POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione
Corso di Studi in Ingegneria Informatica

ServiceRetrieval: ricerca ed esplorazione di grandi collezioni di servizi

Relatore: Prof. Letizia TANCA
Correlatore: Ing. Mirjana MAZURAN

Tesi di laurea di:
Dario FORNARI
matr. 782747

Anno Accademico 2012-2013

Ringraziamenti

Ringrazio la mia famiglia per il loro sostegno, senza di loro non sarei riuscito a raggiungere questo traguardo. Ringrazio in particolare mia mamma per avermi sempre appoggiato nelle scelte e aiutato nelle situazioni di maggiore difficoltà.

Ringrazio mia nonna, per tutto quello che ha fatto per me e per avermi sempre sostenuto nel mio percorso.

Ringrazio la Prof.ssa Letizia Tanca e la Prof.ssa Mirjana Mazuran per l'aiuto e la disponibilità che mi hanno offerto nello sviluppare questo progetto.

Ringrazio i compagni di studio per la collaborazione in alcuni progetti e il loro supporto durante gli studi universitari.

Indice

Elenco delle figure

Elenco delle tabelle

Abstract

Il continuo aumento delle informazioni presenti nei sistemi di memorizzazione ha portato alla necessità di pensare e sviluppare nuovi metodi, diversi da quelli tradizionali, per riuscire a ricavare le informazioni volute. Lo scopo del lavoro sviluppato in questa tesi è proprio proporre nuovi metodi per recuperare i dati, tenendo in considerazione le conoscenze possedute dai diversi tipi di utente. Per questo scopo è stata implementata un'applicazione chiamata ServiceRetrieval. Questa svolge la funzione di recupero di servizi all'interno del progetto SeNSori, nel quale si vuole ottimizzare il consumo energetico in un ambiente e per tale scopo il numero di informazioni presenti nel sistema di memorizzazione è elevato. A seconda delle conoscenze possedute, gli utenti utilizzando l'applicazione ServiceRetrieval possono recuperare i servizi inserendo delle parole chiave per indicare di cosa necessitano oppure possono ricavarli attraverso l'esplorazione, cioè il sistema suggerisce per quale attributo l'utente deve inserire il valore.

Capitolo 1

Introduzione

Il numero di dati presenti nei sistemi di memorizzazione è in continua crescita e ciò rende sempre più difficile riuscire a ricavare le informazioni corrette di cui si necessita. L'avere a che fare con grandi quantità di informazioni è già stato considerato da Alvin Toffler nel 1970 nel suo libro *Future Shock*, il quale utilizzò l'espressione "Information overload" riferendosi alla difficoltà di capire e prendere decisioni quando sono disponibili molte informazioni. A maggior ragione questo è vero se si prendono in considerazione persone con scarsa conoscenza del modo di interagire con i sistemi di memorizzazione. È perciò necessario pensare a nuovi metodi che aiutano l'utente a trovare quello che cerca. Possibili soluzioni vengono introdotte in questa tesi, dove viene sviluppata un'applicazione, chiamata ServiceRetrieval, che cerca di aiutare diversi tipi di utenti a individuare le informazioni a loro utili.
La prima parte dell'applicazione è stata sviluppata per utenti esperti, che hanno una certa conoscenza delle informazioni disponibili, ma che non conoscono in maniera abbastanza approfondita il linguaggio di interrogazione per recuperare le informazioni. La soluzione pensata è di proporre all'utente una lista di interrogazioni in linguaggio naturale, tra le quali egli può selezionare quella da eseguire di suo interesse. La lista proposta è ordinata in base a delle parole chiave inserite dall'utente. All'utente non viene chiesto di inserire direttamente delle query in linguaggio naturale, in quanto si creerebbe il problema di capire il significato corretto dell'interrogazione inserita, operazione computazionalmente costosa e che richiederebbe una complessa fase di addestramento, incrementando il tempo ottenuto per ottenere le informazioni.
La seconda parte dell'applicazione invece è stata sviluppata per utenti ine-

sperti, che non conoscono abbastanza il dominio applicativo per recuperare le informazioni di cui necessitano utilizzando l'applicazione precedente. Questa parte utilizza il concetto di esplorazione. Con l'esplorazione si ottengono le informazioni attraverso continue interazioni col sistema: si instaura così una sorta di dialogo tra l'utente ed il sistema stesso. Ad ogni passo del dialogo, il sistema partendo dai servizi ottenuti al passo precedente, computa il prossimo attributo per il quale l'utente dovrà inserire un valore. Quando tutte le condizioni sono state individuate, i servizi rimasti sono quelli che soddisfano le condizioni espresse dall'utente. Attraverso l'esplorazione l'utente riesce a comprendere il dominio dell'applicazione, riesce a trovare le correlazioni tra i diversi servizi e a capire quali sono le caratteristiche rilevanti, arricchendo in questo modo la sua conoscenza.
Il dominio in cui l'applicazione ServiceRetrieval è stata sviluppata è quello del progetto SeNSori. L'obiettivo di questo è di gestire i dispositivi installati in un ambiente in modo da ridurre il consumo energetico. Per questo scopo è necessario l'utilizzo di un alto numero di sensori e attuatori, ai quali si accede attraverso l'utilizzo di servizi i quali mascherano le differenze tecnologiche presenti. L'applicazione ServiceRetrieval viene utilizzata per il recupero di questi servizi. Il dominio del progetto SeNSori è significativo per lo scopo prefissato perché per ottimizzare correttamente il consumo energetico è necessario memorizzare un alto numero di informazioni, come la posizione dei sensori, il loro tipo, il servizio collegato, ...e in più diversi tipi di utenti con diverse conoscenze possono averci a che fare.

1.1 Organizzazione della tesi

Nel capitolo 2 viene spiegato lo stato dell'arte, descrivendo come l'applicazione sviluppata si pone rispetto allo stato della ricerca.
Nel capitolo 3 vengono descritte le principali tecnologie necessarie per lo sviluppo dell'applicazione ServiceRetrieval. Si parla sia di aspetti teorici, spiegando cosa sono le ontologie e il semantic web, sia di strumenti utilizzati per gestire questi aspetti e per implementare l'applicazione.
Nel capitolo 4 viene esposto il progetto SeNSori, nel quale l'applicazione ServiceRetrieval cerca di recuperare i servizi. Vengono descritti i motivi che hanno portato al suo sviluppo, l'architettura proposta per raggiungere lo scopo per

il quale il progetto SeNSori è stato creato e le ontologie che contengono i dati necessari al corretto funzionamento.
Nel capitolo 5 viene spiegata l'infrastruttura software dell'applicazione Sensori_NL che compone la prima parte dell'applicazione ServiceRetrieval, quella cioè incaricata del recupero di servizi attraverso l'inserimento di parole chiave. Viene descritto il progetto SAFE [17] dal quale Sensori_NL prende l'idea, spiegandone le differenze che hanno portato allo sviluppo di un'applicazione diversa. In seguito viene presentata l'infrastruttura software vera e propria, mostrando l'architettura e descrivendo nel dettaglio le parti di cui è composta, in particolare ci si soffermerà sulla logica sviluppata. Nella parte finale del capitolo vengono esposti i risultati per mezzo di esempi che ne mostrano il funzionamento.
Nel capitolo 6 viene descritta l'infrastruttura software dell'applicazione SensoriExploration che compone la seconda parte dell'applicazione ServiceRetrieval, quella relativa all'esplorazione dei servizi. Si parla inizialmente dell'idea che sta dietro al concetto di esplorazione, spiegandone gli aspetti che la portano ad essere considerata un nuovo campo di ricerca. Quindi si passa alla descrizione dell'infrastruttura software riguardante SensoriExploration. Anche per questa, dopo aver descritto l'architettura si spiegherà la logica sviluppata, in particolare mostrando le diverse distribuzioni implementate. Infine vengono mostrati i risultati per mezzo di esempi, tramite i quali sarà poi possibile effettuare un'analisi delle differenti distribuzioni.

1.2 Contributi della tesi

Attraverso questa tesi sono state affrontate le tematiche di ricerca sul recupero di informazioni quando queste sono presenti in grandi quantità. Nello specifico è stata sviluppata l'applicazione ServiceRetrieval che permette il recupero di servizi del progetto SeNSori. ServiceRetrieval può essere vista come composta da due applicazioni separate: Sensori_NL e SensoriExploration, in quanto pensate per diverse necessità. All'avvio di ServiceRetrieval l'utente seleziona l'applicazione di cui ha bisogno.
Sensori_NL è stata sviluppata per utenti che non hanno la conoscenza del linguaggio di interrogazione necessario per recuperare i servizi presenti nel sistema di memorizzazione. Attraverso l'inserimento di parole chiave, che identificano

le esigenze degli utenti, una lista di query poste in linguaggio naturale viene proposta per recuperare ciò di cui l'utente necessita. Tale lista è ordinata in base alla rilevanza che le query hanno rispetto alle parole chiave inserite. Selezionando la query di cui si necessita vengono recuperati i servizi cercati. Un'estensione di Sensori_NL permette di recuperare servizi attraverso l'inserimento di query poste direttamente nel linguaggio di interrogazione. Questo consente ad utenti molto esperti di recuperare servizi attraverso l'inserimento di vincoli complessi che non sarebbero esprimibili attraverso l'inserimento di parole chiave o di esprimere query che non sono state pensate e quindi non vengono proposte in linguaggio naturale. Questa funzionalità presuppone che gli utenti oltre a conoscere il linguaggio di interrogazione conoscano perfettamente anche il dominio applicativo.
SensoriExploration è stata invece pensata per utenti inesperti, che non conoscono il dominio applicativo e non sarebbero perciò in grado di utilizzare Sensori_NL. Partendo da una query inserita dall'utente, che può anche essere generale (cioè senza specificare vincoli), l'applicazione SensoriExploration suggerisce degli attributi da raffinare, in modo tale da ridurre il numero di servizi proposti. Ad un certo punto, quando il valore di un numero sufficiente di attributi è stato impostato, ritorneranno i servizi utili all'utente, cioè che soddisfano i vincoli a lui necessari. L'uso di SensoriExploration consente perciò di arrivare alla definizione dei vincoli attraverso l'interazione tra utente e sistema. Non è quindi necessario l'inserimento di tutti i vincoli da parte dell'utente attraverso la query scritta nel linguaggio di interrogazione.
Un ultimo contributo portato attraverso questa tesi riguarda la creazione dell'ontologia che contiene i dati e viene utilizzata anche nel progetto SeNSori. Questa è stata creata attraverso protege. In questa ontologia sono stati inseriti i concetti necessari espressi attraverso le classi. Per ciascuna classe sono poi state definite le relazioni esistenti con altre. Infine sono state inserite le istanze appartenenti alle varie classi, esprimendo le relazioni che hanno con altri concetti.

Capitolo 2

Stato dell'arte

In questo capitolo viene spiegato lo stato della ricerca riguardo i temi trattati nell'applicazione ServiceRetrieval sviluppata. Dal momento che questa può essere vista come composta da due sotto applicazioni, questo capitolo viene diviso in due. Nella prima parte viene spiegato lo stato della ricerca rispetto alla parte relativa allo sviluppo di Sensori_NL, quindi del recupero di istanze basandosi sull'inserimento di parole chiave da parte dell'utente. Nella seconda parte invece quello relativo all'applicazione SensoriExploration e cioè riguardante l'esplorazione.

2.1 Recupero di istanze basandosi sull'inserimento di parole chiave

L'inserimento di parole chiave per accedere ai campi di un database è un'area di ricerca nata con lo scopo di permettere un accesso semplificato ad esso rispetto all'utilizzo di SQL. Tanto più un database ha una struttura complessa o sconosciuta all'utente, maggiori sono i vantaggi di questo approccio in quanto non è richiesta la conoscenza della struttura del database e quindi non è necessario scrivere query complesse. Progetti come BANKS [16], DBXplorer [32], Discover [22], SQAK [40] e ObjectRank [20] utilizzano parole chiave inserite dall'utente per recuperare informazioni da un database. In questi progetti le informazioni recuperate sono solo quelle che si riferiscono a tutte le parole chiave inserite. Il problema nell'utilizzo di questo approccio è che l'utente potrebbe non conoscere tutte le parole chiave relative alle istanze di cui necessita,

cosa che limita l'efficacia del processo. La soluzione proposta dall'applicazione Sensori_NL, a differenza dei progetti sopra citati, utilizza un'ontologia invece di un database e risolve il problema perché recupera delle interrogazioni da porre all'ontologia, ordinandole in base alla rilevanza che queste hanno rispetto alle parole chiave: non è quindi necessario inserire tutte le parole chiave relative ad un'interrogazione, ma in base a quelle inserite vengono proposte certe interrogazioni.
Sensori_NL si basa inoltre sulla ricerca semantica [25, 39]. Si parla di ricerca semantica quando si combinano tecniche di recupero di informazioni con tecnologie semantiche. A questo riguardo, in [5] viene utilizzata una tassonomia di riferimenti attraverso la quale dopo l'inserimento di una parola chiave il sistema riesce a calcolare tutti i sinonimi. A questo punto il sistema esegue la ricerca sul database attraverso l'esecuzione di query predefinite utilizzando tutti i termini trovati. Il problema di questo approccio è che richiede l'esecuzione di più query SQL per poter eseguire l'interrogazione considerando tutti i termini. Ciò è dovuto al modello di memorizzazione utilizzato. L'ontologia viene perciò utilizzata come una semplice tassonomia. In questo contesto, un lavoro che più si avvicina a Sensori_NL è WebDB [38]. In esso le parole chiave inserite sono utilizzate per creare query congiuntive in SPARQL, cioè query che per esprimere condizioni non possono contenere operatori di comparazione se non l'uguaglianza e tali condizioni possono essere combinate solo attraverso l'operatore logico AND. Tra le query così generate l'utente selezionerà quella da eseguire sull'ontologia. Questo approccio presenta però lo stesso problema di ambiguità della generazione di query in SQL, essendo SPARQL una sua variante sintattica. Sensori_NL come quest'ultimo progetto propone all'utente la scelta della query da eseguire, ma le query proposte non vengono generate dalle parole chiave inserite, bensì sono già create e attraverso le parole chiave vengono recuperate quelle più rilevanti nel contesto.
Infine in Sensori_NL è necessario adottare un sistema di suggerimento di parole chiave: dal momento che la ricerca tramite parole chiave potrebbe portare incertezza all'utente in quanto egli potrebbe non conoscere le possibili parole da inserire, è necessario fornire un sistema di suggerimento. In [9] si propone che l'inserimento di parole chiave venga auto-completato controllando l'intero contenuto del database. Il problema di questa tecnica è che per essere efficiente richiede un elevato utilizzo di memoria. Per ovviare al problema riportato, in

Sensori_NL si controllano nell'ontologia solo i concetti relativi alle classi e alle relazioni, senza interessarsi delle istanze: l'inserimento per queste riguarderà valori che l'utente deve conoscere (ad esempio se l'utente vuole conoscere la temperatura di una stanza dovrà conoscere il nome della stanza di interesse).

2.2 Esplorazione

L'esplorazione è un'area di ricerca di recente nascita. Questa è stata affrontata dal professor Paolini e proposta nella tesi di dottorato di Spagnolo [35]. Attraverso l'esplorazione si vogliono ricavare informazioni e migliorare la loro comprensione quando queste sono presenti in grandi quantità. Comprendere le informazioni significa anche capire quali dati condividono gli stessi attributi e quali di questi sono rilevanti per scoprire correlazioni tra i dati. Il problema del ricavo di informazioni quando sono presenti in grandi quantità è oggi serio, dato che le informazioni sono in continua crescita. Attraverso l'esplorazione si vogliono estendere i tradizionali sistemi di recupero di informazioni in modo da risolvere il problema che, altrimenti, dato l'alto numero di condizioni che bisognerebbe esprimere per recuperare i dati potrebbe non portare ai risultati voluti. L'esplorazione permette ciò attraverso un'interazione continua tra utente e sistema, in questo modo il sistema di volta in volta filtra i risultati con le scelte dell'utente e quest'ultimo migliora la propria conoscenza dei dati e delle loro caratteristiche.

L'esplorazione cerca di categorizzare i dati basandosi non sulle definizioni estensionali ma su quelle intensionali: mentre le prime mostrano l'insieme di dati che soddisfano le condizioni, le ultime specificano le proprietà necessarie per far si che un oggetto appartenga ad un insieme. Tuttavia non è possibile definire tutte e sole le proprietà che caratterizzano un insieme, ma si possono approssimare. Questa approssimazione è comunque abbastanza potente per supportare l'utente in varie attività, come per esempio il recupero di informazioni su un dominio non noto ad esso. L'approssimazione richiede l'uso della statistica, in modo da trovare correlazioni tra i dati e scoprire nuova conoscenza.

Come detto, l'esplorazione si differenzia rispetto ai sistemi di recupero di informazioni tradizionali. In particolare, rispetto al data mining l'esplorazione si differenzia perché il data mining cerca di trovare correlazioni tra i dati che

possono migliorare la conoscenza dell'oggetto di interesse. Tuttavia la ricerca nel data mining è focalizzata principalmente sul trovare i migliori algoritmi per estrarre conoscenza, ma una volta utilizzata, questa conoscenza non viene più sfruttata. Alcuni progetti come [29, 13], utilizzano tecniche di data mining per estrarre conoscenza da utilizzare per descrivere in maniera intensionale gli insiemi in un modo approssimativo, con l'obiettivo di memorizzare gli apetti intensionali da usare per generare query successive. Tuttavia a differenza degli obiettivi dell'esplorazione, questi progetti non permettono di gestire un inserimento incrementale di parti di query o di confrontare le differenze tra diversi insiemi. Anche rispetto ai sistemi OLAP (On Line Analytical Processing) l'esplorazione si differenzia. I sistemi OLAP infatti sono una tecnica che permette di analizzare le informazioni per prendere decisioni attraverso l'uso di Data Warehouse [18]. In questi ogni query è indipendente dalle altre e un utente scrive la query per ricavare le informazioni con il tipo di granularità di cui necessita. A differenze dell'esplorazione non è perciò possibile analizzare i dati attraverso le proprietà intensionali e non è possibile costruire query in modo incrementale, partendo cioè dall'ultimo risultato ottenuto. Da queste considerazioni, si può notare che l'obiettivo che si pone l'esplorazione non è raggiungibile attraverso l'uso dei moderni sistemi di query e di analisi dei dati, ma l'esplorazione consente, partendo da alcuni dei metodi esistenti, di svilupparne altri per raggiungere gli scopi prefissati.
La principale tecnica per effettuare esplorazione è oggi la navigazione faceted [41]. In questa, ogni informazione è categorizzata in base a dei parametri: gli utenti possono filtrare le informazioni selezionando il valore dei parametri desiderati fino a raggiungere le informazioni necessarie. La ricerca faceted estende la navigazione attraverso l'uso di parole chiave nel sistema di recupero. L'utente selezionando i valori dei parametri può ottenere risultati sempre più specifici. L'output può essere costituito dall'insieme di informazioni che soddisfano le condizioni oppure da altri parametri da raffinare che possono essere combinati con quelli già selezionati per ridurre l'insieme delle risposte. Quest'ultimo tipo di output definisce la navigazione faceted come un metodo di esplorazione piuttosto che un metodo di recupero di istanze e può essere definito come paradigma interrogazione-risposta [43]. Diversi progetti sono nati riguardo l'aspetto della ricerca faceted, i quali si differenziano a seconda del modo con cui i dati strutturali e le query vengono sintatticamente e se-

manticamente modellati. La navigazione faceted di base combina con clausole congiuntive le diverse coppie parametro-valore e viene proposta in progetti come [36, 26]. Altri progetti, come [11], permettono anche la combinazione con clausole disgiuntive, ma solo se le coppie parametro-valore appartengono alla stessa proprietà. La tassonomia dinamica [34] e la Logical Information Systems (LIS) [15], permettono la combinazione dei parametri in diverse forme (come le congiunzioni, le disgiunzioni e le negazioni). gFacet [31], Visor [24] e Sewelis [33] (un'evoluzione di LIS) sono progetti che estendono la navigazione faceted al web semantico. Pur condividendo con la ricerca faceted l'obiettivo (migliorare il meccanismo di interazione in modo da poter costruire query più espressive per facilitare la ricerca delle informazioni necessarie all'utente), a differenza dei progetti basati su di essa, nell'applicazione SensoriExploration i parametri da raffinare vengono suggeriti all'utente, il quale inserisce il valore da assegnare voluto. In questo modo l'utente non deve cercare tra i vari parametri quello di suo interesse, ma semplicemente inserire il valore richiesto dal sistema. In questo modo si crea una specie di dialogo tra l'utente e il sistema.

2.3 Sommario

In questo capitolo è stato mostrato lo stato dell'arte relativo alle tecnologie che sono state utilizzate per sviluppare il progetto. In particolare è stato mostrato lo stato della ricerca rispetto al recupero di istanze basandosi sull'inserimento di parole chiave e sono stati mostrati dei progetti a tal riguardo spiegando in cosa differiscono dall'applicazione Sensori_NL. Successivamente è stato descritto il concetto di esplorazione indicando perché è considerato un aspetto di ricerca diverso dai metodi già esistenti. Quindi sono stati riportati dei progetti relativi all'esplorazione e anche per questi sono state spiegate le differenze rispetto all'applicazione SensoriExploration.

Capitolo 3

Strumenti

In questo capitolo vengono spiegate le varie tecnologie che sono servite per lo sviluppo dell'applicazione ServiceRetrieval. In particolare, nella prima parte vengono descritti gli aspetti più teorici, spiegando cosa sono le ontologie, cos'è il semantic web e come lo si può descrivere. Nella seconda parte vengono invece spiegati i principali strumenti utilizzati per lo sviluppo.

3.1 Fondamenti

La conoscenza è un aspetto critico per poter creare sistemi intelligenti e in molti casi avere una conoscenza di qualità è più importante che avere un buon algoritmo. Per avere sistemi intelligenti, la conoscenza deve venire catturata, processata, riutilizzata (una volta che è stata creata per un dominio è utilizzabile da diverse applicazioni sullo stesso) e scambiata. Le ontologie supportano queste operazioni. In un'ontologia la conoscenza è rappresentata attraverso la logica. Questa permette così di avere un servizio di reasoning: concetti non esplicitamente dichiarati possono essere comunque derivati attraverso regole di inferenza logica. La logica utilizzata prevalentemente per definire ontologie è la description logic [14]. Le ontologie permettono di rappresentare le informazioni in modo che siano capite da agenti. In questo modo si è sviluppato un nuovo modo di rappresentazione delle informazioni: il semantic web. Nel web è presente molta conoscenza. Questa è però fatta in modo che possa essere compresa da esseri umani. Per rendere la conoscenza comprensibile anche ad agenti è stato sviluppato il semantic web. Questi concetti vengono presentati in questa sezione.

3.1.1 Description Logic

Per poter essere processata da un sistema, un'ontologia ha bisogno di essere specificata formalmente (così come un programma è formalmente scritto attraverso un linguaggio di programmazione). Uno dei linguaggi più utilizzati per tal fine è OWL [30]. Il formalismo per lo sviluppo di OWL è la description logic (DL). Quando si deve sceglie un linguaggio per descrivere ontologie bisogna bilanciare espressività che questo consente (quanto riusciamo a descrivere ciò che ci serve) e proprietà computazionali (cercare di mantenere la decidibilità), cose che possono essere in contrasto. La description logic è una buona via di mezzo, essendo decidibile e abbastanza espressiva.
Sistemi di rappresentazione della conoscenza basati sulla DL sono costituiti da due componenti: TBox e ABox. TBox rappresenta la terminologia, cioè la rappresentazione di concetti (o classi, cioè un insiemi di individui) e ruoli (o proprietà, cioè relazioni tra concetti) dell'ontologia; ABox rappresenta le asserzioni tra individui.
La descrizione di ontologie in DL usa costrutti che hanno la semantica della logica predicativa. DL è costituita da diversi lingaggi, ognuno dei quali consente diverse operazioni. Il nome di un linguaggio è in realtà un acronimo: ogni lettera del nome del linguaggio sta ad indicare che quel linguaggio può fare una determinata operazione. La tabella 3.1 mostra il significato di ogni lettera, e usa le lettere A per indicare una classe atomica, C e D per classi arbitrarie, R e S per proprietà, a e b per individui.
Dalla tabella 3.1 si possono capire le differenze tra i tre linguaggi di DL maggiormente utilizzati: $SHIF(D)$, $SHOIN(D)$ e $SROIQ(D)$.
La DL è stata creata stando attenti al precesso di reasoning. Con reasoning si intende il processo di derivazione di fatti che non sono espressi nell'ontologia esplicitamente, ma che da questa possono venire dedotti. Alcune delle operazioni di reasoning sono: verificare la soddisfacilbilità di un concetto (se la sua descrizione non è contradditoria), verificare se un concetto è composto da un sotto insieme di un altro, recupero di individui.

3.1.2 Semantic web

Il semantic web è stato creato con lo scopo di migliorare il web, in modo che i computer possano processare, interpretare e collegare le informazioni qui

Codice	Espressività della DL corrispondente
AL	(i) assiomi di inclusione (C $\sqsubseteq$ D) e di equivalenza (C $\equiv$ D) di classi; (ii) classi atomiche A, classe universale $\top$ e classi complesse formate con i costrutti $C \sqcap D$, $\forall R.C$, $\exists R$; asserzioni $C(a)$, $R(a,b)$, $a = b$, $a \neq b$.
ALC	come AL, più la classe vuota $\bot$ e le classi complesse formate con i costruttori $\neg C$, $C \sqcup D$, $\exists R.C$.
S	come ALC, più la specifica di transitività delle proprietà $Tra(R)$.
H	assiomi di inclusione ($R \sqsubseteq S$) e di equivalenza $R \equiv S$ di proprietà.
R	come H più: (i) disgiunzione di proprietà, riflessività globale, irriflessività e asimmetria di proprietà; (ii) assiomi di sottoproprietà basati su catene; (iii) restrizione di riflessività locale $\exists R.Self$; (iv) asserzioni negative $\neg R(a,b)$.
O	nominali $\{a\}$ ($one\text{-}of$), restrizioni di valore $R \ni a$.
I	proprietà inversa R^-.
F	funzionalità di una proprietà: $\leq 1R$ o $Fun(R)$.
N	restrizioni di cardinalità non qualificate: $\leq nR$, $\geq nR$, $= nR$.
Q	restrizioni di cardinalità qualificate: $\leq nR.C$, $\geq nR.C$, $= nR.C$.
D_n o D	attributi con valori in domini di dati.

Tabella 3.1: Codice degli acronimi della DL

presenti per aiutare le persone a trovare ciò che cercano. Il semantic web ha l'obiettivo di creare una grande base di conoscenza distribuita, condividendo i dati invece dei documenti. La figura 3.1 mostra l'architettura del semantic web.
L'architettura è divisa in più livelli:

- URI (Uniform Resource Identifier) e UNICODE: URI è una stringa che permette di identificare in modo univoco una risorsa mentre UNICODE è lo standard per codificare i caratteri.

- XML (Extensible Markup Language) [37]: garantisce l'uso di una sintassi comune per il semantic web. XML è un linguaggio per documenti che contengono informazioni strutturate.

- RDF (Resource Description Framework) [28]: è un framework per rappre-

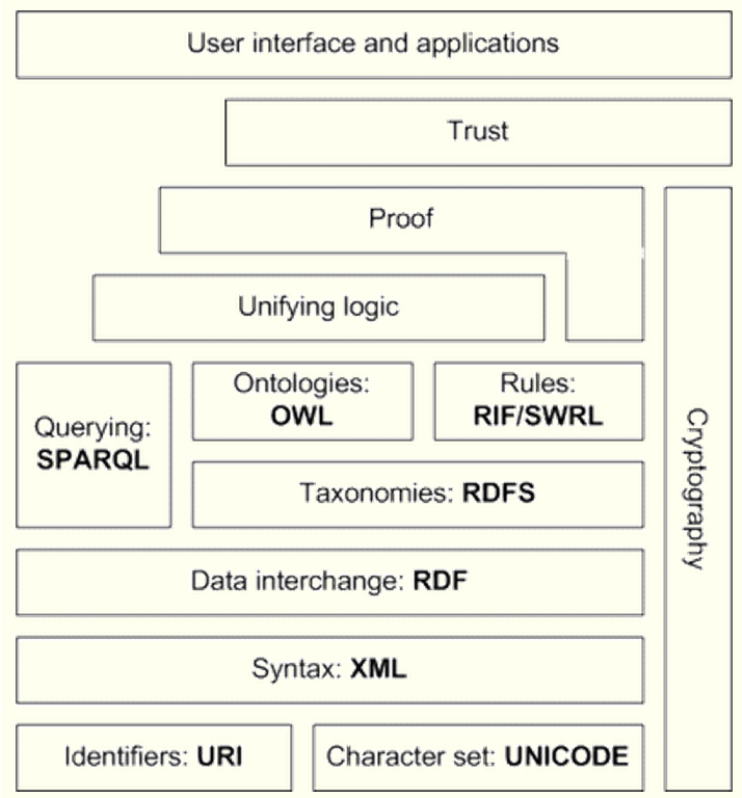


Figura 3.1: Architettura semantic web

sentare informazioni su risorse. Per risorsa si intende qualunque cosa può venire rappresentata attraverso un URI. RDF si basa su triple soggetto, predicato, oggetto per formare le relazioni tra le risorse. Tutti i dati del semantic web usano RDF come linguaggio primario di rappresentazione.

- RDFS (RDF Schema) [8]: RDFS estende RDF permettendo la descrizione di tassonomie di classi e proprietà.

- OWL (Ontology Web Language): è usato per creare ontologie più dettagliate. È un linguaggio che deriva dalla description logics. OWL è sintatticamente incorporato in RDF, quindi come RDFS offre un vocabolario standard esteso di RDF.

- RIF/SWRL (Rule Interchange Format/Semantic Web Rule Language) [27, 23]: servono per fornire regole al di sopra dei costrutti disponibili in OWL e RDFS.

- SPARQL(Simple Protocol and RDF Query Language) [21]: è un linguaggio che serve per interrogare ontologie RDF, RDFS e OWL.

- Trust, Proof: sono livelli che indicano che se l'input fornito è affidabile, lo saranno anche i risultati dedotti attraverso reasoning.
- Crittografia: è necessaria per avere input affidabili.

RDF

RDF è un framework per rappresentare informazioni di risorse. Le risorse sono identificate con un URI. Le informazioni sono rappresentate da triple soggetto-predicato-oggetto. Il soggetto indica la risorsa che si vuole decrivere, il predicato indica una relazione tra il soggetto e l'oggetto, l'oggetto può indicare o un valore o un altro elemento (che potrà essere un soggetto in altre relazioni) al quale il soggetto è legato dal predicato. La figura 3.2 mostra un esempio di tripla in forma grafica.

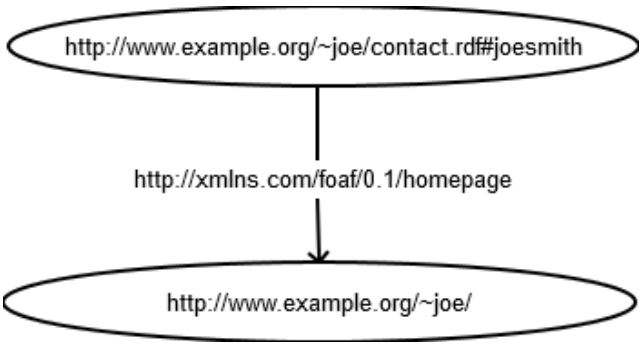


Figura 3.2: Esempio tripla RDF

L'esempio indica che il soggetto Joe Smith ha come homepage (che indica il predicato) l'oggetto http://www.example.org/∼joe. Si può notare che tutti gli elementi della tripla sono identificati da URI. L'oggetto può anche essere un literal, cioè un valore costante.
Il grafico RDF si forma unendo più triple. Oltre che in forma grafica RDF può anche essere espresso in forma sintattica. I linguaggi più utilizzati per tal fine sono RDF/XML [6], N3 [7] e TURTLE [10]. RDF/XML fa uso della sintassi del XML per descrivere un documento RDF, N3 descrive le triple per comporre il documento, TURTLE è un'alternativa a RDF/XML che non fa uso di XML ma di una sintassi più leggibile rappresentando le informazioni attraverso triple. La notazione di TURTLE è valida anche in N3, in quanto TURTLE è un suo sottoinsieme.

RDFS

RDFS estende il vocabolario di RDF per descrivere tassonomie di classi e proprietà. Inoltre estende anche alcune definizioni, per esempio permette di definire dominio e range di proprietà. Tutte le risorse possono essere divise in gruppi chiamati classi. Le classi essendo risorse sono sempre identificate da URI e possono essere descritte da proprietà. I membri della classe sono dette istanze o individui della classe. In RDFS una classe potrebbe anche essere un'istanza di un'altra classe. Le proprietà sono relazioni tra soggetti e oggetti nelle triple RDF, sono cioè i predicati. Le proprietà possono essere definite da dominio e range: il dominio indica che ogni risorsa che ha una certa proprietà è un'istanza della classe, il range indica che il valore di una proprietà è istanza di un'altra classe definita.

OWL

OWL estende RDF e RDFS. Il suo obiettivo primario è portare l'espressività e la potenza del reasoning della DL nel semantic web. Però non tutto ciò che può essere espresso in RDF può essere espresso in DL. Per questo OWL è solo un'estensione sintattica di RDF (mentre RDFS lo è anche semantica). Per risolvere parzialmente questo problema, sono stati definiti tre tipi di OWL:

- OWL Lite: può essere usato per descrivere tassonomie e semplici vincoli. È la forma di OWL più semplice e corrisponde alla DL SHIF.
- OWL DL: consente massima espressività conservando completezza e decidibilità. Questo corrisponde alla DL SHOIN ed è il linguaggio OWL più utilizzato.
- OWL Full: non ha vincoli espressivi, ma non garantisce la decidibilità.

I tre linguaggi possono essere visti a livelli: tutto ciò che può essere detto con OWL Lite può essere detto anche in OWL DL e tutto ciò che può essere detto in OWL DL può essere detto anche in OWL Full. Inoltre ogni ontologia OWL è un valido documento RDF, mentre non tutti i documenti RDF sono validi documenti OWL Lite e OWL DL.

SPARQL

SPARQL è un linguaggio SQL-like per fare interrogazioni su documenti RDF. SPARQL è composto da triple simili a RDF, ma qui ogni componente può essere una variabile, cioè non un elemento definito, ma il cui valore dipenderà dal risultato dell'interrogazione. La figura 3.3 mostra un semplice esempio di interrogazione SPARQL.

```
PREFIX foaf:<http://xmlns.com/foaf/0.1>
SELECT ?name ?mbox
WHERE {
        ?x foaf:name ?name.
        ?x foaf:mbox ?mbox.
}
```

Figura 3.3: Esempio interrogazione SPARQL

La prima riga definisce i prefissi, cioè dei riferimenti all'ontologia; la SELECT indica quali attributi visualizzare delle istanze che soddisfano la clausola WHERE. Le variabili sono precedute dal punto interrogativo. Nella query dell'esempio si stanno cercando *name* e *mbox* di tutte le risorse presenti nell'ontologia per le quali esistono tali proprietà.

3.2 Strumenti utilizzati

In questa sezione vengono introdotti alcuni dei principali strumenti usati per sviluppare l'applicazione ServiceRetrieval.

3.2.1 Protege

Protege [4] è un editor open source per creare e gestire ontologie. Le ontologie con protege possono essere sviluppate in diversi formati tra i quali RDF, RDFS e OWL. Protege organizza le ontologie in un insieme di classi, organizzate secondo una gerarchia. Ciascuna classe può essere associata ad un'altra classe o a valori costanti attraverso relazioni. Infine ogni classe può essere composta da più istanze che indicano gli individui di una classe. Anche ogni individuo può avere relazioni con altri individui o con valori. Protege include una API java che rende possibile utilizzare l'ontologia con altre applicazioni. La figura 3.4 mostra un esempio di ontologia svilluppata in Protege.
Nella figura si vede che protege contiene diverse schede, ognuna consente di

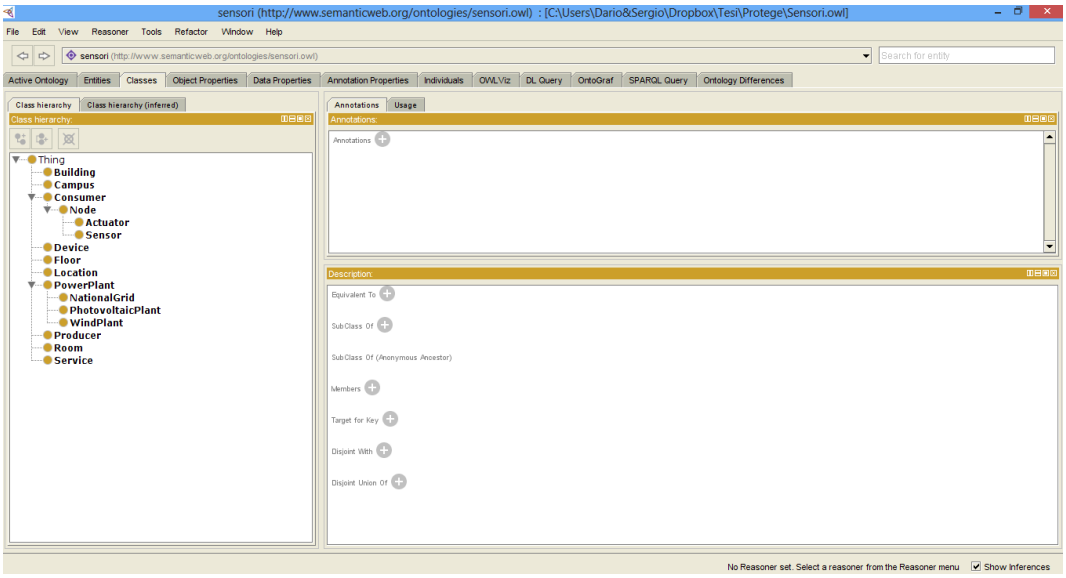


Figura 3.4: Esempio ontologia creata con protege

gestire qualcosa: classi, relazioni con oggetti o valori, individui, interrogazioni SPARQL. Protege include anche vari reasoner, che data l'ontologia permettono di ricavare assunzioni logiche non esplicitamente specificate.

3.2.2 Jena

Jena [1] è un framework java open source creato per lo sviluppo di applicazioni orientate al web semantico. Jena include un API per gestire documenti RDF, RDFS e OWL, un sistema di reasoning, un motore di interrogazioni SPARQL. La figura 3.5 mostra l'architettura di Jena.
Documenti RDF vengono acceduti attraverso l'API Jena RDF. I grafici RDF sono memorizzati internamente attraverso una semplice astrazione. Questo permette a Jena di usare diversi sistemi di memorizzazione equivalenti: come in memoria, in un database SQL, ho in uno storage persistente usando un indice di tuple. La caratteristica chiave del semantic web è l'uso del reasoner. Jena memorizza le inferenze come se queste fossero state aggiunte esplicitamente. Un'interfaccia API fornisce un motore di regole per svolgere questo lavoro. In alternativa l'interfaccia API può essere connessa con un reasoner esterno. Jena permette di fare interrogazioni sull'ontologia utilizzando SPARQL.

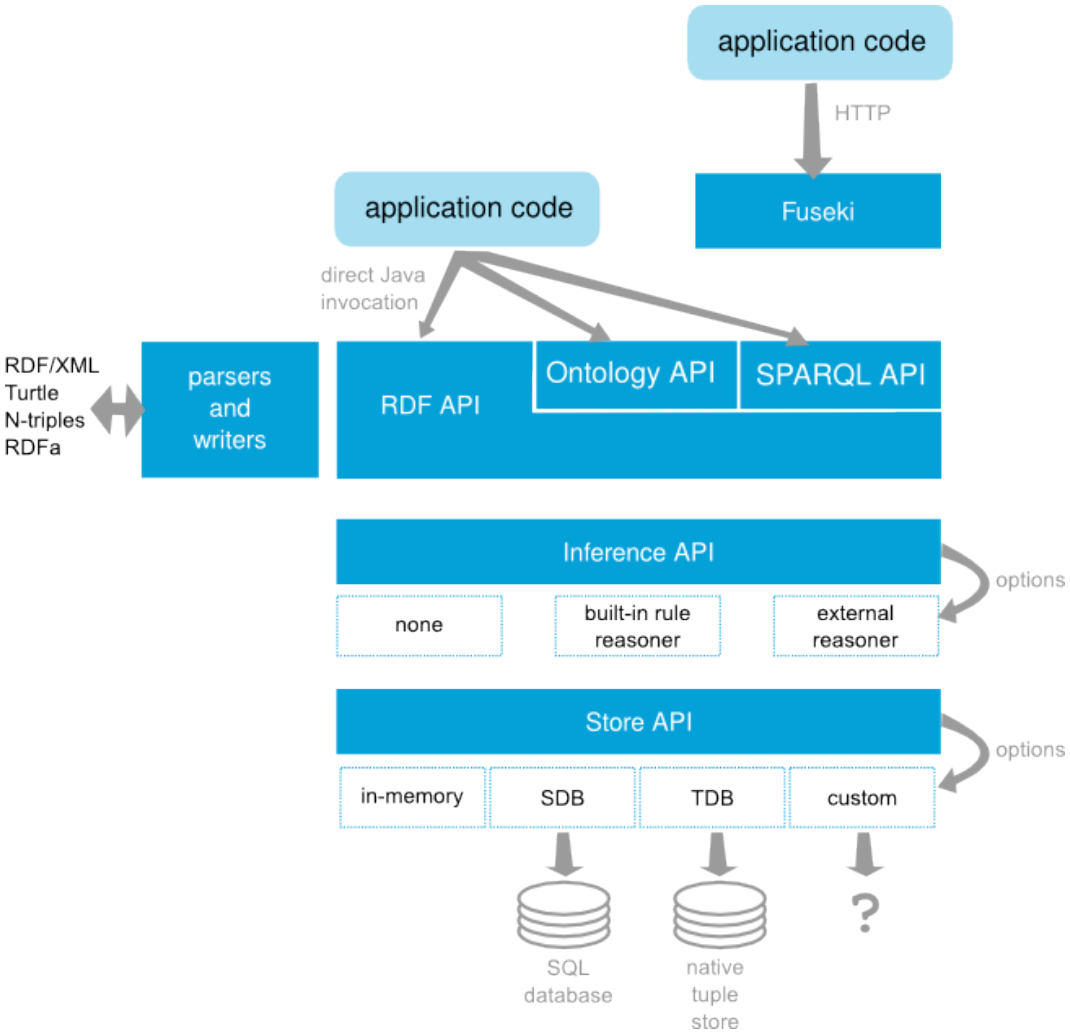


Figura 3.5: Architettura Jena

3.2.3 Postgres

Postgres [3] è un sistema open source per gestire database relazionali ad oggetti. Questo garantisce tutte le proprietà ACIDE (atomicità, consistenza, isolamento, durabilità o persistenza) che devono avere le transazioni. Postgres come la maggior parte dei database usa il linguaggio SQL e conserva i dati su tabelle con chiavi esterne per collegare i dati. Il principale vantaggio rispetto agli altri database è che rende più semplice costruire applicazioni grazie alla sua programmabilità. Postgres permette agli utenti di definire nuovi tipi basati sui normali tipi di dato SQL, permettendo al database stesso di comprendere dati complessi. Permette inoltre l'ereditarietà dei tipi. In Postgres i programmatori possono implementare la logica in uno dei molti linguaggi supportati: PL/pgSQL (linguaggio nativo simile a PL/SQL di Oracle), Wrapper

per linguaggi di scripting come Perl, Python e Ruby, C e C++, R. Il programmatore può inserire il codice sul server come funzioni che rendono il codice riusabile come stored procedure in modo che il codice SQL possa richiamare funzioni scritte in altri linguaggi. I suoi punti di forza sono: incremento delle prestazioni, incremento dell'affidabilità e codice del client più semplice.

3.3 Sommario

In questo capitolo sono state introdotte le principali tecnologie che sono state utilizzate per sviluppare il lavoro. È stato descritto cosa sono le ontologie e il semantic web e quali sono i linguaggi più utilizzati per descriverli; a cosa servono Protege e Jena e perché Postgres è il database utilizzato.

Capitolo 4

Progetto SeNSori

In questo capitolo viene spiegato in cosa consiste il progetto SeNSori, progetto del quale l'applicazione ServiceRetrieval si occupa del recupero di servizi. In particolare viene spiegata la parte che ha richiesto lo sviluppo dell'applicazione ServiceRetrieval.

4.1 Introduzione

Il progetto SeNSori è stato sviluppato con lo scopo di creare una piattaforma per monitorare ed ottimizzare il consumo energetico, inteso in ogni sua possibile forma (acqua, gas, elettricità), in un ambiente. L'ottimizzazione deve avvenire senza creare danno alle persone: non si può ad esempio spegnere un impianto di riscaldamento se poi la persona che ne faceva uso prova freddo. Per raggiungere questo scopo, è necessario installare nell'ambiente dei sensori e degli attuatori: i primi consentono di monitorare i consumi insieme ad altre informazioni riguardanti l'ambiente (come la temperatura); i secondi di operare modifiche sui dispositivi installati (ad esempio aumentare la potenza dell'impianto di riscaldamento se la temperatura misurata è troppo bassa). Il continuo monitoraggio dell'ambiente, ottenuto tramite i sensori installati, consente inoltre di rilevare le anomalie, che possono avvenire a causa di cambiamenti ambientali o strutturali (ad esempio la rottura di un dispositivo).
I sensori installati possono essere costituiti da diverse tecnologie. Per tale ragione l'utente non deve interagire direttamente con questi, altrimenti occorrerebbe implementare un meccanismo ad hoc per ogni tecnologia esistente, ma viene creato un livello di servizi. Questo livello espone dei metodi per gestire

le informazioni raccolte dai sensori e le operazioni da eseguire attraverso gli attuatori che gli utenti possono invocare. Per controllare correttamente gli attuatori, in modo cioè da non creare problemi alle persone presenti nell'ambiente monitorato, è necessario l'uso di tecniche di data mining e di data prediction per l'analisi dei dati provenienti dai sensori. I servizi sono descritti in modo che sia possibile recuperarli, quando necessario, attraverso un meccanismo di ricerca.

4.2 Architettura

L'architettura della piattaforma deve essere progettata tenendo in considerazione che:

- deve essere flessibile, cioè l'installazione nell'ambiente deve essere semplice e scalabile;
- deve consentire il monitoraggio del contesto operativo e adattarlo ai requisiti degli utenti se questi cambiano nel tempo;
- deve prevedere un meccanismo di auto riparazione, cioè deve riconoscere e risolvere possibili malfunzionamenti.

La figura 4.1 mostra l'architettura del progetto, pensata per consentire quanto appena indicato.
Il pannello di controllo (dashboard nella figura) è fornito come Software as a Service (SaaS), quindi ad ogni diverso tipo di utente, ne viene fornita una versione personalizzata. Il pannello di controllo è utilizzato dall'utente per compiere le azioni necessarie a ridurre il consumo energetico (quindi compreso il recupero e l'analisi dei dati rilevati dai sensori). I dati gestiti dalla dashboard vengono forniti da un insieme di nodi. Un nodo raggruppa logicamente e controlla un insieme di reti di sensori (ad esempio le reti di sensori di un edificio). In una rete di sensori sono presenti sia i sensori che gli attuatori installati. Per rispondere al requisito di flessibilità, la rete di sensori può venire estesa o ridotta a seconda delle esigenze.
Il livello SaaS, che è composto da un insieme di servizi REST, fornisce un livello di astrazione. I servizi REST, possono essere invocati da remoto per

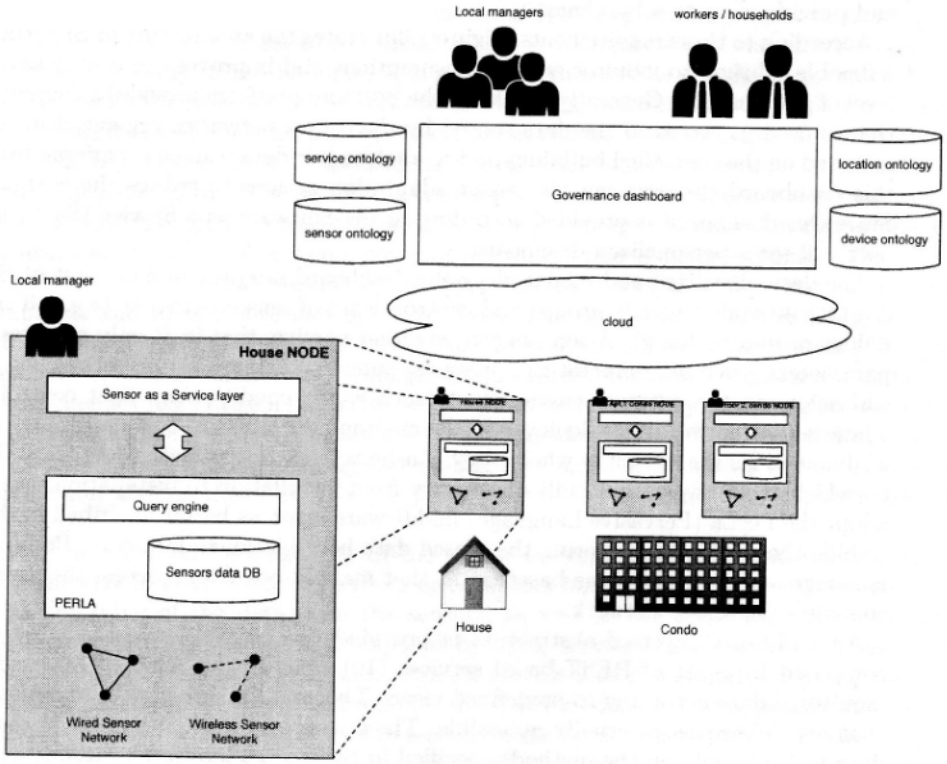


Figura 4.1: Architettura SeNSori

ottenere i dati monitorati in accordo con una vista predefinita. Quest'ultima permette l'esposizione dei soli dati definiti come esternamente accessibili. Questa caratteristica fa si che chi invoca i servizi per ottenere i dati, non deve necessariamente conoscere il modello dei dati e la tecnologia adottata per memorizzarli, ma fa affidamento solo sui metodi esposti nell'interfaccia dei servizi.

Ogni nodo viene gestito da un particolare utente: il manager. Il manager ha la possilbilità di riconfigurare i sensori appartenenti al nodo da lui gestito, decidere quali servizi il livello SaaS offre riguardo il nodo di suo interesse, invocare servizi offerti dal livello SaaS di altri nodi esposti da altri manager. Da notare che ogni nodo gestisce diverse reti di sensori, quindi i sensori installati sono gestiti da diversi manager. I servizi resi disponibili dai nodi, sono utilizzati dalla piattaforma per analizzare e ottimizzare il consumo di energia su un ambiente ampio, attraverso l'uso di servizi avanzati. Per esempio il manager analizzando il consumo storico di energia (possilbile grazie a questi servizi avanzati) può suggerire agli utenti una configurazione che ottimizzi il consumo: ad esempio, calcolando il consumo energetico in un edificio simile a

quello da lui gestito può capire se esistono delle inefficienze.
I servizi che il manager può selezionare, sono prelevati da un registro di servizi. Attraverso il pannello di controllo, il manager può comporre e visualizzare i dati provenienti da questi servizi. Oltre al manager, anche gli utenti che operano nell'ambiente possono accedere al pannello di controllo, ma per questi ne viene fornita una versione con limitate funzionalità che consente di conoscere le informazioni riguardo il consumo energetico.
Nella figura 4.1, accanto al pannello di controllo sono rappresentate quattro ontologie. Queste contengono i concetti principali riguardanti il dominio dell'applicazione: i servizi offerti, i sensori e gli attuatori installati, il consumo dei dispositivi presenti, informazioni riguardo la posizione in cui sia i dispositivi che i sensori sono installati. Attraverso le ontologie, la piattaforma offre al manager un meccanismo per recuperare i servizi di cui necessita per compiere determinate operazioni. In particolare, il pannello di controllo offre al manager tre funzionalità principali:

- **Pubblicazione:** i servizi vengono descritti da un'ontologia che include informazioni riguardo il contesto nel quale i servizi operano, la natura dei dati coinvolti e i dettagli tecnici riguardo a come invocarli.

- **Ricerca:** il registro dei sevizi supporta ricerche che restituiscono un insieme di servizi che soddisfano i vincoli espressi in queste. In particolare, il processo fornisce una lista ordinata di risultati dove più un servizio soddisfa i requisiti inseriti e prima compare in tale lista. In questo modo se non esiste un servizio capace di rispondere esattamente all'interrogazione, viene fornita una lista di possibili approssimazioni.

- **Analisi dei dati:** i dati provenienti dai servizi possono essere analizzati utilizzando tecniche di data mining.

4.3 Ontologie

Come indicato precedentemente, è possibile interagire con i sensori e gli attuatori installati utilizzando un'interfaccia comune che maschera le differenti tecnologie. È inoltre possibile accedere a queste funzionalità da remoto. Queste funzioni sono rese possibili grazie alla definizione del livello Sensor as a

Service. In questo livello, un servizio viene mappato su un sensore, quindi un servizio offre le funzioni che possono essere ottenute dal sensore corrispondente. I manager accedono così ai servizi connessi ai sensori, invece che a questi ultimi direttamente. In questo modo si mascherano le diverse tecnologie dei sensori. Siccome in un ambiente può essere installato un alto numero di sensori ed attuatori, è necessario fornire un meccanismo di recupero dei servizi in base a ciò di cui l'utente necessita. Di conseguenza, un servizio deve possedere una descrizione nella quale includere le informazioni che il manager può indicare per richiamare il servizio necessario. In particolare le informazione che è necessario indicare sono:

- **Sensore:** ogni servizio è collegato ad un sensore (o attuatore) ed è necessario sapere le loro caratteristiche.
- **Posizione:** indica la posizione in cui un sensore è installato. Nota la posizione è possibile rispondere alle richieste dell'utente basandoci solo su alcuni sensori.
- **Dispositivo:** è necessario avere informazioni riguardo i dispositivi che consumano o producono energia perché l'obiettivo della piattaforma è ottimizzarli.

Per supportare la definizione e la gestione di tali informazioni, la piattaforma SeNSori comprende un'ontologia come riportato in figura 4.2.
Per rappresentare i servizi viene estesa l'ontologia OWL-S [12] con caratteristiche riferite al dominio dell'applicazione. OWL-S è un'ontologia del web semantico per descrivere servizi. Per l'obiettivo del progetto, all'ontologia OWL-S vengono aggiunte informazioni riguardo i sensori connessi ai servizi: se un servizio è associato ad un sensore, allora il servizio consente all'utente di accedere alle informazioni misurate dal sensore attraverso un'interfaccia (relazione *dataFrom* nella figura). Altrimenti, se il servizio è associato ad un attuatore, deve permettere all'utente di configurarlo (relazione *configure*). Inoltre, in accordo con la specifica OWL-S, ogni servizio è costituito anche dalle operazioni che può invocare. In questo modo, è possibile sia descrivere la struttura dei servizi che ricavare nuova conoscenza basandosi sui dati disponibili. Tutti i servizi sono memorizzati in un file OWL, essendo questi un'istanza dell'ontologia. Quindi, data un'interrogazione generica, è possibile rispondere

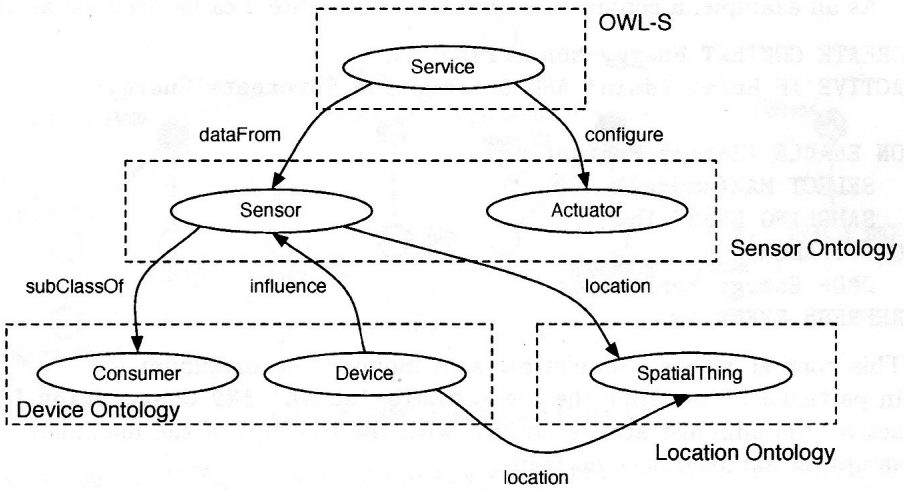


Figura 4.2: Ontologia della piattaforma SeNSori

a questa offrendo l'insieme dei servizi che soddisfano la richiesta.
Anche i sensori sono descritti da un'ontologia che rappresenta sia i sensori che gli attuatori installati nell'ambiente. Inoltre siccome è importante conoscere le posizioni in cui gli elementi sono installati e quali dispositivi sono presenti, vengono definite un'ontologia di posizioni e una di dispositivi. Grazie a queste aggiunte, viene reso possibile arricchire le interrogazioni per il recupero dei servizi inserendo questi aspetti. Di seguito vengono descritte queste tre ontologie, collegate a quella dei servizi dalle relazioni riportate nella figura 4.2.

4.3.1 Ontologia delle posizioni

L'ontologia delle posizioni definisce la posizione nella quale i vari sensori vengono installati. Per questa ragione, l'ontologia deve considerare sia concetti riferiti a caratteristiche di un edificio, sia coordinate geografiche (latitudine, longitudine e altitudine). Definendo questa ontologia è possibile avere informazioni riguardo una specifica posizione ma anche informazioni aggregate di un'ampia area. Inoltre, quando possibile, permette di approsimare la risposta a certe interrogazioni: ad esempio un utente potrebbe essere interessato alla temperatura di una stanza di un edificio, ma in tale stanza potrebbe non esserci il sensore di temperatura, quindi vengono forniti i dati relativi alla temperatura di stanze vicine, magari facendo la media di queste. Per queste ragioni, viene

adottata l'ontologia SOUPA [19]. Questa è rappresentata in figura 4.3.

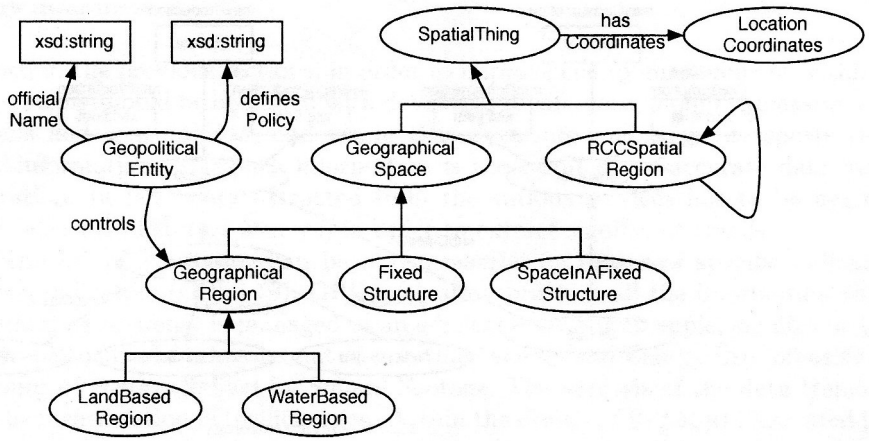


Figura 4.3: Ontologia delle posizioni

SOUPA usa un approccio top-down: descrive concetti generali che vengono poi divisi in concetti più specifici. L'obiettivo di questa ontologia è supportare il reasoning riguardo relazioni spaziali tra varie zone.

4.3.2 Ontologia dei dispositivi

Dal momento che è necessario ottimizzare il consumo di energia, è necessario rappresentare i dispositivi che consumano o producono energia. La figura 4.4 mostra l'ontologia per tale rappresentazione.
Dalla figura 4.4, si nota che l'ontologia è rappresentata da due concetti principali: il consumo e la produzione di energia dei dispositivi. Inizialmente i dati riguardanti il consumo (consumo minimo, massimo e medio) sono presi dalle informazioni tecniche del dispositivo. Questi vengono poi modificati monitorando il dispositivo quando viene installato nell'ambiente. Attraverso queste informazioni si possono filtrare i dispositivi secondo il loro consumo. Inoltre la conoscenza dei consumi permette di trovare anomalie: se, per esempio, certi elementi hanno un consumo massimo di energia molto superiore rispetto a quello medio abbiamo probabilmente un'anomalia. Lo stesso discorso può farsi per la produzione, qui però è importante indicare anche l'origine dell'energia (locale o fornita con tecniche tradizionali).
Un sensore può anch'esso essere definito come consumatore, caratterizzandolo

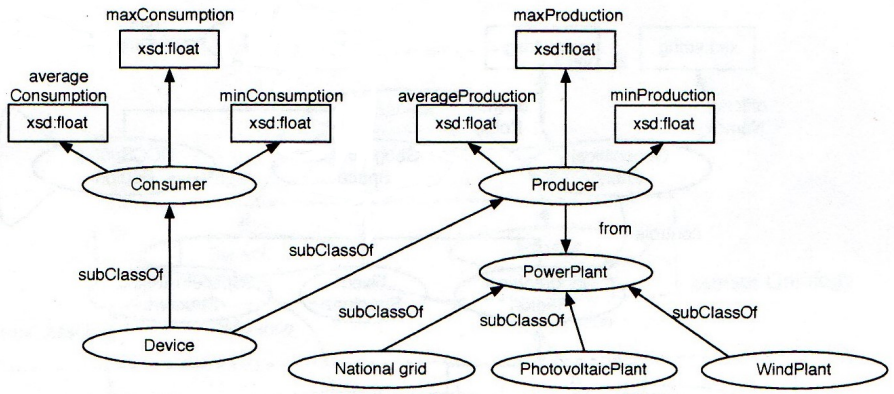


Figure 5. Device ontology.

Figura 4.4: Ontologia dei dispositivi

quindi anche in termini di consumo. Così è possibile ottimizzare la distribuzione di sensori in certi ambienti migliorando il salvataggio di energia senza tuttavia influire sul sistema di monitoraggio. Attraverso la relazione *influence* di figura 4.2, l'ontologia memorizza se un sensore è influenzato dalla variazione del funzionamento di un dispositivo. Così è possibile capire se un dispositivo è rotto verificando che, se si modifica il suo stato, i sensori che dovrebbero esserne influenzati cambiano effettivamente la rilevazione.

4.3.3 Ontologia dei sensori

I servizi permettono di recuperare dati rilevati dai sensori. Conoscendo il sensore che ha generato i dati è possibile arricchire la descrizione di un servizio con le informazioni relative ad esso. È così possibile rispondere ad interrogazioni che richiedono particolari misure (come ad esempio la temperatura o la pressione). Ogni sensore viene perciò caratterizzato in termini di categoria e tipo. La categoria identifica la categoria di consumo che il sensore rileva (acqua, corrente e gas); il tipo permette di specificare meglio il sensore stesso. Lo stesso ragionamento viene utilizzato per descrivere gli attuatori. In questo caso però, non parliamo di controllo del consumo, ma di elementi che lo fanno variare. La figura 4.5 mostra l'ontologia dei sensori.

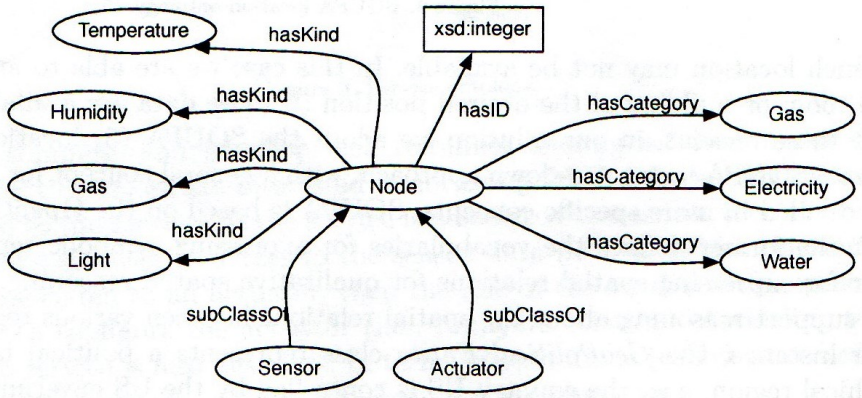


Figura 4.5: Ontologia dei sensori

4.4 Sommario

In questo capitolo è stato presentato il progetto SeNSori, progetto che ha lo scopo di ottimizzare il consumo energetico in un ambiente. È stata descritta l'architettura del progetto, spiegando che nell'ambiente sono installati sensori e attuatori, organizzati in reti. Il risparmio energetico è raggiunto analizzando dapprima i dati provvenienti dai sensori per cercare situazioni che riflettono gli sprechi, quindi compiendo azioni adattive che operano sugli attuatori. È stato mostrato che, per risolvere il problema delle eterogeneità tecnologiche, l'utente interagisce con un livello software anziché con i sensori stessi. Sono state infine presentate le ontologie create. Queste contengono le informazioni per recuperare i servizi necessari per rispondere alle interrogazioni dell'utente.

Capitolo 5

Interrogazioni basate su linguaggio naturale e parole chiave: Sensori_NL

In questo capitolo viene spiegata come è stata sviluppata la prima parte del lavoro, quella riguardante la traduzione di query da linguaggio naturale a linguaggio SPARQL. Queste verranno poi eseguite sull'ontologia per ricavare i risultati. È importante lo sviluppo di questa applicazione per recuperare i servizi di cui gli utenti necessitano: molti utenti che avranno a che fare con il recupero dei servizi infatti, non conosceranno perfettamente come è costituita l'ontologia, tanto meno non sapranno il linguaggio SPARQL, cose necessarie per l'operazione di recupero attraverso le normali interrogazioni. L'applicazione Sensori_NL risolve il problema consentendo la ricerca dei servizi in modo che gli utenti inseriscano delle informazioni (parole chiave) relative al servizio che vogliono recuperare. In base a queste, l'applicazione propone una serie di query descritte in linguaggio naturale e ordinate in base alla rilevanza rispetto alle informazioni inserite, in modo che l'utente possa selezionare quelle utili. A questo punto l'applicazione le eseguirà sull'ontologia, incaricandosi della traduzione nel linguaggio necessario.
Questo capitolo viene così strutturato: nella prima parte viene descritto il progetto SAFE, che include un'applicazione per tradurre query da linguaggio naturale, tuttavia questa presenta alcune caratteristiche che non la rendono applicabile al dominio di interesse. Si passa, nelle sezioni successive, a descri-

vere l'applicazione sviluppata, mostrandone i componenti di cui è composta: l'ontologia del dominio SeNSori, il file xml contentente le corrispondenze tra le query e l'implementazione dell'applicazione. Infine vengono mostrati degli esempi che mostrano il funzionamento dell'applicazione

5.1 SAFE

SAFE (Sistema per l'Anamnesi in Fase di Emergenza) è un progetto sviluppato per supportare medici e personale sanitario durante le situazioni di emergenza, provvedendo le informazioni necessarie per gestirle. Proprio per il fatto che deve funzionare in situazioni di emergenza, l'obiettivo di SAFE è quello di recuperare i dati corretti il più velocemente possibile. La figura 5.1 mostra l'architettura dell'applicazione.

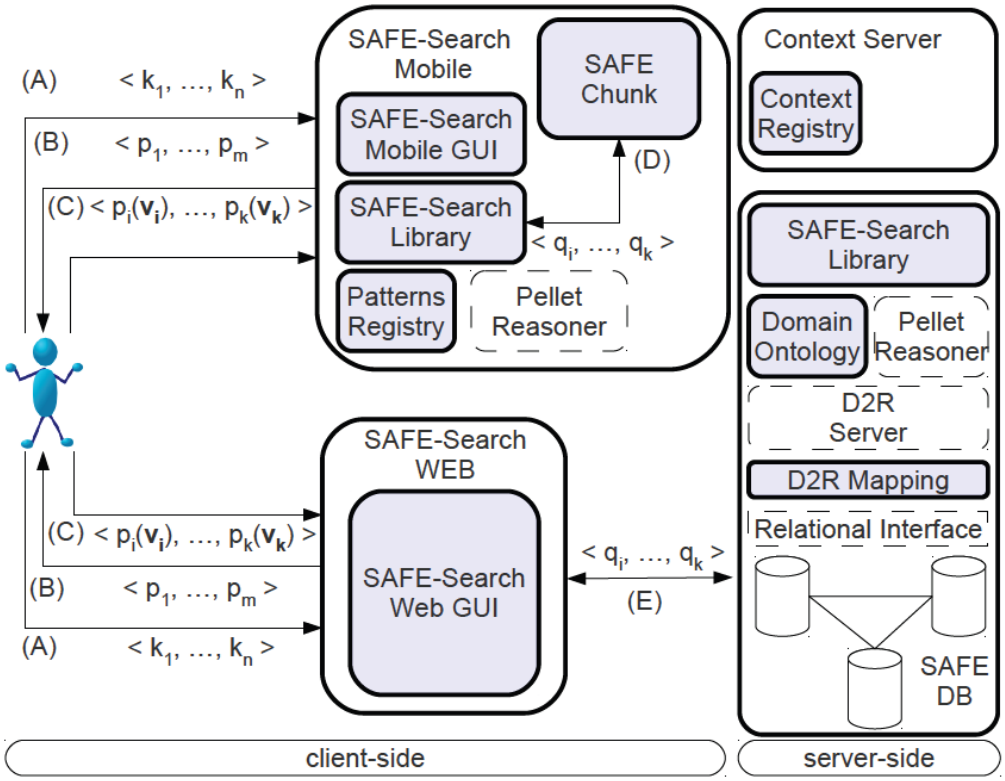


Figura 5.1: Architettura SAFE

Dato un insieme di parole chiave inserite dagli utenti, l'applicazione suggerisce un insieme di interrogazioni. Queste sono presentate in linguaggio naturale e ordinate secondo la rilevanza, basandosi sulle parole chiave inserite. L'utente

seleziona le interrogazioni proposte di suo interesse. Queste vengono poi tradotte in linguaggio SPARQL dall'applicazione, la quale usa poi D2R Server [2] per eseguire l'interrogazione sul database relazionale sul quale sono presenti i dati. SAFE può funzionare anche su device mobili. Per questi, viene memorizzata una piccola parte dei dati basandosi sul contesto di interesse dell'utente, ricavato attraverso una smart card.

5.1.1 Motivazioni

I motivi per i quali è stata sviluppata l'applicazione Sensori_NL e non è stata utilizzata SAFE sono diversi:

- Mentre in SAFE i dati sono presenti in un database relazionale, nel progetto SeNSori i dati sono presenti in un'ontologia, quindi le interrogazioni SPARQL devono venire eseguite direttamente su questa, quindi senza bisogno di utilizzare D2R Server.

- In SAFE esistevano ontologie per ricercare parole chiave, in Sensori_NL un'unica ontologia è usata sia per trovare i suggerimenti durante l'immisione di parole chiave sia per restituire i risultati dell'interrogazione.

- In SAFE per capire se una parola chiave indica un valore da assegnare ad un attributo (cioè una variabile) anziché un attributo, viene usata la distanza Jaro-Winkler [44]: se la distanza tra una keyword inserita e la parola più vicina presente nell'ontologia è maggiore di una soglia, la keyword è considerata una variabile. Questo approccio non è fattibile nel dominio SeNSori perché si possono presentare parole chiave che vogliono essere variabili ma vengono scambiate per attributi. Ciò dipende dell'alta similarità tra i nomi delle variabili e quello degli attributi. Perciò è stato usato un confronto più semplice: se la parola digitata dall'utente non è presente nell'ontologia come classe, relazione o come particolare annotazione, allora viene considerata una variabile. Attraverso l'utilizzo di un sistema di suggerimento di parole chiave si preserva la significatività perché l'utente capisce quali parole indicano gli attributi.

- L'ordine con cui vengono presentate le query in linguaggio naturale è stato modificato: a differenza di SAFE in cui è importante la velocità di

risposta, in sensori_NL è stata privilegiata la precisione, quindi maggiore è il numero di parole chiave che fanno riferimento ad una query, prima tale query si presenterà.

- In SAFE mancano alcune funzionalità che possono risultare utili: la possibilità di inserire query direttamente nel linguaggio di interrogazione, la possibilità di inserire e rimuovere pattern direttamente dall'applicazione, una lista di suggerimenti di variabili significative che possono venire inserite nella query selezionata, la possiblità di rimuovere qualsiasi keyword inserita.

5.2 Infrastruttura software

L'obiettivo principale è la creazione di un'applicazione, chiamata Sensori_NL, che permette ad un utente che non conosce in maniera abbastanza approfondita il linguaggio SPARQL da utilizzare per interrogare l'ontologia, di ricavare le informazioni di cui ha bisogno. Per far questo l'applicazione mostra una serie di interrogazioni scritte in linguaggio naturale ricavate da parole chiave inserite dall'utente. Per parola chiave si intende una parola che indica cosa sta cercando l'utente: ad esempio se l'utente vuole ricavare i servizi della Stanza14, possibili parole chiave possono essere servizi, stanza, Stanza14. Tra le interrogazioni mostrate, l'utente selezionerà quella di suo interesse. A questo punto l'applicazione tradurrà la query da linguaggio naturale in linguaggio SPARQL e la eseguirà sull'ontologia, mostrando infine i risultati.
Di seguito viene descritta l'architettura, spiegando i componenti di cui è composta e le operazioni che vengono eseguite dall'applicazione per arrivare al risultato voluto.

5.2.1 Architettura

La figura 5.2 mostra l'architettura dell'applicazione Sensori_NL.
La figura 5.2 riporta anche la sequenza con cui le varie operazioni vengono compiute:

1. L'applicazione Sensori_NL apre il file patternDB.xml che contiene le corrispondenze tra query in linguaggio naturale e query SPARQL. Carica

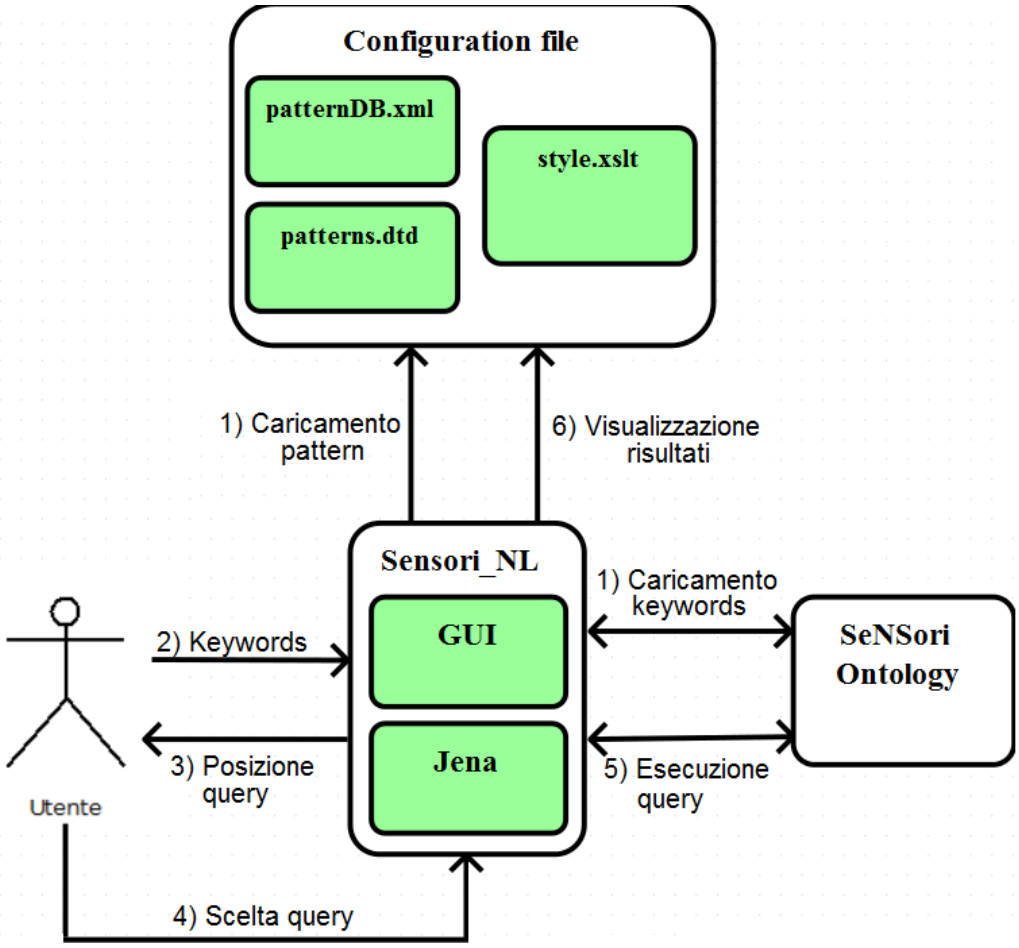


Figura 5.2: Architettura applicazione Sensori_NL

quindi le corrispondenze in una sua classe interna e successivamente le parole chiave dall'ontologia.

2. L'utente inserisce le parole chiave nell'applicazione.

3. L'applicazione ritorna una lista contenente delle interrogazioni descritte in linguaggio naturale, ordinate in base alla rilevanza che hanno sulle parole chiave inserite dall'utente al passo precedente. Queste interrogazioni permettono di recuperare i servizi utili agli utenti.

4. L'utente seleziona le interrogazioni di suo interesse.

5. Le interrogazioni selezionate vengono tradotte dall'applicazione in linguaggio SPARQL ed eseguite sull'ontologia.

6. L'applicazione utilizza il file style.xslt per mostrare i risultati dell'interrogazione

Questi passi vengono spiegati in maniera più dettagliata successivamente.

5.2.2 Ontologia

La prima cosa che è stata fatta è stata la creazione dell'ontologia e l'inserimento delle istanze. L'ontologia è stata creata a partire dalle specifiche ontologie rappresentate nel capitolo precedente. Per gestire l'ontologia è stato usato Protege. All'ontologia è stato dato l'URI: *http://www.semanticweb.org/ontologies/2013/1/SeNSori-Ontology#*. Oltre alle classi, alle proprietà e alle istanze, l'ontologia contiene un'annotazione appositamente creata chiamata *idQuery*. Questa può essere messa come annotazione di ogni classe o proprietà (sia di tipo oggetto che di tipo data), e ognuna di queste ne può contenere anche più di una oppure nessuna. Questa annotazione contiene un riferimento ad una query nel file patternDB.xml. Questo riferimento consente di legare ogni parola chiave che verrà inserita dall'utente ad una o più query: se ad esempio l'utente inserisce la parola chiave "Room" si guarderà nell'ontologia la classe Room quali query referenzia, in modo che una volta inserite tutte le parole chiave si riesca a decidere il ranking delle query. È stata usata anche l'annotazione già esistente *label* con lo scopo di indicare tutti i sinonimi: così se l'utente inserisce al posto di "Room" la parola "Stanza" (per esempio perché è italiano), l'applicazione capisce comunque che è una parola chiave e compie le dovute azioni. La figura 5.3 mostra le annotazioni appena descritte per la classe "Room".

5.2.3 patternDB.xml

Il file patternDB.xml contiene tutte le corrispondenze tra le query in linguaggio naturale mostrate all'utente e le query in linguaggio SPARQL che verranno eseguite sull'ontologia. La figura 5.4 mostra un esempio di come è costituito il file.

Nell'esempio, l'interrogazione permette di recuperare i servizi ed i sensori legati a questi che sono della categoria inserita dall'utente in fase di selezione (per questo indicata col campo var, che indica appunto i riferimenti che l'utente deve inserire) e che sono installati nella stanza che verrà sempre inserita dall'utente al momento della selezione. Come si può intuire, questa è la rappresentazione in linguaggio naturale alla quale è poco dopo indicata quella in linguaggio

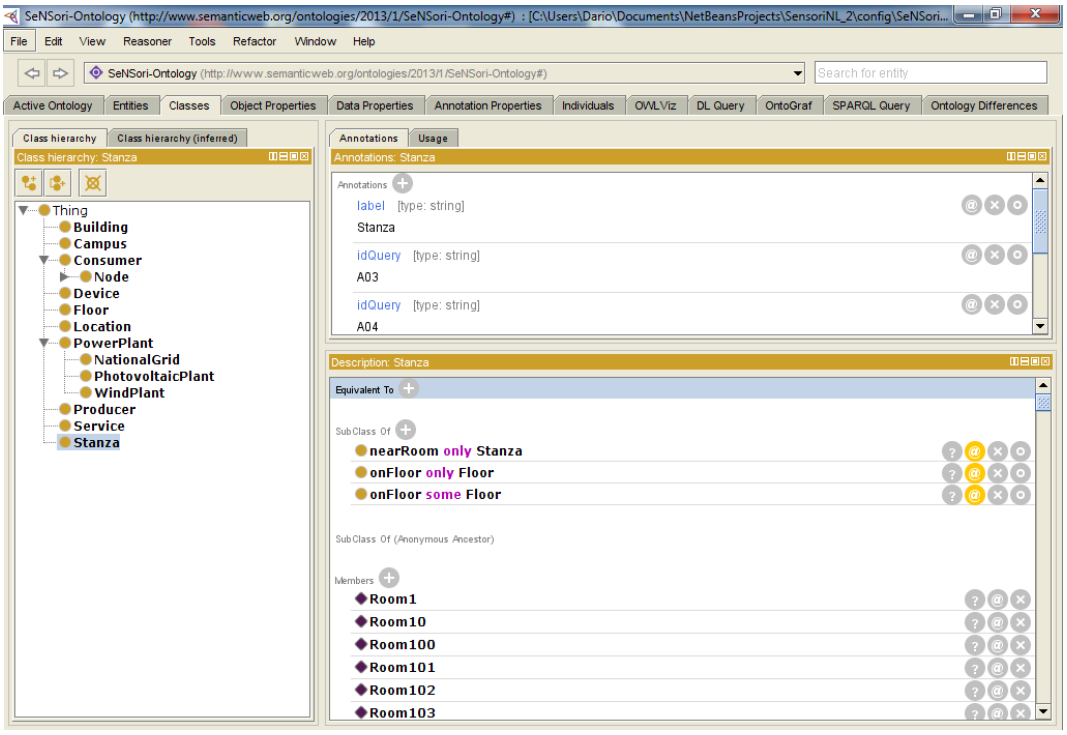


Figura 5.3: Annotazioni presenti nell'ontologia per la classe Room

SPARQL.

Di seguito vengono spiegate in maniera dettagliata le parti di cui è composto il file patternDB.xml:

- **nlquery:** È la classe padre: contiene infatti tutti gli altri componenti. Viene definita inserendo l'attributo *id* che indica l'identificativo del pattern a cui si farà riferimento nell'ontologia.

- **sentence:** Contiene al suo interno i componenti che servono per descrivere la query in linguaggio naturale. Questa può essere composta da testo fisso e variabile (cioè il cui contenuto verrà poi riempito dall'utente). Contiene anche l'attributo *description*, il quale contiene una descrizione della query.

  - **fixed:** Contiene le parti di query in linguaggio naturale il cui testo non può essere modificato dall'utente.

  - **var:** Contiene le parti di query in linguaggio naturale il cui testo può essere modificato dall'utente. È costituito dall'attributo *ref* che contiene una stringa che fa riferimento all'identificativo del campo successivo che descrive le variabili.

```
<nlquery id="A04">
  <sentence description="Services and sensors of a certain category taken from a specific room">
    <fixed>Show services and sensors from room</fixed>
    <var ref="p3"/>
    <fixed>which belong to category</fixed>
    <var ref="p4"/>
  </sentence>
  <variables>
    <variable classname="Room" default="roomName" id="p3" type="String"/>
    <variable classname="" default="category" id="p4" type="String"/>
  </variables>
  <formalQuery>
    <query>
      <queryPart>
                  SELECT ?service ?sensor
                  WHERE {
                      ?service base:dataFrom ?sensor.
                      ?sensor base:hasLocation ?location.
                      ?location base:inRoom base:</queryPart>
      <fvar ref="p3"/>
      <queryPart>.
                      ?sensor base:category ?category.
                      FILTER regex (?category, "</queryPart>
      <fvar ref="p4"/>
      <queryPart>")
                  }
              </queryPart>
    </query>
    <vars>
      <formalVar modelRef="http://www.semanticweb.org/ontologies/sensori.owl#Service"/>
      <formalVar modelRef="http://www.semanticweb.org/ontologies/sensori.owl#Sensor"/>
      <formalVar modelRef="http://www.semanticweb.org/ontologies/sensori.owl#Location"/>
      <formalVar modelRef="http://www.semanticweb.org/ontologies/sensori.owl#Room"/>
    </vars>
  </formalQuery>
  <style file="./config/style/tab-style.xslt"/>
</nlquery>
```

Figura 5.4: Esempio di pattern contenuto nel file patternDB.xml

- **variables:** Al suo interno contiene le descrizioni delle variabili.
- **variable:** È contenuto nel campo precedente e indica una singola variabile. Al suo interno ha quattro attributi che servono per descrivere la variabile: *id* è l'identificatore della variabile; *classname* contiene il nome della classe dell'ontologia a cui la variabile si riferisce, tale campo serve poi all'applicazione per capire i valori che sono ammissibili dalla variabile. Questo attributo può anche essere vuoto se il suo valore fa riferimento al range di una data properties dell'ontologia, quindi può assumere un insieme infinito di valori; *default* indica il valore di default della variabile; *type* indica che tipo di valore assumerà la variabile (String o Number).
- **formalQuery:** È composta dai seguenti campi:
  - **query:** Contiene i campi per descrivere la query in linguaggio SPARQL. A sua volta, come la query in linguaggio naturale, è costituita dai campi fissi e variabili:

        * **queryPart:** È composta dai campi fissi della query SPARQL.
        * **fvar:** È composta dai campi variabili della query SPARQL, come per il campo *var* della descrizione della query in linguaggio naturale, contiene un riferimento alla variabile da rappresentare.
      - **vars:** Al suo interno contiene un insieme di riferimenti all'ontologia.
      - **formalVar:** Indica un riferimento singolo all'ontologia. Tale campo non serve se i riferimenti dell'ontologia ai pattern vengono scritti a mano, se invece vengono scritti dall'applicazione servono per capire quali elementi dell'ontologia dovranno avere l'annotazione "idQuery" che si riferisce al pattern dichiarato.

- **style:** Indica il percorso ad un file xslt che serve per mostrare i risultati dell'interrogazione dell'ontologia in una forma più "elegante".

Per permettere la corretta definizione di nuove immissioni nel documento XML, è presente il file patterns.dtd. Questo ha lo scopo di validare il file patternDB.xml, controllando che i campi necessari siano presenti e nel corretto formato.

5.2.4 Applicazione Sensori_NL

La figura 5.5 mostra la pagina che appare lanciando l'applicazione Sensori_NL. Questa può essere vista come composta da quattro parti: la prima serve per inserire e rimuovere le parole chiave (keyword); la seconda permette di inserire nuovi pattern e rimuovere quelli presenti; la terza permette di inserire una query per interrogare l'ontologia direttamente in SPARQL; l'ultima mostra le interrogazioni in linguaggio naturale e ne permette la selezione e la modifica dei campi variabili. Inizialmente le query in linguaggio naturale che vengono mostrate sono le prime otto presenti nel file patternDB.xml.

Settaggio impostazioni iniziali

All'avvio dell'applicazione, viene creata la classe *Configuration*, nella quale vengono eseguite le operazioni iniziali di settaggio dei parametri: viene aperta l'ontologia e viene creata la classe *OntologySuggestor* che è responsabile della

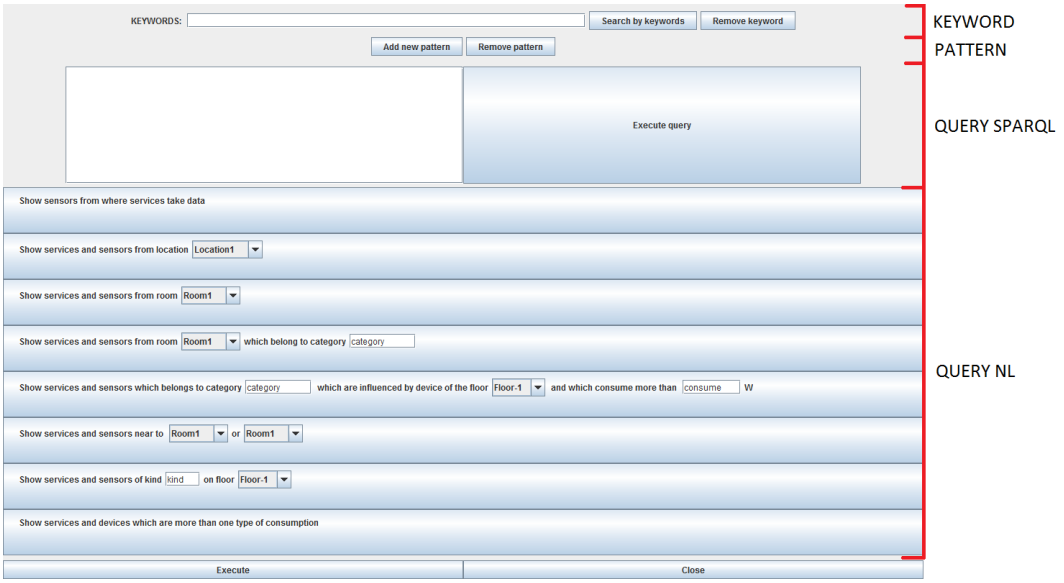


Figura 5.5: Pagina iniziale dell'applicazione Sensori_NL

gestione delle keyword e del calcolo della posizione delle query in linguaggio naturale. In particolare, inizialmente vengono caricate dall'ontologia tutte le parole chiave che verranno considerate tali e non variabili: i nomi delle classi, quelli delle proprietà e le annotazioni label che sono state inserite nell'ontologia. A questo punto viene aperto il file patternDB.xml e vengono caricate le sue informazioni all'interno della classe *NLQuery*. Quest'ultima, contiene anche il punteggio di ogni query, che sta ad indicare in che posizione verrà visualizzata la determinata query. Questo punteggio viene calcolato in base alle parole chiave che verranno inserite dall'utente successivamente; inizialmente vale zero. Quindi viene creata l'interfaccia grafica, come appare in figura 5.5, creando, nella parte relativa alle query in linguaggio naturale, un menù a tendina per le variabili che contengono l'attributo *classname* nel file patternDB.xml non vuoto e un'area di testo per le altre. Il menù a tendina conterrà tutte le istanze della classe dell'ontologia a cui l'attributo *classname* fa riferimento.

Inserimento keyword

A questo punto l'utente si trova davanti la schermata iniziale. L'uso classico dell'applicazione consiste nel ricercare le interrogazioni che interessano all'utente ed eseguirle. Per raggiungere questo scopo, l'utente deve inizialmente inserire nel campo "keywords" delle parole chiave che identificheranno l'interrogazione che l'utente sta cercando. Per ogni lettera inserita, l'applicazione,

che aveva nel passo di settaggio precedentemente spiegato caricato dall'ontologia tutte le possibili keywords, cerca tutte le parole chiave che iniziano con quella parte iniziale. La figura 5.6 mostra un esempio del suggerimento.

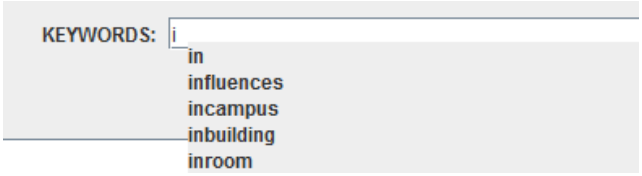


Figura 5.6: Esempio di suggerimento parole chiave

Vengono così suggerite le parole chiave all'utente. Se la parola chiave inserita dall'utente non è presente tra quelle suggerite, questa viene considerata una variabile e verrà sostituita nei campi variabili della query in linguaggio naturale. In particolare se la parola inserita dall'utente è un numero, viene sostituita nei campi variabili che devono contenere numeri, altrimenti in quelli che devono contenere stringhe.
Una volta inserite le parole chiave, cliccando il tasto "Search by keywords", l'applicazione calcola in che ordine mostrare le interrogazioni in linguaggio naturale. Per fare questo l'applicazione prende, per ogni parola chiave inserita, le annotazioni dall'ontologia denominate "idQuery" che fanno riferimento alla parola chiave. Quindi per ogni annotazione presente, va a vedere quale query si riferisce all'id riportato dall'annotazione e salva tale query su un attributo temporaneo. Controlla quindi quali query della classe *NLQuery* corrispondono all'attributo e di quelle incrementa il punteggio. Una volta fatto ciò aggiorna l'ordine delle query da mostrare basandosi sul punteggio. L'algoritmo 1 mostra il processo di aggiornamento dei punteggi appena spiegato.

Nel caso in cui l'utente non trovi ancora la query di interesse, può inserire ulteriori parole chiave, così i risultati verranno aggiornati tenendo conto sia delle vecchie che delle nuove parole chiave inserite.
Oltre a aggiungere parole chiave, l'utente può anche eliminarne alcune già inserite cliccando il tasto "Remove keyword". In questo caso l'utente selezionerà la parola da eliminare tra quelle da lui inserite. L'applicazione ricalcolerà quindi l'ordine di visualizzazione delle query, togliendo la parola da quelle inserite, reimpostando tutti i punteggi di query ed eseguendo nuovamente l'algoritmo 1.

Interrogazioni basate su linguaggio naturale e parole chiave: Sensori_NL

Algorithm 1 Aggiornamento posizione query in linguaggio naturale

Input: Lista di parole chiave inserite.

Output: Lista di query in linguaggio naturale ordinate in base alla rilevanza delle parole chiave inserite.

```
for Ogni parola chiave inserita do
  if La parola chiave è una variabile then
    if La variabile è un numero then
      Sostituisci la parola inserita in ogni valore numerico non ancora
      modificato
    else
      Sostituisci la parola inserita in ogni valore di tipo stringa non ancora
      modificato
    end if
  else
    Prendi dall'ontologia il contenuto delle annotazioni chiamate "idQuery"
    della risorsa inserita dall'utente (la parola chiave);
    for Ogni elemento trovato do
      for Ogni query presente do
        if Elemento=id query then
          Salva idQuery
        end if
      end for
    end for
    for Ogni query presente do
      for Ogni id salvato do
        if id salvato=id query then
          Incrementa punteggio query
        end if
      end for
    end for
    Aggiorna ordine di visualizzazione query
  end if
end for
```

Selezione query in linguaggio naturale

Tra le query proposte, l'utente potrà ora selezionare quelle di suo interesse scegliendole direttamente. Se alcune di queste contengono ancora come valore di variabile quello di default oppure un valore non corretto, l'utente potrà inserire il valore corretto direttamente nella query. Quando l'utente ha scelto le query potrà lanciarne l'esecuzione. Ogni query presente è una rappresentazione della classe *NLQuery*. Quando ne viene selezionata una, l'attributo *selected* della classe *NLQuery* viene messo a *true*. In questo modo, quando l'utente esegue le query selezionate, l'applicazione sa quali query eseguire. Quando l'utente clicca il tasto "Execute", l'applicazione eseguirà le query selezionate: per ognuna, prende la query in linguaggio SPARQL e la esegue sull'ontologia. La query SPARQL è ottenuta lasciando le parti fisse come sono state dichiarate, mentre sostituendo il nome della variabile (quella definita nel file patternDB.xml) con il valore corretto definito dall'utente. Per poter eseguire le query sull'ontologia, è stato usato ARQ, un query engine per Jena che supporta le interrogazioni SPARQL su RDF. ARQ è costituito da quattro classi principali, presenti nel package *com.hp.hpl.jena.query*:

- **Query:** è il contenitore per tutti i dettagli della query.
- **QueryFactory:** crea l'oggetto *Query* partendo dalla stringa contenente la query in SPARQL.
- **QueryExecution:** rappresenta un'esecuzione di una query.
- **QueryExecutionFactory:** ottiene gli oggetti *QueryExecution* dagli oggetti *Query*.

Quindi da una stringa contenente la query SPARQL, attraverso il metodo *create* della classe *QueryFactory* è stata ottenuta un'istanza dell'oggetto *Query*. Da questo, insieme al modello dell'ontologia, con il metodo *create* di *QueryExecutionFactory* è stata ricavata un'istanza di *QueryExecution* e da qui attraverso il metodo *execSelect* i risultati dell'esecuzione della query vengono ricavati. Questi risultati sono di tipo *ResultSet*, un iteratore anch'esso contenuto nel package *com.hp.hpl.jena.query*. I risultati presenti in *ResultSet*, vengono sistemati nel formato XML attraverso il metodo *outputAsXML* della classe *ResultSetFormatter*. Quindi viene utilizzato il campo *style* presente nel

file patternDB.xml per recuperare il foglio di stile della query eseguita e mostrarne i risultati in accordo. La figura 5.7 mostra un esempio del risultato di un'interrogazione.

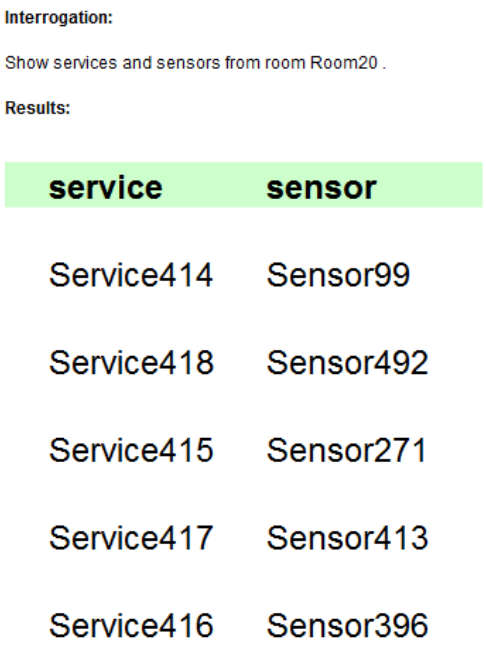


Figura 5.7: Esempio risultato dell'esecuzione di una query

Inserimento interrogazione SPARQL

L'applicazione consente anche di inserire interrogazioni direttamente nel linguaggio SPARQL. Tale interrogazione può essere inserita nell'area di testo presente nella parte "QUERY SPARQL" nella figura 5.5. Cliccando il tasto "Execute Query", l'applicazione eseguirà la query inserita. I passi compiuti dall'applicazione per eseguire la query sono simili ai passi spiegati precedentemente quando l'utente seleziona una query in linguaggio naturale. Qui non c'è però bisogno di ricavare la query SPARQL perché già presente.
Questa funzionalità è stata inserita per permettere a utenti esperti, che hanno bisogno di trovare cose molto specifiche non presenti tra le query in linguaggio naturale proposte, di ottenerle, oppure che preferiscono scrivere direttamente la query anziché cercarla attraverso le parole chiave.

Inserimento e rimozione di pattern dall'applicazione

Nuovi pattern possono essere inseriti nel file patternDB.xml e mettendo poi i relativi riferimenti nell'ontologia. Tuttavia, questo procedimento non garantisce che gli id inseriti nel file xml siano univoci. L'applicazione quando eseguita lo può capire e comunicare all'utente, il quale potrebbe poi dover modificare nuovamente sia il file xml che l'ontologia. Per risolvere questo problema, l'applicazione consente di inserire e rimuovere pattern direttamente da essa. Così una volta inserite le informazioni, l'applicazione penserà sia alla modifica del file patternDB.xml che a quella dell'ontologia. Queste operazioni sono consentite nella parte "PATTERN" mostrata in figura 5.5.
Cliccando il tasto "Add new pattern" l'applicazione consente l'inserimento di nuovi pattern. Per prima cosa all'utente appare la finestra dove viene chiesto di inserire l'id del nuovo pattern (figura 5.8).

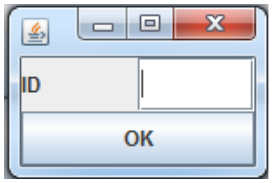


Figura 5.8: Inserimento dell'id del nuovo pattern

Una volta inserito, l'applicazione controlla se questo era già stato usato verificando il documento xml. In caso positivo viene mostrato un errore, altrimenti verrà mostrata la finestra di inserimento della query in linguaggio naturale (figura 5.9) dove sarà possbile inserire la descrizione, la parte fissa e quella variabile della query in linguaggio naturale. Per l'aggiunta di queste ultime due, l'utente dovrà cliccare gli appositi bottoni, in modo che sia lui a comporre la query come vuole.

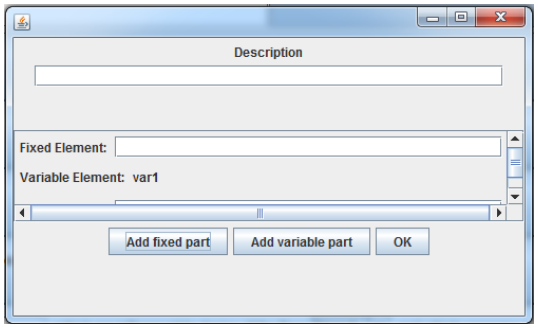


Figura 5.9: Inserimento della query in linguaggio naturale del nuovo pattern

Per inserire le variabili, viene mostrata un'ulteriore finestra (figura 5.10) dove l'utente deve inserire id, valore di default assunto, tipo di variabile (stringa o numerica) e se è necessario anche la classe a cui si riferisce nell'ontologia.

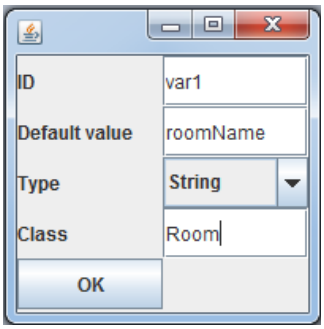


Figura 5.10: Inserimento delle variabili della query in linguaggio naturale

Anche in questo caso l'applicazione controllerà l'assenza nel file xml dell'id della variabile inserita. A questo punto viene chiesto di inserire la query in linguaggio SPARQL. La finestra che appare è composta da un'area di testo, dove l'utente deve semplicemente inserire la query. Le variabili, possono essere inserite racchiudendole tra parentesi quadre. Infine appare l'ultima finestra dove l'utente potrà inserire quali classi e proprietà dell'ontologia fanno riferimento alla query appena inserita, in modo che l'id della query venga inserito anche nell'ontologia nel posto corretto. Una volta finiti gli inserimenti dei vari campi, l'applicazione inserisce il nuovo pattern nel file xml e i riferimenti nell'ontologia. Per inserire il pattern nel file xml, l'applicazione fa uso di DOM (Document Object Model, che permette di vedere il file xml come un albero) [42], come in ogni altro caso in cui fa riferimento al file patternDB.xml: partendo dalla radice aggiunge un nuovo ramo che rappresenta il nuovo pattern, costruito in modo top-down: partendo dall'id della query si aggiungono via via gli elementi inseriti in modo da formare la struttura descrita nella sezione che presentava il documento xml. Per inserire i riferimenti nell'ontologia, l'applicazione prende le risorse di questa basandosi su ciò che ha inserito l'utente nell'ultima finestra e ne aggiunge l'annotazione *idQuery* inserendoci l'id del pattern.
Oltre a poter aggiungere pattern, l'applicazione consente anche la loro rimozione. Dopo aver cliccato il bottone "Remove pattern" nella pagina iniziale, all'utente viene mostrato un menù a tendina come in figura 5.11.
Qui l'utente potrà selezionare quale pattern eliminare. L'applicazione prende l'id del pattern scelto dall'utente, apre il file patternDB.xml e, sempre

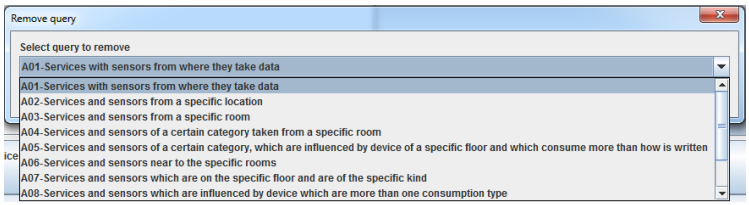


Figura 5.11: Rimozione pattern inseriti

utilizzando DOM, cerca l'id corrispondente per eliminare tutto il ramo che rappresenta. Poi, deve eliminare i riferimenti nell'ontologia. Per fare questo vengono cercate le annotazioni "idQuery" che corrispondono all'id selezionato, cercandolo prima tra le classi dell'ontologia, poi tra le object property e infine tra le data property. Tutte le annotazioni corrispondenti trovate vengono eliminate.

5.3 Risultati

Per capire se l'applicazione funzionasse correttamente, è stata testata con alcuni casi reali significativi. Di seguito ne vengono presentati due.

5.3.1 Recupero servizi

Supponiamo che un utente abbia bisogno di capire la temperatura attuale nelle stanze vicine alla stanza "Room112" perché ha aumentato la potenza dell'impianto di raffreddamento, ma il sensore di temperatura di quella stanza continua a misurare la stessa temperatura. Vuole allora capire se è il sensore o l'impianto ad aver problemi analizzando i sensori vicini.
Nella schermata iniziale dell'applicazione (rappresentata nella figura 5.5) l'utente non trova nessuna query utile per il suo obiettivo. Allora comincia a scrivere delle parole chiave che identificano la sua necessità. Parole chiave possono essere:

- **room:** è alla ricerca di stanze;
- **near:** cerca stanze che sono vicine ad una specifica stanza;
- **kind:** cerca sensori di tipo temperatura;
- **time output:** cerca servizi che restituiscono il tempo di lettura attuale.

L'utente comincia allora a scrivere nel campo "KEYWORDS" le parole chiave room e near. Da notare che ad ogni inserimento di lettera l'applicazione presenta all'utente le parole chiave ammissibili, quindi per esempio, quando l'utente digita la parola "near", l'applicazione gli propone nearroom e nearbuilding, indicando così che near deve essere specificato meglio (figura 5.12).

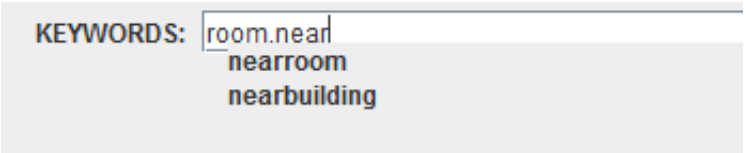


Figura 5.12: Inserimento parola chiave "near"

Supponiamo ora che l'utente voglia provare a vedere se inserendo solo queste due parole chiave l'applicazione restituisce già un risultato utile. La figura 5.13 mostra i primi risultati forniti dall'applicazione.

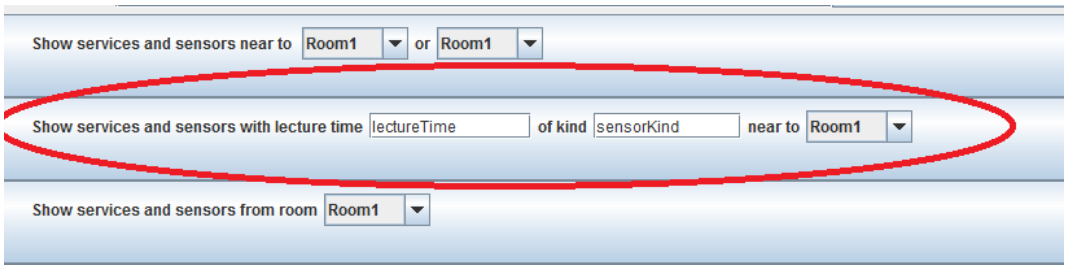


Figura 5.13: Risultato fornito dall'applicazione dopo l'inserimento delle parole chiave "nearRoom" e "room"

Dalla figura 5.13 si vede che la query ricercata è in seconda posizione, in quanto anche la prima è costituita dalle stesse parole chiave. Tuttavia può capitare che l'utente non si accorga della presenza della query voluta perché egli controlla solo quelle in prima posizione. Allora inserisce altre keyword, come "kind" e "timeoutput". Il risultato è mostrato in figura 5.14.
Stavolta l'interrogazione cercata compare in prima posizione. L'utente deve ora inserire i valori nelle variabili. Per far questo può o inserirli direttamente nei campi della query oppure inserirli dal campo "KEYWORDS". Una volta inseriti si ottiene l'interrogazione come in figura 5.15.
Ora basta eseguirla per ottenere i servizi richiesti come mostrato in figura 5.16.

Oltre ai servizi vengono ritornate alcune informazioni che riguardano i servizi stessi.

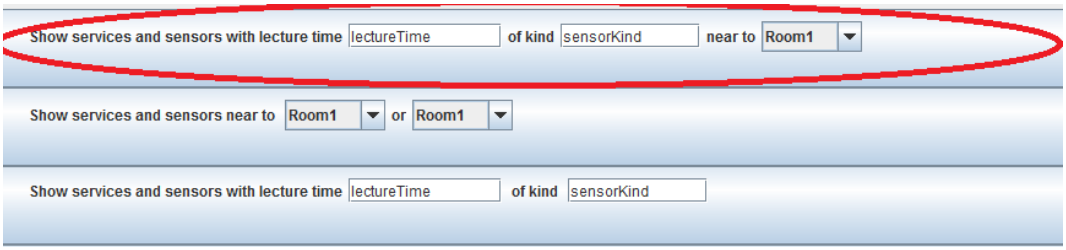


Figura 5.14: Risultato fornito dall'applicazione dopo l'inserimento di tutte le parole chiave identificate

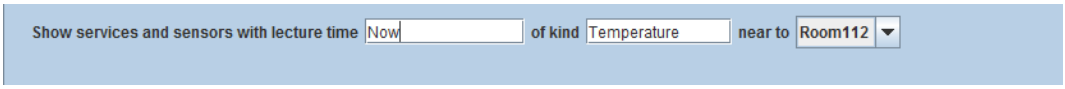


Figura 5.15: Query cercata pronta per essere eseguita

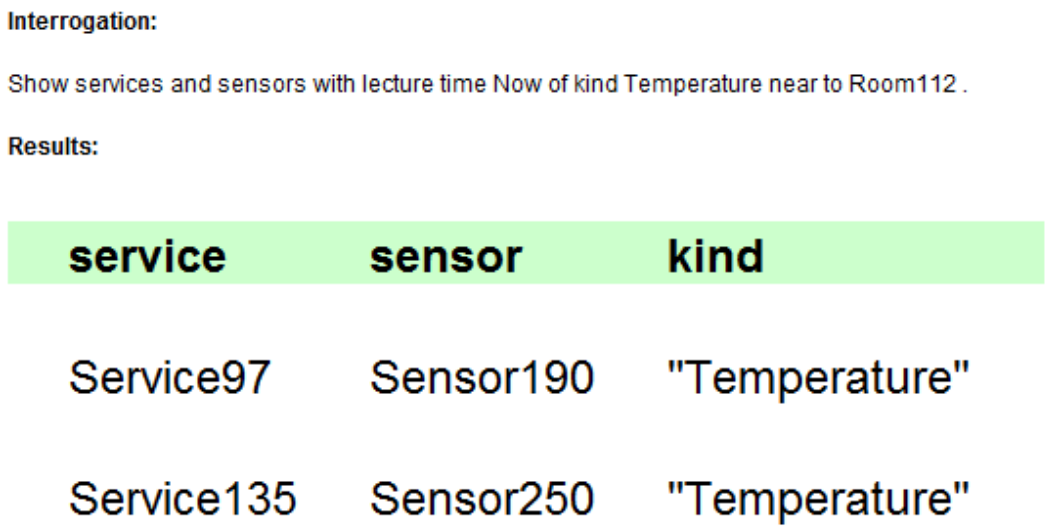


Figura 5.16: Servizi che soddisfano la query eseguita dall'utente

Per essere sicuri che il risultato sia corretto si può controllare nell'ontologia quali sono i servizi che soddisfano la query inserita dall'utente. Vicino alla "Room112" abbiamo le stanze "Room111" e "Room116" (figura 5.17).
Queste due stanze contengono i servizi mostrati in figura 5.18.
Di questi servizi solo tre restituiscono il tempo di lettura attuale (figura 5.19).
Dei servizi rimasti, solo il servizio 97 e il 135 (quelli restituiti dalla query eseguita dall'utente) si riferiscono a sensori che misurano le temperature (figura 5.20).
Se l'utente fosse esperto e conoscesse sia il dominio dell'applicazione che il

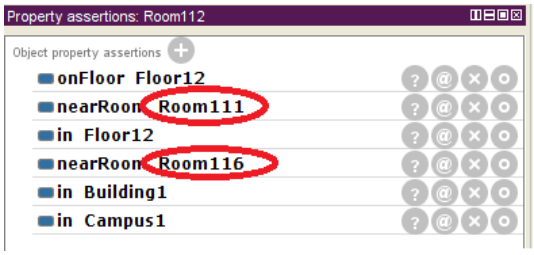


Figura 5.17: Stanze vicine alla "Room112"

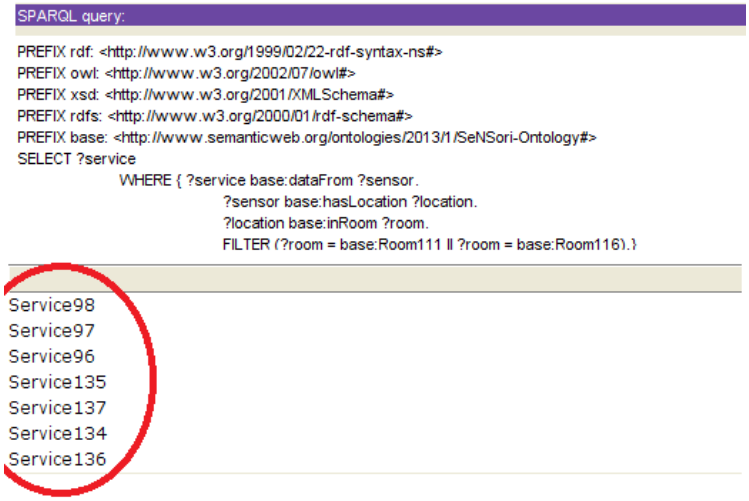


Figura 5.18: Servizi delle stanze vicine alla "Room112"

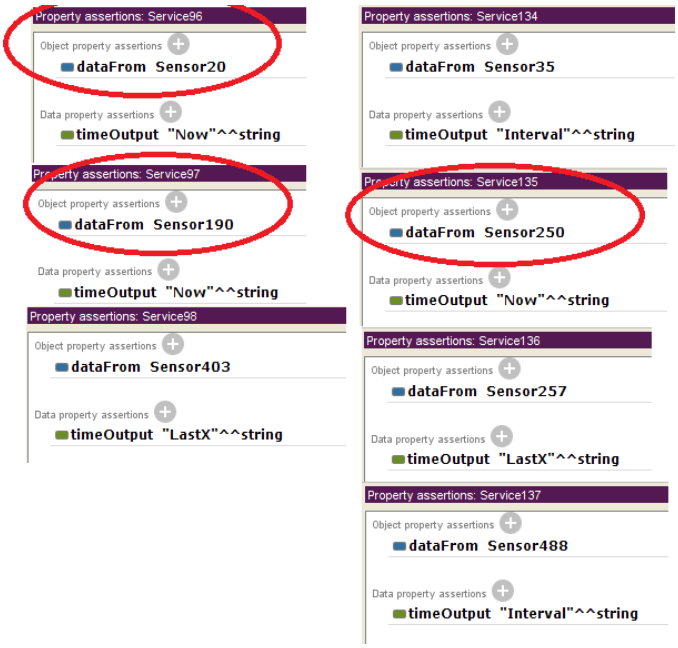


Figura 5.19: Caratteristiche dei servizi delle stanze vicine alla "Room112"

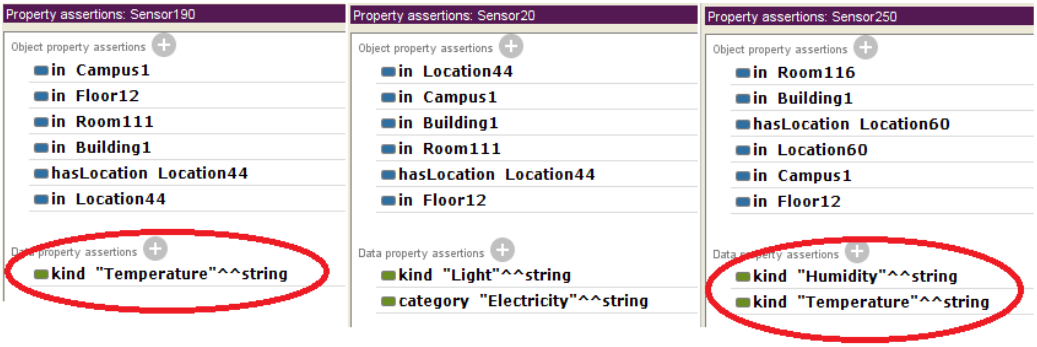


Figura 5.20: Caratteristiche dei sensori collegati ai servizi che tornano il tempo attuale delle stanze vicine alla "Room112"

linguaggio SPARQL per interrogare l'ontologia, può inserire la query in questo linguaggio direttamente, come mostrato in figura 5.21.

```
PREFIX base: <http://www.semanticweb.org/ontologies/2013/1/SeNSori-Ontology#>
SELECT ?service ?sensor ?kind
        WHERE{?service base:dataFrom ?sensor.
                ?service base:timeOutput ?time.
                ?sensor base:kind ?kind.
                ?sensor base:hasLocation ?location.
                ?location base:inRoom ?room.
                ?room base:nearRoom base:Room112.
                FILTER regex(?time, "^Now" ).
                FILTER regex(?kind, "^Temperature" ).
        }
```

Figura 5.21: Query SPARQL che ritorna i servizi vicino alla stanza "Room112" che misurano la temperatura all'istante attuale

Eseguendola si ottengono gli stessi servizi ottenuti precedentemente, come mostra la figura 5.22.

5.3.2 Inserimento nuovi pattern

Supponiamo ora che all'amministratore dell'applicazione di un edificio, siano giunte molte lamentele dovute alla mancanza di interrogazioni che servono agli utenti dell'edificio. Una di queste lamentele riguarda il fatto che non esiste un'interrogazione che permette di ricavare i servizi che gestiscono attuatori installati in una generica stanza utilizzati per gestire un particolare dispositivo. L'amministratore allora decide di inserire questa funzionalità. Dopo aver scelto l'opzione "Add new pattern" nella pagina iniziale, l'amministratore comincia ad inserire i campi necessari. Inserisce l'id "A12", quindi nella finestra successiva

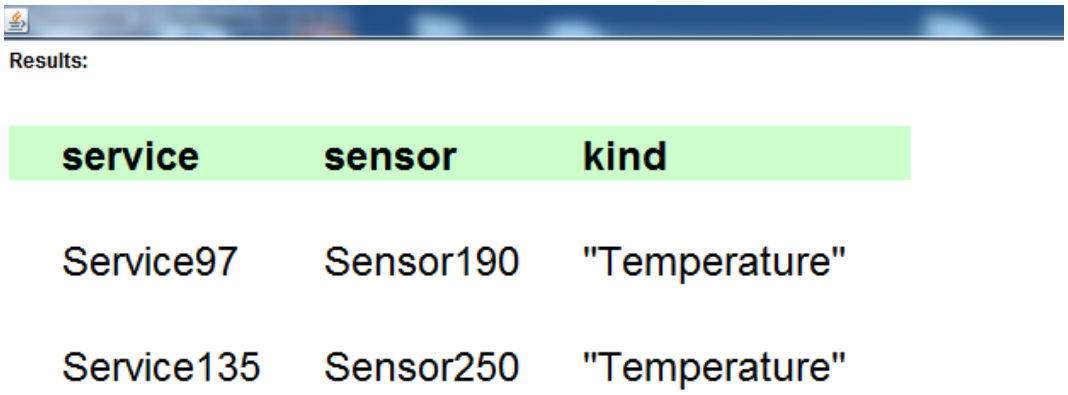


Figura 5.22: Risultati esecuzione query SPARQL che ritorna i servizi vicino alla stanza "Room112" che misurano la temperatura all'istante attuale

la descrizione "Service and actuators which acts upon a device in a specific room" e le parti fisse e variabili dell'interrogazione, come mostrato nelle figure 5.23 e 5.24.

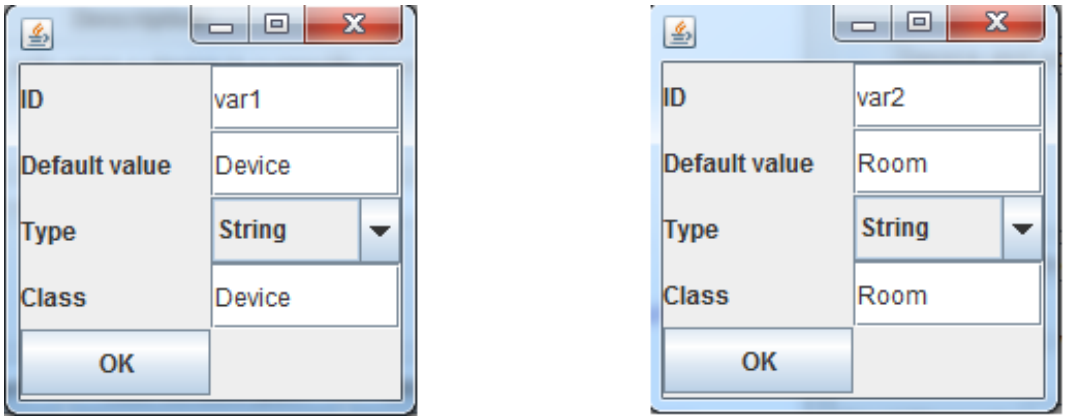


Figura 5.23: Inserimento delle due variabili "Room" e "Device" nella descrizione della query

Dovrà ora inserire la querry in SPARQL (figura 5.25) e infine i riferimenti nell'ontologia (figura 5.26).
Ora l'applicazione conterrà il nuovo pattern. Inserendo quindi nelle keyword alcune delle parole chiave si otterrà in prima posizione la query mostrata in figura 5.27.
Impostando, per esempio, come variabili il dispositivo "LampCompactFluorescent287" e la stanza "Room126" , eseguendo la query si ottiene il risultato di figura 5.28.

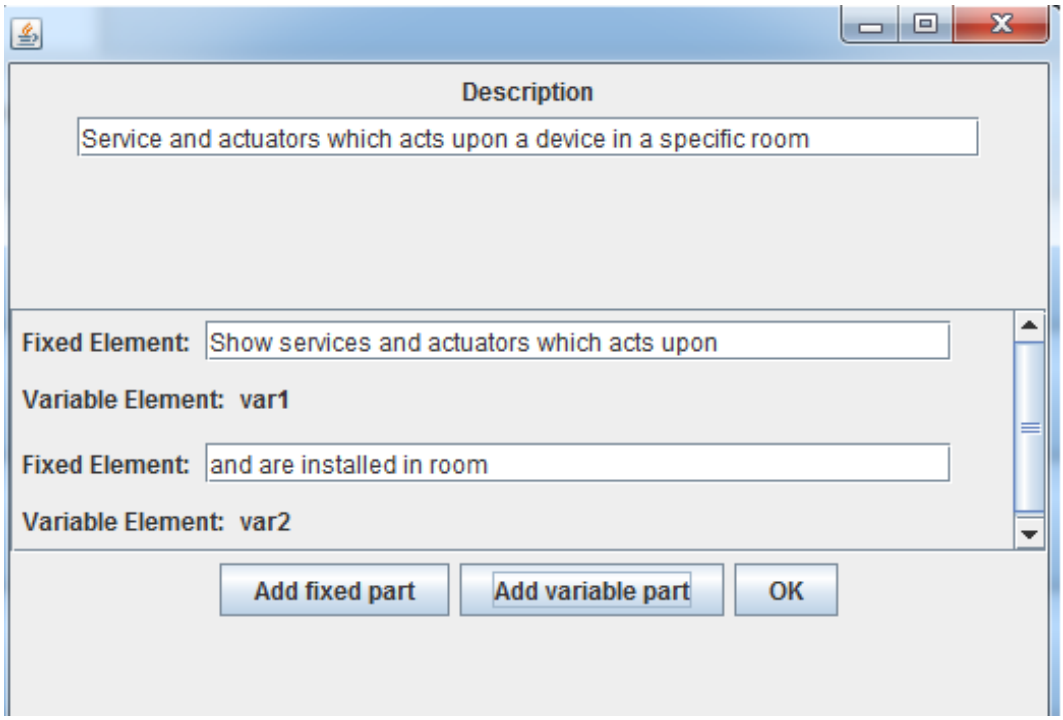


Figura 5.24: Inserimento delle parti fisse e variabili nella descrizione della query

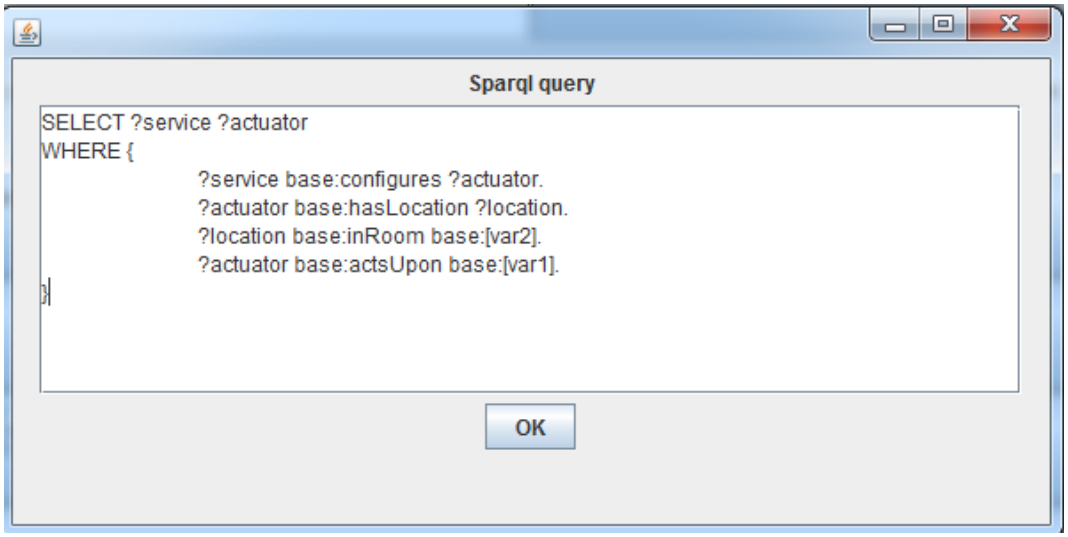


Figura 5.25: Inserimento della query SPARQL

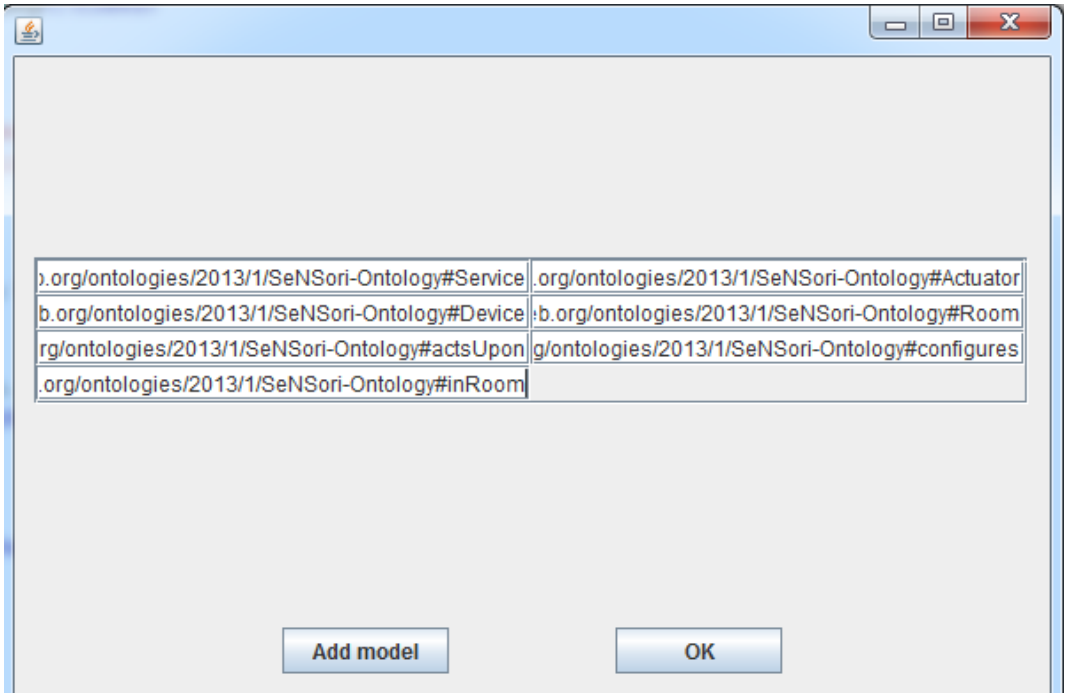


Figura 5.26: Inserimento dei riferimenti dell'ontologia

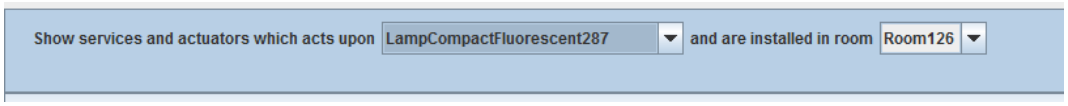


Figura 5.27: Interrogazione inserita nell'applicazione e pronta per essere eseguita

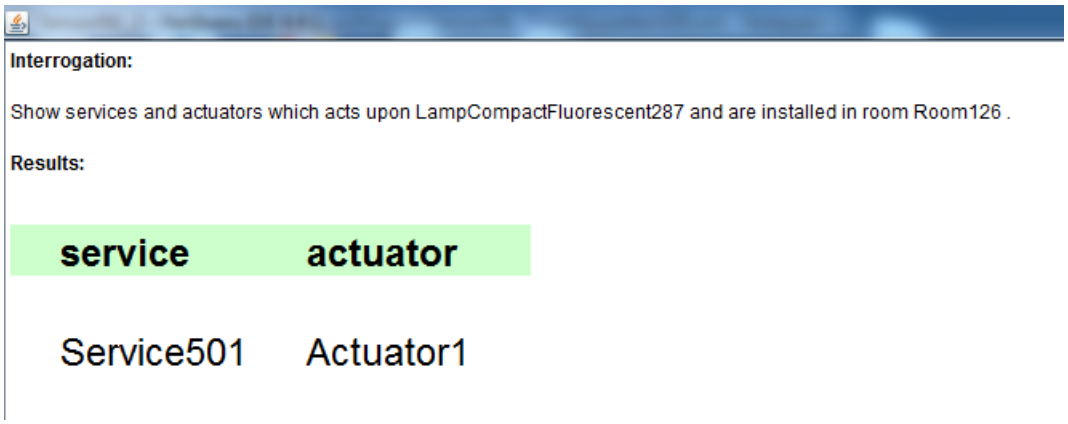


Figura 5.28: Risultato dell'esecuzione dell'interrogazione inserita dall'amministratore

5.4 Sommario

In questo capitolo è stata descritta l'infrastruttura software sviluppata per permettere l'esecuzione di interrogazioni sull'ontologia del dominio SeNSori partendo da query descritte in linguaggio naturale. È stato dapprima presentato il progetto SAFE, da cui l'applicazione sviluppata ha preso spunto, mostrando tuttavia perché questa non poteva essere usata nel dominio di interesse. Sono poi state descritte le parti da cui è composta l'applicazione, mostrando l'architettura e spiegando a cosa servono le singolo parti: l'ontologia per contenere la struttura e i dati del dominio, il file patternDB.xml per le corrispondenze tra query in linguaggio naturale e query in linguaggio SPARQL, l'applicazione vera e propria che si interfaccia con il file xml e l'ontologia e che consente di interrogare quest'ultima sia attraverso query scritte direttamente in SPARQL, sia attraverso la scelta di query in linguaggio naturale. Infine sono stati mostrati i risultati ottenuti dall'applicazione per mezzo di due esempi che ne mostrano le principali funzionalità.

Capitolo 6

Esplorazione e interrogazione interattiva: SensoriExploration

Nella prima parte di questo capitolo viene spiegato cosa si intende per esplorazione, fornendo alcuni esempi e spiegando perché è importante e perché non è fattibile con le tecniche note. Nella seconda parte viene invece spiegato come è stata sviluppata l'applicazione SensoriExploration che consente di fare esplorazione del dominio del progetto SeNSori, spiegando i vari passi dell'applicazione e le tecniche utilizzate.
L'applicazione SensoriExploration è stata sviluppata sempre per lo scopo di recupero di servizi, ma rispetto a Sensori_NL, gli utenti per i quali è stata pensata sono meno esperti, in quanto non conoscono in maniera sufficientemente approfondita il dominio SeNSori per poter utilizzare Sensori_NL. SensoriExploration consente quindi di partire da una query definita dall'utente. Ad ogni passo cerca di arricchire l'interrogazione con ulteriori condizioni, in modo da arrivare a definire la query di cui necessita l'utente.

6.1 Esplorazione

Con esplorazione si intende un nuovo campo di ricerca, che ha come obiettivo il miglioramento della comprensione di ampi e ricchi insiemi di dati. Non sempre infatti l'utente è ha conoscenza del dominio in studio, ha perciò bisogno dell'aiuto del sistema che lo indirizza su come muoversi per acquisire maggiore conoscenza del dominio. Per tale ragione l'esplorazione implica un'interazione

continua tra utente e sistema, come avviene tra le persone quando discutono. Per spiegare l'importanza dell'esplorazione, di seguito vengono illustrati due esempi.

- In un liceo, una professoressa di lettere vorrebbe introdurre la tecnologia nella sua didattica. Accede perciò ad un portale che raccoglie diverse esperienze educative insieme ai materiali relativi. Ogni esperienza è descritta attraverso una serie di documenti e classificata sulla base di molti parametri (es. materia, benefici, tecnologia ...). Inizialmente la professoressa vuole farsi un'idea generale dell'insieme totale (S0), ottenuto osservando i parametri di classificazione. Poi crea un primo sotto-insieme (S1), selezionando le tecnologie di interesse, come ad esempio Lavagne Interattive e Tablet. Tra i parametri, nota i benefici motivazionali (di cui non era consapevole) e "lavoro di gruppo" (strategia che ha sempre voluto fare); raffina allora l'esplorazione selezionando questi parametri, creando il sotto insieme S2. Nota però che la maggior parte delle esperienze riguardano materie scientifiche, che non sono di suo interesse. Torna allora all'insieme S1 precedente, dal quale seleziona materie umanistiche ed esclude "immigranti" come problema (non avendone in classe). Ottiene l'insieme S3 di suo interesse, il quale contiene poche istanze e può quindi essere analizzato facilmente.

- Un'applicazione permette di visualizzare siti archeologici di interesse in Italia, distinguendoli secondo tre variabili: tipo di sito, posizione, età di riferimento. Consideriamo il caso in cui un utente vuole capire quali civiltà erano presenti in Italia e come erano distribuite nel territorio. Si può notare che l'utente non è interessato ad una ricerca, perché l'utente non è interessato a trovare oggetti che soddisfino sue condizioni, ma piuttosto vuole investigare sul dominio. L'utente può combinare i filtri delle tre variabili e confrontare i risultati. Ad esempio impostando "Musei e siti archeologici nel nord Italia" può notare che la civiltà prevalente erano i Romani, impostando invece "Musei e siti archeologici nel sud Italia" nota che anche la civiltà greca ha avuto un'influenza importante in questa zona.

Nel primo esempio si può notare che la docente non era in cerca di qualcosa di specifico ma di ispirazione; era interessata ai "profili" dei vari insiemi (cioè

certe caratteristiche che possono essere identificate dagli attributi) e non alla loro "estensione" (ovvero gli elementi specifici); in base alle risposte fornite dal sistema, la sua interazione ha subito modifiche. Nel secondo si nota che l'utente acquisice conoscenza in base a ciò che emerge dall'interazione col sistema. Dagli esempi si può capire che tali esperienze d'uso non sarebbero ottenibili con gli attuali meccanismi di ricerca, nei quali l'obiettivo è quello di recuperare specifiche informazioni, ma che possono invece essere considerate simili a quanto si ottiene interagendo con un esperto del dominio. Cioè con l'esplorazione non si vuole semplicemente recuperare informazioni, ma far si che l'utente impara che certi oggetti che hanno certe caratteristiche (quelle specificate dall'utente con una query iniziale), ne hanno anche altre.

6.1.1 Differenze con tecniche note

Durante l'esplorazione l'enfasi viene data sugli aspetti intensionali (il profilo degli elementi) anziché estensionali (lista degli elementi che soddisfano certe condizioni) e sul flusso di interazioni, cioè ciascuna azione si basa su azioni precedenti.
Dal momento che l'esplorazione implica una continua interazione tra utente e sistema, questa non può essere realizzata con tecniche di accesso ai dati note in letteratura, come l'interrogazione, la navigazione o le raccomandazioni, in quanto esse presuppongono che l'utente abbia già un'idea chiara in mente; inoltre le tecniche di data mining utilizzate senza alcun supporto per l'utente, permettono di estrarre pattern frequenti con la possibilità di disorientare l'utente inesperto, invece di agevolarlo nel trovare soluzioni anche non attese.
Attraverso l'esplorazione si potrà migliorare la qualità delle esperienze degli utenti in tutti i settori in cui grandi quantità di informazioni devono essere gestite da utenti e amministratori il cui ruolo è quello di prendere decisioni velocemente, capire andamenti di fenomeni, etc.

6.2 Infrastruttura software

L'obiettivo è quello di creare un'applicazione, chiamata SensoriExploration, che suggerisce ad un utente che ha una bassa conoscenza del dominio quale attributo selezionare per raffinare la sua ricerca. Per far questo l'utente deve

inizialmente inserire una query, che può anche essere la più generale possibile. Se il numero di istanze che rispondono alla query è maggiore di una soglia allora il risultato deve essere migliorato aggiungendo delle condizioni sui campi della query. Per capire quali condizioni è meglio aggiungere, l'applicazione calcola il test chi-quadro su un insieme di attributi ritenuti utili a tal fine; quello con test minore sarà l'attributo da proporre all'utente.

6.2.1 Architettura del sistema

La figura 6.1 mostra l'architettura dell'applicazione SensoriExploration.

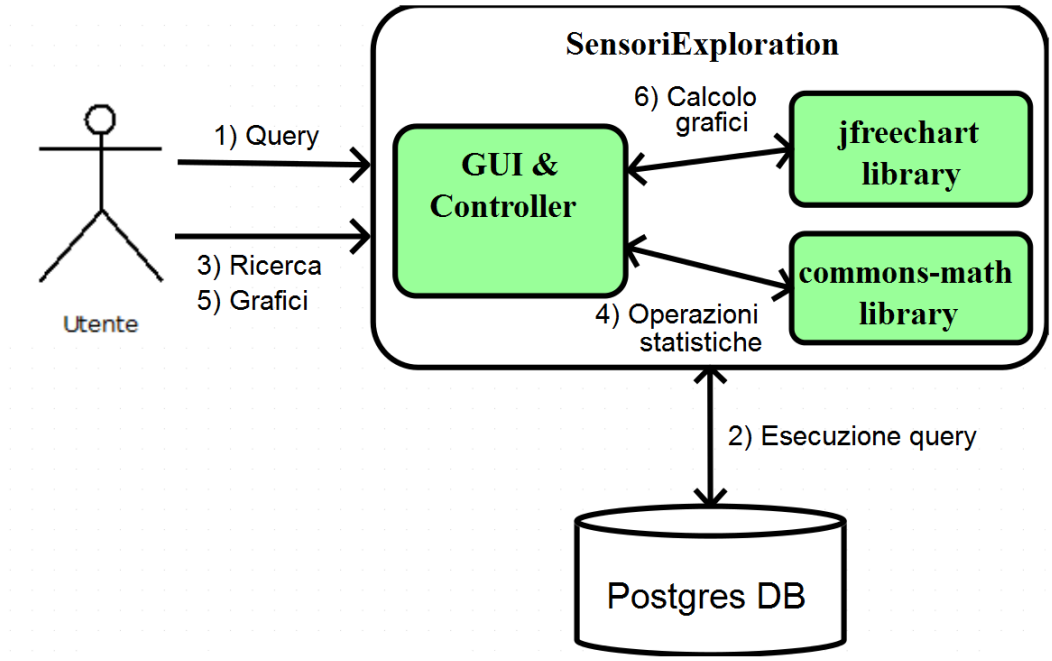


Figura 6.1: Architettura applicazione SensoriExploration

Come per l'architettura mostrata per l'applicazione Sensori_NL, viene riportato l'ordine di esecuzione delle operazioni, le quali verrano poi spiegate in maniera maggiormente dettagliata:

1. L'utente inserisce la query iniziale.
2. L'applicazione esegue la query, inserita dall'utente nel passo precedente, sul database e ne mostra i risultati.
3. L'utente vorrà arricchire la sua ricerca, altrimenti l'utilizzo di questa applicazione non sarebbe spiegato. Quindi chiede all'applicazione di cercare l'attributo da raffinare.

4. L'applicazione esegue sugli attributi presenti una serie di operazioni statistiche, che verranno spiegate in seguito, alcune delle quali fornite dalla libreria esterna *commons-math*. Queste hanno lo scopo di capire qual'è l'attributo migliore da restituire all'utente per venire raffinato. Una volta ottenuti i risultati, SensoriExploration ritorna all'utente tale attributo. L'utente inserisce il valore desiderato per esso, quindi se è la prima volta che viene eseguito questo passo l'applicazione interroga il database con la query completa (come spiegato nel paragrafo successivo), altrimenti l'applicazione elimina dai risultati calcolati precedentemente quelli che non rispettano la nuova condizione. In entrambi i casi vengono poi mostrati i risultati. Queste ultime due operazioni possono essere eseguite fino al termine degli attributi selezionabili o finché l'utente non ha trovato quello che cerca.

5. L'applicazione consente anche di visualizzare dei grafici per vedere quante istanze hanno determinati valori di attributi.

6. L'applicazione facendo uso della libreria esterna *jfreechart* mostra i grafici desiderati. Da notare che i punti riguardanti la ricerca e la visualizzazione dei grafici non devono per forza seguire l'ordine proposto, ma possono essere scambiati.

Come appena indicato, l'applicazione SensoriExploration fa uso di due librerie esterne: *jfreechart* e *commons-math*. La prima è utilizzata per compiere le operazioni necessarie per disegnare i grafici contenenti le distribuzioni degli attributi, la seconda per compiere le operazioni statistiche (ad esempio il calcolo del test chi-quadro o della distribuzione normale) che porteranno alla selezione di un'attributo da raffinare. Entrambe le operazioni vengono eseguite su richiesta dell'utente. Da notare che l'accesso al database postgres viene fatto due sole volte: all'inserimento della query iniziale da parte dell'utente e alla prima esecuzione del passo di raffinamento. Le successive interrogazioni per ricavare le informazioni, vengono sempre fatte sulle istanze restituite durante quest'ultima interrogazione al database tenendo in considerazione solo gli elementi non eliminati dai raffinamenti. Questo consente di velocizzare il processo di recupero dei dati.

6.2.2 Recupero istanze

All'avvio, l'applicazione offre la possilbilità di impostare due parametri che saranno utilizzati successivamente:

- **Soglia:** se il numero di istanze che soddisfano la query (o i raffinamenti di questa) è maggiore della soglia, allora l'applicazione suggerisce di continuare il processo di raffinamento.
- **Funzione di riferimento:** funzione con la quale il risultato ottenuto verrà confrontato nel test chi-quadro.

Se nessun parametro viene modificato, l'applicazione imposta la soglia uguale a 20 e come funzione di riferimento considera l'uniforme.
Una volta impostati i paramentri, l'utente dovrà inserire una query. È quindi necessario che l'utente abbia una conoscenza, seppur minima, del linguaggio SQL. Per quanto riguarda il dominio, non è indispensabile la sua conoscenza, in quanto l'obiettivo dell'esplorazione è quello di far navigare utenti in domini sconosciuti. L'applicazione eseguirà la query sulla base di dati, ne riceve i risultati e li visualizzerà. Se il numero di istanze che soddisfano la query è troppo elevato (maggiore della soglia), verrà detto all'utente che è meglio raffinare la ricerca. Per questo scopo è necesario eseguire la query completa, quella cioè ottenuta facendo tutti i join tra le tabelle utili (quelle che contengono attributi utili all'esplorazione), selezionando solo le istanze che soddisfavano la query inserita dall'utente. La query completa è mostrata in figura 6.2, dove ID_SELECTED_SERVICE è una stringa contenente gli id dei service che soddisfavano la query inserita dall'utente.

```
SELECT service.id, service.time_output, sensors.id_sensortype, devices.description,
        area.city, area.provincia, area.region, area.country, area.area,
        building.floor, building.id_building_type,
        times.second, times.minute, times.hour, dates.day, dates.month, dates.year
FROM service, sensors, installed_devices, devices, locations, area, building
        measurements, timestamps, times, dates
WHERE service.id_sensor = sensors.id AND sensors.id_installeddevice = installed_devices.id
        AND installed_devices.id_device = devices.id
        AND installed_devices.id_location = locations.id AND locations.id_area = area.id
        AND locations.id_building = building.id
        AND measurements.id_service = service.id AND measurements.id_timestamp = timestamps.id
        AND timestamps.id_date = dates.id AND timestamps.id_time = times.id
        AND and service.id in ID_SELECTED_SERVICE
```

Figura 6.2: Query completa

6.2.3 Raffinamento della ricerca

Per capire quale attributo proporre all'utente, l'applicazione si basa su un approccio statistico, analizzando la distribuzione degli attributi. In particolare un attributo viene considerato rilevante (cioè la sua selezione è utile ai fini dell'esplorazione) se la sua distribuzione è diversa da una distribuzione di riferimento. Sono state analizzate tre distribuzioni di riferimento: la distribuzione uniforme, la distribuzione dell'attributo al passo precedente e la distribuzione normale. La prima è implementata nel metodo $calculateDistributionRespectUniform$, la seconda nel metodo $calculateDistributionRespectOldValue$ e la terza nel metodo $calculateDistributionRespectNormal$. Tutti i tre metodi appartengono alla classe $ResultsController$.

Distribuzione uniforme

Rispetto a questa distribuzione, si può dire che l'attributo che presenta una distribuzione più lontana da quella uniforme è l'attributo da selezionare. Infatti se si considera un attributo qualsiasi, se questo è composto da 50% dei valori di un tipo e 50% di un altro probabilmente non è un attributo significativo perché non differenzia i dati, mentre se fosse composto dal 90% dei valori di un tipo e 10% di un altro lo sarebbe.
Per capire quanto un attributo ha distribuzione diversa dall'uniforme, l'applicazione usa il test chi-quadro. In particolare, per ogni attributo l'applicazione controlla quanti valori può assumere e conta il numero di istanze che contengono quel valore nell'interrogazione corrente (includendo cioè solo le istanze che compaiono nelle situazione corrente). Calcola quindi il numero di volte in cui ogni valore dovrebbe comparire se la distribuzione fosse uniforme, ottenendo $expectedCount$:

$$expectedCount \quad = \quad totalCount * \frac{1}{numberOfElementNoZero} \qquad (6.1)$$

dove $totalCount$ è il numero totale delle istanze della selezione corrente e $numberOfElementNoZero$ è il numero totale di valori assunti dall'attributo dove $count \neq 0$ ($count$ è il numero di volte che compare quel valore per quell'attributo, è stato calcolato precedentemente). Calcola poi la probabilità

reale (*prob*) e la differenza tra questa e quella uniforme (*pAtt*) di ogni valore:

$$prob = \frac{count}{totalCount} \tag{6.2}$$

$$pAtt = \sum_{\substack{j=0,\\ count\neq 0}}^{k} (prob_j - \frac{1}{numberOfElementNoZero})^2 \tag{6.3}$$

dove k è il numero totale di valori assunti dall'attributo considerato, quindi la sommatoria viene fatta sulla probabilità del verificarsi del valore dell'attributo. Quindi salva questi dati perché potrebbero servire al passo successivo se si fa il confronto con la distribuzione del passo precedente. Calcola poi il chi-quadro richiamando la funzione $ChiSquareTest(double[]exp, long[]obs)$. Questa vuole in ingresso un array di double e uno di long che indicano i valori attesi e osservati precedentemente calcolati (*expectedCount* e *count*). La funzione *ChiSquareTest* calcola inizialmente $chiValue$:

$$chiValue = \sum_{r=1}^{R} \frac{(obs[r] - exp[r])^2}{exp[r]} \tag{6.4}$$

dove R è il numero totale di elementi negli array, imposta poi i gradi di libertà *gdl*:

$$gdl = R - 1 \tag{6.5}$$

e controlla nella tabella dei valori del chi-quadro, con i gradi di libertà appena calcolati, qual'è la probabilità di avere $chiValue$. Minore è la probabilità trovata, piú significativo sarà l'attributo. Questi vengono poi ordinati per valore crescente, avendo all'inizio quelli più significativi. Nel caso in cui degli attributi abbiano lo stesso valore di probabilità vengono ordinati a seconda della *pAtt* in modo decrescente.

Distribuzione passo precedente

Rispetto a questa distribuzione, si può dire che l'attributo che presenta una distribuzione più lontana da quella del passo precedente è l'attributo da selezionare. Chiaramente questo confronto può venire fatto dal secondo passo in poi perché al primo non si ha nessun valore di confronto.
Come nel caso del confronto con la distribuzione uniforme, per capire quanto

un attributo ha distribuzione diversa da quella precedente, l'applicazione usa il test chi-quadro. Prima di procedere al calcolo vero e proprio è necessario calcolare il conteggio di ogni valore di ogni attributo, la probabilità reale con cui il valore si presenta e salvare tali dati per successivi confronti. Ora è necessario recuperare i dati del passo precedente per poter fare il test. Oltre a recuperarli è però necessario normalizzarli rispetto al nuovo numero totale di istanze, per far ciò si può moltiplicare la vecchia probabilità di ogni valore con il numero totale degli elementi attuali:

$$oldCount \quad = \quad probOld * newNumberOfElement \tag{6.6}$$

Come nel caso precedente viene calcolata la *pAtt* che verrà controllata nel caso in cui due attributi diano lo stesso risultato del test:

$$pAtt \quad = \quad \sum_{w=0}^{k1} \sum_{\substack{j=0, \\ nameOld_w = nameNew_j}}^{k2} (probOld_w - probNew_j)^2 \tag{6.7}$$

dove $k1$ e $k2$ sono il numero di valori assunti dall'attributo considerato nel passo precedente e attuale rispettivamente, *nameOld* e *nameNew* indicano i nomi del valore dell'attributo e *probOld* e *probNew* le probabilità nel passo precedente e attuale del valore dell'attributo. Inoltre se il numero di valori assunti dall'attributo nel passo prima è maggiore di quello del passo presente, a *pAtt* viene sommata la $probOld^2$ dei valori mancanti nel passo presente. Quindi viene calcolato il chi-quadro come nel caso di distribuzione uniforme, l'unica differenza è che *chiSquareTest* stavolta prende in ingresso *oldCount* e *newCount* anziché *count* e *expectedCount*. Infine gli elementi vengono ordinati come nel caso precedente.

Distribuzione normale

Il confronto rispetto alla distribuzione normale è applicabile soltanto per gli attributi numerici. Per quelli categorici si continua ad utilizzare la distribuzione uniforme. Come per il confronto con la distribuzione uniforme, si considera l'attributo migliore da selezionare quello che presenta una distribuzione più lontana da quella normale. Tale confronto è sempre ottenuto con il test del chi-quadro. Il passo iniziale è uguale a quello eseguito nei due precedenti casi: per ogni attributo, l'applicazione controlla la quantità di valori che può assumere l'attributo e per ognuno di questi valori conta il numero di istanze che

lo contengono nell'interrogazione corrente. A questo punto, per gli attributi non numerici viene calcolato il chi-quadro rispetto alla distribuzione uniforme come spiegato precedentemente, per gli attributi numerici invece è necessario calcolare media (avg) e deviazione standard (sd):

$$avg = \frac{\sum_{j=0}^{k}(elementValue_j * count_j)}{totalCount} \tag{6.8}$$

$$sd = \sqrt{\sum_{j=0}^{k}(elementValue_j - avg)^2} \tag{6.9}$$

dove $elementValue_j$ e $count_j$ indicano rispettivamente il valore j-esimo assunto dall'attributo e il numero di volte in cui tale valore compare nell'attributo, $totalCount$ indica il numero totale di istanze della selezione corrente e k il numero totale di valori assunti dall'attributo in esame. Avendo media e deviazione standard è adesso possibile calcolare il numero di volte con cui ci si aspetta compaia ogni valore dell'attributo ($expectedCount$):

$$expectedCount = totalCount * f(elementValue) \tag{6.10}$$

dove f indica la funzione di densità della distribuzione normale. Quindi vengono calcolati, come nel caso uniforme, la probabilità reale ($prob$) e la differenza tra questa e quella normale ($pAtt$) di ogni valore:

$$prob = \frac{count}{totalCount} \tag{6.11}$$

$$pAtt = \sum_{\substack{j=0,\\ count \neq 0}}^{k} (prob_j - f(elementValue))^2 \tag{6.12}$$

Viene poi calcolato il chi-quadro come per i due casi precedenti, passando alla funzione $chiSquareTest$ $expectedCount$ e $count$. Prima di far ciò è però necessario controllare che la deviazione standard dell'attributo non valga zero: se questo fosse il caso, significa che l'attributo assume un solo valore e non serve perciò analizzarlo. Come nei due casi precedenti, gli attributi vengono infine ordinati in maniera crescente basandosi sul risultato del test.

Complessità computazionale

Per quanto riguarda la complessità computazionale, nel caso di distribuzione uniforme, si possono analizzare tre parti separatamente: il calcolo del numero di valori che compaiono per ogni attributo, il calcolo del test del chi-quadro e l'ordinamento degli attributi.

- Calcolo numero valori per attributo: per ogni attributo viene calcolato il conteggio con cui ogni valore compare, è perciò necessario controllare tutte le istanze che soddisfano la query. La complessità di questa prima parte è del tipo $O(kmn)$, dove k è il numero di attributi, m è il numero di valori diversi assunto dall'attributo k_i, quindi $k * m$ è il numero totale di valori diversi considerando tutti gli attributi e n è il numero di istanze.

- Calcolo test chi-quadro: Per quanto riguarda il calcolo del chi-quadro, anche questo viene fatto per ogni attributo. Il $chiValue$ viene calcolato su un array di m elementi, dove m, come prima, indica il numero totale di valori diversi assunti dall'attributo in esame. È quindi necessario andare nella tabella del chi-quadro, cercare la riga con gradi di libertà gdl e in tale riga cercare il valore $chiValue$. Ora assumendo che il numero di righe della tabella sia j e che il numero di colonne sia h la complessità sarà $O(k(m + j + h))$, dove k è il numero di attributi.

- Odinamento attributi: L'ordinamento ha complessità $O(k * log(k))$ (con k numero di attributi) perché costruisco un nuovo array analizzando un attributo per volta, trovando la sua posizione corretta.

Quindi unendo le parti, la complessità del processo è $O(kmn + k(m + j + h) + k * log(k)) = O(kmn)$. Si può assumere che inizialmente $k * m < n$ in quanto nell'esplorazione l'utente non ha una grossa idea del dominio, quindi avrà inizialmente un numero elevato di istanze, questo non vale invece verso la fine del processo, dove il numero di istanze sarà basso, ma in questa fase la complessità sarà molto minore di quella iniziale. Si può perciò assumere che la complessità del processo sia $O(n^2)$.
Allo stesso risultato si giunge analizzando le altre distribuzione, in quanto il numero di operazioni da fare è lo stesso.

6.2.4 Selezione valore attributo

Una volta calcolato l'attributo da raffinare, l'applicazione chiede all'utente che valore deve inserire per l'attributo, quindi aggiunge alla query completa la nuova condizione. Se all'utente non interessa specificare meglio quell'attributo, l'applicazione gli presenta quello che viene considerato il secondo più importante e così via fino ad esaurire gli attributi. Una volta scelto il valore dell'attributo, alla query viene aggiunta questa nuova condizione e il processo si ripete.

6.2.5 Verifica dei risultati

Per verificare che l'attributo selezionato sia quello che ha test migliore, l'applicazione consente di visualizzare le distribuzioni degli attributi tramite un grafico. Su di esso, per ogni valore dell'attributo viene riportato il numero di istanze che hanno tale valore. Ad esempio dopo aver scritto la query più generale possibile (SELECT * FROM service), l'applicazione dice di raffinare il campo "Country". Assegnando a questo valore "Italy", otteniamo dei grafici, alcuni riportati in figura 6.3. Facendo una nuova ricerca rispetto alla distribuzione uniforme otteniamo i risultati della tabella 6.1.

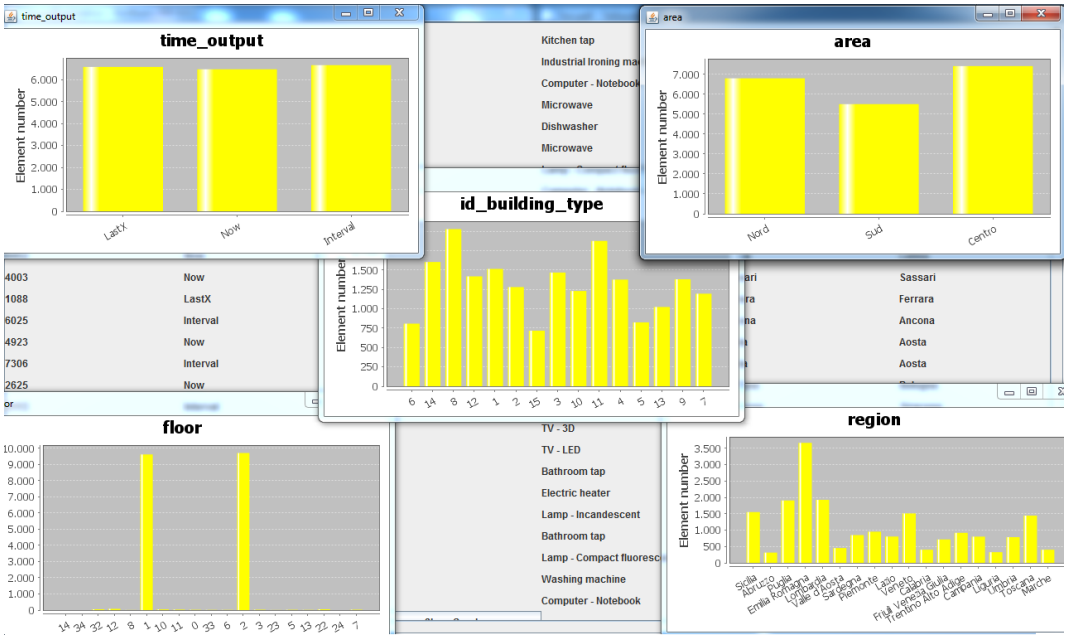


Figura 6.3: Grafici selezione Contry='Italy'

Analizzando i grafici ci si aspetta che "time_output" abbia una distribuzione

Attributo	Risultato test chi-quadro
floor	0.0
region	0.0
description	0.0
id_building_type	0.0
area	0.0
city	0.0
provincia	0.0
id_sensortype	0.00126
minute	0.22739
time_output	0.25798
second	0.32340
hour	0.74869
country	1000.0
day	1000.0
month	1000.0
year	1000.0

Tabella 6.1: Risultati chi-quadro rispetto alla distribuzione uniforme

simile ad un'uniforme, cosa che non vale per gli altri, infatti il risultato del chi-quadro riportato in tabella 6.1 da un valore alto per quell'attributo. Da notare che alcuni attributi hanno test chi-quadro uguale a 1000. Questo indica che tali attributi presentano un solo valore e quindi la loro selezione sarebbe inutile.

6.3 Risultati

Per capire se l'applicazione SensoriExploration funzionasse in maniera corretta, questa è stata testata con dei casi significativi. Di seguito ne vengono mostrati due, i quali saranno analizzati per ogni tipo di distribuzione. In ognuno dei due esempi mostrati, vengono descritti due diversi tipi di utenti: un utente che non ha nessuna conoscenza del dominio, quindi partirà inserendo la query generale ed un utente che invece conosce il dominio (anche se in misura minima, altrimenti l'uso dell'applicazione SensoriExploration non sarebbe spiegato), il

quale inserirà una query un po' più completa.

6.3.1 Caso1: servizi di Palermo che monitorano il consumo energetico di freezer

In questo primo caso, l'utente vuole recuperare i servizi che monitorano i sensori installati a Palermo che controllano il consumo energetico dei dispositivi freezer. Inoltre l'utente non ha nessuna conoscenza del dominio SeNSori. Per cominiciare allora scriverà la quey più generale possibile: *SELECT * FROM service*, che restituisce tutti i servizi presenti (figura 6.4).

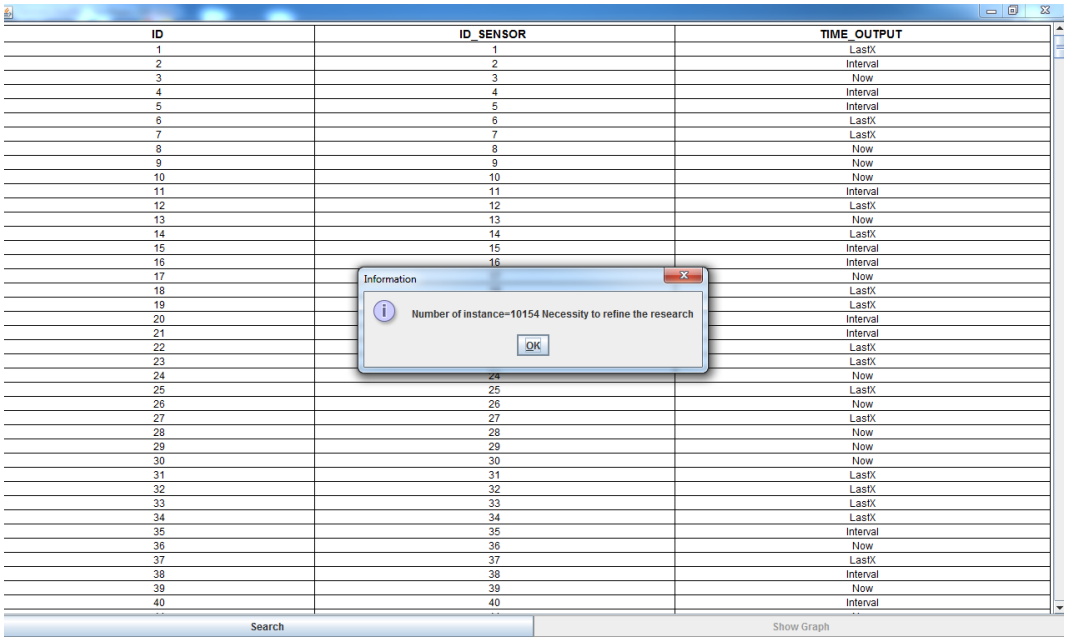


Figura 6.4: Servizi recuperati digitando la query "generale"

La figura mostra inoltre che l'applicazione SensoriExploration suggerisce di raffinare la ricerca in quanto il numero di istanze che soddisfano la query è maggiore della soglia inserita, che è stata impostata col valore 20. Una volta che l'utente sceglie di raffinare la ricerca (selezionando il tasto "Search"), l'applicazione suggerisce per ogni distribuzione di raffinare il campo "country" (figura 6.5).
Verrà selezionato "Italy" e l'applicazione indicherà ancora che il numero di servizi è superiore alla soglia. Stavolta l'applicazione suggerisce diversi raffinamenti per le distribuzioni: se la distribuzione di confronto scelta è la uniforme o la normale viene detto di specificare "floor" (figura 6.6), altrimenti, se è quella

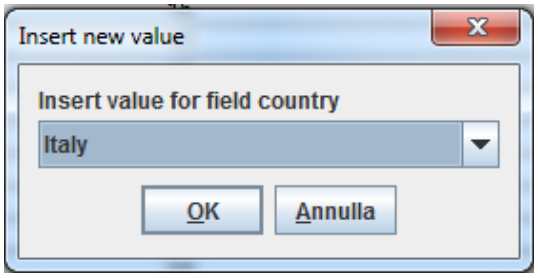


Figura 6.5: Suggerimento di raffinare il campo ”country”

rispetto al valore precedente l'attributo scelto è ”area” (figura 6.7).

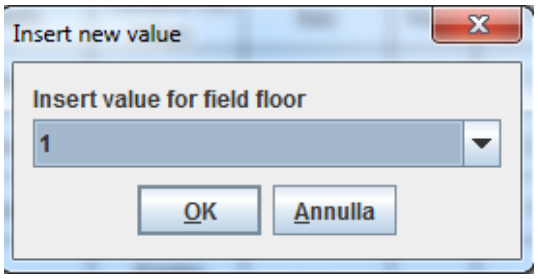


Figura 6.6: Suggerimento di raffinare il campo ”floor”

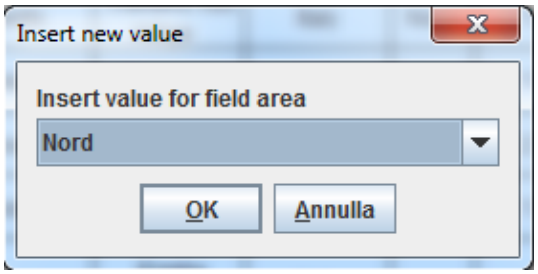


Figura 6.7: Suggerimento di raffinare il campo ”area”

Tuttavia il raffinamento del piano non interessa, quindi viene cliccato il tasto ”Annulla” e l'applicazione suggerisce il secondo attributo che ritiene migliore, cioè ”measurand”, che indica il tipo di misura monitorata dal sensore. Per questo attributo l'utente seleziona ”Power consumption - Average”, mentre per l'area, suggerita nel caso di distribuzione rispetto al vecchio valore, sceglierà ”Sud”. Il numero di istanze risulta ora molto minore dal numero di istanze iniziali, vicino alle 20000 (figure 6.8 e 6.9), ma è ancora troppo elevato, cosa che fa proseguire la ricerca.
Per le distribuzioni uniforme e normale il sistema propone ancora di raffinare il campo ”floor” e dopo averlo scartato il campo ”region”. Per la distribuzione rispetto al vecchio valore verrà invece suggerito subito il campo ”region”. La

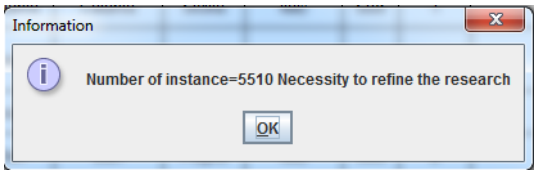


Figura 6.8: Numero di istanze dopo aver raffinato il campo "area"

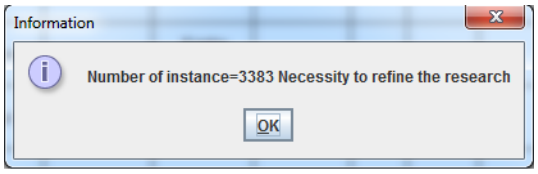


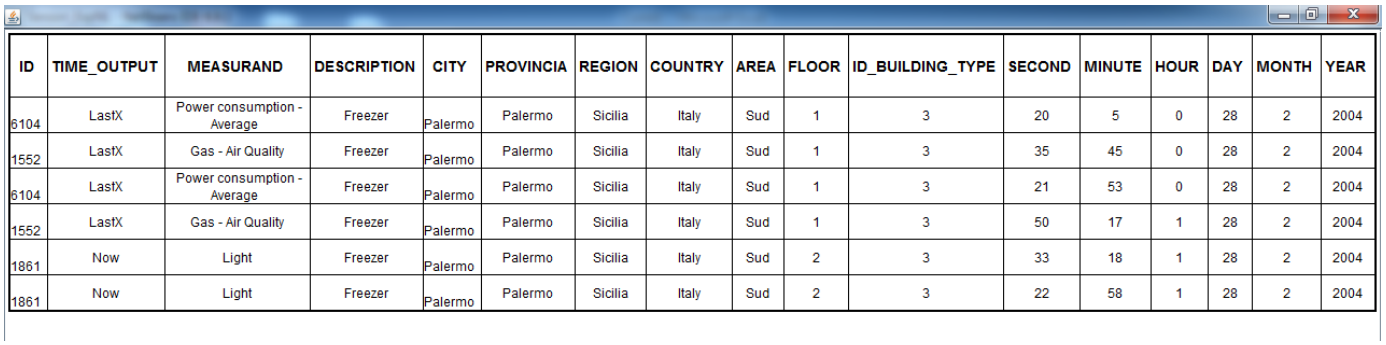
Figura 6.9: Numero di istanze dopo aver raffinato il campo "measurand"

scelta del valore sarà "Sicilia". Di nuovo, il numero di istanze cala sensibilmente, ma ancora una volta è superiore della soglia, infatti per la distribuzione rispetto al vecchio valore vale 1552 e per le altre due 238. Proseguiamo il raffinamento. Per le distribuzioni uniforme e normale viene indicato l'attributo "description", mentre per quella rispetto al vecchio valore "città". Una volta selezionato il valore "Freezer" per l'attributo "description" otteniamo il risultato mostrato in figura 6.10.

ID	TIME_OUTPUT	MEASURAND	DESCRIPTION	CITY	PROVINCIA	REGION	COUNTRY	AREA	FLOOR	ID_BUILDING_TYPE	SECOND	MINUTE	HOUR	DAY	MONTH	YEAR
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	20	5	0	28	2	2004
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	21	53	0	28	2	2004

Figura 6.10: Servizi ottenuti alla fine del processo utilizzando la distribuzione uniforme o la distribuzione normale

Per la distribuzione del vecchio valore, che indicava di inserire il valore di "città", viene selezionato "Palermo" e ancora abbiamo 424 istanze, cosa che fa proseguire nella ricerca. Allora viene proposto di inserire il tipo di edificio, ma questo non interessa, quindi l'attributo suggerito diventa "description". Impostato "Freezer", si ottiene il risultato di figura 6.11.
È stato così ottenuto un risultato contenente sei servizi. Essendo il numero di servizi rimasti minore della soglia impostata, l'applicazione dice di aver trovato il risultato. Tuttavia si nota dalla figura 6.11 che sono stati proposti anche servizi che non misurano il consumo energetico, in quanto tale raffinamento non è stato chiesto dall'applicazione. Questo non è comunque un problema,



ID	TIME_OUTPUT	MEASURAND	DESCRIPTION	CITY	PROVINCIA	REGION	COUNTRY	AREA	FLOOR	ID_BUILDING_TYPE	SECOND	MINUTE	HOUR	DAY	MONTH	YEAR
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	20	5	0	28	2	2004
1552	LastX	Gas - Air Quality	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	35	45	0	28	2	2004
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	21	53	0	28	2	2004
1552	LastX	Gas - Air Quality	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	50	17	1	28	2	2004
1861	Now	Light	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	2	3	33	18	1	28	2	2004
1861	Now	Light	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	2	3	22	58	1	28	2	2004

Figura 6.11: Servizi ottenuti alla fine del processo utilizzando la distribuzione rispetto al valore calcolato nel passo precedente

in quanto impostando la soglia ad un livello sufficientemente basso, i servizi possono essere analizzati ed esclusi se non di interesse. Se non si è ancora contenti del risultato si può comunque proseguire con il raffinamento oppure si può ripetere tutto il processo impostando una soglia minore. Sempre dalla figura 6.11 si nota che i servizi vengono presentati più di una volta. Questo fatto si verifica perché sono stati utilizzati per leggere il valore dei sensori in diversi istanti di tempo. Questa informazione è stata mantenuta perché può darsi che sia significativa per qualche ricerca (ad esempio si vuole recuperare il servizio usato in una specifica data). Un'ultima nota merita il fatto che l'applicazione può suggerire in passi successivi un attributo che è stato scartato precedentemente. Questo perché può darsi che un utente usi l'esplorazione per approfondire la conoscenza di un dominio. Quindi può accadere che un'informazione che non ritiene utile in un certo momento lo diventi in passi successivi. Ad esempio il piano di un edificio suggerito dopo aver specificato il paese può non essere rilevante, ma una volta specificata la città e il tipo di edificio lo può diventare.
La tabella 6.2 riassume i passi di raffinamento proposti dall'applicazione SensoriExploration appena descritti.
Ad ogni passo, per ogni distribuzione analizzata, sono mostrati gli attributi da raffinare proposti, il valore da inserire ed il numero di istanze rimangono valide. Gli attributi lasciati con righe vuote indicano che quell'attributo non era interessante e quindi si è passati a quello successivo proposto.

Passo	Distribuzione Uniforme			Distribuzione normale			Distribuzione vechio valore		
	Attr	Val	Ist	Attr	Val	Ist	Attr	Val	Ist
1	country	Italy	19731	country	Italy	19731	country	Italy	19731
2	floor measure	PowerCons-avg	3383	floor measure	PowerCons-avg	3383	area	Sud	5510
3	floor region	Sicilia	238	floor region	Sicilia	238	region	Sicilia	1552
4	desription	Freezer	2	description	Freezer	2	city	Palermo	424
5							building description	Freezer	6

Tabella 6.2: Passi di raffinamento degli attributi proposti nel caso 1

Utente con buona conoscenza del dominio SeNSori

In questo caso supponiamo che l'utente abbia una certa conoscenza del dominio SeNSori. Quindi, sempre per rispondere alla richiesta di recuperare i servizi di Palermo che monitorano il consumo energetico di freezer, scriverà una query un po' più specifica, in particolare inserendo la città di interesse:
SELECT service.id
FROM service, sensors, installed_ devices, locations, area
WHERE service.id_ sensor = sensors.id AND sensors.id_ installeddevice = installed_ devices.id AND installed_ devices.id_ location = locations.id AND locations.id_ area = area.id AND area.city = 'Palermo'
La figura 6.12 riporta il risultato che si ha lanciando l'applicazione SensoriExploration con questa query.
Iniziando il processo di raffinamento, tutte le distribuzioni utilizzate suggeriscono di selezionare la misura da monitorare. Scegliendo il consumo energetico medio, l'applicazione torna 60 istanze. Ora, per quanto riguarda la distribuzione rispetto al vecchio valore, l'attributo da inserire è la descrizione. Una volta impostata si ottengono i servizi voluti, come mostrato in figura 6.13.
Per quanto riguarda le distribuzioni uniforme e normale, viene chiesto di inserire il tipo di edificio, cosa che non interessa. Cliccando "Annulla" il sistema proporrà allora la descrizione. Inserendo *Freezer* si ottiene lo stesso risultato della distribuzione rispetto al vecchio valore mostrato in figura 6.13.
La tabella 6.3 riassume i passi di raffinamento proposti dall'applicazione SensoriExploration per questo esempio.

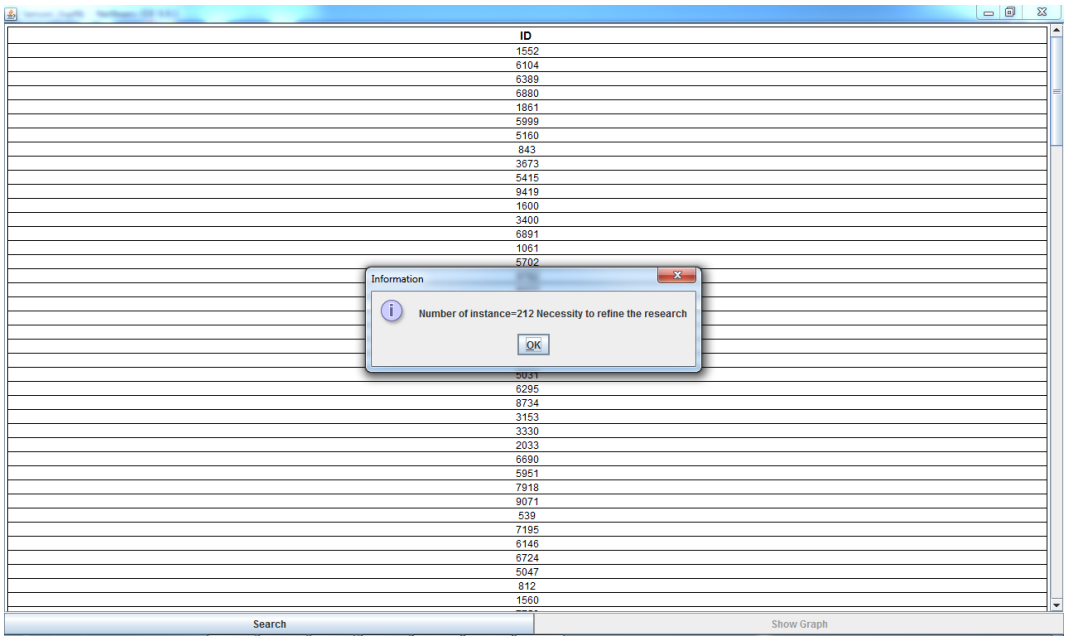


Figura 6.12: Servizi ottenuti lanciando la query per recuperare i servizi della città di Palermo

ID	TIME_OUTPUT	MEASURAND	DESCRIPTION	CITY	PROVINCIA	REGION	COUNTRY	AREA	FLOOR	ID_BUILDING_TYPE	SECOND	MINUTE	HOUR	DAY	MONTH	YEAR
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	20	5	0	28	2	2004
6104	LastX	Power consumption - Average	Freezer	Palermo	Palermo	Sicilia	Italy	Sud	1	3	21	53	0	28	2	2004

Figura 6.13: Servizi ottenuti alla fine del processo utilizzando la distribuzione rispetto al valore calcolato nel passo precedente partendo dalla conoscenza della città

Passo	Distribuzione Uniforme			Distribuzione normale			Distribuzione vechio valore		
	Attr	Val	Ist	Attr	Val	Ist	Attr	Val	Ist
1	measure	PowerCons-avg	60	measure	PowerCons-avg	60	measure	PowerCons-avg	60
2	building description	Freezer	2	building description	Freezer	2	description	Freezer	2

Tabella 6.3: Passi di raffinamento degli attributi proposti nel caso 1 in cui l'utente ha una buona conoscenza del dominio

6.3.2 Caso2: servizio di Cremona legato al sensore di tipo luce dell'edificio 5 che monitora il consumo di lampade fluorescenti utilizzato alle ore 1:43:46

In questo caso l'utente vuole recuperare il servizio che monitora il sensore installato a Cremona che controlla il consumo di luce di una lampada fluorescente

ed è stato utilizzato alle ore 1:43:46 (ore, minuti, secondi). Per questa prima parte, si suppone che l'utente non conosca il dominio in esame, quindi i risultati ottenuti dopo aver inserito la query "generale" saranno come quelli riportati in figura 6.4. La tabella 6.4 mostra i passi di raffinamento degli attributi che l'applicazione propone.

Passo	Distribuzione Uniforme			Distribuzione normale			Distribuzione vechio valore		
	Attr	Val	Ist	Attr	Val	Ist	Attr	Val	Ist
1	country	Italy	19731	country	Italy	19731	country	Italy	19731
2	floor measure	Light	2729	floor measure	Light	2729	area	Nord	6803
3	floor region	Lombardia	319	floor region	Lombardia	319	region	Lombardia	1927
4	floor description	LampFluo	116	floor description	LampFluo	116	city	Cremona	423
5	city	Cremona	102	city	Cremona	102	floor description	LampFluo	269
6	floor building	5	14	floor building	5	14	floor measure	Light	102
7	floor hour	1	8	floor time_out hour	1	8	hour	1	54
8	floor time_out minute	43	2	floor time_out minute	43	2	building	5	8
9	second	46	1	second	46	1	floor minute	43	2
10							second	46	1

Tabella 6.4: Passi di raffinamento degli attributi proposti nel caso 2

La figura 6.14 mostra il servizio trovato alla fine dei passi con le tre distribuzioni.

ID	TIME_OUTPUT	MEASURAND	DESCRIPTION	CITY	PROVINCIA	REGION	COUNTRY	AREA	FLOOR	ID_BUILDING_TYPE	SECOND	MINUTE	HOUR	DAY	MONTH	YEAR
15	Interval	Light	Lamp - Compact fluorescent	Cremona	Cremona	Lombardia	Italy	Nord	32	5	46	43	1	28	2	2004

Figura 6.14: Servizio restituito dall'applicazione SensoriExploration alla fine del processo

Utente con buona conoscenza del dominio SeNSori

Come nell'esempio precedente viene ora analizzato il caso in cui l'utente abbia una buona conoscenza del dominio. In particolare l'utente inserisce "Cremona"

come città e l'identificativo 5 dell'edificio. La query sarà perciò:
SELECT service.id
FROM service, sensors, installed_ devices, locations, area, building
WHERE service.id_ sensor = sensors.id AND sensors.id_ installeddevice = installed_ devices.id AND installed_ devices.id_ location = locations.id AND locations.id_ area = area.id AND locations.id_ building = building.id AND area.city = 'Cremona' AND id_ building_ type = '5'
La figura 6.15 mostra i risultati ottenuti inserendo questa query.

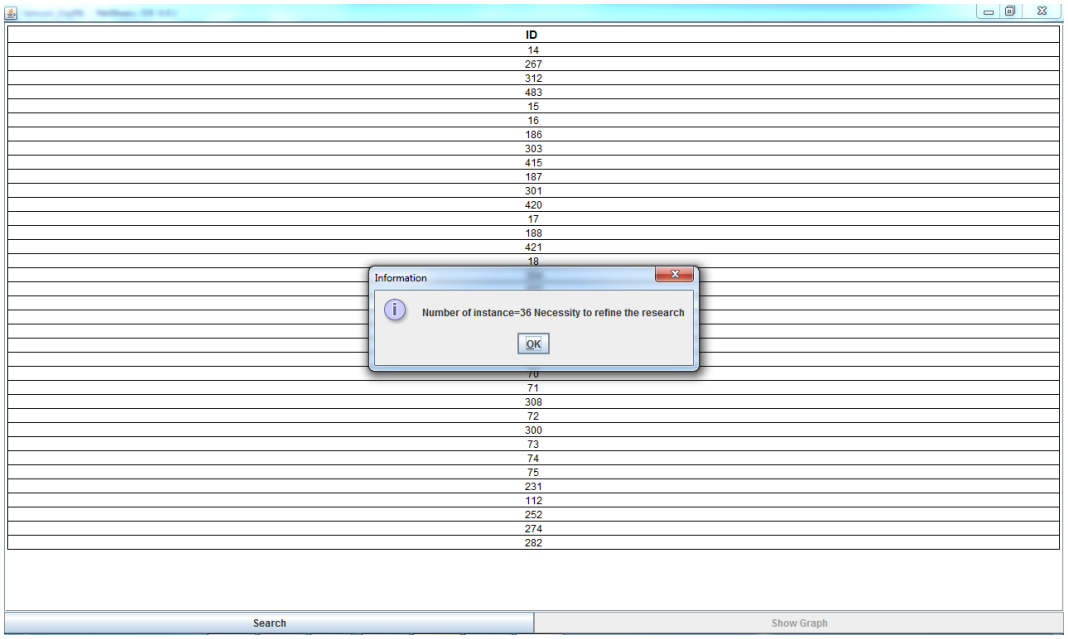


Figura 6.15: Servizi della città di Cremona i cui sensori sono installati in edifici di tipo 5

La tabella 6.5 mostra i passi di raffinamento proposti dall'applicazione SensoriExploration per le tre distribuzioni.
Il servizio ottenuto è lo stesso raggiunto nel caso in cui l'utente non ha conoscenza del dominio ed è mostrato in figura 6.14.

6.4 Analisi dei risultati

Per analizzare i risultati vengono presi in considerazione gli esempi mostrati nella sezione precedente. Una prima considerazione generale riguarda il fatto che, come è logico aspettarsi, più un utente ha conoscenza del dominio e quindi più specifica sarà la query inserita inizialmente e meno passi gli algoritmi

Passo	Distribuzione Uniforme			Distribuzione normale			Distribuzione vechio valore		
	Attr	Val	Ist	Attr	Val	Ist	Attr	Val	Ist
1	description	LampFluo	35	description	LampFluo	35	description	LampFluo	35
2	measure	Light	14	measure	Light	14	floor		
							hour	1	15
3	floor			floor			minute	43	3
	hour	1	8	time_out					
				hour	1	8			
4	floor			floor			measure	Light	2
	time_out			time_out					
	minute	43	2	minute	43	2			
5	second	46	1	second	46	1	second	46	1

Tabella 6.5: Passi di raffinamento degli attributi proposti nel caso 2 in cui l'utente ha una buona conoscenza del dominio

eseguono per arrivare al risultato. Per confrontare invece i risultati ottenuti dalle diverse distribuzioni, queste vengono analizzate a coppie: prima la distribuzione uniforme con la distribuzione normale e poi la distribuzione uniforme con quella del passo precedente.

6.4.1 Distribuzione uniforme vs distribuzione normale

Come si vede dagli esempi mostrati, i raffinamenti calcolati attraverso la distribuzione uniforme e quelli attraverso la distribuzione normale non hanno grandi differenze. Questo è dovuto al fatto che per gli attributi categorici, la distribuzione normale non può essere calcolata. Scegliendo questa distribuzione, l'applicazione SensoriExploration la utilizza come confronto per gli attributi numerici, ma continua ad utilizzare la distribuzione uniforme per quelli categorici. Essendo questi ultimi presenti in maggior frequenza ed essendo considerati più rilevanti nella maggior parte dei casi, l'applicazione non nota le differenze utilizzando queste due distribuzioni. Tuttavia, il secondo esempio mostra che il suggerimento degli attributi numerici avviene dopo nella distribuzione normale rispetto all distribuzione uniforme. Questo avviene nel passo 7 nel caso in cui l'utente non conosca il dominio e nel passo 3 nel caso in cui lo conosca almeno minimamente. Analizzando i valori del test del chi-quadro in questi passi, abbiamo che per l'attributo "hour", che è l'attributo che da differenze di posizione nel suggerimento, il valore è di 0.59 per la distribuzione uniforme e di 0.75 per la normale. Non è però corretto dire che utilizzando la distribuzione uniforme, si ottengono per attributi numerici sempre valori di test chi-quadro

più elevati: ad esempio il valore dell'attributo "minute" nel primo passo del secondo esempio vale 0.24 se si usa la distribuzione uniforme e 0.13 se si usa la normale. In conclusione l'utilizzo della distribuzione uniforme non presenta grandi differenze rispetto all'uso della normale nel dominio considerato. In certi casi una può essere migliore dell'altra, ma comunque sempre in maniera limitata.

6.4.2 Distribuzione uniforme vs distribuzione passo precedente

Dal momento che la scelta tra distribuzione uniforme e normale porta risultati simili, il confronto con la distribuzione rispetto al passo precedente viene fatto solo con la uniforme. Dagli esempi si nota come la distribuzione uniforme porta al raggiungimento del risultato in meno passi. Però rispetto alla distribuzione del passo precedente propone attributi da raffinare in modo meno "sensato": ad esempio nel primo caso dopo aver chiesto di impostare un valore per il paese, chiede di inserire il piano dell'edificio e poi la misura monitorata dal sensore. Al passo successivo chiede di inserire la regione senza aver detto niente dell'area, che ricopre una zona più ampia. Questo fatto invece non si verifica nella distribuzione del passo precedente: in entrambi gli esempi mostrati viene proposto di raffinare prima tutti gli attributi dell'area (paese, area, regione, città). Per gli attributi del tempo nel secondo esempio questa differenza non si nota più perché gli attributi da suggerire sono ormai pochi. Sempre dagli esempi si può notare che pur impiegando più passi per arrivare alla soluzione, il numero di attributi suggeriti dalla distribuzione del passo precedente è minore rispetto a quelli della distribuzione uniforme, la quale suggerisce quindi più attributi che non sono ritenuti utili. Le tabelle 6.6 e 6.7 mostrano i valori di chi-quadro ottenuti dopo aver selezionato il paese negli esempi mostrati precedentemente. La prima tabella riporta il valore di chi-quadro calcolato facendo uso della distribuzione uniforme, mentre la seconda utilizzando la distribuzione rispetto al passo precedente.

Confrontando i valori del chi-quadro ottenuti, si nota che partendo da una stessa situazione, utilizzando la distribuzione rispetto al passo precedente si riesce a differenziare meglio i valori rispetto all'uso della distribuzione uniforme. Inoltre si vede che attributi considerati importanti con l'uso della distribuzione

Attributo	Valore chi-quadro
floor	0
measure	0
region	0
description	0
building	0
area	0
city	0
minute	0.23
time_output	0.26
second	0.32
hour	0.75

Tabella 6.6: Valori chi-quadro dopo aver selezionato il paese nel caso di distribuzione uniforme

Attributo	Valore chi-quadro
area	$2.22 * 10^{-16}$
region	$3.71 * 10^{-9}$
floor	$3.89 * 10^{-7}$
city	0.12
hour	0.95
time_output	1
building	1
measure	1
description	1
second	1
minute	1

Tabella 6.7: Valori chi-quadro dopo aver selezionato il paese nel caso di distribuzione rispetto al passo precedente

uniforme non lo sono per quella rispetto al passo precedente. Alcuni esempi sono gli attributi "measure" e "description" che hanno valore di chi-quadro pari a zero nella prima e pari a uno nella seconda. Ciò accade perché dato l'alto numero di valori che hanno un conteggio molto diverso tra loro (figura 6.16), la distribuzione uniforme ne risulta influenzata, cosa che non accade nell'altra

distribuzione perché i valori hanno un conteggio simile a quelli avuti nel passo precedente (figura 6.17).

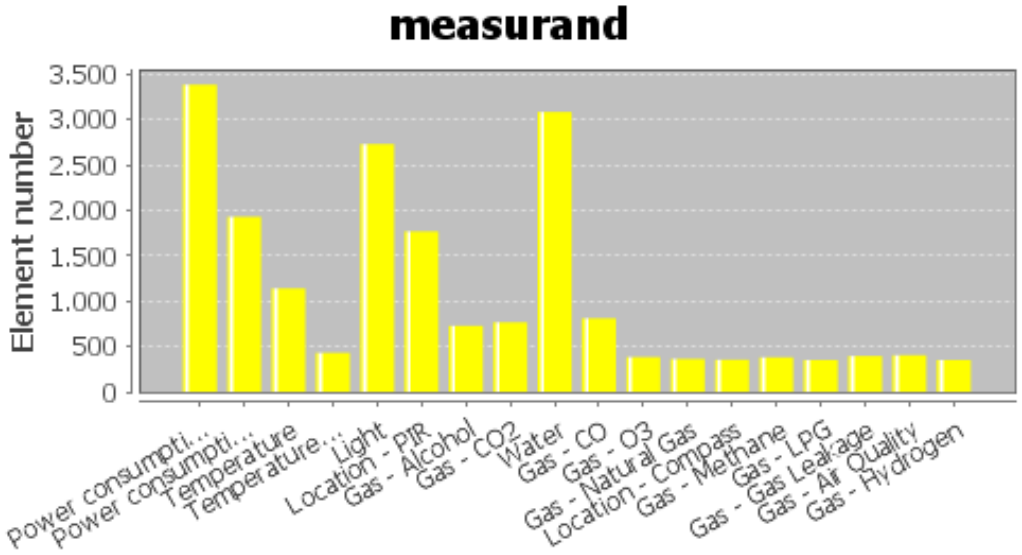


Figura 6.16: Grafico dei valori dell'attributo "measure"

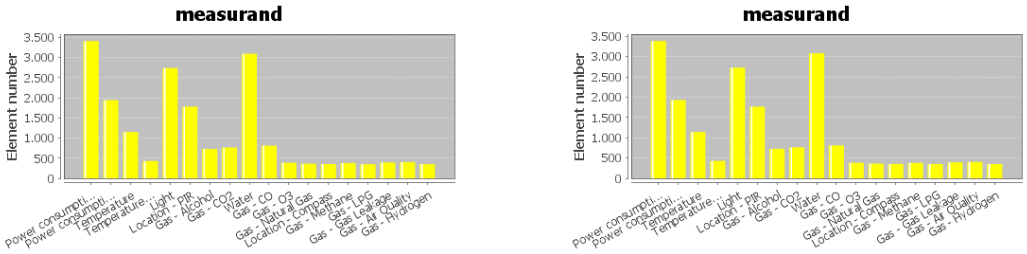


Figura 6.17: Grafico dei valori dell'attributo "measure" prima e dopo la selezione del paese

6.5 Sommario

In questo capitolo è stato descritto cosa è l'esplorazione e perché serve. È stata quindi spiegata l'applicazione sviluppata, specificando le fasi che compie per consentire all'utente di esplorare il dominio. È stata posta particolare attenzione al calcolo eseguito per decidere quale attributo selezionare per raffinare la ricerca, evidenziando che l'applicazione può utilizzare tre distribuzioni di confronto per eseguire il test chi-quadro: quella uniforme, quella normale e quella rispetto al vecchio valore. Infine sono stati mostrati degli esempi, grazie

ai quali sono state confrontate le distribuzioni utilizzate. Da questi confronti è emerso che l'uso della distribuzione uniforme porta risultati simili all'uso della normale. Mentre rispetto alla distribuzione del passo precedente è stato notato che la distribuzione uniforme richiede meno passi per arrivare al risultato finale, ma suggerisce più attributi, li differenzia in modo peggiore nel test chi-quadro e risulta più sensibile alla presenza di valori di attributi con conteggio molto diverso.

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Conclusioni

In questa tesi sono state discusse alcune tecniche di recupero di servizi, pensate per soddisfare utenti con diverse conoscenze. Inizialmente è stato mostrato lo stato dell'arte rispetto alle tematiche trattate, relativo quindi ai concetti di recupero di istanze inserendo parole chiave e di esplorazione, spiegando perché quest'ultima è considerata diversa dalle tecnologie già esistenti.
Sono stati illustrati gli aspetti e gli strumenti utilizzati per lo sviluppo del lavoro. Sono state presentate le ontologie, indicando che in esse la conoscenza è espressa attraverso la logica e ciò permette la possibilità di utilizzare il servizio di reasoning. Quindi è stata presentata la Desciption Logic, cioè la logica utilizzata nella descrizione di ontologie. È stato poi presentato il semantic web, un modo di rappresentare le informazioni in modo che siano capite anche da agenti, spiegando le parti di cui è composto. Per quanto riguarda gli strumenti utilizzati per lo sviluppo dell'applicazione sono stati spiegati Protege (un'editor che permette di operare sulle ontologie), Jena (un framework java utilizzato per lo sviluppo di applicazione orientate al web semantico) e Postgres (un sistema per la gestione di database relazionali).
È stato presentato il progetto SeNSori, utilizzato come scenario applicativo per lo sviluppo dell'applicazione ServiceRetrieval, la quale in questo progetto si occupa del recupero di servizi. Di questo sono stati spiegati gli aspetti principali, l'architettura e le ontologie di cui è costituito.
È stata descritta l'infrastruttura software sviluppata per l'applicazione

ServiceRetrieval. Inizialmente è stata spiegata la parte riguardante il recupero di istanze attraverso la selezione di query poste in linguaggio naturale e poi sul recupero di servizi attraverso l'esplorazione. Nella prima parte è stato inizialmente presentato il progetto SAFE da cui si è preso spunto per creare Sensori_NL. Quindi si è passati alla descrizione dell'architettura di Sensori_NL. È stata illustrata l'ontologia contenente le informazioni e utilizzata anche per associare le parole chiave alle query presenti nel file patternDB.xml, il quale contiene appunto le associazioni tra query in linguaggio naturale e query in linguaggio SPARQL utilizzate per recuperare i dati dall'ontologia. In seguito è stata descritta l'applicazione sviluppata, spiegando il funzionamento del suggerimento di parole chiave, del sistema di ranking di query proposte in linguaggio naturale, del ricavo di query in linguaggio SPARQL partendo da quelle selezionate in linguaggio naturale e del calcolo dei risultati. Vengono anche descritte le funzioni aggiuntive sviluppate come l'inserimento di query direttamente in linguaggio SPARQL e l'immissione di nuovi pattern. Infine sono presentati i risultati dell'applicazione per mezzo di esempi.
Per quanto riguarda la seconda parte, quella riguardante il recupero di servizi attraverso l'eplorazione, dopo aver spiegato il concetto di esplorazione e le differenze rispetto alle tecniche classiche, viene descritta l'architettura dell'applicazione SensoriExploration. Quindi viene presentato l'inserimento della query iniziale e vengono spiegate le operazioni eseguite per calcolare quale sarà l'attributo da proporre da raffinare. Per questo scopo è stato utilizzato il test chi-quadro, nel quale le distribuzioni dei valori degli attributi vengono confrontate con delle distribuzioni di riferimento. Come per la parte precedente, vengono mostrati i risultati attraverso esempi. Infine gli stessi esempi vengono utilizzati per analizzare i risultati ottenuti dalle diverse distribuzioni.

Per quanto riguarda l'applicazione Sensori_NL, i risultati mostrati fanno capire che l'applicazione consente effettivamente di recuperare i servizi voluti. Più parole chiave relative ad un servizio vengono inserite e più la query in linguaggio naturale associata compare in cima alla lista. Non è fondamentale inserire tutte le parole chiave relative al servizio per recuperarlo e anche l'inserimento di qualche parola errata non è necessariamente un problema.
Riguardo l'applicazione SensoriExploration anche in questa, indipendentemen-

te dalla distribuzione utilizzata, si arriva al risultato voluto. Le diverse distribuzioni portano al raggiungimento del risultato in modo diverso, ma nessuno può essere definto sbagliato: alcune in certe situazioni raggiungono il risultato prima rispetto ad altre, ma in altre occasioni possono portate ad un numero superiore di suggerimenti errati: ad esempio è stato mostrato nel caso2 che con la distribuzione uniforme si è arrivati al risultato in meno passi rispetto alla distribuzione del passo precedente, tuttavia dopo il suggerimento dell'attributo "country" viene chiesto "region" senza passare da "area". Se per esempio si volessero recuperare i servizi del nord Italia il suggerimento di "region" sarebbe inutile e perciò sarebbe necessario chiederne un altro. Dal momento che, come è stato mostrato, la distribuzione rispetto al passo precedente propone raffinamenti successivi più "sensati" rispetto alle altre analizzate e riesce a differenziare meglio i valori ottenuti dal test chi-quadro, questa è quella da preferire.

7.2 Sviluppi futuri

Per quanto riguarda l'applicazione Sensori_NL possibili sviluppi sono:

- Utilizzare un sistema di autenticazione, in questo modo è possibile capire alcune caratteristiche dell'utente che sta utilizzando l'applicazione, come per esempio il tipo o il luogo in cui si trova. Così facendo si riescono a categorizzare gli utenti. È possibile che certe categorie di utenti utilizzino sempre un sotto insieme di interrogazioni. È così possibile tenere in considerazione questo aspetto per modificare il sistema di ranking, in modo che queste interrogazioni abbiano maggiore priorità.

- Introdurre un sistema di suggerimento di parole chiave "evoluto", che in base alle parole chiave già inserite ne propone nuove. In questo modo un utente potrebbe arrivare al recupero del servizio desiderato in un tempo minore.

Riguardo l'applicazione SensoriExploration gli sviluppi possibili sono:

- Introdurre nuove distribuzioni di confronto, utile in particolare su domini che presentano un insieme di attributi di tipo numerico più ampio rispet-

to al dominio studiato. In questo modo si potrebbero ottenere risultati migliori.

- Al momento della presentazione dell'attributo da raffinare, si potrebbe pensare ad un modo per presentare i valori per quell'attributo ordinati secondo una rilevanza. Un modo potrebbe essere ancora l'utilizzo di un sistema di autenticazione: in base ai valori selezionati precedentemente da utenti della stessa categoria si possono proporre i valori in un determinato ordine.

Bibliografia

[1] Apache jena. http://jena.apache.org/.

[2] D2r server: Accessing databases with sparql and as linked data. http://d2rq.org/d2r-server.

[3] Postgresql. http://www.postgresql.org/.

[4] Protégé. http://protege.stanford.edu/.

[5] L. Lim A. Kementsietsidis and M. Wang. Supporting ontology-based keyword search over medical databases. In *American Medical Informatics Association Symp.*, pages 409–413, 2008.

[6] D. Beckett. Rdf/xml syntax specification (revised). http://www.w3.org/TR/rdf-syntax-grammar/.

[7] T. Berners-Lee and D. Connolly. Notation3 (n3): A readable rdf syntax. http://www.w3.org/TeamSubmission/n3/.

[8] D. Brickley and R. V. Guha. Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/.

[9] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *Proc. of the 35th Intl Conf. on Management of Data*, pages 707–718, 2009.

[10] E. Prud'hommeaux D. Beckett, T. Berners-Lee and G. Carothers. Turtle. http://www.w3.org/TR/turtle/.

[11] D. Karger D. Huynh and R. Miller. Exhibit: lightweight structured data publishing. In *Proc. of the 16th international conference on World Wide Web, WWW '07*, pages 737–746, 2007.

[12] J. Hobbs O. Lassila D. McDermott S. McIlraith S. Narayanan M. Paolucci B. Parsia T. Payne E. Sirin N. Srinivasan D. Martin, M. Burstein and K. Sycara. Owl-s: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/.

[13] E. Quintarelli L. Tanca E. Baralis, P. Garza. Answering xml queries by means of data summaries. In *ACM Transaction on information systems, ACM*, volume 25, pages 1–33, 2007.

[14] D. L. McGuinness D. Nardi F. Baader, D. Calvarese and P. F. Patel-Schneider. *The Description Logic Handbook. Theory, Implementation and Applications.* 2nd edition.

[15] S. Ferré and O. Ridoux. Introduction to logical information systems. In *Information Processing & Management*, volume 40, pages 383–419, 2014.

[16] C. Nakhe S. Chakrabarti G. Bhalotia, A. Hulgeri and S. Sudarshan. Keyword searching and browsing in databases using banks. In *18th Intl Conf. on Data Engineering*, pages 431–440, 2002.

[17] L. Tanca G. Orsi and E. Zimeo. Keyword-based, context-aware selection of natural language query patterns. In *14th International Conference on Extending Database Technology*, pages 189–200, 2011.

[18] M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies.* McGraw-Hill, 2009.

[19] T. Finin H. Chen, F. Perich and A. Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services*, pages 258–267, 2004.

[20] V. Hristidis H. Hwang and Y. Papakonstantinou. Objectrank: A system for authority-based search on databases. In *26th Intl Conf. on Management of Data*, pages 796–798, 2006.

[21] S. Harris and A. Seaborne. Sparql 1.1 query language. http://www.w3.org/TR/sparql11-query/.

[22] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *28th Intl Conf. on Very Large Data Bases*, pages 670–681, 2002.

[23] H. Boley S. Tabet B. Grosof I. Horrocks, P. F. Patel-Schneider and M. Dean. Swrl: A semantic web rule language. combining owl and ruleml. http://www.w3.org/Submission/SWRL/.

[24] W. Hall I. Popov, M.Schraefel and N. Shadbolt. Connecting the dots: A multi-pivot approach to data exploration. In *The Semantic Web-ISWC 2011*, pages 553–568, 2011.

[25] J. Bernad J. A. Royo, E. Mena and A. Illarramendi. Searching the web: From keywords to semantic queries. In *3rd Intl Conf. on Information Technology and Applications*, pages 244–249, 2005.

[26] K. Li M. Hearst K. Yee, K. Swearingen. Faceted metadata for image search and browsing. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, pages 401–408, 2003.

[27] M. Kifer and H. Boley. Rif overview (second edition). http://www.w3.org/TR/rif-overview/.

[28] G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts/.

[29] L. Tanca M. Mazuran, E. Quintarelli. Data mining for xml query-answering support. In *IEEE Transactions on knowledge and data engineering*, volume 24, pages 1393–1407, 2012.

[30] D. L. McGuineness and F. van Harmelen. Owl web ontology language. http://www.w3.org/TR/owl-features/.

[31] T. Ertl P. Heim and J. Ziegler. Facet graphs: Complex semantic querying made easy. volume 6088, pages 288–302. Springer, 2010.

[32] S. Chaudhuri S. Agrawal and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *18th Intl Conf. on Data Engineering*, pages 5–16, 2002.

[33] A. Hermann S. Ferré and M. Ducassé. Semantic search: Reconciling expressive querying and exploratory search. In *10th International Semantic Web Conference*. Springer, 2011.

[34] G. M. Sacco and Y. Tzitzikas. *Dynamic Taxonomies and Faceted Search: Theory, Practice and Experience*. Springer, 2009.

[35] L. Spagnolo. *Exploratory computing : designing engines for discovery driven user experiences*. Tesi di dottorato, Politecnico di Milano, 2012/2013.

[36] M. Stefaner and B. Muller. Elastic lists for facet browsers. In *Database and Expert Systems Applications*, pages 217–221, 2007.

[37] C. M. Sperberg-McQueen E. Maler T. Bray, J. Paoli and F. Yergeau. Extensible markup language (xml) 1.0 (fifth edition). http://www.w3.org/TR/xml/.

[38] S. Rudolph T. Tran, H. Wang and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *25th Intl Conf. on Data Engineering*, pages 405–416, 2009.

[39] S. Rudolph T. Tran, P. Cimiano and R. Studer. Ontology-based interpretation of keywords for semantic search. In *6th Intl Semantic Web Conf.*, pages 523–536, 2007.

[40] S. Tata and G. M. Lohman. Sqak: doing more with keywords. In *28th Intl Conf. on Management of Data*, pages 889–902, 2008.

[41] D. Tunkelang. Faceted search synthesis lectures on information concepts, retrieval, and services. volume 1, pages 1–80, 2009.

[42] A. van Kesteren and L. Hunt. Dom4. http://www.w3.org/TR/dom/.

[43] R. White and R. Roth. Exploratory search: Beyond the query-response paradigm. In *Synthesis Lectures on Information Concepts, Retrieval, and Services*, volume 1, pages 1–98, 2009.

[44] W. E. Winkler. The state of record linkage and current research problems. 1999.