le quali la più famosa e importante è sicuramente Wikipedia, la più grande e vasta (in termini di pubblico, lingue, argomenti) enciclopedia mai creata dall'umanità.
- Social network, quali Facebook, Twitter e LinkedIn, che hanno permesso a chiunque non solo di creare contenuti, ma anche e soprattutto di creare collegamenti e relazioni con altre persone e di condividere informazioni e risorse con la propria rete di conoscenze.

Vediamo brevemente i fattori che hanno decretato il successo del web in prima istanza e della sua evoluzione poi. Questa trattazione verrà utile al momento di delineare le caratteristiche desiderabili in un sistema di RDF authoring.

2.1.1 I fattori che hanno decretato il successo del Web 1.0

Il world wide web ha visto una crescita esplosiva nel giro di pochissimo tempo dalla sua definizione e prima implementazione nel 1991 [ILS15]. Tra le cause del suo successo, si possono sicuramente annoverare [CAB10]:

- La tecnologia semplice e "open": alla base del web ci sono HTTP, HTML e URL, protocolli semplici da implementare e non protetti da brevetti.
- L'accessibilità: basta un accesso ad internet e l'URL di una risorsa per poterla reperire e visualizzare.
- La natura distribuita: il web non siede sotto alcuna autorità centrale, ma qualunque persona o entità può decidere di rendere accessibili i propri documenti in perfetta autonomia, senza dover richiedere alcun tipo di approvazione.

- I collegamenti fra le risorse: il web ha esteso il concetto già esistente di ipertesto con la possibilità di far riferimento a una qualunque risorsa identificata da un URL.

2.1.2 I fattori che hanno permesso l'evoluzione verso il Web 2.0

Si può tranquillamente affermare che oggi la gran parte degli utenti del World Wide Web abbia confidenza con la creazione e l'editing di contenuti e che la creazione di informazioni sia diventata un'attività quotidiana nella comunità di internet. [PEW14]

Ma quali sono stati i fattori determinanti di questo cambio di paradigma nel modo con cui è utilizzato il Web? Sicuramente ve ne sono innumerevoli [BOR08], ma tra i principali possiamo sicuramente citare:

- La disponibilità di strumenti di facile utilizzo che permettessero la produzione di contenuti anche ad utenti non esperti.
- La spinta a creare informazioni che fossero accessibili a chiunque, sia per i fini più nobili quali la condivisione di conoscenza su Wikipedia, sia che per fini ludici, quali il mettere a disposizione del mondo intero l'immagine raffigurante l'ultimo pasto consumato.

2.2 World Wide Web: Verso il Semantic Web

Se il moltiplicarsi delle possibilità per gli utenti del web di essere autori oltre che fruitori dei contenuti del web ne ha decretato l'evoluzione dal cosiddetto Web 1.0 al Web 2.0, lo step evolutivo ad oggi in corso verso quello che è stato definito Web 3.0 vede il cambio di focus da contenuti pensati solo per gli utenti umani a contenuti pensati per essere intelleggibili e fruibili anche dalle macchine.

I contenuti, per lo più documenti, presenti sul web sono nell'ordine delle decine di biliardi [SBR14]. Numeri di questo ordine di grandezza rendono ovviamente impossibile gestire

questa conoscenza da parte dei fruitori umani per cui questa è stata concepita. Da qui l'esigenza di renderli processabili e sintetizzabili da macchine. [BER09]

Il problema che sorge a questo punto è che il formato originale, il documento, non è adatto a questo genere di utilizzo. Per permettere l'utilizzo della vastissima mole di informazioni presenti sul web bisogna iniziare a strutturarle non più come documenti, ma come dati dotati di un significato comprensibile alle macchine. [BHL01]

Di seguito analizzeremo il metodo proposto dal World Wide Web Consortium per ottenere questo risultato e quale sia la sua attuale diffusione.

2.3 Rappresentare la conoscenza: Ontologie e RDF

Per permettere al sogno del Semantic Web di realizzarsi è necessario dare alla conoscenza una struttura che sia comprensibile e processabile alle macchine. Questo ha portato allo studio e allo sviluppo di ontologie formali quali strumenti per descrivere le entità rilevanti e le relazioni tra di essi nell'ambito di un dominio preso in esame. Un'ontologia è definita come "la descrizione (come una formale specifica di un programma) dei concetti e delle relazioni che possono formalmente esistere per un agente o una comunità di agenti" [GR93].

I componenti strutturali che compongono un'ontologia sono gli individui (istanze), le classi (i concetti), gli attributi e le relazioni.

Dopo aver affrontato come descrivere a livello teorico delle informazioni, approfondiamo come rappresentarle a livello informatico.

2.3.1 Lo standard RDF

RDF è uno standard W3C proposto al fine di rappresentare la conoscenza. Adottato come recommendation nel 1999, è giunto nel 2014 alla sua versione 1.1 [W3C14a].

L'idea che sta alla base di RDF è semplice ma estremamente potente: tutte le informazioni possono essere rappresentate come asserzioni riguardanti risorse. Queste asserzioni hanno la forma di triple formate da soggetto, predicato, oggetto.

Nello standard RDF ogni risorsa è identificata da un'URI, così come anche i predicati. L'oggetto di un'asserzione può essere sia un letterale, come ad esempio un numero o una stringa, sia una risorsa, creando di fatto dei collegamenti fra risorse.

Questi collegamenti fra risorse sono alla base del concetto di Linked Data [BER06], che descrive come, tramite il concetto di URI per identificare le risorse, il protocollo HTTP per deferenziarle e RDF per descriverne proprietà e relazioni (quindi, più generalmente, informazioni), sia possibile dar vita al Semantic Web.

Degno di menzione in questa dissertazione è come una tripla possa essere oggetto di un'altra tripla, grazie al concetto di reificazione (permettendo quindi di esprimere informazioni che consentano di dedurre il grado di confidenza assegnabile ad una asserzione, come ad esempio chi ne è l'autore o quando è stata creata questa asserzione [(1)]) e che tramite un'estensione di RDF, OWL [W3C04], sia possibile definire e descrivere un'ontologia.

2.4 Attuale natura dei Dataset RDF

Dal suo concepimento, la proposta del Semantic Web ha raccolto molte adesioni. Sono nati progetti volti a rendere pubblicamente disponibili in formato RDF basi di dati preesistenti. Tra i progetti più importanti per quantità di dati processabili esposti ricordiamo:

- DBPedia (3 bilioni di triple [DBP16]): side project di Wikipedia volto a rendere

(1) Come rendere realizzabile la reificazione è tuttavia stato oggetto di dibattito rispetto alla proposta originale dello standard RDF [WN06] [SBH10] fino arrivare alla definizione di una ontologia a se stante, la Provenance Ontology [W3C13]

accessibii le informazioni contenute nei riquadri laterali delle pagine Wikipedia come triple RDF.

- US Census (1 bilione di triple [TAU07]): progetto del governo degli Stati Uniti per rendere disponibili in formato RDF i dati del censimento del 2000.
- Bio2RDF (11 bilioni di triple [DUM15]): progetto volto a fornire la più grande collezione di dataset RDF riguardante le scienze della vita.

È doveroso ricordare in questa sede che l'utilizzabilità di questi dati è fortemente legata alla loro qualità (oltre che al contesto di utilizzo). Per approfondire è possibile consultare l'indagine [ZRM12].

Tuttavia, nonostante la straordinaria importanza di questi dataset per la diffusione e la processabilità della conoscenza, bisogna ricordare che la maggior parte di questi dati sono solo dei "porting" in RDF di database già esistenti, in cui le informazioni continuano ad essere prodotte con le metodologie precedenti e, soprattutto, risultano accessibili in sola lettura. Fanno eccezione alcuni casi notevoli tra cui Wikidata [VK14] (che analizzeremo meglio in seguito), che si prefigge di creare un dataset RDF parallelo ed integrato agli altri progetti della galassia Wikimedia creato tramite le logiche di editing collaborative che hanno portato al successo di Wikipedia.

Altra peculiarità di questi dataset sempre dovuta agli scopi per cui sono nati, è la loro eterogeneità. I collegamenti fra i vari dataset non sono ancora, ad oggi, pervasivi [SBP14] e la loro creazione è appannaggio dei soli enti che hanno prodotto i dati contenuti nei dataset.

2.5 RDF data authoring

Gli attuali dataset RDF sono per lo più trasformazioni in sola lettura applicati a database esistenti. Per garantire la diffusione dei cosiddetti Linked Data e, quindi, del Semantic Web in generale, si sente la necessità di strumenti in grado di rendere agevole, pratica e conveniente la creazione, la visualizzazione e l'editing di dati nativamente RDF.

Vediamo nel dettaglio quali caratteristiche possono risultare auspicabili in un sistema di Data Authoring (d'ora in poi RDF-DA) volto alla creazione e l'editing di dataset RDF.

2.5.1 Caratteristiche desiderabili

Perché possa avere successo, l'evoluzione verso il Semantic Web deve godere degli stessi fattori di forza che hanno favorito l'affermarsi del Web 1.0 prima e del Web 2.0 poi, analizzati nel dettaglio nei capitoli 2.1.1 e 2.1.2.

Seguendo questi esempi, è possibile affermare che, perché possa essere adottato estensivamente, uno strumento di Data Authoring dovrebbe essere:

- General purpose: dati i limiti dell'attuale modello ontologico [JHY10], è importante che lo strumento utilizzato per l'inserimento dei dati non sia legato ad alcuna ontologia specifica e possa evolvere con l'evoluzione della rappresentazione della concettualizzazione sottostante.
- Collaborativo: deve permettere a più persone l'accesso simultaneo ai dati e il contributo di ciascuno deve potersi basare e beneficiare di quanto già fatto dagli altri.
- Distribuito: ciò che ha favorito il diffondersi del world wide web è stata la possibilità per chiunque di creare informazioni e renderle accessibili a chiunque volesse fruirle. I dataset sono innumerevoli, come innumerevoli sono le autorità che li gestiscono e i fini che si prefiggono. Ognuno di questi nodi deve potersi riferire liberamente agli altri e avere la completa autonomia nella creazione dei propri dati.
- Accessibile: i dati contenuti in ciascun nodo devono poter essere accessibili a chiunque (posto che ne abbia le autorizzazioni), ovunque e, eventualmente, utilizzato per creare o estendere informazioni.
- User friendly: perché raggiunga una massa critica di utilizzatori, ciascuna tecnologia deve essere pratica e facilmente adottabile. Più è bassa la curva di apprendimento, più è alta la probabilità che venga adottata.

Analizziamo ora quale sia il panorama attuale degli strumenti di RDF Data Authoring e quale contributo possa essere dato al fine di migliorarlo.

2.6 Panorama attuale dell'RDF Data Authoring

Fin dagli albori del Semantic Web, la sua comunità ha riconosciuto l'intrinseca difficoltà che le sue concettualizzazioni e le sue tecnologie presentano per il grande pubblico (e non solo il grande pubblico composto da non tecnici, ma anche quello composto da "tecnici").

Le pubblicazioni in merito [QHK03] [MEG03] [LH09] e le altre riportate di seguito hanno analizzato i fattori necessari per una larga diffusione del Semantic Web, giungendo a conclusioni simili a quanto precedentemente analizzato per quanto riguarda le caratteristiche desiderabili di uno strumento di data authoring.

Vediamo brevemente i progetti di data authoring che fanno capo a queste pubblicazioni e la loro conformità ai principi esposti sopra.

2.6.1 Haystack

Proposto nel 2003, Haystack [QHR03] è forse il progetto più vicino a quanto realizzato da Bird-A, come ad esempio il meccanismo basato su form configurabili grazie ad un opportuno linguaggio e l'accesso a un dataset condiviso.

A quanto è possibile dedurre dal paper, il progetto mirava a creare un programma similare a Protégé [PRO16] per l'editing di istanze RDF di un'ontologia.

Purtroppo il progetto non sembra mai essere stato sviluppato e sotto questo nome attualmente si trova Haystack Project [HAY16] sempre riguardante il tag semantico e attivamente supportato da varie aziende quali Siemens, ma specializzato per le esigenze dei sensori e dei microcontrollori.

2.6.2 Mangrove

Scopo di questo progetto [MEG03] è fornire uno strumento di tagging di pagine HTML già esistenti con dati RDF. Nonostante sia anche stato sviluppato un prototipo java funzionante, il progetto non sembra essere più in vita.

Sebbene il focus sia sul tagging anziché sulla produzione di individui di una qualunque ontologia, questo lavoro merita una menzione qui per l'analisi svolta del problema e per gli obiettivi che si prefiggeva.

2.6.3 Loomp

Partendo dagli stessi problemi che mira a risolvere Bird-A, è stata creata una proposta per un sistema di tagging RDF questa volta basato su un'applicazione HTML [LH09].

Non sembrano esserci prototipi funzionanti e, ad oggi, il progetto pare abbandonato.

2.6.4 SPARQL Web Pages

SPARQL Web Pages [KNU16] è un progetto volto a creare delle interfacce utente configurabili di visualizzazione di form a partire da dati RDF.

L'approccio ad elementi riusabili (views) è simile a quanto realizzato da Bird-A e alcune intuizioni possono essere prese come spunto per prossime estensioni di Bird-A. Purtroppo lo scopo di sola visualizzazione non soddisfa le esigenze in esame.

2.6.5 TopBraid Platform

La TopBraid Platform [TQ16] è una suite commerciale sviluppata da TopQuadrant, Inc che mira a rendere fruibili i dati RDF a livello enterprise grazie a trasformazioni di dati esistenti, visualizzazioni configurabili (grazie a SPARQL Web Pages) e webservice di interrogazione.

Non pare ad oggi essere presente nello stack un componente di definizione di form di inserimento.

2.6.6 Semantic Wiki

Una realtà importante dell'RDF data authoring è costituita dai cosiddetti Semantic Wiki. Questi progetti mirano a portare il processo di editing collaborativo tipico dei wiki nella creazione di dati RDF (si veda [KHD05] per una trattazione più approfondita).

Di seguito vengono riportati in sintesi i progetti di maggior rilievo in questo ambito:

- Wikidata [VK14]: Un interessante progetto che sta avendo un discreto successo è Wikidata, che si propone di collezionare dati RDF strutturati per arricchire gli altri progetti Wikimedia. Nonostante questo obiettivo, l'approccio di editing a statement proprietà / valore (oltre a qualificatori delle proprietà come ad esempio la data, la fonte, ecc) lo rende uno strumento piuttosto flessibile.
- Semantic MediaWiki [VKV06]: Sempre nella sfera Wikimedia è da ricordare Semantic MediaWiki, l'estensione di MediaWiki (piattaforma di wiki su cui gira, tra gli altri, Wikipedia) che permette annotazioni RDF su pagine wiki esistenti. Da citare, inoltre, i Semantic Form [KG16] che non solo permettono l'inserimento di dati ma anche la loro ricerca.
- AceWiki [BSB09]: AceWiki è un'estensione a Semantic MediaWiki per fornire un'interfaccia di inserimento di informazioni semantiche tramite il linguaggio naturale controllato (CNL: Controlled Natuarl Language) ACE, Attempto Controlled English. Progetto molto interessante (anche se forse di difficile adozione da parte di un largo pubblico) i cui sviluppi si sono interrotti nel 2013.
- Makna [DPT06]: Makna è un progetto simile a Semantic MediaWiki per la piattaforma wiki JSPWiki.
- Rhizome [SOU05]: Rhizome è un wiki pensato per creare risorse RDF, con l'ottica di accedere a dataset RDF federati. Purtroppo non è andato oltre il prototipo e gli sviluppi si sono arrestati nel 2006.

- IkeWiki (rinominato poi KiWi) [SCH06]: progetto simile ai precedenti, ha visto gli sviluppi interrompersi nel 2010.
- OntoWiki [ADR06]: OntoWiki è un progetto attivamente sviluppato e ricco di funzionalità che mira a creare una piattaforma wiki di editing collaborativo tramite strumenti WYSIWYG. Si focalizza su creare dati RDF piuttosto che sul tagging di testi.
- SweetWiki [BGE08]: altro wiki con scopi simili ai precedenti il cui sviluppo è stato abbandonato.
- UFOWiki [PL08]: progetto che mira a realizzare una wikifarm i cui wiki condividano le informazioni semantiche. A quanto è possibile capire il progetto non ha mai superato la fase di prototipo. Non è possibile rintracciare le evoluzioni del progetto.

2.6.7 Altri progetti

Degni di menzione per l'argomento in esame sono infine:

- Metasaur: un tool web-based per il tagging semantico di documenti pensato come strumento didattico [KL03].
- WikiOnt [HGO05] Ontologia per citare e scambiare articoli di Wikipedia.

2.7 Gaffe

Diverso dai tool elencati precedentemente, nei quali i form per l'inserimento di dati RDF erano per lo più sviluppati ad hoc per la specifica esigenza o per le ontologie per cui erano intesi, è il progetto Gaffe, Generator of Automatic Forms - Final Edition [BID09].

Gaffe mira a portare il popolare pattern MVC (Model-View-Controller) nella progettazione ed implementazione di editor di metadati attraverso due ontologie, gestite dall'applicazione (il "Controller"):

- L'ontologia di dominio (il "Model") di cui i metadati creati sono espressione.
- L'ontologia GUI (la "View") che specifica le proprietà grafiche degli elementi del form e dei widget e del loro mapping nell'ontologia di dominio.

Il modello Gaffe ha visto per ora due differenti implementazioni. La prima, chiamata Gaffe for Word Processor, è un plugin per l'editing di metadati all'interno dei documenti per le piattaforme Microsoft Word e OpenOffice Writer [BID09]. Il secondo, chiamato OWiki (da Ontological Wiki) è un plugin per la piattaforma MediaWiki volta ad inserire metadati associati a pagine wiki secondo la sintassi dei wiki template [DMP10a] [DMP11] [DMP10b].

2.8 Considerazioni

Nonostante la ricchezza di strumenti wiki-based per l'editing di istanze RDF, si denota la mancanza di strumenti stand-alone per questo scopo che siano general-purpose, semplici da usare e con un'architettura MVC che separi distintamente la parte di Model (l'ontologia o le ontologie target) dalla parte di View (l'ontologia che descrive i form e i widget e il mapping nel model).

Il progetto Bird-A nasce per colmare questa lacuna fornendo uno strumento di facile utilizzo, general purpose e collaborativo, che si avvale di un'architettura decentrata e scalabile.

Vedremo nel prossimo capitolo quali soluzioni progettuali siano state adottate per soddisfare i requisiti posti in 2.5.1 e su quali tecnologie preesistenti esse siano basate.

3. Il progetto Bird-A

Bird-A nasce per proporre uno strumento di RDF data authoring conveniente per l'amministratore di dataset RDF e di facile utilizzo per l'autore di contenuti RDF.

Nel precedente capitolo abbiamo visto quali siano gli strumenti ad oggi disponibili per svolgere questa mansione, ma di come essi siano per lo più legati ad una specifica esigenza o ad uno specifico sistema (ad esempio wiki).

Introdurremo dapprima l'approccio con cui si è affrontato il problema, con la filosofia di divisione dei ruoli come suo aspetto fondativo in accordo con i principi enunciati da Gaffe [BID09]. Vedremo poi come è stata progettata l'architettura di Bird-A al fine di garantire un approccio federato in cui ciascun soggetto sia libero di decidere come strutturare i dati e a chi consentirne la modifica.

Si procederà poi con l'analisi della struttura del backend e della struttura e dell'interfaccia del frontend, chiudendo infine con la descrizione ad alto livello dell'ontologia e delle API esposte dal backend.

3.1 L'approccio di Bird-A

Per rendere possibile il soddisfacimento dei requisiti del paragrafo 2.5.1, il progetto Bird-A ha puntato innanzitutto alla semplificazione del problema, ricercando un minimo comune denominatore autosufficiente che fosse però facilmente estendibile a seconda delle esigenze.

Per far questo è venuto in aiuto lo standard RDF e la sua filosofia di organizzare le informazioni come un insieme di triple che hanno come soggetto delle risorse. Il significato semantico di queste triple è noto al fruitore (sia esso umano o meno) ed è formalizzato in una o più ontologie, ma non è per forza necessario alla loro creazione.

In RDF, dato un soggetto (qualunque esso sia) è possibile specificarne le proprietà come il tipo (ad es rdf:type foaf:Person) o qualunque altro attributo (quali foaf:givenName e foaf:familyName) senza essere in alcun modo legati all'ontologia che ne descrive il significato semantico (nell'esempio FOAF [BM14]).

Il significato semantico di queste proprietà (che a livello pratico sono semplicemente delle URI) viene stabilito da chi fruisce di questi dati. Nel caso di Bird-A, è sufficiente che sia noto al creatore del form di presentazione e modifica dei dati, non all'applicazione.

Fatto ancora più importante, questo significato semantico deve poter esser ignoto anche all'utente che si occuperà di inserire i dati.

Esempio:

> Per chiarificare meglio questi concetti si pensi ad un form creato per permettere l'inserimento di informazioni relative a cantanti lirici. Il creatore del form predisporrà un campo chiamato "Nome", suggerirà all'utente "Inserisci qui il nome del cantante", imporrà che tale campo non contenga caratteri speciali quali il "%" o parentesi ed infine indicherà che il valore che immetterà l'utente sarà il valore della proprietà foaf:givenName del soggetto di tipo foaf:Person in questo momento sotto esame. Il compito dell'applicazione è creare un form che permetta l'inserimento di questo dato secondo i vincoli imposti, senza curarsi che il dominio di foaf:givenName sia effettivamente foaf:Person, mentre il compito del fruitore del form è riempire le caselle con le informazioni che gli sono richieste e non di sapere cosa sia una foaf:Person o addirittura un'ontologia.

Questo esempio ci aiuta ad introdurre un concetto molto importante di Bird-A: i widget.

3.1.1 I Widget

In Bird-A ogni componente del form ed il form stesso sono rappresentati sia a livello ontologico (come vedremo più avanti) che a livello applicativo come oggetti chiamati widget.

Ogni form si riferisce ad una risorsa RDF che sarà soggetto di una o più triple indicanti le sue proprietà. Ogni widget ha il compito di descrivere una o più di queste proprietà, avendo di fatto il controllo diretto e autonomo sulla creazione delle triple coinvolte.

Questa stretta divisione di responsabilità permette da un lato la semplificazione del lavoro di ciascun widget, dall'altro permette una buona estensibilità. Nello specifico:

- Per aggiungere una proprietà al soggetto basta aggiungere un widget al form.
- Per creare controlli specializzati o presentazioni grafiche particolari basta implementare un nuovo tipo widget dotato delle proprie proprietà indipendenti da quelle di qualunque altro widget.

I widget costituiscono quindi il tramite fra l'ontologia, le triple RDF create a database e l'utente che si occupa di inserire i dati. Questa strategia permette di raggiungere un risultato fondamentale per la riuscita del progetto: la divisione fra i ruoli.

La configurazione dei widget è demandata alla Bird-A Ontology (affrontata con maggiore dettaglio più avanti) a cui spetta il compito di specificarne l'organizzazione visiva, determinare quali siano le triple generate e le validazioni da imporre agli inserimenti da parte dell'utente.

3.1.2 La separazione dei ruoli

Al fine di raggiungere la facilità di utilizzo da parte dell'utente finale e l'autonomia di ciascun amministratore di dataset, occorre fornire la possibilità di dividere i soggetti fruitori in quattro tipologie distinte: chi definisce le ontologie, chi definisce come visualizzare ed inserire dati che fanno riferimento ad un'ontologia, chi amministra il dataset e chi dovrà effettivamente usare o modificare questi dati.

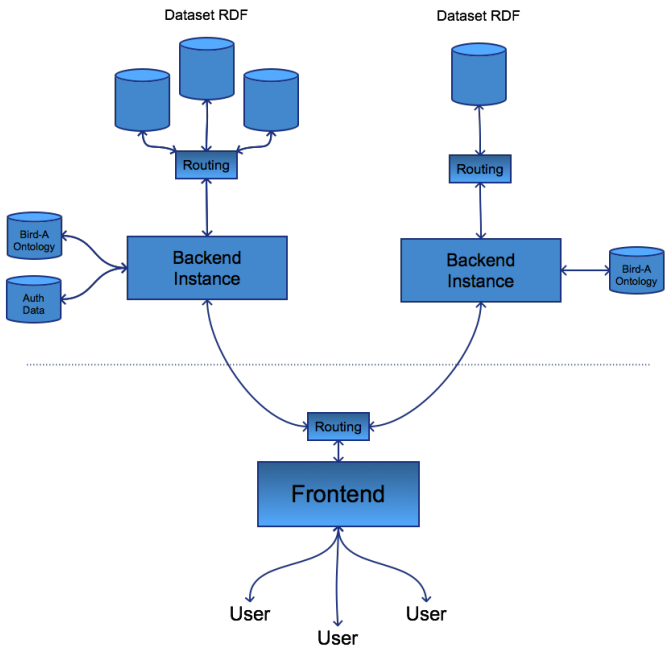
Vediamo nel dettaglio queste figure e il loro ruolo nella creazione di dati semantici RDF:

- Progettisti di ontologie: sono coloro che si occupano di definire le entità, le proprietà e le relazioni, specificando i vincoli di ciascuno di questi e definendo quale sia il loro significato semantico. Possono essere sia soggetti singoli che organizzazioni come il W3C.
- Progettisti di form: sono le figure che decidono come debbano essere create le triple RDF dell'ontologia (o le ontologie) di interesse e di come questi dati debbano essere presentati all'utente. Questo è il ruolo che definisce ad esempio le indicazioni agli utenti per l'inserimento dei dati, le validazioni sugli input e le proprietà effettive che ciascun widget mappa. Lo strumento a disposizione di queste persone è la Bird-A Ontology.
- Amministratore di dataset: è la figura che si occupa di predisporre il dataset RDF per l'accesso in sola lettura o in lettura / scrittura all'applicativo. Questo richiede un dataset RDF esistente o la sua istanziazione, l'installazione del backend e la sua configurazione, l'eventuale scelta di form con cui si vogliono presentare e far inserire i dati e l'eventuale amministrazione delle autorizzazioni (creazione di utenti, dei loro ruoli, dei privilegi di lettura / scrittura associato a ciascun ruolo, ecc.).
- Utente finale: dopo essersi eventualmente autenticato e sfruttando il frontend messo a disposizione da Bird-A, si occupa di inserire i dati in un comunissimo form html ignorando tutti i passaggi precedenti.

È importante notare che queste sono solo le categorie di fruitori. Per nessuno di loro è richiesto il coinvolgimento nello sviluppo di Bird-A. Il progettista di form ad esempio non deve possedere alcuna competenza di HTML, CSS o Javascript ma solo conoscere le funzionalità messe a disposizione da ciascun widget.

Per capire come si sia cercato di raggiungere questa separazione dei ruoli, analizzeremo Bird-A dapprima a livello architetturale, per passare poi al funzionamento di backend e frontend per concludere infine delineando come si articola l'ontologia.

3.2 L'architettura di Bird-A



L'architettura di Bird-a segue la filosofia di separazione forte fra frontend e backend. Sebbene la trattazione di questo schema e dei vari suoi elementi sia demandata al capitolo successivo, conviene fare una rapida introduzione a quali siano i ruoli svolti dai due componenti principali.

3.2.1 Backend

In Bird-A il ruolo del backend è quello di servire uno o più dataset RDF (o, con una definizione più rigorosa, endpoint SPARQL) tramite le opportune API che verranno trattate più avanti.

In questa prima configurazione, l'amministratore di dataset ha la possibilità di servire più dataset RDF tramite regole di routing (non ancora implementate)

A questa collezione di dataset RDF può essere associato un dataset di informazioni

riguardanti le policy di autorizzazione all'accesso e all'edit di questi dati specificati tramite un'opportuna ontologia correlata alla Bird-A Ontology (non ancora implementata).

Nella sua configurazione full-featured infine, può essere specificato un dataset Bird-A Ontology che permetta di presentare form e widget secondo delle opportune API nelle varie serializzazioni RDF e, soprattutto, in un formato JSON pensato appositamente per una fruizione da parte della GUI fornita dal Frontend.

In quest'ultima fase subentra sia il progettista di form, che definisce i widget e il loro mapping nelle ontologie target, sia l'amministratore di dataset, che decide quali form rendere disponibili.

3.2.2 Frontend

Al frontend spetta il compito di avvalersi delle API messe a disposizione dei vari backend tramite un meccanismo di routing (come si diceva prima non ancora implementato) per:

- Reperire i form disponibili dai backend che li mettono a disposizione.
- Per ciascun form presentare gli individui (o risorse) che possono essere presentate / editate tramite quel form.
- Dato un individuo ed un form, renderizzare e permettere la modifica delle informazioni del primo in accordo con quanto sancito dal secondo.

Naturalmente queste operazioni sono subordinate alla disponibilità dei necessari privilegi di accesso o modifica definite dal relativo backend. Il frontend deve consentire l'autenticazione dell'utente verso i backend che lo richiedono tramite opportune API, sempre seguendo il precetto che ciascun backend deve poter avere pieno controllo degli utenti abilitati all'accesso dei dati e delle ACL (Access Control List) relative.

Per non violare l'obiettivo di semplicità si consiglia di utilizzare un servizio di Single Sign-On (SSO) negli sviluppi futuri.

L'architettura proposta è pensata per avere un solo frontend a fronte di multipli backend, ma nulla vieta di averne molteplici. È quindi possibile avere istanze di frontend per intranet aziendali che presentano il logo dell'azienda e che si avvalgono di soli backend selezionati (interni od esterni), quanto è possibile avere il frontend pacchettizzato come app per dispositivi mobili (utilizzando ad esempio Ionic [ION16] e Apache Cordova [AC16]).

Per quanto riguarda la portabilità su dispositivi mobili è da riportare che il design di Bird-A è responsive, anche se rimane da affinare la visualizzazione e l'interfaccia utente per le risoluzioni più piccole.

Vediamo ora in cosa consistono ad alto livello la Bird-A Ontology e le Bird-A API, prima di fare una rapida analisi di come Bird-A raggiunga gli obiettivi analizzati nel paragrafo 2.5.1 (argomento che verrà approfondito nel capitolo 5).

3.3 Interfaccia Utente

Il frontend di Bird-A si presenta attualmente come un comune sito web, con una homepage di presentazione del progetto, la pagina dei contatti, la pagina di login (non implementata) e con la possibilità di ricercare i form. Ecco come si presenta quest'ultima pagina:

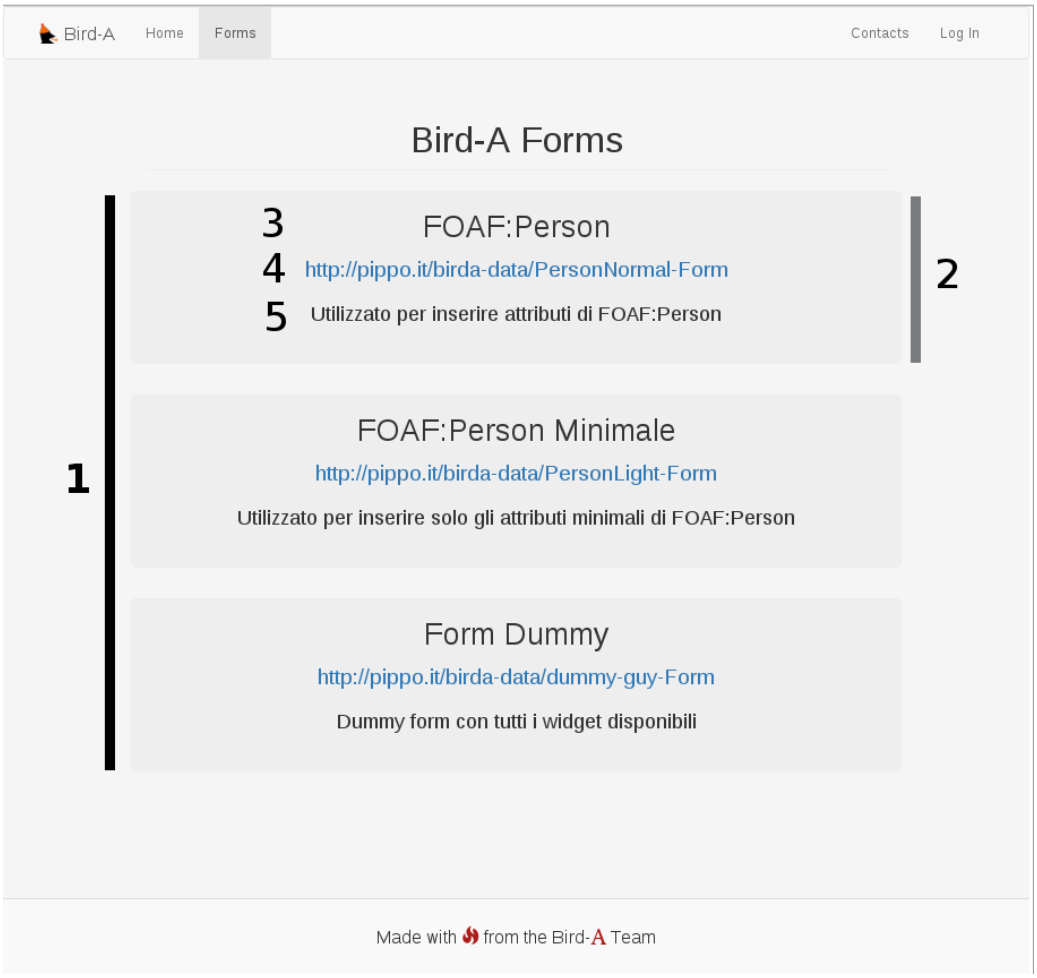


Figure 3.1: Screenshot della lista dei form. Tramite Gimp verranno evidenziati con dei numeri alcune aree

Legenda:

1. Lista dei form.
2. Ciascun riquadro indica uno specifico form e le informazioni ritenute, al momento, di interesse, ovverosia:
3. Label del form.
4. Description del form.
5. URI del form (il form è una risorsa RDF).

Nota: Questa pagina è pensata per consentire una ricerca dei form disponibili tramite opportuni filtri specificabili dall'utente, anche se questa funzionalità non è stata ancora implementata.

Quando l'utente clicca su di un form, viene presentata la pagina di elenco di istanze, riportata sotto:

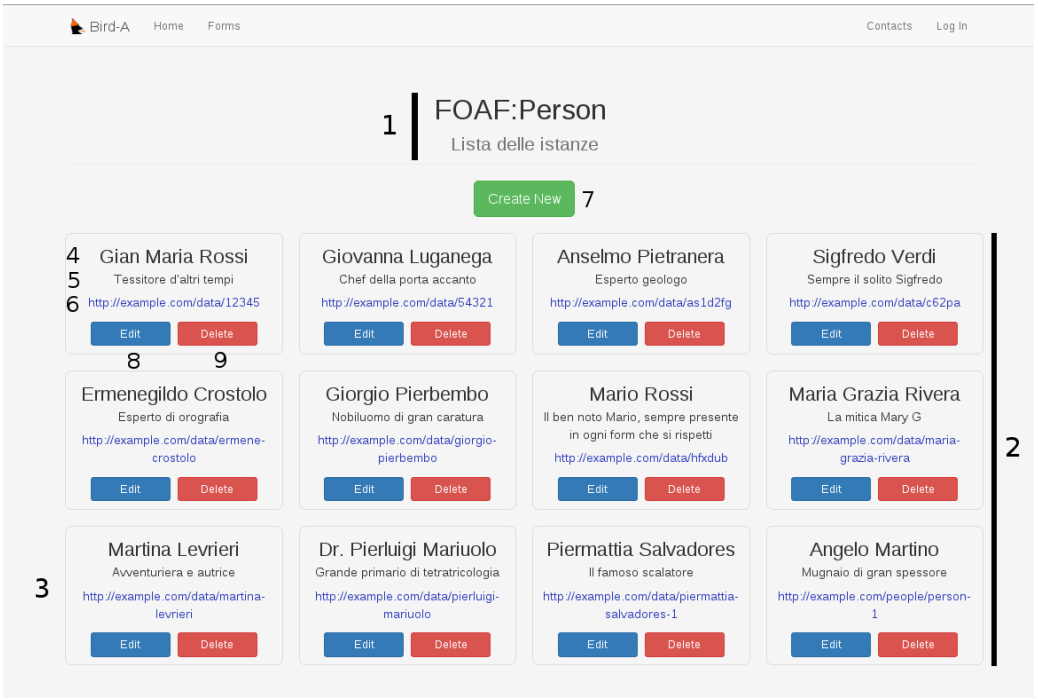


Figure 3.2: Screenshot della lista di istanze.

Legenda:

1. Label e description del form.
2. Lista di istanze.
3. Ciascun riquadro indica un'istanza e le informazioni necessarie ad una rapida identificazione dell'individuo, ovverosia:
4. Label dell'individuo.
5. Description dell'individuo.
6. URI dell'individuo.

7. Pulsante per la creazione di un nuovo individuo tramite il form corrente.

8. Pulsante per l'edit di un individuo esistente tramite il form corrente.

9. Pulsante per la cancellazione dell'individuo esistente e di tutte le proprietà ad esso associate.

Nota: Questa pagina è pensata per consentire una ricerca dei form disponibili tramite opportuni filtri specificabili dall'utente, anche se questa funzionalità non è stata ancora implementata.

Al clic del pulsante "New" o del pulsante "Edit", all'utente verrà presentata la pagina di edit del form, vuota nel primo caso, compilata con le informazioni dell'individuo nel secondo:

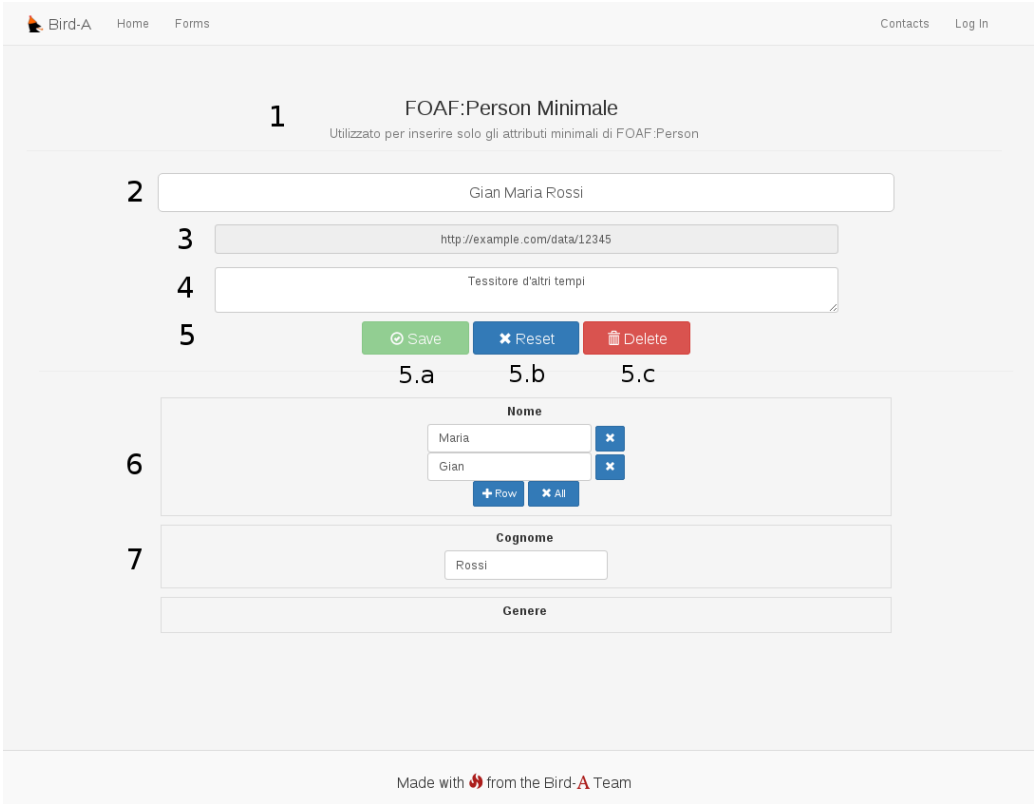


Figure 3.3: Screenshot della pagina di edit.

Legenda:

1. Label e description del form.
2. Label dell'istanza (il cui mancato inserimento genera un warning ma è comunque ammesso).
3. URI dell'istanza, editabile al momento della creazione di una nuova istanza e in sola lettura negli edit successivi.
4. Description dell'istanza.
5. Azioni effettuabili sul form. Nello specifico:
    5.1. Save: causa la validazione del form e, se questa va a buon fine, la creazione di un nuovo individuo o la modifica di uno esistente.
    5.2. Reset: causa il ricaricamento del form con le informazioni presenti a database.
    5.3. Delete: causa la cancellazione dell'individuo a database.
6. Widget di inserimento di una proprietà dell'individuo, in questo caso il nome (inserito tramite un input testuale), con più valori possibili. In questo caso si possono vedere i pulsanti generici "Add Row" per aggiungere un nuovo valore (fino a raggiungere la massima numerosità della relazione) e "Delete All" per ripristinare la situazione di default (con ad esempio tre campi vuoti nel caso la numerosità della relazione sia minimo tre) e il pulsante "Delete" associato a ciascun valore per consentirne l'eliminazione.
7. Widget di inserimento di una proprietà con un solo valore possibile, in questo caso il cognome (solo a titolo d'esempio). In questo caso il campo è statico e i pulsanti indicati sopra non vengono mostrati.

Da notare come, per consentire una facile interazione con l'utente, una grande enfasi è posta su etichette e descrizioni. Esse servono per mostrare e introdurre i form e per mostrare in modo sintetico gli individui nelle ricerche, per questo il loro inserimento nel form viene così fortemente suggerito tramite l'importanza visiva dei campi e tramite dei warning in fase di validazione.

3.4 La Bird-A Ontology

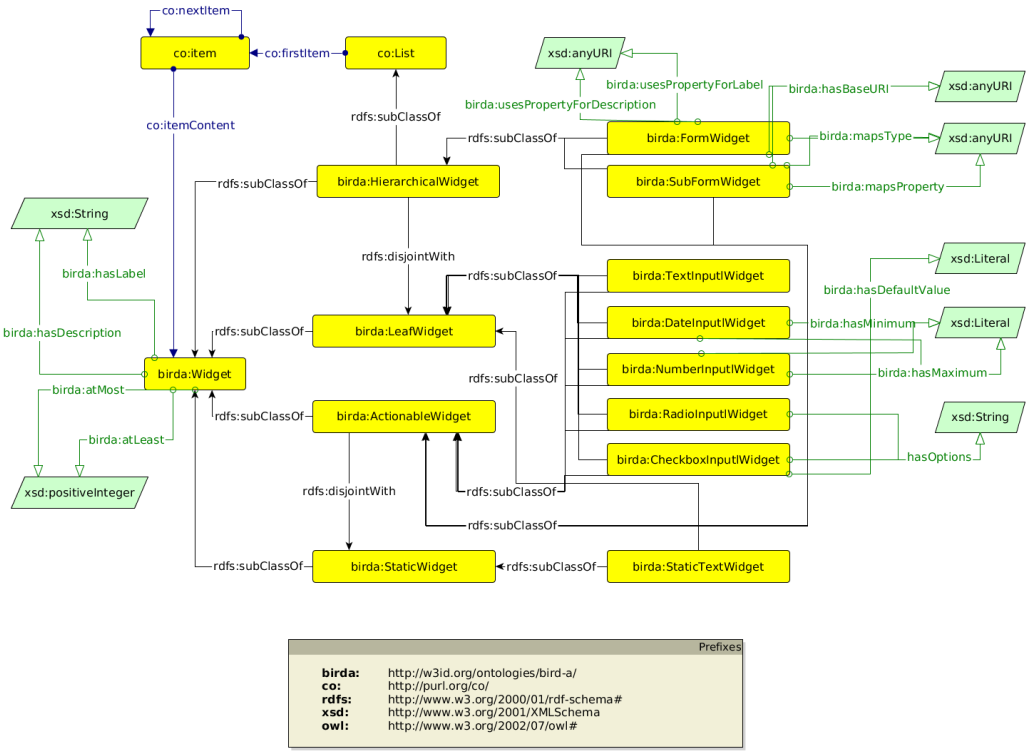


Figure 3.4: Rappresentazione grafica della Bird-A Ontology tramite il framework grafico Graffoo [FGP14]

La Bird-A Ontology serve per definire i form e le componenti del form (widget) a livello sia presentazionale che semantico, indicando come mostrare le informazioni, come validare gli input e cosa suggerire all'utente da una parte, sia quali siano le proprietà e i tipi RDF mappati da ciascun widget dall'altra.

In questa prima versione dell'ontologia l'entità principale è costituita dal Widget, di cui tutte le altre entità sono discendenti. I widget si differenziano in HierarchicalWidget, che possono contenere al proprio interno altri widget, e LeafWidget.

Il concetto chiave di Bird-A è il FormWidget. Il FromWidget è uno HierarchicalWidget che definisce una risorsa RDF e le sue proprietà quali il tipo RDF, la base uri, le regole per creare una uri di una nuova risorsa, ecc.

Il FormWidget fa capo ad un insieme di widget organizzati secondo una struttura ad albero, ciascuno dei quali definisce una proprietà RDF riferita alla risorsa RDF descritta dal form. Le informazioni associate ad una proprietà RDF possono essere di validazione (ad es numero massimo di valori per la data proprietà) o di presentazione (ad esempio visualizzarla come una lista statica o come un input text editabile).

Gli HierarchicalWidget fanno riferimento ai widget in essi contenuti tramite la Collection Ontology List, stabilendo quindi un ordine di presentazione e permettendo allo stesso widget di comparire ad esempio in più form (feature molto importante per evitare di duplicare inutilmente le informazioni associate ad una proprietà).

Altro HierarchicalWidget molto importante è il SubForm, che permette di definire una proprietà che si riferisce ad una risorsa anziché ad un valore. Queste risorse possono essere ricercate tramite il tipo RDF associato al SubForm stesso e grazie alle proprietà definite dai widget suoi diretti discendenti.

3.5 Le Bird-A API

Le API Bird-A si dividono attualmente in due categorie: le API di gestione dei form e quelle di gestione degli individui (risorse) target. A queste due tipologie si aggiungono le API di autenticazione e le API di esposizione dei namespace (necessaria per il routing lato frontend), non ancora tuttavia definite.

Le API riguardanti i form attualmente definite ed implementate sono due:

- Ricerca di form.
- Get di tutto il form con tutte le informazioni necessarie alla GUI per renderizzarlo, validarlo, ecc.

Le API riguardanti gli individui definite e implementate sono:

- Ricerca di individui.
- Creazione di un individuo.

- Get di un individuo.
- Modifica di un individuo.
- Cancellazione di un individuo.

Tutte le API sfruttano la metodologia REST e utilizzano per lo scambio dati il formato JSON. Il backend e le API sono state implementate per consentire più lingue.

3.6 Aderenza agli obiettivi progettuali

Riprendendo quanto scritto nel paragrafo 2.5.1, analizziamo in maniera sintetica come il progetto Bird-A soddisfa gli obiettivi progettuali delineati (per una trattazione più esaustiva si rimanda al capitolo 5):

- General purpose: come Gaffe [BID09], in Bird-A i form sono generati runtime avvalendosi di un'ontologia di definizione basata su widget (pensata per essere ignara del significato semantico dei dati RDF creati). Basta installare Bird-A, creare un'opportuna istanza di Bird-A Ontology e si dispone di un sistema in grado di popolare e presentare un dataset RDF.
- Collaborativo: Bird-A è pensato per essere multi-utente, anche se tuttavia nelle future release dovrà essere migliorata la gestione dell'editing concorrente degli individui.
- Distribuito: L'architettura di Bird-A consente la creazione di una costellazione di backend tra loro autonomi (gestibili da entità diverse) i cui dati possono essere gestiti dal medesimo frontend. Da notare che è possibile il linking fra risorse appartenenti a più backend.
- Accessibile: Accedendo ad un solo frontend, l'utente può ricercare, visualizzare ed eventualmente editare tutti i dataset RDF serviti dai backend registrati.
- User friendly: L'utente finale può disinteressarsi di cosa sia un'ontologia, di come operi lo standard RDF e di dove i dati effettivamente risiedano e siano acceduti. Il

"trait d'union" fra i dati generati e le ontologie target è delineato dal progettista di form e messo in opera dagli amministratori di dataset.

Molte di queste feature non sono ancora implementate, per altre l'implementazione è migliorabile (come vedremo nel capitolo 5), ma l'architettura del progetto è in grado di rispondere alle esigenze di genericità e configurabilità per cui è nato, oltre a soddisfare requisiti desiderabili come la divisione dei ruoli e la scalabilità.

Nel prossimo capitolo si analizzeranno nel dettaglio la Bird-A Ontology, le Bird-A API e i dettagli implementativi di frontend e backend.

4. Bird-A: Dettagli implementativi

In questo capitolo analizzeremo con maggiore dettaglio l'implementazione di Bird-A. In prima istanza si introdurranno le tecnologie utilizzate nel progetto, mentre a seguire si esamineranno nel dettaglio la Bird-A Ontology e le Bird-A API. Per concludere sarà presentata la struttura dei sorgenti e il ruolo svolto da ciascun file significativo, con un rapido accenno alla procedura di build.

Le valutazioni sul progetto, gli aspetti migliorabili e gli sviluppi futuri sono invece demandate al capitolo successivo.

4.1 Lo stack tecnologico

Il backend è stato realizzato in linguaggio Python 2.7 [PYT10] avvalendosi del framework WSGI [PYT03] Pyramid [PYL16], scelto per la sua flessibilità. Attualmente i dataset RDF sono mantenuti su file acceduti tramite la libreria Python rdflib [PYT15a], ma l'implementazione di connettori per altri triplestore è semplice e trasparente per il resto del backend.

Per lail processo di build del backend che si occupa di installare le dipendenze, configurare il pacchetto Python e infine predisporre gli script, è stato utilizzato Python setuptools [PYT15b].

Gli scambi fra backend e frontend avvengono tramite servizi REST [FT02] esposti dal primo. Tali servizi usano JSON [JSO16] come formato di scambio dati.

Il frontend è composto da file interamente statici, il che li rende servibili tramite una CDN (Content Delivery Network, ovvero un servizio di distribuzione di contenuti statici ad alte prestazioni [PBV08]) e pacchettizzabili come app.

Il framework javascript utilizzato è AngularJS [ANG16], che si occupa di gestire:

- routing: determina quale vista deve essere invocata in base all'url richiesta dall'utente.
- templating: analizza il codice HTML in cerca di opportune direttive e le sostituisce con il contenuto dinamico, come ad esempio il contenuto di una tabella.
- validazione: fornisce strumenti facilitati di validazione dell'input dell'utente.
- AJAX: gestisce le richieste HTTP asincrone con il backend.

Gli stylesheet sono realizzati intramite SASS [SAS15], un'estensione al linguaggio CSS che viene compilata in quest'ultimo (il compilatore utilizzato in questo progetto è compass [COM16]), e si avvalgono del framework Bootstrap [GET16] (i cui i sorgenti SASS sono inclusi e compilati nello stylesheet principale, in modo da essere facilmente estesi o modificati).

Il processo di build del frontend è gestito da Grunt [GRU16] e si occupa di:

- Installare le dipendenze di Bird-A tramite Bower [BOW16] (e di inserirle nel file index.html con la giusta versione).
- Di lanciare i test (non implementati).
- Di effettuare la compilazione degli stylesheet, la loro unione in un singolo file (detta "merge") e la loro "miniaturizzazione" (detta "minify"), ovvero il processo di rimozione di tutto ciò che è superfluo al funzionamento, come spazi, ecc.
- Di effettuare il merge e il minify dei Javascript.
- Di effettuare il minify dell'HTML.

Le librerie lato server occorrenti per fare tutto ciò sono gestite tramite NPM (Node Package Manager [NPM16]), il gestore di pacchetti e dipendenze di NodeJS [NOD16].

Infine, per automatizzare installazione, test e run sia del backend che del frontend, è

stato utilizzato GNU Make [GNU14] (di fatto tutti i comandi si lanciano eseguendo "make <cmd>" nella directory principale).

Il progetto utilizza GIT [GIT16a] come sistema di controllo versione ed è ospitato su Github [GIT16b] come progetto opensource (con licenza GPL [GNU07]). Il progetto è raggiungibile all'URL https://github.com/caiogit/bird-a.

4.2 La Bird-A Ontology

Abbiamo già parlato precedentemente della Bird-A Ontology e della sua struttura a widget. In questo capitolo analizzeremo in dettaglio le entità e le proprietà dell'ontologia., reperibile all'URL https://w3id.org/bird-a/

4.2.1 Entità

Tutti i widget discendono da una classe che rappresenta il generico widget chiamata, appunto, Widget.

Data la sua naturale struttura ad albero, in un form saranno presenti widget che contengono altri widget, chiamati HierarchicalWidget, e widget finali di presentazione e inserimento, chiamati LeafWidget. Oltre a questa tipologia si può anche distinguere tra widget che richiedono un'interazione con l'utente, chiamati ActionableWidget e widget invece statici, chiamati StaticWidget.

Le tipologie di widget analizzate fino ad ora sono classi dirette discendenti della classe Widget vista inizialmente. Tutti gli altri widget ereditano da una classe per tipologia. Vediamo nel dettaglio i widget disponibili e da quali classi ereditano:

- FormWidget (eredita da HierachicalWidget e ActionableWidget): Descrive il form, ovverosia la radice da cui discenderanno tutti i widget presenti nella pagina. Un form ha associato un tipo RDF e diverse proprietà specifiche, come accennato nel capitolo 3.4. Un form descrive inoltre le proprietà riferite ad un singolo individuo.

- SubFormWidget (eredita da HierachicalWidget e ActionableWidget): Descrive una object property dell'individuo. Il valore di questa proprietà è una URI indicante una particolare risorsa (di cui il SubForm deve specificare il tipo). Nel form però dovranno anche essere mostrate alcune proprietà caratteristiche della risorsa volte a renderla di immediata comprensione per l'utente. Queste proprietà sono specificate dai widget contenuti all'interno del subform. Ultimo appunto, la valorizzazione di questa risorsa richiede una ricerca delle risorse di quel tipo già create; i filtri di questa ricerca saranno appunto le proprietà indicate nei widget contenuti nel subform.
- TextInputWidget, TextAreaInputWidget, NumberInputWidget, DateInputWidget (ereditano da LeafWidget e ActionableWidget): richiedono all'utente l'inserimento di un valore della tipologia specificata (con opportune facilitazioni, come ad esempio il DatePicker per i campi DateInputWidget).
- CheckboxInput, RadioInput, SelectInput (ereditano da LeafWidget e ActionableWidget): richiedono all'utente di selezionare uno o più valori proposti.
- StaticTextWidget: indica un campo di testo statico presentato all'utente (detto testo potrà essere in futuro espresso tramite opportune ontologie, come ad esempio DoCO [CPP15]).

4.2.2 Proprietà

Di seguito elencate le proprietà della Bird-A Ontology, le entità a cui si riferiscono e la loro funzione:

- atLeast, atMost: dominio Widget, codominio xsd:nonNegativeInteger. Per un dato widget, indica quanti valori possa ammettere la data proprietà.
- hasDefaultValue: dominio ActionableWidget, codominio Literal. Se presente, indica il valore con cui deve essere inizialmente popolato un determinato input.
- hasLabel, hasDescription: dominio Widget, codominio xsd:string. Indica la label e la description del widget da mostrare all'utente.

- hasMinimum, hasMaximum: dominio NumberInput e DateInput, codominio Literal. Se presenti, indicano rispettivamente il valore minimo e il valore massimo assumibili da un dato valore.
- hasOptions: dominio ActionableWidget, codominio xsd:string. Indica, per i widget che lo prevedono, un set di valori possibili tra cui l'utente può scegliere. Attualmente questi valori sono contenuti in una stringa in formato CSV (da migliorare in futuro).
- hasPlaceholder: ActionableWidget, codominio xsd:string. Se presente, indica il placeholder da presentare all'utente in un campo non valorizzato.

Di seguito, le proprietà specifiche di FormWidget e SubFormWidget:

- mapsType: codominio xsd:anyURI. Indica quale sia il tipo (rdf:type) degli individui descritti.
- hasBaseIRI: codominio xsd:anyURI. Indica quale sia l'URI di base con cui creare le nuove istanze degli individui definiti.
- usesPropertyForLabel, usesPropertyForDescription: Nel generare gli individui, indica quali siano la proprietà da utilizzare per specificare label e description.

Come ultimo appunto, trattato più approfonditamente nel capitolo delle valutazioni, sono definite delle proprietà per la generazione assistita delle URI dei nuovi individui, ma richiedono dei miglioramenti. Tale funzionalità non è ancora implementata in Bird-A.

4.2.3 Esempio di Bird-A Ontology

Di seguito si riporta un esempio di Bird-A Ontology che descrive un form con queste caratteristiche:

- L'individuo generato è di tipo foaf:Person
- Richiede in input il nome, che andrà a valorizzare la proprietà foaf:givenName

- Richiede in input il cognome, che andrà a valorizzare la proprietà foaf:familyName
- Accetta in input il sesso della persona, che andrà a valorizzare la proprietà foaf:gender
- Permette l'inserimento di uno o più persone conosciute, sempre di tipo foaf:Person, che andranno a popolare la proprietà foaf:knows. Per ognuna di queste persone, visualizza (e permette di ricercare per) nome e cognome, definiti come sopra.

```
@prefix binst: <http://pippo.it/birda-data/> .
@prefix birda: <http://w3id.org/ontologies/bird-a/> .
@prefix co: <http://purl.org/co/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

binst:GivenName-TextInput a birda:TextInputWidget ;
    birda:atLeast 1 ;
    birda:hasLabel "Name"@en,
        "Nome"@it ;
    birda:mapsProperty foaf:givenName .

binst:FamilyName-TextInput a birda:TextInputWidget ;
    birda:atLeast 1 ;
    birda:hasLabel "Family Name"@en,
        "Cognome"@it ;
    birda:mapsProperty foaf:familyName .

binst:Gender-RadioInput a birda:RadioInputWidget ;
    birda:atMost 1 ;
    birda:hasLabel "Gender"@en,
        "Sesso"@it ;
    birda:hasOptions "\"Male\", \"Female\", \"Unknown\""@en,
        "\"Maschio\", \"Femmina\", \"Sconosciuto\""@it ;
    birda:mapsProperty foaf:gender .

---

binst:PersonKnowsSubForm-SubForm a co:List ,
        birda:SubFormWidget ;
    co:firstItem binst:PersonKnowsSubForm-SubForm-el1 ;
    co:item binst:PersonKnowsSubForm-SubForm-el1 ,
        binst:PersonKnowsSubForm-SubForm-el2 ;
```

```
    birda:hasLabel "Connections"@en,
        "Conoscenze"@it ;
    birda:mapsProperty foaf:knows ;
    birda:mapsType foaf:Person .

binst:PersonKnowsSubForm-SubForm-el1 co:itemContent binst:GivenName-TextInput ;
    co:nextItem binst:PersonKnowsSubForm-SubForm-el2 .

binst:PersonKnowsSubForm-SubForm-el2 co:itemContent binst:FamilyName-TextInput .

---

binst:PersonNormal-Form a co:List,
        birda:FormWidget ;
    co:firstItem binst:PersonNormal-Form-el1 ;
    co:item binst:PersonNormal-Form-el1,
        binst:PersonNormal-Form-el2,
        binst:PersonNormal-Form-el3,
        binst:PersonNormal-Form-el4 ;
    birda:hasBaseIRI "http://pippo.it/target-data/"^^xsd:anyURI ;
    birda:hasDescription "Used to insert FOAF:Person attributes"@en,
        "Utilizzato per inserire attributi di FOAF:Person"@it ;
    birda:hasLabel "FOAF:Person"@en,
        "FOAF:Person"@it ;
    birda:mapsType foaf:Person ;
    birda:usesPropertyForDescription rdfs:comment ;
    birda:usesPropertyForLabel skos:prefLabel .

binst:PersonNormal-Form-el1 co:itemContent binst:GivenName-TextInput ;
    co:nextItem binst:PersonNormal-Form-el2 .

binst:PersonNormal-Form-el2 co:itemContent binst:FamilyName-TextInput ;
    co:nextItem binst:PersonNormal-Form-el3 .

binst:PersonNormal-Form-el3 co:itemContent binst:Gender-TextInput ;
    co:nextItem binst:PersonNormal-Form-el4 .

binst:PersonNormal-Form-el4 co:itemContent binst:PersonKnowsSubForm-SubForm .
```

4.3 Le Bird-A API

Vediamo ora le API messe a disposizione dal backend. Si analizzeranno prima i formati JSON di scambio e, successivamente, i servizi che si avvalgono di tali JSON.

4.3.1 Formati JSON

4.3.1.1 Forms Simple

Restituisce una lista di form le informazioni basilari di ciascuno.

```
{
  "forms": [
    {
      "uri": "http://www.birda.it/form-person-1",
         ⇒ URI del FormWidget
      "type": "http://xmlns.com/foaf/0.1/Person",
         ⇒ rdf:type gestito dal Form
      "label": "...",
      "description": "..."
    },
    ...
    ...
  ]
}
```

4.3.1.2 Form Full

Restituisce tutte le informazioni (definite con la Bird-A Ontology) riguardanti uno specifico form.

```
{
  "form_uri": "http://birda.com/form-person-1",
     ⇒ URI del FormWidget
```

```
"maps_type": "http://xmlns.com/foaf/0.1/",
   ⇒ rdf:type gestito dal Form
"base_uri": "http://ex.com/",
   ⇒ URI base che dovranno avere i nuovi individui
"label": "...",
"description": "...",
"label_property": "http://www.w3.org/2004/02/skos/core#prefLabel",
   ⇒ URI della proprietà che indica la label nell'individuo
"descr_property": "http://www.w3.org/2000/01/rdf-schema#comment",
   ⇒ Come sopra, però per la description
"lang": "it",
   ⇒ Lingua delle descrizioni del JSON
"fields": [
  {
    "widget_uri": "http://birda.com/person-givenName-1",
    "w_type": "text-input",
       ⇒ Tipo del widget
    "property": "http://xmlns.com/foaf/0.1/givenName",
       ⇒ Proprietà mappata dal widget
    "label": "Nome",
    "description": "Usare un campo diverso per ogni nome",
    "default": "",
    "placeholder": "Nome della persona (ad es. \"Pino\")",
    "at_least": 1,
       ⇒ Minimo numero di valori per questa proprietà
    "at_most": 10,
       ⇒ Massimo numero di valori per questa proprietà
    "validation": {
       ⇒ Validazioni da effettuare sull'input dell'utente
      "max_length":25,
      "required": true,
      ...
    },
    "choices": [
       ⇒ Valori tra cui scegliere (solo per radio e checkbox)
      {
        "label": "Scelta 1",
        "description": "",
        "type": "xsd:string",
        "value": "http://w3id.com/gender-ontology/male",
          ⇒ Valore effettivamente salvato a DB
        "default": true
          ⇒ Indica il valore selezionato di default
      },
  },
  {
    "widget_uri": "http://birda.com/person-knows-1",
```

```
      "w_type": "subform",
      "maps_property": "http://xmlns.com/foaf/0.1/knows",
         ⇒ URI di questa object property
      "maps_type": "http://xmlns.com/foaf/0.1/",
         ⇒ rdf:type degli individui riferiti in questa proprietà
      "label": "...",
      "description": "...",
      "at_least": 0,
      "at_most":50,
      "fields": [
         ⇒ Ricorsivamente come sopra (vedi campo "fields")
         ...
         ...
      ]
    }
  ],
  "local_name":{
     ⇒ Oggetto contenente le indicazioni per creare l'URI di un nuovo individuo
    "fields": [
       ⇒ Widget di cui concatenare i valori per ottenere il local_name
      "http://birda.com/person-givenName-1",
      "http://birda.com/person-familyName-1"
    ],
    "tokenSeparator": "(\.|\s|-)+",
       ⇒ Indica l'espressione regolare per suddividere i vari valori in token
    "separator": "_",
       ⇒ Separatore fra i token
    "renderer": "[lowercase|uppercase|camelcase]"
       ⇒ Come renderizzare i token prima di concatenarli
  }
}
```

4.3.1.3 Individuals Query

Permette la ricerca di uno o più individui. Questa ricerca è pensata per essere il più general purpose possibile.

```
{
  "properties": [
     ⇒ Lista di proprietà da restituire in output
    {
```

```
      "uri": "http://xmlns.com/foaf/0.1/givenName"
    },
    {
      "uri": "http://xmlns.com/foaf/0.1/familyName"
    },
    ...
  ],
  "filters": [
     ⇒ Lista di filtri di ricerca da applicare
    {
      "property": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
        ⇒ Proprietà su cui effettuare il filtro
      "value": "http://xmlns.com/foaf/0.1/Person",
        ⇒ Valore per cui filtrare
      "match": "exact"
        ⇒ Tipo di match (per ora "exact" e "starts_with")
    },
    {
      "property": "http://xmlns.com/foaf/0.1/familyName",
      "value": "http://xmlns.com/foaf/0.1/Person",
      "match": "starts_with"
    },
    ...
  ],
  "order_by":[
     ⇒ Lista di criteri di ordinamento da applicare ai risultati
    {
      "property": "http://xmlns.com/foaf/0.1/familyName",
        ⇒ Proprietà per cui ordinare
      "order": "desc"
        ⇒ Ordinamento ("desc" o "asc")
    },
    ...
  ]
  "limit": 20,
    ⇒ Limita il numero di risultati
  "offset": 0
     ⇒ Indica da che offset partire (utile per il pagination)
}
```

4.3.1.4 Individuals Infos

Contiene le informazioni di uno o più individui.

```
{
  "individuals": [
     ⇒ Lista di individui che rispondono ai filtri
    {
      "uri": "http://ex.com/john-max-smith",
        ⇒ URI dell'individuo
      "type": "http://xmlns.com/foaf/0.1/Person",
        ⇒ rdf:type dell'individuo
      "lang": "en",
        ⇒ Lingua dei valori stringa di questo dizionario
      "label": "John Max Smith",
        ⇒ Valore della label per questo individuo / lingua
      "description": "Famous actor",
        ⇒ Valore della description per questo individuo / lingua
      "last_modified": "2015-11-25 14:33:01",
        ⇒ Data di ultima modifica dell'individuo
      "authors": [
         ⇒ Autori che hanno collaborato all'edit dell'individuo
        {
          "uri": "http://bigio-bagio.it#me",
          "label": "Bigio Bagio"
        },
        ...
      ],
      "properties": [
         ⇒ Proprietà dell'individuo con i rispettivi valori
        {
          "uri": "http://xmlns.com/foaf/0.1/givenName",
          "values": ["John", "Max"]
        },
        {
          "uri": "http://xmlns.com/foaf/0.1/familyName",
          "values": ["Smith"]
        },
        ...
      ]
    },
    ...
  ]
}
```

4.3.2 Servizi

Vediamo nel dettaglio i servizi messi a disposizione del backend. Il significato dei parametri della querystring è, laddove presenti:

- offset: indica l'offset da cui iniziare a restituire un certo set di valori.
- limit: indica a quanti valori limitare il risultato.
- lang: indica la lingua del risultato atteso nella forma codice ISO a due caratteri.
- form: indica il form o il subform di riferimento.

Tutte le API iniziano per "/api/v1". Per ogni servizio è fornito un alias senza versione, cioè con solo "/api", che rimanda sempre all'ultima versione disponibile.

4.3.2.1 GET /api/v1/forms

`Example: /api/v1/forms?offset=<n>&limit=<m>&lang=<l>`

Reperisce la lista dei form. Da estendere in futuro per consentire la ricerca secondo opportuni filtri e l'ordinamento.

Parametri querystring: offset, limit, lang.

4.3.2.2 GET /api/v1/forms/<form_uri>

`Example: /api/v1/forms/http://www.birda.it/form-person-1?lang=<l>`

Fornisce tutte le informazioni a disposizione per un dato form.

Parametri querystring: lang.

4.3.2.3 POST /api/v1/individuals/search

`Example: /api/v1/individuals?form=<form_uri>&lang=en&offset=10&limit=10`

Ricerca gli individui che soddisfano i parametri in ingresso e ne restituisce le proprietà richieste (più altre restituite sempre di default). In questa prima implementazione richiede anche la specifica di un form o un subform.

Accetta in POST il JSON "Individuals Query" e restituisce in output il JSON "Individuals Infos".

Parametri querystring: offset, limit, lang, form.

4.3.2.4 PUT, GET, POST, DELETE /api/v1/individuals/<individual_uri>

`Example: /api/v1/individuals/http://www.birda.it/john-smith?form`
`=http://www.birda.it/form-person-1&lang=en`

Effettua le operazioni CRUD (Create Read Update e Delete) su uno specifico individuo. Nello specifico:

- PUT: crea un nuovo individuo.
- GET: reperisce un individuo esistente.
- POST: Modifica i dati di un individuo esistente.
- DELETE: Cancella tutte le proprietà di un individuo esistente.

PUT e POST si aspettano il JSON "Individuals infos" come payload, mentre GET, PUT e POST ritornano sempre il JSON "Individuals infos" (nel caso di PUT e POST potrebbero essere state variate delle informazioni come il modification time, in questo modo si evita di fare una GET per aggiornare queste informazioni sul frontend).

Parametri querystring: form, lang.

4.4 Viste Frontend

Di seguito si vedranno brevemente le viste messe a disposizione dal frontend, tra cui, le più importanti, sono state oggetto di analisi del paragrafo 3.3.

4.4.1 Home e altre pagine

Le pagine semplici offerte per ora dal frontend sono:

- /: homepage contenente una rapida presentazione del progetto.
- /contact: pagina in cui sono presenti i contatti e il link al progetto GitHub di Bird-A https://github.com/caiogit/bird-a.
- /404: pagina di defualt mostrata se la view risulta sconosciuta.

4.4.2 Lista dei form

`Route: /forms-list`

(v. fig. 3.x1) Presenta la lista dei form e permette, per ciascuno di questi, di accedere alla lista delle istanze.

4.4.3 Lista delle istanze

`Route: /individuals-list?form=<form_uri>`

(v. fig. 3.x2) Dato un form passato come URI nella querystring, ne mostra le istanze. Permette la creazione di nuove istanze oltre alla modifica o la rimozione di quelle presenti.

4.4.4 Edit di un'istanza

`Route: /individuals-list?form=<form_uri>[&individual=<individual_uri>]`

(v. fig. 3.x3) Se viene fornito il parametro "individual", allora utilizza il form per renderizzare le informazioni dell'individuo e permetterne l'editing. In caso non sia presente, renderizza un form vuoto e permette la creazione di un nuovo individuo.

Si veda il capitolo 5 sulle considerazioni riguardanti la "canonicizzazione" di questo tipo di route (utile per i motori di ricerca).

4.5 Struttura directory di Bird-A

Vista l'estensione del progetto, conviene analizzarne i file e le directory dividendole in tre sezioni. La trattazione è svolta sul contenuto del progetto a seguito di un "git clone" del repository presente sulla pagina Github https://github.com/caiogit/bird-a.

Si analizzeranno solo i file significativi per il funzionamento di Bird-A. Come convenzione le directory saranno rappresentate in grassetto.

4.5.1 Struttura directory generale

Per prima cosa analizziamo il contenuto della directory radice del progetto:

- **backend**: si veda la sezione 4.5.2.
- **frontend**: si veda la sezione 4.5.3.
- **config**: directory contenente tutti i file (o i link ai file) che devono essere modificati per configurare Bird-A.
- **db**: contiene i database, attualmente gestiti su file, necessari al funzionamento del backend.

  - birda.turtle: dataset RDF contenente l'istanza di Bird-A Ontology servita dal backend.
  - indiv.turtle: dataset RDF contenente le triple delle ontologie target generate da Bird-A.
- **doc**: contiene schemi e documentazione varia.
  - **ontology**: contiene la Bird-A ontology in formato XML.
    - **w3id.org**: contiene i file inviati via push al progetto W3ID [W3I16].
- **log**: contiene i log dei due webserver, backend e frontend.
- Makefile: file di configurazione per GNU Make che accentra le operazioni di build fatte su frontend e backend (analizzate nel paragrafo 4.6).

4.5.2 Struttura e funzionamento Backend

Nella directory "backend" sono contenuti tutti i sorgenti e i file di configurazione necessari al funzionamento del backend. Ogni directory ha il proprio file __init__.py (eventualmente vuoto) al fine di farla riconoscere all'interprete python come pacchetto, ma di seguito verranno trattati solo i file __init__.py contenenti una qualche logica. Nel dettaglio:

- development.ini / production.ini: configurazioni Pyramid e Bird-A valide su tutto il backend.
- setup.py: configuratore del pacchetto backend.
- **birda**: è la directory dei sorgenti utilizzati dal webserver WSGI.
  - __init__.py: contiene l'inizializzazione dell'applicazione WSGI e l'istanziazione dei servizi singleton come FormsFacotry e IndividualsFactory. Effettua il load del file di configurazione e le inizializzazioni ad esso legate.
  - views.py: contiene le definizioni delle view html.

- **bController**: contiene la logica di "controllo" degli oggetti form e individual.
  - forms_factory.py: contiene l'oggetto FormsFactory responsabile della creazione degli oggetti di tipo Widget rappresentanti Form e SubForm. Questo oggetto si occupa della caching degli oggetti creati.
  - individuals_factory.py: contiene l'oggetto IndividualsFactory responsabile della creazione degli oggetti di tipo Individual e della ricerca di individui. Questo oggetto si occupa della caching degli oggetti creati (non ancora implementata).
- **bModel**: contiene la logica di "modellazione" degli oggetti "Widget" e "Individual" e della loro rappresentazione a DB.
  - __init__.py: contiene la definizione dei namespace da utilizzare in tutto il backend, compresi il namespace della Bird-A Ontology e quello dell'istanza di Bird-A ontology (nel futuro dovrà essere reperito dalla configurazione).
  - individual.py: definisce la classe Individual che si occupa di reperire l'individuo dal DB se presente, esportarlo in JSON, fare il load di un JSON esistente e inserirlo a DB o andare in update su quanto già esistente.
  - ontology.py: contiene le funzioni di utilità per la creazione di istanze di Bird-A Ontology.
  - widget.py: contiene la definizione della classe Widget. Questa classe è responsabile del reperimento delle informazioni comuni a tutti i widget e, per i widget gerarchici, della discesa ricorsiva nei widget figli (al fine di creare una struttura ad albero). Le proprietà specifiche di ciascun widget devono essere implementate come specializzazioni della classe Widget tramite ereditarietà. A tali oggetti è demandata la responsabilità di creare ricorsivamente il JSON "Form Full".
  - **widget_catalog**: contiene gli oggetti che ereditano dalla classe Widget.
    - checkbox_input.py

    - form.py
    - subform.py
    - text_input.py

- **models**: Contiene gli oggetti ORM SQLAlchemy di accesso al database SQL (in questo caso sqlite dove sono mantenuti gli utenti).
    * __init__.py: Contiene alcune inizializzazioni.
    * users.py: Contiene il model che rappresenta gli Users (con logica di verifica della password).
- **scripts**: Contiene gli script di utilità.
    * create_test_ontologies.py: Crea l'istanza di Bird-A di test (da mettere poi manualmente nella directory /db).
    * initialize_db.py: Inizializza il database SQL.
    * test_service.sh: Esegue l'interrogazione "detached" (senza bisogno del server in esecuzione) di un service e va in errore se lo status http restituito è diverso da 200. Utile nel Makefile per interrompere l'esecuzione in caso di errore in un test.
- **services**:
    * __init__.py: Contiene funzioni di utilità condivise fra i vari servizi.
    * forms.py: Contiene i servizi */forms*.
    * individuals.py: Contiene i servizi */individuals*.
    * test.py: Contiene dei servizi utili per testare il comportamento di Cornice (la libreria accessoria di Pyramid che si occupa della gestione delle API REST) o del routing di Pyramid.
    * **jsons**: Contiene le definizioni degli oggetti della libreria Colander utilizzati per validare i JSON.

        - __init__.py: Contiene funzioni di utilità condivise fra i vari moduli di definizione dei JSON.
        - forms.py: Contiene gli oggetti JSON "Form Full" e "Forms Simple".
        - individuals.py: Contiene gli oggetti JSON "Individuals Query" e "Individuals Infos".

- **storage**: contiene la logica di accesso al dataset RDF e le funzioni di utilità relative.
    * __init__.py: Definisce le classi di base condivise (come ad esempio Results) e le interfacce da implementare. Tra queste la più importante è Connection . Per implementare un DB Connector basta infatti implementare un oggetto che rispetti questa interfaccia senza dover modificare le funzionalità soprastanti.
    * file_storage.py: Implementa il DB Connector verso dataset RDF contenuti su file (di cui riconosce il formato di serializzazione grazie all'estensione).
    * utils.py: Contiene molte funzioni di utilità che servono ad astrarre l'oggetto rdf.Graph (da spostare, in futuro, nella directory utils).
- **templates**: contiene i template per generare le view html.
    * home.jinja2: Genera una semplice home informativa sul backend.
    * proxy.jinja2: Genera il file proxy necessario per il funzionamento di xDomain (libreria che migliora la gestione di chiamate a servizi cross-domain).
- **utils**: Raccoglie vari moduli di utilità.
    * ascii_utils.py: utilità di output testuale.
    * lock.py: utilità per la gestione dei lock.
    * generic.py: utilità che non ricadono nelle precedenti categorie.

Nel codice del backend può convenire, in futuro, operare un po' di refactoring per:

- Riorganizzare le funzioni di utilità.
- Rinominare "storage" in qualcosa di più significativo.
- Rinominare "bController" in qualcosa di più significativo.

Come ultimo appunto, per trattare al meglio le stringhe unicode si è deciso di utilizzare la gestione messa a disposizione dal ramo 3 di Python. Ogni sorgente riporta in testata alcune istruzioni di inizializzazione. Laddove si siano resi indispensabili degli interventi volti a far funzionare le librerie attualmente utilizzate (implementate per Python 2.7), viene sempre riportato "Patch for unicode strings".

4.5.3 Struttura e funzionamento Frontend

Nella directory "frontend" sono contenuti tutti i sorgenti, i file statici e i file di configurazione necessari al funzionamento del "frontend". Per capire nel concreto le spiegazioni sottostanti è necessaria una conoscenza di base di AngularJS.

È importante sottolineare come il frontend sia interamente statico, cioè i file serviti non subiscono variazioni serverside. La directory effettivamente servita dal webserver è "app". Nel dettaglio:

- Gruntfile.js: contiene l'inizializzazione degli script Grunt che presiedono al processo di buid del frontend.
- package.json: contiene la definizione delle dipendenze serverside necessarie al processo di build. L'installazione di queste dipendenze è gestita dal tool NPM.
- bower.json: contiene le dipendenze effettivamente necessarie clientside, quali angularjs, ecc. L'installazione di queste dipendenze è gestita dal tool Bower.
- app: Questa è la directory che viene effettivamente servita dal webserver e contiene tutti i file utili al funzionamento del frontend.
  - config.js: Contiene le configurazioni, le inizializzazioni e alcune semplici funzioni che si avvalgono di tali configurazioni (come ad esempio il concatena-

mento di indirizzo, porta e APIs root directory del backend per ottenere la "baseuri" dei servizi).

- index.html: è il file principale del frontend che ne definisce header, nav e footer. La sezione main contiene la direttiva ng-view in cui verranno inseriti dinamicamente i contenuti in base alla route e ai dati reperiti dal backend. Qui è contenuta l'importazione delle librerie e le direttive che serviranno al processo di build per sostituirle (con, ad esempio, la versione minimizzata e concatenata o il reperimento via CDN).
- **images**: Contiene le immagini utilizzate dal frontend.
- **locale**: Contiene i file di traduzione delle stringhe statiche del frontend (feature non ancora implementata).
- **styles**: contiene gli stylesheet utilizzati nel progetto (formato .css o .scss).
  - birda.scss: tutti gli stili di Bird-A sono per ora racchiusi qui. Al suo interno importa i sorgenti scss di Bootstrap, con la possibilità di cambiare gli stili di default di quest'ultimo prima della compilazione in un unico css.
- **views**: contiene le view html che verranno inserite nella direttiva ng-view a seconda della route.
  - home.html: View che presenta l'homepage.
  - 404.html: View che notifica che la route selezionata non esiste.
  - contact.html: View che presenta la pagina dei contatti.
  - forms-list.html: View che presenta la pagina che mostra i form a disposizione.
  - individuals-list.html: View che presenta tutti gli individui appartenenti ad un dato form scelto in precedenza.
  - edit.html: View che renderizza un form e presenta le informazioni di un dato individuo.

    * **directives**: Contiene i template html utilizzati dalle direttive (un esempio di direttive sono i vari tipi di widget).
        · date-input.html, text-input.html, ecc.

- **templates**: contiene i template html non utilizzati come view.
    * modal_xhr_error.html: contiene l'html del modal utilizzato per presentare l'errore di una http request.

- **scripts**: Contiene tutti gli script javascript (ad eccezione di config.js).
    * app.js: contiene l'inizializzazione di AngularJS e la definizione delle routes.
    * utils.js: contiene varie funzioni di utilità utilizzate all'interno dell'app.
    * dummy_jsons.js: contiene JSON di test delle Bird-A API utili per costruire risposte fittizie dal backend (ad esempio per provare feature non ancora implementate lato backend).
    * **controllers**: contiene i controller AngularJS che contengono la logica di "business" (interazione con l'utente, cambi di stato in seguito a determinati eventi, ecc) di determinate aree della pagina, come ad esempio le view o le direttive.
        · MainController.js: È il controller generale dell'applicazione che presiede, ad esempio, al menù.
        · FormsListController.js: Controller che presiede alla pagina di lista dei form.
        · IndividualsListController.js: Controller che presiede alla pagina della lista di individui.
        · EditController.js: Controller che presiede alla pagina di edit di un'istanza.
    * **directives**: contiene le inizializzazioni delle direttive AngularJS. Ogni direttiva definisce il proprio controller. Le direttive che rappresentano dei widget, ereditano da un controller generico contenuto in utils.js che si

occupa di gestire l'aggiunta e la rimozione di valori e il reset ai valori di default.

  - dateInput.js, textInput.js, subForm.js, ecc.

* **filters**: contiene i filtri AngularJS definiti da Bird-A.

  - filters.js: contiene per ora tutti i filtri Bird-A.

* **services**: contiene i service AngularJS preposti al reperimento e stoccaggio dei dati dell'applicazione utilizzati poi dai controller.

  - ConfigService.js: Servizio che si occupa del reperimento della configurazione (config.js) e della sua esposizione tramite la dependency injection di AngularJS.

  - UIService.js: Servizio che mette a disposizione funzionalità di interazione con l'utente come ad esempio i modal d'errore.

  - FormService.js: Servizio che si occupa di reperire tutte le informazioni di un determinato form.

  - FormsService.js: Servizio che si occupa di reperire la lista di form a disposizione e le loro informazioni basilari.

  - IndividualsSearchService.js: Servizio che permette la ricerca di individui.

* **factories**: Contiene le factory AngularJS che servono per creare dei service AngularJS.

  - IndividualsFactory.js: Factory che produce servizi di tipo IndividualService (in essa definito) che si occupano di reperire tutti i dati legati ad un individuo e di presentarli all'applicazione. L'esigenza di una factory è giustificata dal fatto che le informazioni di più individui possono essere attive contemporaneamente sulla stessa pagina. Gli individualService si occupano anche delle azioni di PUT, POST e DELETE sugli individui.

4.6 Build e Run

Bisogna in realtà ricordare che Bird-A si articola in due progetti praticamente indipendenti, frontend e backend, i quali hanno processi di build e run separati.

Per permettere uno sviluppo più agevole, si è però utilizzato lo strumento GNU Make per lanciare entrambi, appunto, in maniera agevole, dando quindi la possibilità di eseguire tutti i comandi eseguendo dalla directory principale del progetto "make <cmd>".

Vediamo i principali comandi messi a disposizione. Come suggerito nel file README.md si consiglia di eseguire i comandi del backend in un virtual environment Python avendo cura di settare la variabile d'ambiente $VENV. Ad ogni esecuzione del comando make verrà chiesto se si sta utilizzando il comando in ambiente di devel o in ambiente di produzione (cambiando il comportamento e le configurazioni in modo opportuno). Ecco l'elenco dei comandi:

- Backend:
  - make make-be: installa il backend e tutte le dipendenze.
  - make test-be: lancia i test del backend.
  - make run-be: esegue il backend.
  - make clean-be: esegue effettua una pulizia del backend (attenzione, rende necessario rieseguire "make make-be").
- Frontend:
  - make make-fe: installa npm, grunt e bower (oltre a tutte le loro dipendenze) e li esegue.
  - make test-fe: lancia le unittest del frontend (non ancora implementate).
  - make run-fe: lancia il demone http che servirà il frontend secondo le specifiche del file di configurazione.
  - make clean-fe: lancia una pulizia del frontend.

È da menzionare che il comando "make run-fe" in modlaità produzione è ancora difettoso (si veda il capitolo 5 a riguardo).

4.7 Conclusione

In questo capitolo sono stati approfonditi i dettagli progettuali introdotti nel capitolo precedente. Per commenti e spiegazioni più precise di ciascun modulo o classe, conviene consultare direttamente la documentazione presente nel codice.

Dopo la trattazione dei dettagli tecnici, è possibile ora effettuare una valutazione sul progetto. Il prossimo capitolo verte appunto su questo argomento.

5. Bird-A: Valutazioni

Il capitolo 3 e il capitolo 4 hanno presentato il progetto Bird-A, la sua struttura, la sua interfaccia utente e, infine, la sua implementazione.

Con queste informazioni è ora possibile valutare se e come il progetto soddisfi le caratteristiche desiderabili analizzate nel capitolo 2, oltre alle caratteristiche desiderabili di un qualunque sistema software come manutenibilità, efficienza e flessibilità.

Conclusa questa analisi, verranno indicati gli sviluppi futuri necessari a rendere il sistema completo e pienamente utilizzabile.

5.1 Soddisfacimento dei requisiti

Nel cap 2.5.1 sono stati analizzati i requisiti funzionali desiderabili per un sistema di RDF data authoring, ovverosia general purpose, collaborativo, distribuito, accessibile e user friendly. Vediamo nel dettaglio come è stato affrontato ciascun obiettivo.

5.1.1 General purpose

Bird-A genera i propri form sulla base di una configurazione espressa da un'ontologia presentazionale. Tale ontologia è svincolata dai significati semantici espressi dalle ontologie target, il che la rende appunto in grado di gestire qualunque ontologia target desiderata.

Oltre a questo aspetto (già realizzato da Gaffe, come visto nel capitolo 2), Bird-A è stato progettato per essere un sistema "stand alone", non si appoggia cioè ad alcun sistema esistente (come ad esempio MediaWiki). Questo costituisce in parte un vantaggio e in parte uno svantaggio.

È un vantaggio, ad esempio, per un ente che deve predisporre ex-novo uno strumento di data authoring, non dovendo per forza predisporre un sistema pensato per l'editing di documenti (strutturato a pagine, incapace di accedere a dataset multipli, mancanza di ricerche sofisticate di dati, ecc).

Lo svantaggio invece risiede nella refrattarietà che tutti noi abbiamo nell'interessarci ad uno strumento nuovo e non consolidato, portandoci sostanzialmente a preferire uno strumento magari non ideale ma che conosciamo già o che è già largamente utilizzato.

5.1.2 Collaborativo

La struttura di Bird-A è pensata per consentire l'editing collaborativo, cioè per consentire sia a più utenti di accedere agli stessi dati (anche se i meccanismi di gestione delle collisioni non sono ancora stati implementati), sia per consentire ad una qualunque entità la creazione di dati basati su dataset su cui non possiede autorità.

Un esempio a riguardo potrebbe essere una comunità di appassionati di serie televisive che vogliono arricchire un database RDF già esistente, eventualmente curato da un'entità che ha un preciso standard qualitativo, con informazioni aggiuntive con pretese di qualità molto più "flessibili". In questo caso, alcuni individui della fanbase con le necessarie competenze tecniche potrebbero predisporre un dataset RDF vergine, configurare dei form Bird-A per consentire di visualizzare le informazioni provenienti dal dataset più blasonato e infine configurare il backend (oltre a fare in modo che almeno un frontend possa accedervi). A questo punto il resto della fanbase vedrà le informazioni dei due dataset presentate sullo stesso form, anche se alcune di queste non saranno editabili

[1].

5.1.3 Distribuito

Come si è già accennato precedentemente e come si è potuto osservare nell'esempio proposto poco fa, un grande vantaggio di questa architettura distribuita è appunto la capacità di soggetti diversi di gestire dataset RDF in modo completamente indipendente gli uni dagli altri, dando però modo all'utente di accedervi (e modificare i dati) senza dover conoscere questa eterogeneità.

Ogni backend può essere gestito da un'autorità differente, che decide quali dati mettere a disposizione, in che modo presentarli e a chi.

5.1.4 Accessibile

L'utente finale potrebbe in linea teorica avere accesso a tutti i backend Bird-A semplicemente collegandosi ad un unico frontend. L'eterogeneità dei sistemi sottostanti viene quasi completamente nascosta al fruitore, che può dedicarsi a cercare e editare i propri dati senza dovere in prima istanza trovare i dataset di suo interesse (quelli cioè che contengono gli individui di quella particolare ontologia), reperire poi i dati e infine effettuarne il merge.

(1) Naturalmente si possono predisporre meccanismi per aggirare questa limitazione, ad esempio popolando alcuni campi ricercando dapprima sul dataset "master" della fanbase e, se qui non è presente, accedere al database "slave" gestito dall'autorità esterna. Questo richiederebbe un'interazione fra i due backend, strada di sicuro percorribile e assolutamente interessante, ma che va oltre gli scopi di questa dissertazione.

5.1.5 User friendly

Oltre alla comodità di reperimento dei dati esposta nel paragrafo precedente, Bird-A fa si che al fruitore non siano richieste conoscenze specifiche, né di RDF e addirittura nemmeno delle concettualizzazioni espresse dalle ontologie sottostanti. È il progettista di form che decide di quali proprietà richiedere l'inserimento, come presentarle e come queste debbano essere espresse in RDF. Al progettista di form è fornita inoltre la possibilità di mostrare label, descrizioni e tooltip (questi ultimi ancora da implementare) per facilitare il compito dell'utente.

Fra i fattori di usabilità da riportare, vi è anche l'appeal grafico di Bird-A. Seppure sia ancora migliorabile, l'integrazione con il framework CSS Bootstrap conferisce un look moderno ed elegante, che migliora l'engagement diminuendo il tasso di abbandono dello strumento. La responsività del design nelle varie risoluzioni permette inoltre di fruire dei contenuti su varie device, quali tablet e smartphone.

È doveroso specificare che l'usabilità dell'attuale implementazione non è ancora ad un livello accettabile, mancando di validazioni ed opportune segnalazioni all'utente nel caso vi sia qualche problema.

Infine, anche quando saranno ultimati gli sviluppi, la reale usabilità andrà testata sul campo tramite opportuni test di usabilità con gli utenti.

5.2 Valutazioni implementative

Oltre ai requisiti di alto livello, un prodotto software deve rispondere anche a precisi requisiti, solitamente impliciti ma non meno importanti, di affidabilità, efficienza e facilità di adozione e sviluppo.

5.2.1 Affidabilità

La divisione fra backend e frontend ha come ulteriore vantaggio lo sviluppo e il test disgiunto delle due componenti, il che segue il principio del "divide et impera".

Il backend è stato testato approfonditamente ed è stata creata una piccola suite di test (anche se questa andrebbe sofisticata in futuro, come riportato sotto), mentre il frontend è stato testato solo tramite interazioni manuali, il che, nonostante esse siano state per quanto possibili approfondite, non fornisce sufficienti garanzie. Anche in questo caso, la creazione di una suite di test è necessaria nel prossimo futuro.

L'affidabilità del codice è stata comunque tenuta in grande considerazione durante gli sviluppi, portando quando possibile ad adottare soluzioni che favorissero il loose coupling dei componenti sia del backend che del frontend.

5.2.2 Efficienza

Anche l'efficienza è stata tenuta in grande considerazione durante gli sviluppi. Ancora una volta, l'architettura ha dato un significativo apporto da questo punto di vista: il backend è sgravato dalla generazione dell'HTML finale, riducendo le risorse necessarie, mentre sul frontend, una volta caricata l'applicazione, il tempo di risposta percepito è più basso visto che non vi è bisogno di ricaricare completamente la pagina.

Il backend è stato implementato per consentire il caching dei due principali oggetti generati: form e individui. I form vengono attualmente caricati allo startup dell'applicazione e possono essere serviti immediatamente, senza accessi al db (anche se questo approccio può andar bene fino a qualche centinaio di form, dopodiché bisognerà adottare qualche tecnica più sofisticata).

Il processo di build del frontend minimizza ed effettua il merge del CSS e dei sorgenti javascript e minimizza l'HTML. La staticità dei file, rende peraltro semplice servirli via CDN, per una rapidità percepita dall'utente nettamente superiore rispetto alla generazione server side.

Come ultimo appunto, è doveroso specificare che nonostante molto sia stato fatto per garantire l'efficienza, molti sono ancora gli aspetti migliorabili su questo fronte. Nella sezione riguardante gli sviluppi futuri se ne affronteranno i principali.

5.2.3 Facilità di adozione e sviluppo

La facilità di configurazione e di run sono fondamentali perché Bird-A possa un giorno veder crescere le installazioni e i contributi di sviluppo.

Per agevolare questo, in primo luogo si è cercato di ridurre il "dependency hell" adottando tool di reperimento automatico delle dipendenze: nel backend si utilizzano "virtualenv", "pip" (Python Package Index) e i "setuptools" per ottenere questo obiettivo, nel frontend si utilizza "npm" (Node Package Manager), utilizzato per reperire la corretta versione di se stesso (per sistemi con versioni di npm non aggiornate, il che è una casistica piuttosto comune) e di tutti i tool successivi come "bower" e "grunt".

I tool sopracitati sono per di più gli standard di fatto nei rispettivi ambiti nel mondo dei progetti open source, favorendo l'ambientamento di sviluppatori che hanno familiarità con questo mondo.

Vi è quindi un solo comando per sottosistema (backend e frontend) da eseguire per reperire e configurare i pacchetti ed un solo comando (sempre per sottosistema) per lanciare il server. Questi comandi sono inoltre gestiti tramite "GNU Make", che automatizza la differenziazione per ambiente (sviluppo e produzione), anche qui per evitare allo sviluppatore di dover ricordare più comandi e quale usare in ciascuna situazione.

I sorgenti sono discretamente commentati, seguono rigorosamente il principio "DRY" (Don't Repeat Yourself) e, dove possibile, quello di "least astonishment" (con opportuni commenti a supporto dello sviluppatore laddove per un motivo o per l'altro questo principio non fosse applicabile).

5.3 Sviluppi futuri

Dopo aver preso in considerazione tutto ciò che è stato fino ad ora implementato, è possibile valutare quanto ancora manca.

Bird-A utilizza un'architettura distribuita per consentire un approccio decentrato e modulare al problema della creazione di dati semantici RDF, tuttavia vi sono ancora diverse sfide da affrontare per renderlo un sistema "production ready" ed è sulla base di queste sfide che si modelleranno gli sviluppi futuri. Vediamo di seguito le più importanti.

5.3.1 Bug noti

L'unico bug ad oggi noto riguarda la gestione del cross site scripting. Per consentire l'architettura di Bird-A, il frontend deve poter interrogare backend non serviti dal suo stesso dominio. Sono state provate due soluzioni: CORS e xdomain. Dato che il focus del progetto non era sulla sicurezza o sulla ricerca del migliore standard ad oggi disponibile per affrontare questo problema, si è deciso di perseguire la strada che portasse al risultato nel minor tempo possibile. Questa strada è stata xdomain.

xdomain richiede tuttavia che sul frontend vengano dichiarati i backend prima dell'importazione di qualunque libreria javascript (andando a modificare l'oggetto core javascript httpRequest) e che ciò debba essere specificato come attributo del tag <script> di importazione. Questo porta a dover modificare il file index.html oltre che il file config.js in fase di configurazione (entrambi i file sono comunque linkati nella directory config/ per ricordare questa particolarità) e a non potere tenere questa libreria insieme alle altre librerie bower, dato che il processo di build ha libertà di riorganizzare e sostituire questi contenuti (perdendo gli attributi definiti originariamente nel tag <script>.

Perché Bird-A sia production ready occorre perciò decidere quale strada intraprendere e apportare gli opportuni sviluppi (ovviamente si può anche far servire frontend e backend dallo stesso dominio, ma così si traviserebbe la filosofia architetturale del progetto).

5.3.2 Sviluppi necessari

Oltre alla problematica precedentemente esposta, sono necessari (o comunque fortemente consigliati) questi sviluppi per rendere il sistema production ready:

- Object properties: le object property rivestono un ruolo fondamentale nella creazione di un dataset e sono alla base dei Linked Data esposti nel capitolo 2. Al fine di rendere Bird-A un progetto completo bisogna gestire i subform, la ricerca degli individui oggetto della relazione e infine la creazione di nuovi se l'individuo cercato non esiste.

- Routing frontend: occorre creare un repository in cui elencare i backend disponibili ed i namespace da essi serviti. Questo si può ottenere realizzando un progetto apposito su GitHub in cui mantenere un file sul quale registrare i vari backend. Questo file verrà servito da una CDN (sfruttando ad esempio RawGit) e verrà acceduto dal frontend per ottenere questa lista di repository.

- Routing backend: occorre far si che, al momento della richiesta di un individuo, il backend abbia modo di identificare (in base al namespace richiesto) su quale dataset accedere fra quelli a sua disposizione.

- Attualmente il backend mantiene le triple RDF su un file, meccanismo poco efficiente e che consente un basso grado di parallelismo. Occorre quindi implementare connettori verso i più comuni database triplestore.

- Autenticazione e autorizzazione: necessarie al fine di garantire che l'edit (ed eventualmente anche la visualizzazione) delle risorse sia garantita solo a chi ne possiede i privilegi e per consentire l'audit delle revisioni di edit.

- Ricerca lato frontend: per permettere all'utente di orientarsi in un contesto in cui vi sono molti form e in cui per ciascuna tipologia di form vi siano molti individui, occorre sofisticare la pagina di "forms list" e "individuals list" fornendo dei filtri di ricerca.

- Validazioni: la chiave della coerenza dei dati su ciascun sistema è la validazione degli input esterni. Nello specifico:

- Nel backend la validazione dei JSON è solo parziale. Per un rilascio in produzione occorre creare dei controlli più stringenti.
- Nel frontend le validazioni sono attualmente minimali e molte validazioni indicate dall'ontologia non sono implementate. In un sistema di produzione, l'utente non dovrebbe avere la possibilità di inserire valori non coerenti con quanto indicato dal progettista di form.

5.3.3 Migliorie apportabili

Si riporta di seguito una lista di migliorie non strettamente necessarie alla messa in produzione ma che renderebbero il sistema più affidabile ed efficace:

- Unit Test: al fine di ottenere un'alta affidabilità del codice e una buona tranquillità nello sviluppo di nuove feature, bisognerebbe creare una test suite appropriata sia per il frontend che per il backend.
- Encryption: cifrare lo scambio di dati servendo sia backend che frontend tramite https.
- Creazione assistita delle URI dei nuovi individui: può essere utile all'amministratore di dataset dare una precisa indicazione all'utente di come dovrebbero essere battezzate le URI dei nuovi individui. In particolare, oltre a utilizzare l'URI di base specificata nel form, bisognerebbe poter indicare:
  - La lista di widget che, con i valori delle proprietà da essi mappate, concorreranno a comporre l'URI.
  - L'espressione regolare da utilizzare per suddividere ogni valore in token (ad esempio "Marco Antonio" deve essere spezzato in due token diversi).
  - Il modo di renderizzare questi token nell'URI (ad esempio in camel case o in lower case).
  - Il separatore fra i vari token (ad esempio "_", "-" oppure anche nessun separatore).

- Caching lato backend: per migliorare i tempi di risposta verso il frontend, bisognerebbe gestire la cache degli individui oltre a quella dei form (l'argomento caching è vasto e aperto a vari scenari che non verranno trattati in questa sede).
- Accorpamento richieste frontend: attualmente il frontend richiede un individuo per volta. Questo può risultare inefficiente per form che presentano un alto numero di valori per le relazioni di object property. Per ridurre questo problema, bisognerebbe accorpare tutte queste richieste.
- Gestione metadati riguardanti gli individui: ad esempio autori, date di modifica, revisioni, ecc.

5.3.4 Feature auspicabili

Gli ambiti di sviluppo presentati in questo paragrafo non sono immediatamente necessari per una versione 1.0 di Bird-A, ma saranno sicuramente utili a rendere il sistema maggiormente ergonomico e ricco nelle prossime release:

- Filtri personalizzabili: nella lista delle istanze sarebbe opportuno prevedere la possibilità di avere dei filtri personalizzabili legati al form di riferimento. La personalizzazione potrebbe essere implementata con un'estensione della Bird-A ontology che permetta di creare filtri anche complessi (come la spunta su "maggiorenne" per ottener tutti gli individui in cui la proprietà "età" sia maggiore o uguale a "18").
- Autocomplete: come è risaputo, l'autocomplete facilita l'inserimento e aumenta l'engagement dell'utente.
- Gestione migliorata hasOptions: attualmente le opzioni della Bird-A ontology sono trattate come stringhe CSV. Sebbene questo possa andar bene per i casi più semplici, non gestisce feature desiderabili quali: differenziazione fra nome e descrizione presentata all'utente e valore scritto a database, possibilità di avere più traduzioni per il medesimo valore scritto a database (al fine di non richiedere il reinserimento di detto valore per tutte le lingue), possibilità di avere dei valori di tipo risorsa, ecc.

- Creazione agevolata di form: come si è già indicato all'inizio, la creazione manuale di dati RDF è un'operazione complessa che richiede una forte competenza tecnica. Non fa eccezione la creazione di istanze Bird-A. Per ridurre gli errori manuali commettibili durante questo inserimento, si è creata una libreria apposita nel backend (come indicato nel capitolo 4), anche se per effettuare questo inserimento occorre comunque creare uno script e anche questa non è un'operazione banale e priva di possibili errori. Per una vasta adozione sarebbe quindi auspicabile lo sviluppo di un tool di configurazione grafico che permetta la creazione di istanze Bird-A anche ad utenze non qualificate.

- Breadcrumb: nel caso l'utente stia inserendo una fattura e debba creare l'anagrafica di un cliente non ancora inserita a database, il subform relativo dovrà portarlo in un form di inserimento di questa anagrafica. Questa anagrafica potrebbe aver bisogno di riferirsi ad un altro individuo non ancora inserito a database e così via. Per consentire all'utente di non smarrirsi durante questo genere di operatività, è utile presentare un "breadcrumb" (la rappresentazione del percorso fatto dall'utente per arrivare a trovarsi dove si trova in questo momento).

- Canonicizzazione URL frontend: se in futuro si vorranno rendere accessibili i contenuti di Bird-A ai motori di ricerca, conviene pensare ad un modo di rendere le URL esposte dal frontend più "canoniche". Attualmente i form e le istanze sono espresse utilizzando le URI per esteso. Un metodo più "SEO oriented" potrebbe prevedere degli alias per le base URI delle istanze (ed eventualmente dei form) e presentare la pagina di edit / visualizzazione di un'istanza come "http://www.example.com/edit/istanza/foaf/nome-cognome ?form=<form_uri>" anziché come "http://www.example.com/edit?individual =http://foo.com/istanza/foaf/0.2/nome-cognome&form=<form_uri>"

- Metadati sugli individui: nell'editing degli individui, prevedere la memorizzazione di informazioni quale autori, timestamp di ultima modifica e cronologia delle revisioni.

5.4 Punti aperti

Di seguito sono indicate le problematiche aperte non affrontate nei punti precedenti perché necessitano di considerare le varie opzioni disponibili, prendere delle scelte e prioritizzare gli sviluppi. Questo tipo di trattazione sarebbe andata oltre gli scopi di questa dissertazione, ma conviene comunque menzionarle:

- Merging dei dati provenienti da più backend: la possibilità di avere più backend è sicuramente interessante, ma richiede di rispondere ad alcune domande, tra cui:
  - Come gestire risorse diverse con medesime URI?
  - Come gestire la stessa risorsa a cui sono associate più URI (il caso del "same as")? Questa problematica è comunque da gestire anche a livello di singolo backend.
  - Come gestire risorse le cui informazioni sono sparse fra più backend?
- Autenticazione e autorizzazione: sotto questo fronte due sono i punti da dirimere:
  - Introdurre un meccanismo di single sign-on fra i vari backend? Per evitare all'utente di doversi autenticare su ciascun backend si possono usare strumenti di single sign-on come quelli messi a disposizione da Google, Facebook, Open ID, ecc
  - Come gestire le autorizzazioni "distribuite"? Il problema riguarda come dare a ciascun amministratore di dataset la facoltà di decidere chi possa accedere ai dati e con quali limitazioni. Una soluzione suggerita è la creazione di un'ontologia satellite della Bird-A ontology volta alla definizione dei privilegi di accesso per ciascun Widget (con criteri la propagazione di questi privilegi anche agli eventuali wiget discendenti salvo ridefinizioni). Le istanze di questa ontologia dovrebbero essere popolate direttamente da chi gestisce il dataset.
- Come gestire le risorse con più tipi? Per come è attualmente organizzato BIrd-A, ad una stessa URI non possono riferirsi form diversi (di fatto la creazione di un nuovo individuo tramite un determinato form viene impedita in caso l'URI abbia

già dei valori a database).

- Come gestire le risorse con diversi tipi RDF? Attualmente bird-A nella creazione di una nuova risorsa controlla solamente che l'URI non esista già. Se esiste ne richiede la modifica prima di procedere al salvataggio. Questo purtroppo non consente di creare risorse con più tipi, come ad esempio foaf:Person e dc:creator.

- Come gestire i valori "speciali"? In questo caso con il termine "speciali" si indicano quei valori che veicolano informazioni particolari quali:

  – "sconosciuto": un esempio a riguardo è il luogo di nascita di una persona nata prima dell'introduzione dei registri di nascita. In questo caso deve essere distinto dal caso in cui non siano stati inseriti dati dall'utente. Questa informazione deve essere trattata a parte dai ragionatori, che sarebbero altrimenti indotti a inferire che due persone con luogo di nascita sconosciuto siano nati nello stesso luogo.

  – "nessuno": un esempio a riguardo può essere il caso in cui una persona non abbia fratelli. Anche in questo caso i ragionatori devono essere istruiti a considerare tale informazione in modo speciale, per evitare di considerare tutte le persone senza fratelli come tra loro fratelli (per la transitività della relazione "è fratello di").

- Quali metadati conservare sulle triple RDF modificate e come gestirli? Come è noto una tripla RDF indica un'asserzione. Al fine di valutare l'attendibilità di questa asserzione, bisognerebbe prevedere la gestione di informazioni di "provenance" (affrontante ad esempio dalla raccomandazione del W3C "Provenance Ontology" [W3C13]) quali:

  – Autore: chi è l'autore o qual è l'ente che gestisce il dataset.

  – Metodo di acquisizione: ad esempio rilevazione statistica, osservazione diretta, ecc.

  – Fonte: nel caso l'informazione sia stata tratta da un lavoro preesistente.

  – Data di acquisizione: utile in quei contesti in cui la conoscenza di una certa in-

formazione è in evoluzione (ad esempio il numero stimato di galassie nell'universo) o nel caso in cui un valore sia suscettibile a mutamenti nel tempo (ad esempio la popolazione di una nazione).

5.5 Conclusione

In questo capitolo si è analizzato come il progetto Bird-A miri a soddisfare i requisiti esposti nel capitolo 2 e quale sia lo stato attuale degli sviluppi, compreso ciò che ancora manca a rendere questo sistema pienamente utilizzabile in un contesto di produzione.

Dopo questa trattazione è infine possibile trarre alcune conclusioni, argomento del prossimo capitolo.

6. Conclusioni

Come visto nei capitoli iniziali, lo scopo del progetto Bird-A è fornire un'interfaccia web collaborativa di visualizzazione e editing di dati RDF, che sia general purpose e distribuita.

Si sono in seguito analizzati i requisiti desiderabili per questo tipo di applicazione e si sono passati in rassegna buona parte dei progetti messi in campo per facilitare sempre più il fiorire di dati semantici tra loro connessi, o Linked Data, che sarà il fattore determinante che porterà il Web a salire il prossimo gradino evolutivo, verso ciò che è noto come Semantic Web.

Abbiamo visto che i tool attualmente a disposizione non sono stati progettati per fornire uno strumento di generazione di form general purpose e come questa mancanza sia stata affrontata da Bird-A tramite la messa a punto di un'ontologia puramente presentazionale, la Bird-A Ontology, agnostica da qualsiasi ontologia target (non si lega cioè ad alcuna ontologia particolare) e quindi potenzialmente in grado di raggiungere un potere espressivo tale da descrivere la creazione di dati RDF facenti riferimento ad una qualunque ontologia (seguendo il principio del "the least power" proprio dello standard RDF), mettendo in relazione i form e i suoi widget con entità esterne (ad es. classi) in modo da popolarle con opportuni individui.

Ci si è in seguito soffermati sul focus, da parte della maggior parte dei sistemi attuali, su di un singolo dataset RDF, limitazione che riduce le potenzialità di qualsiasi tool di presentazione e di editing, costringendo a creare istanze differenziate e non cooperanti di questi tool nel caso si voglia garantire a più entità autonome il controllo sui propri

dati.

È stata quindi presentata la strategia messa a punto con il progetto Bird-A per superare questo ostacolo, ovverosia l'architettura distribuita a più backend e più frontend, ciascuno ipoteticamente gestito da un'entità autonoma e con pieno controllo dei propri dati (eventualmente sparsi fra più dataset).

Nel capitolo 5 si è infine valutato il progetto Bird-A nei confronti di queste ambizioni. Come si è potuto constatare lo sviluppo del progetto Bird-A è solo all'inizio. Diversi sono gli sviluppi necessari a renderlo un sistema utilizzabile in un contesto di produzione e molte sono le sfide da affrontare per renderlo un sistema efficace per la gestione dell'authoring RDF.

Questo primo prototipo richiederà ancora molti sforzi per raggiungere gli obiettivi che il progetto si prefigge, tuttavia la speranza è che questa base sia sufficientemente interessante da stimolare gli sviluppi che potrebbero portarlo ad essere adottato in quei contesti in cui la necessità di creare e modificare dati RDF diventa sempre più sentita.

7. Bibliografia

[AC16] Apache Cordova - Apache Cordova
https://cordova.apache.org/ (Ultima visita febbraio 2016)

[ADR06] Sören Auer, Sebastian Dietzold, Jens Lehmann, Thomas Riechert (2006) - "OntOWiki - A Tool for Social, Semantic Collaboration". In Cruz, I. F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L. (Eds.), Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Lecture Notes in Computer Science 4273: 736-749. Berlin, Germany: Springer. DOI: 10.1007/11926078_53.
http://www2007.org/workshops/paper_91.pdf

[ANG16] Angularjs.org (2016) - AngularJS by Google https://angularjs.org/

[BER06] Tim Berners-Lee (27 Luglio 2006) - "Linked Data". W3C 27 Luglio 2006
https://www.w3.org/DesignIssues/LinkedData.html

[BER09] Tim Berners-Lee (2009) - "The Next Web". Ted Conferences, LLC Febbraio 2009
http://www.ted.com/talks/tim_berners_lee_on_the_next_web?language=en

[BFM05] T. Berners-Lee, R. Fielding, L. Masinter (Gennaio 2005) - Uniform Resource Identifier (URI): Generic Syntax
https://tools.ietf.org/html/rfc3986

[BGE08] M. Buffa, F. Gandon, G. Ereteo, P. Sander, C. Faron (2008) - SweetWiki: "A semantic wiki. In Journal of Web Semantics: Science, Services and Agents on the World Wide Web" 6 (1): 84-97. doi: 10.1016/j.websem.2007.11.003.

[BHL01] Tim Berners-Lee, James Hendler, Ora Lassila (17 Maggio 2001) - "The Semantic Web". Scientific American Magazine 2001.
http://www.scientificamerican.com/article/the-semantic-web/

[BID09] V. Bolognini, A. Di Iorio, S. Duca, A. Musetti, S. Peroni, F. Vitali (2009) - Gaffe: "Exploiting Ontologies To Deploy User-Friendly and Customized Metadata Editors". In Proceedings of the IADIS Internet/WWW 2009 conference, eds. B. White, P. Isaías, M. B. Nunes. Lisbon, Portugal: IADIS 2009
http://speroni.web.cs.unibo.it/publications/bolognini-2009-exploiting-ontologies-deploy.pdf

[BM14] Dan Brickley, Libby Miller (14 January 2014) - "FOAF Vocabulary Specification 0.99" http://xmlns.com/foaf/spec/

[BOR08] Valerio Bordin (2008) - "Caratteristiche sociali e culturali del Web 2.0"
http://vitali.web.cs.unibo.it/viewfile/LabInt08/ConsegnaRelazioni?rev=1.3&filename=RelazioneBordin.pdf

[BOW16] Bower.io - Bower: A package manager for the web
http://bower.io/ (Ultima visita marzo 2016)

[BSB09] Jie Bao, Paul R. Smart, Dave Braines, Nigel R. Shadbolt (2009) - "A Controlled Natural Language Interface for Semantic Media Wiki Using the Rabbit Language". In Proceedings of the 3rd Annual Conference of the International Technology Alliance (ACITA2009). September 23-24, 2009, Maryland, USA. http://www.usukita.org/pa

[CAB10] R. Cailliau, JF. Abramatic, Tim Berners-Lee (2010) - "History of the Web".
http://pchospital.co.uk/wp-content/uploads/2015/04/History-of-the-World-Wide-Web.pdf

[COM16] Compass-style.org - Compass

http://compass-style.org/ (Ultima visita marzo 2016)

[CPP15] Alexandru Constantin, Silvio Peroni, Steve Pettifer, David Shotton, Fabio Vitali (2015) - "The Document Components Ontology (DoCO)". eds. Oscar Corcho http://speroni.web.cs.unibo.it/publications/constantin-in-press-document-components-ontology.pdf

[DAT16] data.gov - Data.gov http://www.data.gov/ (Ultima visita marzo 2016)

[DBP16] DBPedia - "DBPedia: Facts and Figures". http://wiki.dbpedia.org/about/about-dbpedia/facts-figures (Ultima visita febbraio 2016)

[DMP10a] A. Di Iorio, A. Musetti, S. Peroni, F. Vitali (2010) - OWiki: "Crowdsourcing semantic content: a model and two applications". In Proceedings of the 3rd International Conference on Human System Interaction (HSI10): 563-570, eds. T. Pardela. Washington, District Columbia, USA: IEEE Computer Society. doi: 10.1109/HSI.2010.5514513. http://speroni.web.cs.unibo.it/publications/di-iorio-2010-crowdsourcing-semantic-content.pdf

[DMP10b] A. Di Iorio, A. Musetti, S. Peroni, F. Vitali (2010) - "Ontology-driven generation of wiki content and interfaces". In New Review of Hypermedia and Multimedia, 16 (1): 9-31. DOI: 10.1080/13614568.2010.497194. http://speroni.web.cs.unibo.it/publications/di-iorio-2010-ontology-driven-generation-wiki.pdf

[DMP11] A. Di Iorio, A. Musetti, S. Peroni, F. Vitali (2011) - "OWiki: enabling an ontology-led creation of semantic data". In Human-Computer Systems Interaction: Backgrounds and Applications 2, eds. Z. S. Hippe, J. L. Kulikowski, T. Mroczek. Berlin, Germany: Springer. ISBN: 3642231711. http://speroni.web.cs.unibo.it/publications/di-iorio-2012-owiki-enabling-ontology-led.pdf

[DPT06] Karsten Dello, Elena Paslaru Bontas Simperl, and Robert Tolksdorf (2006) - "Creating and using Semantic Web information with Makna". In Proceedings of the 1st Workshop on Semantic Wikis – From Wiki To Semantics eds. M. Völkel, S. Schaffert. CEUR Workshop Proceedings, vol. 206: Aachen, Germany: CEUR-WS.org.
http://www.ceur-ws.org/Vol-206/paper4.pdf

[DUM15] Michel Dumontier (13 Settembre 2015) - "Bio2RDF" http://bio2rdf.org/. https://github.com/bio2rdf/bio2rdf-scripts/wiki

[FGP14] Riccardo Falco, Aldo Gangemi, Silvio Peroni, David Shotton, Fabio Vitali (2014) - "Modelling OWL ontologies with Graoo"
http://speroni.web.cs.unibo.it/publications/falco-2014-modelling-ontologies-with.pdf

[FT02] Roy T. Fielding, Richard N. Taylor (2002) - REST: "Principled Design of the Modern Web Architecture" In CM Transactions on Internet Technology, Vol. 2, No. 2, May 2002, Pages 115–150.
http://www.ics.uci.edu/ taylor/documents/2002-REST-TOIT.pdf

[GET16] Getbootstrap.com - Bootstrap http://getbootstrap.com/ (Ultima visita marzo 2016)

[GIT16a] Git-scm.com - GIT https://git-scm.com/ (Ultima visita marzo 2016)

[GIT16b] Github.com (2016) - Github https://github.com/

[GNU07] Gnu.org (2007) - GNU General Public License
http://www.gnu.org/licenses/gpl-3.0.en.html

[GNU14] Gnu.org (2014) - GNU Make https://www.gnu.org/software/make/

[GR93] Thomas R. Gruber (Giugno 1993) - "A translation approach to portable ontology specifications". Appeared in Knowledge Acquisition, 5 (2):199-220, 1993.

http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[GRU16] Gruntjs.com - Grunt: The JavaScript Task Runner http://gruntjs.com/ (Ultima visita marzo 2016)

[GUA98] Nicola Guarino (1998) - "Formal Ontology in Information Systems: Proceedings of the First International Conference (FOIS'98), June 6-8, Trento, Italy". EditoreIOS Press, 1998
https://books.google.it/books?hl=it&lr=&id=Wf5p3_fUxacC&oi=fnd&pg=PR5&dq=ontology&ots=nmVG-WoDKH&sig=ogbCo6YgXngQDoll8R1_LIBw0uM#v=onepage&q=ontology&f=false

[HAY16] Project Haystack - Project Haystack http://project-haystack.org/ (Ultima visita febbraio 2016)

[HB11] Tom Heath, Christian Bizer (2011) - "Linked Data: Evolvingthe Web into a Global Data Space" (1st edition). SynthesisLectures on the Semantic Web: Theory and Technology, 1:1,1-136. Morgan & Claypool.
http://info.slis.indiana.edu/ dingying/Teaching/S604/LODBook.pdf

[HGO05] Andreas Harth, Hannes Gassert, Ina O'Murchu, John G. Breslin, Stefan Decker (2005) - "WikiOnt: An Ontology for Describing and Exchanging Wikipedia Articles". In Voss, J., Lih, A., Klein, S., Ma, C. (Eds.), Proceedings of Wikimania 2005.
http://meta.wikimedia.org/wiki/Wikimania05/Paper-IM1

[ILS15] Internet Live Stats (2015) - Internet Users
http://www.internetlivestats.com/internet-users/

[ION16] Ionic - Ionic: Advanced HTML5 Hybrid Mobile App Framework
http://ionicframework.com/ (Ultima visita febbraio 2016)

[JHY10] Prateek Jain, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, and Amit P. Sheth.

(2010) - "Linked Data Is Merely More Data". In: Dan Brickley, Vinay K. Chaudhri, Harry Halpin, and Deborah McGuinness: Linked Data Meets Artificial Intelligence. Technical Report SS-10-07, AAAI Press, Menlo Park, California, 2010
http://www.aaai.org/Library/Symposia/Spring/ss10-07.php

[JSO16] Json.org - Introduzione a JSON http://www.json.org/json-it.html (Ultima visita marzo 2016)

[KG16] Yaron Koren, Stephan Gambke and others - "Semantic Forms"
http://www.mediawiki.org/wiki/Extension:Semantic_Forms (Ultima visita febbraio 2016)

[KHD05] Bertin Klein, Christian Hoecht, Björn Decker (2005) - "Beyond Capturing and Maintaining Software Engineering Knowledge – "Wikitology" as Shared Semantics". Presented during the Workshop on Knowledge Engineering and Software Engineering (KESE 2005). 11 Settembre 2005, Koblenz Germany.
http://www.dfki.uni-kl.de/ klein/papers/finalKESE05.pdf

[KL03] Judy Kay, Andrew Lum (Novembre 2003) - "An Ontologically Enhanced Metadata Editor". University of Sidney, School of Information Technologies, Technical Report 541. Novembre 2003.
http://www.it.usyd.edu.au/research/tr/tr541.pdf

[KNU16] Holger Knublauch - "SPARQL Web Pages (SWP, aka UISPIN)"
http://uispin.org/index.html (Ultima visita febbraio 2016)

[LH09] Markus Luczak-Rosch, Ralf Heese (2009) - "Linked Data Authoring for Non-Experts".
http://ceur-ws.org/Vol-538/ldow2009_paper4.pdf

[MEG03] Luke McDowell, Oren Etzioni, Steven D. Gribble, Alon Halevy, Henry Levy, William Pentney, Deepak Verma, and Stani Vlasseva (2003) - "Evolving the Semantic

Web with Mangrove".
http://homes.cs.washington.edu/ etzioni/papers/www12submission.pdf

[NOD16] Nodejs.org - Node JS https://nodejs.org/en/ (Ultima visita marzo 2016)

[NPM16] Npmjs.com - NPM: Node Package Manager https://www.npmjs.com/ (Ultima visita marzo 2016)

[PBV08] Mukaddim Pathan, Rajkumar Buyya, Athena Vakali (2008) - "Content Delivery Networks: State of the Art, Insights, and Imperatives" In Content Delivery Networks Volume 9 of the series Lecture Notes Electrical Engineering pp 3-32, eds Springer Berlin Heidelberg. doi:10.1007/978-3-540-77887-5_1
http://link.springer.com/chapter/10.1007/978-3-540-77887-5_1

[PEW14] Pew Research Center (Gennaio 2014) - "Social Networking Fact Sheet"
http://www.pewinternet.org/fact-sheets/social-networking-fact-sheet/

[PL08] Alexandre Passant, Philippe Laublet (2008) - UFOWiki: "Towards an Interlinked Semantic Wiki Farm". In Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008), CEUR Workshop Proceedings 360: 1-14. eds. C. Lange, S. Schaffert, H. Skaf-Molli, M. Völkel. Aachen, Germany: CEUR-WS.org. http://ceur-ws.org/Vol-360/paper-19.pdf

[PRO16] Protege.standford.edu - Protégé http://protege.stanford.edu/ (Ultima visita: febbraio 2016)

[PYL16] Pylonsproject.org - Pyramid: Pylons project http://www.pylonsproject.org/ (Ultima visita marzo 2016)

[PYT03] Python.org (7 Dicembre 2003) - PEP 0333 – Python Web Server Gateway Interface v1.0
https://www.python.org/dev/peps/pep-0333/

[PYT10] Python.org (2010) - Python 2.7
https://www.python.org/download/releases/2.7/

[PYT15a] Python.org (2015) - RDF: rdflib 4.2.1 https://pypi.python.org/pypi/rdflib

[PYT15b] Python.org (2015) - Python setuptools 20.2.2
https://pypi.python.org/pypi/setuptools

[QHK03] Dennis Quan, David Huynh, and David R. Karger (2003) - "Haystack: A Platform for Authoring End User Semantic Web Applications"
http://haystack.csail.mit.edu/papers/iswc2003-haystack.pdf

[RE79] Reenskaug, T. (1979). Model-View-Controller: XEROX Park 1978-1979.
http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html.

[SAS15] Sass-lang.com (2015) - SASS http://sass-lang.com/

[SBH10] Satya S. Sahoo, Olivier Bodenreider, Pascal Hitzler, Amit Sheth, Krishnaprasad Thirunarayan (2010) - "Provenance Context Entity (PaCE): Scalable Provenance Tracking for Scientific RDF Data". In: Michael Gertz, Bertram Ludäscher: Scientific and Statistical Database Management Volume 6187 of the series Lecture Notes in Computer Science pp 461-470
http://link.springer.com/chapter/10.1007/978-3-642-13818-8_32#page-1

[SBP14] Max Schmachtenberg, Christian Bizer, Heiko Paulheim (2014) - "State of the Linked Open Data Cloud 2014"
http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/

[SBR14] Statistic Brain Research Institute (2014) - "Total Number of Pages Indexed by Google"
http://www.statisticbrain.com/total-number-of-pages-indexed-by-google/

[SCH06] Sebastian Schaffert (2006) - "IkeWiki: A Semantic Wiki for Collaborative

Knowledge Management". In Proceedings of 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006): 388-396. Washington, District Columbia, USA: IEEE Computer Society. DOI: 10.1109/WETICE.2006.46.
https://pdfs.semanticscholar.org/33a0/b467c0f405c961b2bd2401260587f99ad503.pdf

[SOU05] Adam Souzis (2005) - "Building a Semantic Wiki". In IEEE Intelligent Systems, eds. Steffen Staab 20 (5): 87-91. doi: 10.1109/MIS.2005.83.
http://www.ualberta.ca/ reformat/ece627w2013/papers/IEEEXplore-17.pdf

[TAU07] Joshua Tauberer (28 Agosto 2007) - "2000 U.S. Census in RDF. (rdfabout.com)"
https://datahub.io/dataset/2000-us-census-rdf

[TQ16] TopQuadrant - "TopBraid Platform Overview". TopQuadrant, Inc.
http://www.topquadrant.com/technology/topbraid-platform-overview/ (Ultima visita febbraio 2016)

[VK14] Denny Vrandecic, Markus Krötzsch (2014) - "Wikidata: A Free Collaborative Knowledgebase"
http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42240.pdf

[VKV06] M. Všlkel, M. Krštzsch, D. Vrandecic, H. Haller, R. Studer (2006) - Semantic wikipedia. In Proceedings of the 15th international conference on World Wide Web (WWW 2006), eds. L. Carr, D. De Roure, A. Iyengar, C. A. Goble, M. Dahlin, 585-594. New York, ACM. doi:10.1145/1135777.1135863

[W3C04] W3C (10 Febbraio 2004) - "OWL Web Ontology Language Reference" https://www.w3.org/TR/owl-ref/

[W3C12] W3C (11 Dicembre 2012) - OWL 2 Web Ontology Language
https://www.w3.org/TR/owl2-overview/

[W3C13] W3C Recommendation (30 Aprile 2013) - "PROV-O: The PROV Ontology". https://www.w3.org/TR/prov-o/

[W3C14a] W3C (25 Febbraio 2014) - "RDF 1.1 Concepts and Abstract Syntax" https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[W3C14b] W3C (25 Febbraio 2014) - RDF Schema 1.1 https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[W3I16] w3id - "Permanent Identifiers for the Web: Secure, permanent URLs for your Web application that will stand the test of time". In w3id.org https://w3id.org/ (Ultima visita febbraio 2016)

[WN06] E. Rowland Watkins, Denis A. Nicole (2006) - "Named Graphs as a Mechanism for Reasoning About Provenance" http://eprints.soton.ac.uk/261935/1/APWEB655.pdf

[ZRM12] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann and Sören Auer (2012) - "Quality Assessment for Linked Data: A Survey. A Systematic Literature Review and Conceptual Framework". http://www.semantic-web-journal.net/system/files/swj773.pdf