Progetto "Casa intelligente per una longevità attiva ed indipendente dell'anziano"
DGR 1464, 7/11/2011

Ambient-Aware LIfeStyle tutor, Aiming at a BETter Health

(Tutoraggio dello stile di vita basato sulla intelligenza ambientale, per una salute migliore)

Risultato D. 4.3

Prototipo di analisi di dati per il monitoraggio di un'indagine scelta dalla cooperativa

Rev. N.1, del 25/02/2015

REGIONE MARCHE

INRCA Istituto di Ricovero e Cura a Carattere Scientifico
HEALTH AND SCIENCE ON AGING

Negli ultimi anni, l'utilizzo di sistemi capaci di controllare gli ambienti domestici e gli edifici registra un progressivo incremento. In altre parole, viene installata una rete di sensori che permette di monitorare e registrare sia le condizioni fisiche e ambientali sia l'interazione tra l'utente e il sistema. In questo scenario, si colloca sia i programmi Ambient Assisted Living [1], che iniziative volte a monitorare i consumi al fine di attuare il risparmio energetico. In entrambi i casi, per raggiungere gli obiettivi, un insieme eterogeneo di sensori e oggetti intelligenti è opportunamente selezionato e installato nell'ambiente. Tali sensori sono interconnessi tra loro attraverso protocolli di connessione standards, come ad esempio IP, KNX, ZigBee HA1.2 eHL7.

A causa della natura della network, risulta evidente come l'ingente quantitativo di dati presenti una natura piuttosto eterogenea. Infatti, i sensori misurano grandezze fisiche differenti e l'acquisizione dei dati avviene in tempi diversi: la natura del sensore determina, quindi, la frequenza di campionamento. Ovvero, ci sono sensori che acquisiscono dati con frequenza regolare e altri che acquisiscono informazioni "occasionalmente". Inoltre, i dati registrati possono essere incompleti, affetti da rumore, ma anche sovrabbondanti. Per esempio, potrebbe accadere che un fenomeno è stato registrato da più di un dispositivo e a causa delle loro caratteristiche fisiche e strumentali, i dati registrati sembrano appartenente diversi. In tale contesto, a causa dell'eterogeneità delle informazioni provenienti dagli oggetti intelligenti e del complesso ambiente in cui essi vengono installati, la sola registrazione dei dati stessi nel *data repository* non è completamente esaustiva per l'analisi. L'insieme delle informazioni raccolte necessita dunque di una visione e classificazione globale. Pertanto, una specifica ontologia di sistema risulta essere necessaria poichè facilita l'interoperabilità semantica e la costruzione delle inferenze, abilitando l'uso di strumenti, detti reasoner, che consentono di esplicitare la conoscenza implicita. Inoltre, l'uso delle ontologie permette di integrare le informazioni registrate dai dispositivi con la conoscenza a priori del dominio e, in particolare, con le relazioni fra le componenti del sistema. Pertanto, l'ontologia risulta essere uno strumento per connettere questi due aspetti, poichè fornisce una formalizzazione condivisa del dominio. L'approccio generale è illustrato nella Figura 1.

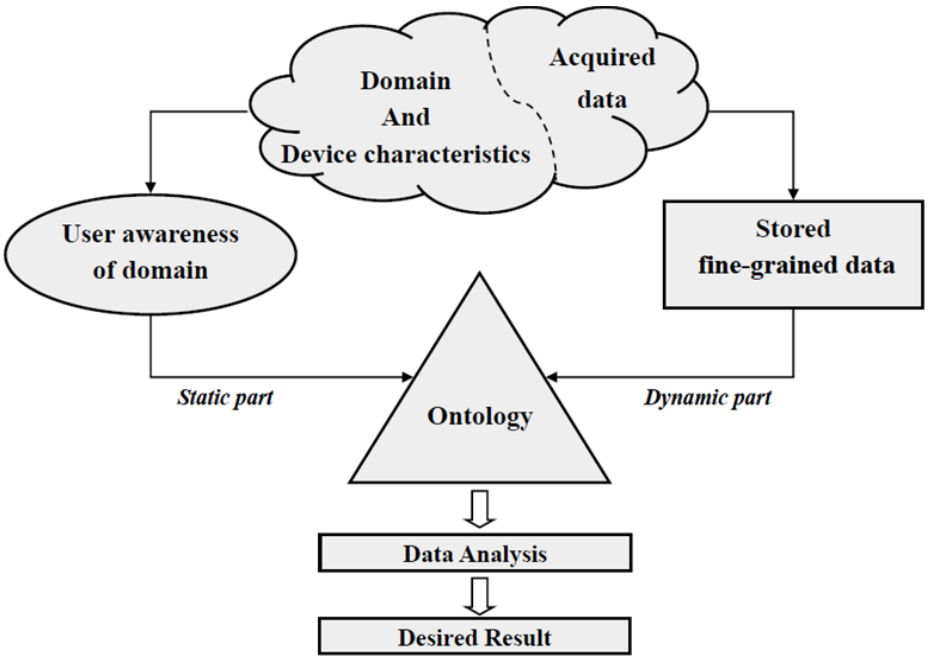


Figura 1: Approccio generale: dal mondo reale all'ontologia

In questo scenario appena descritto si colloca il progetto AALISABETH [2], che intende monitorare gli stili di vita degli anziani, non soggette a malattie croniche o a disabilità importanti. In particolare, all'interno del sistema domotico installato nelle case degli utenti sono presenti sensori domotici, sensori di parametri clinici e fisiologici, sensori personali ed indossabili.

I dati generati da tali dispositivi sono registrati nel database di sistema AALISABETH opportunamente sviluppato [4]. Pertanto, al fine di raggiungere gli obiettivi del progetto, è stata sviluppata e testata una nuova metodologia per analizzare i dati. Nello specifico, l'analisi è in grado di individuare particolari comportamenti, comparare questi con l'evolvere del tempo e determinare l'ordine in cui gli eventi si verificano. In tali situazioni di monitoraggio, le informazioni geo-spaziali sono altrettanto importanti, poichè permettono di localizzare gli eventi nello spazio. Per questo motivo, la metodologia integra un'ontologia [2] di dominio basata su delle regole per la rappresentazione condivisa del sistema con un motore capace di gestire eventi complessi (Complex Event Process Engine, in inglese). In tale infrastruttura, la conoscenza è organizzata in classi e può essere interrogata, scoperta e condivisa con altre applicazioni. L'ontologia è introdotta, come già anticipato, per esplicitare la conoscenza condivisa del dominio e, quindi, permette un'interoperabilità semantica. Nello specifico, essa è stata sviluppata seguendo una struttura piramidale, in modo tale

da distinguere la conoscenza statica dalle informazioni generate dell'interazione fra utente e ambiente in modo tale da riorganizzare i dati in base alla propria semantica e selezionarli durante la fase di preprocesing. D'altra parte, il motore CEP è stato introdotto nel framework a causa della limitazioni espressive dell'ontologia, cioè la mancanza di un reasoning temporale. Infatti, i metodi tradizionali non permettono tali tipi di reasoning, i quali risultano essere necessari per catturare caratteristiche importanti di un flusso di dati ed eventi.

METODOLOGIA

Architettura del framework

Il progetto AALISABETH intende affrontare i temi della prevenzione e della gestione di alcune fra le più comuni patologie diffuse tra la popolazione anziana. In questo scenario risulta evidente che uno delle principali obiettivi è quello di riconoscere le attività umane dell'anziano che si sta monitorando all'interno della propria abitazione. A tale scopo, proponiamo il framework raffigurato in Figura 2, nella quale si mostra le principali componenti che lo costituiscono. E' possibile notare che il sistema è sostanzialmente formato da due differenti componenti interconnesse: **l'Ontologia** ed il **motore CEP**, collegate al database AALISABETH. Nello specifico, i dati collezionati nel database di tipo MySQL opportunamente costruito per il progetto, vengono mappati in OntoAALISABETH, l'ontologia di dominio appositamente sviluppata per il progetto. Inizialmente si trova una corrispondenza tra gli elementi del DB e quelli dell'ontologia. Tale ontologia è costruito seguendo una precisa struttura che verrà descritta in dettaglio in seguito.

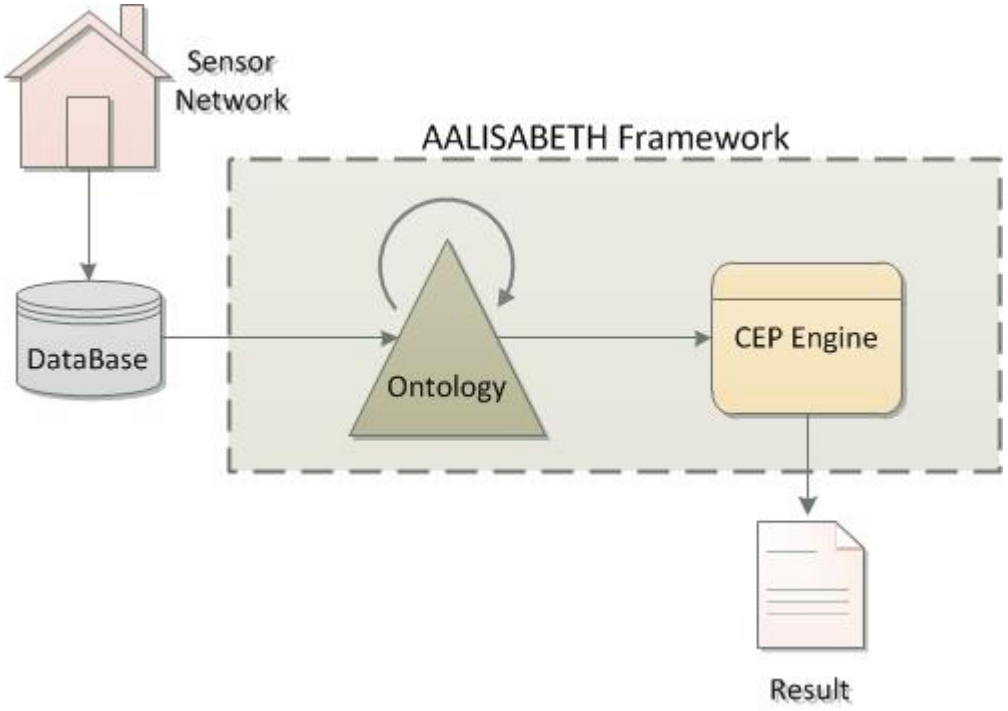


Figura 2: Schema semplificato dall'architettura del Framework

Una volta che i dati sono stati caricati all'interno dell'ontologia, essi vengono aggregati in modo tale che un insieme di dati possa essere salvata nell'ontologia come l'evento di un sensore "virtuale". Un sensore virtuale rappresenta un sensore fittizio dove un record rappresenta l'aggregazione di più dati derivanti da uno o più tipi di sensori. Per esempio, per rilevare che un utente sta utilizzando la toilet durante la notte, è necessario trovare un insieme di azione atomiche e un insieme di luoghi che individuano tale attività. Inoltre, tali eventi sono dipendenti dal tempo. Nello specifico, è necessario che l'utente si alzi dal letto, vada in bagno e usi lo sciacquone. Dunque, il sensore virtuale "Usare la toilet durante la notte" aggrega dati che si riferiscono ai sensori coinvolti nelle azioni atomiche precedentemente citate.

La generazione di questi eventi risulta essere di notevole importanza. Infatti, mentre l'ontologia classifica e riorganizza i dati, il riconoscimento dei pattern comportamentali viene eseguito da un motore Complex Event Processing (CEP). Infatti, ricordando che il nostro obiettivo è quello di riconoscere le attività umane all'utente e osservando che sono strettamente legate al tempo, è necessario che il framework sia in grado di gestire le restrizioni temporali. Per tale scopo è stato introdotto il motore CEP, il quale permette l'analisi di un flusso alla ricerca di pattern.

COMPONENTI DEL FRAMEWORK

Come anticipato nella sezione precedente, il framework è composto da differenti componenti interconnesse, come mostrato in Figura 3.

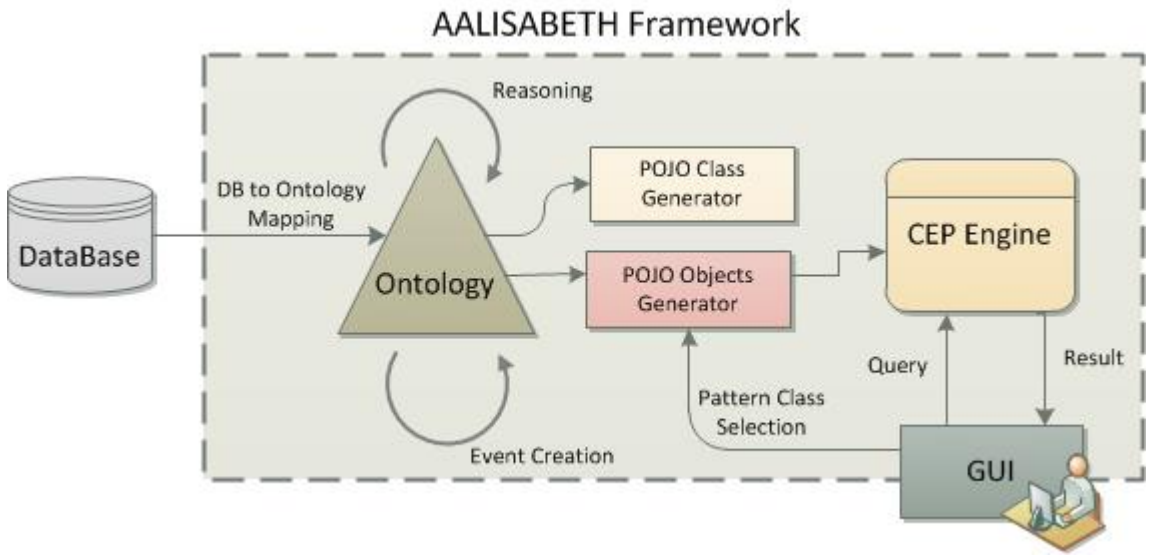


Figura 3: Architettura del Framework

Nello specifico, la prima fase è composta dall'estrazione dei dati contenuti nel database MySQL e nella mappatura degli stessi all'interno di OntoAALISABETH, l'ontologia di dominio opportunamente sviluppata per il progetto. Questa fase viene eseguita tramite l'utilizzo di uno schema di mapping, il quale permette di creare una corrispondenza tra i record del DB e le instance dell'ontologia, sfruttando un linguaggio di mappatura chiamato d2rq **[6].** Nello specifico, D2RQ è capace di supportare mappature condizionali, mappature di colonne multiple in una stessa proprietà, di gestire strutture di tabelle normalizzate, dove una instanza è distribuita su diverse tabelle e infine l'utilizzo di tabelle di traduzione durante il processo di mappatura. In questo modo si viene a creare una corrispondenza biunivoca tra gli individui del DB ed le instanze dell'ontologia permettendo quindi una riorganizzazione dei dati secondo la loro semantica. Inoltre, al fine di catalogare, dividere e combinare gli individui nell'ontologia è possibile sfruttare anche le potenzialità delle rules, poichè è possibile creare delle specifiche rules semantiche.

La seconda fase è rappresentata dal filtraggio e selezione dei dati, durante la quale i dati vengono modellati in differenti modi. In particolare, i dati vengono aggregati e scelti in modo tale da generare eventi più complessi, chiamati "macro eventi". Durante questa fase, i macro eventi vengono ulteriormente selezionati e filtrati. Questo viene fatto attraverso l'uso delle rules e dei reasoner. Infatti, come descritto in precedenza, l'ontologia gioca un ruolo di preprocessing e di filtraggio semantico.

Infine, il framework permette anche l'interazione con utenti non esperti. Infatti, attraverso un'interfaccia grafica, è possibile scegliere due importanti parametri, il pattern da analizzare e la query. In questo modo, Esper [5] in accordo con la query di input analizza il flusso degli eventi cercando il pattern che corrisponde con la query inserita.

a. Mapping DB-ontologia

I dati caricari nel database del sistema AALISABETH vengono mappati nell'ontologia sfruttando il linguaggio D2RQ [6]. In particolare, permette di mappare elementi relazionali (tabelle, colonne, id, join e così via) su elementi ontologici (classi, relazioni, URI, tipo, XSD, ect). Il primo step per la mappatura è la connessione database ontologia, nello specifico è necessario specificare la classe del driver JDBC , l'url del database e relativo nome utente per la connessione, mentre la funzione *keepAlive* permette di tenere attiva la connessione fra ontologia e database. Nel nostro caso, la connessione db-ontologia è data dal seguente codice

```
map:database a d2rq:Database;
      d2rq:jdbcDriver "com.mysql.jdbc.Driver";
      d2rq:jdbcDSN "jdbc:mysql://localhost:3306/aal";
      d2rq:username "aalisabeth";
      .
```

I driver di connessione sono quelli mysql, poichè il database è sviluppato nella piattaforma mysql e "jdbc:mysql://localhost:3306/aal" indica l'url del database salvato in locale, mentre "aalisabeth" rappresenta il nome utente.

Il mapping, invece, consiste essenzialmente nella definizione di due oggetti: **Class Map** e **PropertyBridge**.

Un **d2rq:ClassMap** è di fatto l'oggetto che specifica il mapping rispetto ad un certo **d2rq:Database** associando tutti gli identificativi delle istanze a una particolare classe del modello ontologico. A ogni **ClassMap** possono essere associati più **PropertyBridge**.

ClassMap permette solo di definire la modalità di generazione (condizionale o non) degli identificatori delle istanze e delle associazioni rispetto alle classi di modello. Per valorizzare poi le proprietà delle istanze create rispetto alle proprietà/relazioni del modello è necessario associare al **ClassMap** delle **PropertyBridge**.

In pratica, un **PropertyBridge** è un elemento del mapping che dichiara come valorizzare una certa proprietà di modello (sia **owl:DatatypeProperty** che **owl:ObjectProperty**) per le istanze di una certa classe (associata al**ClassMap**) e in funzione di regole condizionali, join ed eventuali espressioni SQL per i mapping più complessi.

Di seguito viene riportato due estratti di codice che rappresenta rispettivamente una Class Map e una PropertyBridge implementata al fine di avere gli elementi del database AALISABETH all'interno dell'ontologia OntoAALISABETH. Il primo frammento di codice permette di mappare tutti i record presenti nella tabella DataSensor del Database nell'omonima classe dell'ontologia. Inoltre, ad ogni elemento dell'ontologia viene associato un nome ed un uriPattern, nel caso specifico il nome di ogni individuals viene fornito dal suo valore del campo della tabella **record_id**. Il secondo frammento di codice permette di mappare il campo "timespamp" della tabella DataSensor come DataProperty degli individui appartenti alla rispettiva classe. E' possibile inoltre anche modificare il nome della corrispettiva DataProperty. Occorre, però, specificare il tipo, nel caso specifico si tratta di un timestamp.

```
map:DataSensor a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "http://AALont#@@data_rc_2_test.record_id@@";
      d2rq:class db_prog:DataSensor;
```

.

```
map:timestamp a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:DataSensor;
      d2rq:property db_prog:timestamp;
      d2rq:column "data_rc_2_test.timestamp";
      d2rq:datatype xsd:dateTime;
```

b. Ontologia

La principale componente del framework è rappresentata dall'ontologia, **OntoAALISABETH**, che definisce formalmente il dominio considerato generando una base di conoscenza condivisa per tutte le componenti del dominio. Tale ontologia ha un particolare approccio come mostrato in Figura 4. In essa, vengono modellati quattro domini (User, Environment, Activity and Device), che rappresentano la conoscenza di base del contesto di Ambient Assisted Living.

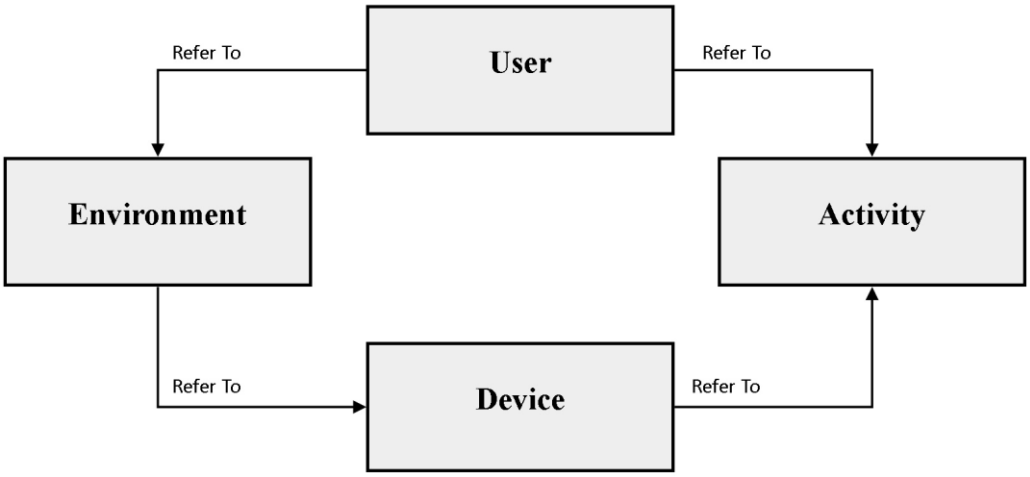


Figura 4: Schema generale dell'ontologia

Il dominio "User" formalizza tutti i concetti relativi al profito dell'utente, mentre "Acrivity" descrive diverse attività domestiche che sono necessarie per scoprire dei comportamenti considerati anormali. Queste due parti giocano un ruolo centrale. Poi, Environment e Device descrivono l'abitazione dell'utente e la rete di sensori installata. Di conseguenza, questa ontologia mostra

differenti livelli di astrazione che composti insieme formano una struttura piramidale, dove ogni livello speciallizza quello immediatamente superiore. L'architettura, come mostrata in Figura 5, è formata da due componenti principali:

- Livello statico (AAL and AAL-Building Specific ontology);
- Livello Dinamico (data and view ontology).

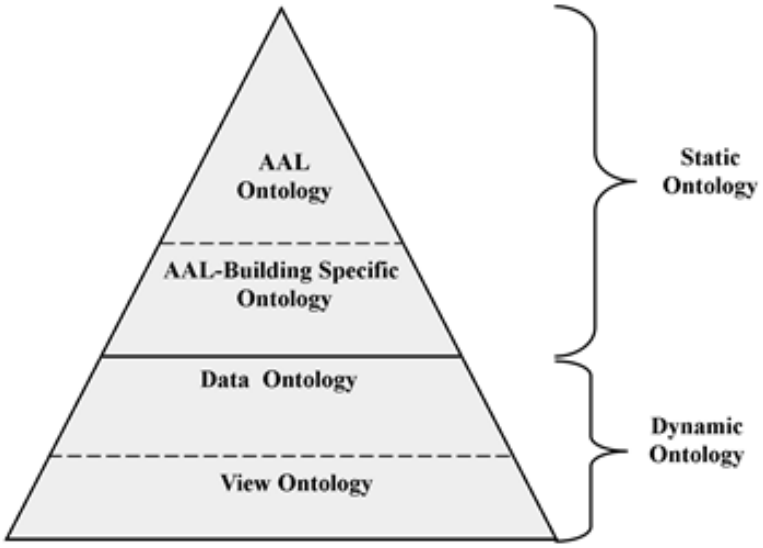


Figura 5: Struttura piramidale dell'ontologia

Ogni parte ha un ruolo specifico al fine di rispondere alle diverse esigenze, come descritto di seguito.

1. *AAL Ontology*. Il primo livello costituisce l'ontologia di dominio superiore, che contiene informazioni statiche. Infatti, vengono formalizzati i concetti generali di contesto AAL. Pertanto, questa parte di OntoAALISABETH può essere riutilizzata in ambienti simili.

   Come suggerisce la letteratura, al fine di velocizzare lo sviluppo dell'ontologia e il tempo di realizzazione, si è scelto di ridefinire ed estendere DogOnt Ontology [7]. Tale ontologia, sviluppata in OWL, è stata implementata dal gruppo e-Lite dell'Università degli Studi di Torino, per descrivere ogni dispositivo di una Smart Home, ovvero un'abitazione dotata di sensori ed attuatori, di hardware e software dedicati, allo scopo di realizzare interazioni di automazione. Infatti, in essa per ogni dispositivo reale viene creata una descrizione dettagliata che ne esplica il funzionamento attraverso l'uso di un linguaggio comune e condiviso. In particolare, l'ontologia OntDog è in grado di descrivere:

- dov'è è situato un dispositivo domotico;
- la capacità e la funzionalità di un dispositivo;
- le specifiche tecnologiche necessarie per interfacciarsi con un dispositivo;
- le possibili configurazioni che un dispositivo può assumere;
- come è strutturata l'abitazione (piani, stanze, muri);
- gli elementi architetturali e di arredamento presenti nell'ambiente.

La descrizione di ogni dispositivo è quindi composta da due componenti: una parte astratta per descrivere il comportamento del dispositivo ed una parte tecnologica necessaria per la comunicazione. Questo tipo di approccio favorisce l'interoperabilità tra i dispositivi in quanto permette uniformità nel linguaggio utilizzato per la loro descrizione. DogOnt, pur descrivendo dettagliatamente una Smart Home, non soddisfa i nostri obbiettivi, poiché non formalizza interamente il tipo di conoscenza che si vuole rappresentare. Infatti, essa non permettere di descrivere gli abitanti della casa, come ad esempio il peso corporeo, l'età, la pressione cardiaca, il tasso glicemico, le relazioni familiari che intercorrono, l'indice di saturazione di ossigeno, le frequenze cardiache e respiratorie. Inoltre, attraverso essa non si riesce a descrivere le abitudini alimentari e personali, come ad esempio con quale frequenza l'anziano monitorato effettua attività fisica e, in particolare, se essa è influenzata dall'ambiente familiare e/o dalle condizioni meteorologiche. Per le precedenti ragioni, a partire da DogOnt è stata sviluppata OntoAALISABETH. Tale Ontologia eredita le principali classi da DogOnt e, in aggiunta, ne presenta delle nuove. Le classi ereditate sono:

- **Building Thing:** per modellare qualsiasi elemento controllabile e elemento non controllabile. Si definisce elemento controllabile (o dispositivo) un elemento che può essere controllato dal sistema domotico. I dispositivi sono suddivisibili in due categorie: quelli appartenenti ad uno degli impianti domestici (la lampada, ad esempio, viene cataloga come appartenente all'impianto elettrico) e gli elettrodomestici. Ogni dispositivo possiede diverse funzionalità (Functionality, rintracciate dalle relazioni hasFunctionality) e può assumere diverse configurazioni di stato (State, identificate attraverso la relazione hasState). La quantità ed il tipo di dati e di funzionalità dipendono ovviamente del dispositivo; in generale, ogni elemento controllabile mette a disposizione almeno una Query-Functionality che

permette di fornire informazioni riguardanti la configurazione. Gli elementi non controllabili possono essere di tipo architetturale per la definizione dell'ambiente (muri, pavimenti, ecc.) e di arredamento (mobili, scrivanie, tavoli, ecc);

- **Building Enviroment**: per modellare la posizione di un elemento. Contiene una descrizione delle varie tipologie di ambiente: vengono definiti gli ambienti giardino (classe Garden), garage (classe Garage) e appartamento (classe Flat). Ogni piano può contenere una o più stanze (cucina, camera da letto, bagno, ecc.);

- **Functionality**: per modellare le operazioni che si possono effettuare su un dispositivo. Tutte le funzionalità definiscono i comandi necessari per modificare o per interrogare una specifica proprietà del dispositivo. Sono suddivise in diverse categorie in relazione al loro scopo;

- **State**: per modellare le configurazioni che un dispositivo può assumere. Come per le funzionalità, anche gli stati vengono classificati in base ai valori che possono assumere: i ContinuosState, che riguardano quelle proprietà che possono assumere valori continui, ed i DiscreteState, associati alle proprietà che assumono sono valori discreti;

- **Notification**: per modellare tutte le notifiche che i dispositivi possono generare in seguito ad una variazione di stato;

- **Command**: per modellare tutti i comandi per controllare ed interrogare i dispositivi.

Le classi aggiunte al fine di raggiungere l'obiettivo sono principalmente quelle che descrivono la persona e i suoi comportamenti:

- **Activity**: per modellare le principali attività della persona può svolgere all'interno della propria abitazione: camminare, camminare velocemente, dormire, riposare, mangiare, stare seduto, o fare altre attività;

- **ConsumableThing**: per monitorare cosa gli abitanti ingeriscono Medicine, pasti e bevande.

- **EnviromentProfile:** per modellare le caratteristiche degli abitanti della casa e le condizioni meteorologiche. Tale classe è organizzata in due sottoclassi: **Person** (che descrive le caratteristiche della persona, come età, corporatura) e **Natural**, che ha sua volta, ha due classi-figlie **Season** e **Weather**. Tali classi sono state introdotte con lo scopo di capire le abitudini della persona, ma soprattutto da cosa sono influenzate.

- **Meal**: per monitorare quando la persona mangia.

In questa parte di OntoAALISABETH sono state introdotte un insieme di regole che sono in grado di formalizzare la conoscenza a priori. Per esempio, se si vuole definire formalmente l'attività di "preparare un pasto", è necessario identificare un insieme di azioni atomiche e uno o più luoghi dove tale azione può avvenire. Questo tipo di informazioni sono ottenute dalla combinazione dei dati prodotti dai sensori e dal loro significato semantico (SENSORE VIRTUALE).

2. *AAL-Building Specific ontology.* Il secondo livello, *AAL-Building Specific ontology*, estende diverse proprietà statiche introdotte nel livello precedente. In questa parte si definisce formalmente il particolare dominio che si sta considerando, ovvero vengono descritte le varie componenti della Smart Home considerata: la reale struttura dell'ambiente e la disposizione delle stanze, le informazioni personali relative agli abitanti della casa, quali sensori sono stai istallati nella rete e come cominciano tra di loro. Inoltre, la completa conoscenza del dominio permette allo sviluppatore di aggiungere nell'ontologia nuovi elementi e relazioni, le quali non possono essere descritte nelle tecnologie di memorizzazione dati. A differenza dell'AAL Ontology, questa componente di OntoAALISABETH è sviluppata per modellare uno specifico ambiente. Di conseguenza, essa cambia per ogni particolare dominio considerato e non può essere riutilizzata.

3. *Data ontology.* Il terzo livello, DataOntology, estende le precedenti livelli di ontologia introducendo il concetto che ogni dispositivo genera dei dati granulari. Nello specifico, le

precedenti classi sono popolate dagli individui che presentano una corrispondenza uno a uno con ogni records memorizzati nel data repository. Questa procedura è permessa da un mapping tra DB e ontologia, attraverso il linguaggio D2RQ. In particolare, esso permette, come descritto nel paragrafo "Linguaggio D2RQ", di stabilire una connessione fra l'ontologia e diverse sorgenti e di importare i relativi attributi e proprietà. Dunque, i dati memorizzati nel data repository vengono caricati nell'ontologia secondo la propria semantica. Inoltre, tale acquisizione avviene in un intervallo di tempo costante, permettendo quindi di eseguire un'analisi dinamica in tempo reale. In conclusione, si ottiene i dati con la propria semantica che permette di avere nel livello successivo delle viste personalizzate del sistema.

4. *View ontology.* L'ultimo livello di OntoAALISABETH, view ontology, consiste nella creazione di viste alternative della conoscenza. Questa fase è basata sulla procedura di prepocessing, dove il reasoner gioca un ruolo principale. Infatti, esso è in grado di inferire conseguenze logiche da un insieme di fatti o di assiomi. Nel nostro caso, le regole e gli assiomi sono definiti attraverso le regole, con lo sviluppo di opportune builtins, e le classi equivalenti. Infatti, un insieme di builtins, paradigma alternativo per modellare la conoscenza, è introdotto per acquisire nuova conoscenza stabilendo nuove connessione fra entità indipendenti. Le regole, che in grado di estendere l'espressività di OWL, sono valutate periodicamente in fase di esecuzione e nuovi fatti sono aggiunti nell'ontologia. La definizione di classe equivalente è guidata dallo scopo di classificare le istanze in base a determinate proprietà e relazioni; cioè queste classi sono popolate da individui desiderati. Invece, la fase di prepocessing è basata sull'abilità del reasoner di fare query sull'ontologia ed estrarre informazioni necessarie per l'analisi. Si osserva che le query sull'ontologia nello step finale della metodologia corrispondono a selezionare un insieme di dati generati da un sensore virtuale, secondo l'interpretazione dell'utente.

c. Rules

Per riorganizzare i dati, che sono stati caricati all'interno dell'ontologia sfruttando il linguaggio D2RQ, sono state introdotte un insieme di regole nell'ontologia. Nello specifico, sono state implementate ed introdotte un insieme di nuove built-ins, funzioni che implementano particolari algoritmi ed invocate nelle rules, all'interno del framework Jena [**9**]. In particolare, tali regole permetto di

raggruppare e riorganizzare i dati dipendenti da restrizioni temporali e sono introdotte anche per acquisire nuova conoscenza stabilendo anche nuove connessioni fra entità non in relazione. Queste regole, che sono in grado di estendere l'espressività del'OWL, sono valutate ad intervalli periodici durante l'esecuzione e permettono di aggiungere nuovi fatti all'interno dell'ontologia. In generale, in Jena non è possibile gestire i valori temporali. Per questo motivo, nuove builtins sono state implementate, come illustrato Risultato D.4.2.

d. Motore CEP

L'ontologia è estremamente potente, ma ha delle limitazioni espressive, nello specifico manca di un reasoning temporale. Considerando la natura della nostra analisi e l'aggiornamento dinamico del dataset, i metodi tradizionali non permettono di eseguire reasoning temporali e spaziali. Per questo motivo, nella metodologia è introdotto il motore CEP. Questi permette di combinare i dati provenienti da più sensori per dedurre eventi o modelli che inducono a situazioni più complicate. Infatti, il nostro principale obiettivo è quello di riconoscere pattern di eventi. Queste identificazioni potrebbero essere riutilizzate per scoprire eventi ancora più complessi, attraverso l'utilizzo del motore CEP. Con eventi più complessi si intendono quegli eventi che sono composti da più eventi semplici.

e. Componenti Java

L'intero framework è stato sviluppato tramite il linguaggio Java. La scelta del linguaggio di programmazione è stata obbligata in quanto sono presenti delle librerie che permetto la gestione di ontologie OWL, come ad esempio la libreria Jena API.

L'idea iniziale era quella di creare un framework completamente "context-free", ossia che non dipendesse dalle variabili del sistema (ontologia, tipi di sensori, abitazione). Per questo motivo sfruttando le librerie Javassist è possibile creare a runtime le classi Java che serviranno poi come struttura dati per le istanze degli eventi. Infatti per poter sfruttare tutte le potenzialità del motore CEP, gli eventi devono essere passati come oggetti Plain Old Java Objects (POJO). In questo modo il sistema risulterà estremamente dinamico alle diverse variabili dell'ambiente.

f. L'interfaccia

In Figura **6,** viene mostrata la Graphical User Interface (GUI), ossia l'interfaccia che vedrà l'utente. L'interfaccia è composta da differenti bottoni ed aree di testo. L'esecuzione è sostanzialmente divisa in tre fasi principali:

1. cliccando sul bottone "Load Data", i record contenuti nel database vengono estrati e salvati nell'ontologia durante la fase di mappatura;
2. nella parte destra l'utente può selezionare quale tipo di pattern e utente vuole analizzare;
3. infine l'utente può definire una query ed attendere i risultati dopo aver cliccato sul bottone "Execute query";
4. i risultati verrano poi visualizzati nel box sottostante.

Questa è una semplice GUI che permette di sfruttare le principali caratteristiche del framework.

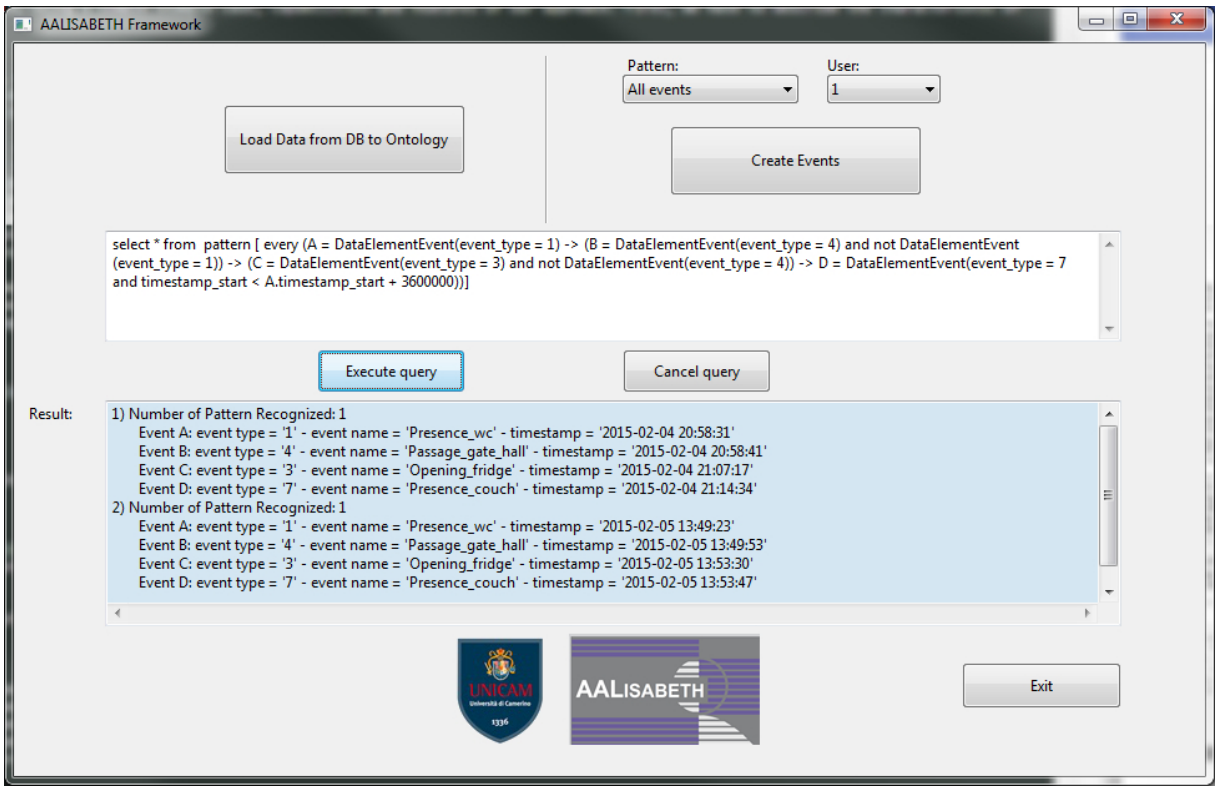


Figura 6: Interfaccia grafica del framework

Tools & Technologies

In questo paragrafo andremo a descrivere brevemente i vari tipi di tool e tecnologie utilizzate durante lo sviluppo. Per iniziare l'ontologia OWL è sviluppata e testata su Protégé 4.3 [10], un editor di ontologie, insieme al Pellet Reasoner Plugin [8], che permette la creazione e popolazione delle classi di equivalenza. Per quanto riguarda la definizione di una configurazione della mappatura sfruttiamo una procedura (sistema OBDA) per definire delle istruzioni che permettono di mappare il database all'interno dell'ontologia. Come descritto prima il framework è stato implementato in Java, linguaggio che ci ha permesso di sfruttare diverse librerie per i diversi componenti. L'ontologia viene gestita grazie alle OWL API. Inoltre il reasoner Pellet viene invocato tramite l'utilizzo di Jena [9] che permette l'attività di "reasoning" nell'ontologia con i dati già caricati. Riassumendo, utilizzando la libreria Jena, l'ontologia, costruita con Protégé , viene caricata in oggetto Java (l'interfaccia OntModel).
Per quanto riguarda il motore CEP, abbiamo scelto Esper [5] per diversi motivi: il fatto che ha una libreria Java, può essere utilizzato per diverse sorgenti di dati e applicazioni CEP, accetta diversi formati di input per la rappresentazione degli eventi. L'intero framework è stato sviluppato utilizzato Eclipse IDE.

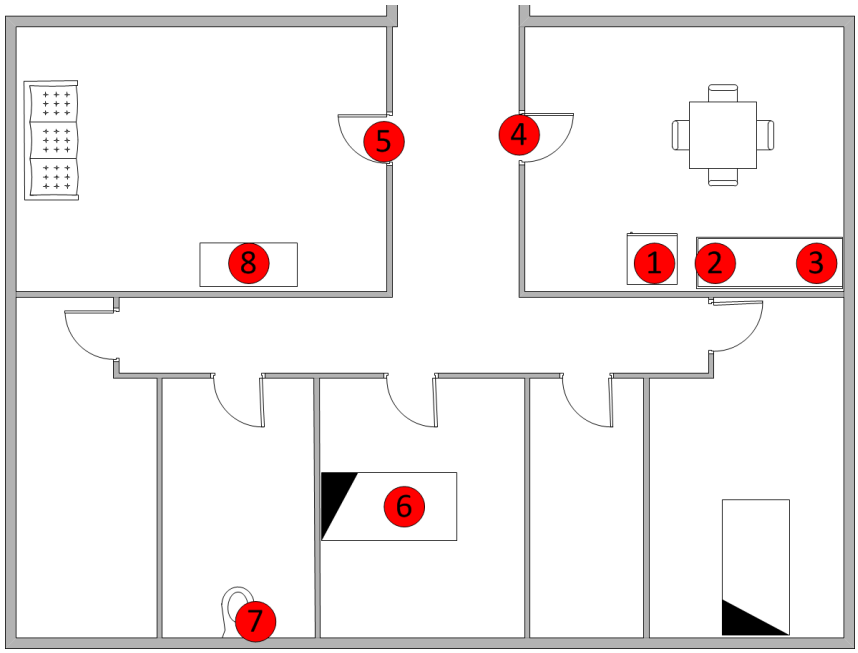
Test & Validation

a. Descrizione pilot

Le abitazioni pilot prese in esame sono tre per le quali è stato effettuato un periodo di un mese di raccolta dati. Di seguito verranno descritte brevemente le diverse strutture delle case ed elencati i sensori installati e presi in considerazione durante la creazione e ricerca dei pattern.

PILOT 1

l Pilot 1 è appartamento di circa 100 mq composto da 8 stanze: cucina, sala, ingresso e corridoio, 3 camere da letto e da 2 bagni. In figura, sono stati disegnati i sensori installati, i quali sono stati



numerati e contrassegnati nella pianta della casa con dei bollini rossi. Nello specifico, 1: Sensore apertura frigorifero

2: Bilancia

3: Tavoletta

4: Varco sala

5: Varco cucina

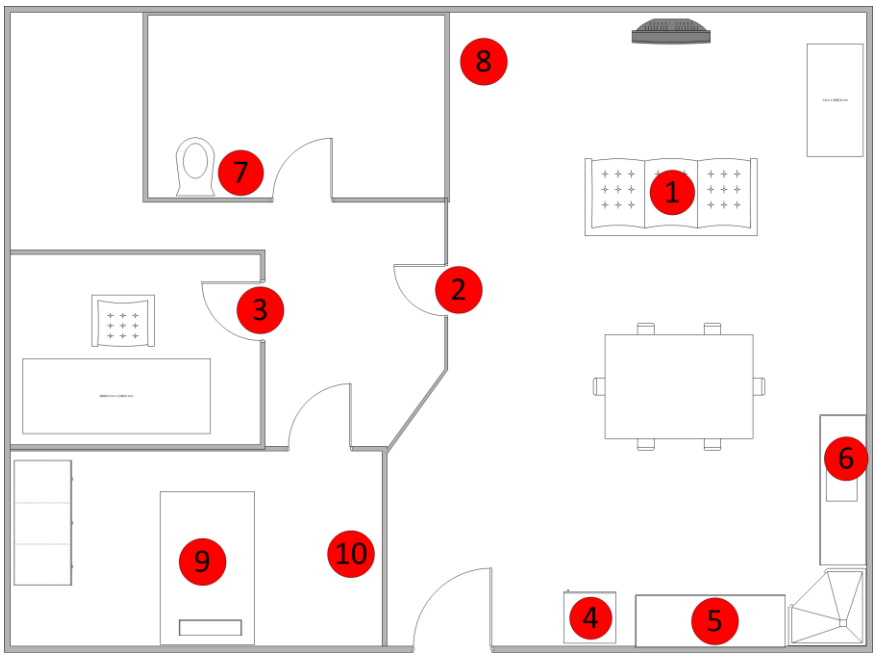
6: Sensore presenza letto

7: Sensore sciacquone

8: Sensori clinici

PILOT 2

Il Pilot 2 è un appartamento di circa 100 mq. L'appartamento è composto da una cucina non separata dalla sala, bagno, stanzino, studio e camera da letto. In figura, sono stati disegnati i sensori installati, i quali sono stati numerati e contrassegnati nella pianta della casa con dei bollini rossi. Nello specifico,



1: Sensore presenza divano

2: Varco sala

3: Varco studio

4: Sensore apertura frigorifero

5: Tavoletta

6: Bilancia

7: Sensore WC

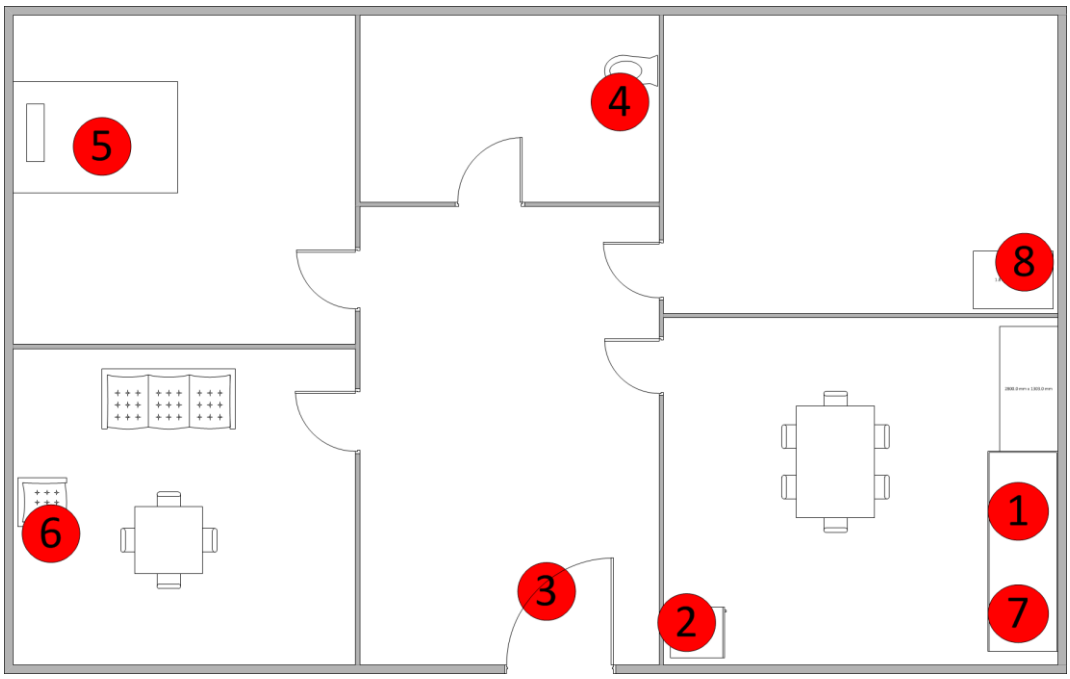
8: Sensori clinici

9: Sensore presenza letto

10: Bilancia pesa persone

PILOT 3

Il Pilot 3 è un appartamento di circa 80 mq. L'appartamento è composto da una cucina, sala, bagno, ingresso e 2 camere da letto. In figura, sono stati disegnati i sensori installati, i quali sono stati numerati e contrassegnati nella pianta della casa con dei bollini rossi. Nello specifico,



1: Bilancia

2: Sensore apertura frigo

3: Varco ingresso

4: Sensore presenza bagno

5: Sensore presenza letto

6: Sensore presenza poltrona

7: Tovaglietta

8: Sensori clinici

b. Descrizione pattern

Per effettuare una analisi comparativa tra i vari appartamenti, è stato scelto un pattern che potesse essere rilevato dai sensori installati nelle diverse case. E' stato quindi scelto un pattern semplice in modo che potesse essere riconosciuto in tutti e tre gli appartamenti, considerando la natura dei tipi di sensori installati.

Il pattern in esame è formato da azioni semplici che sono:

- andare in bagno;
- aprire il frigorifero;
- aprire il cassetto della dispensa;
- pesare una pietanza.

Di seguito, è mostrata la query che viene eseguita:

```
select * from pattern [ every (A = DataElementEvent(event_type=1) -> ((B =
DataElementEvent(event_type=2) or C = DataElementEvent(event_type=3)) and
not DataElementEvent(event_type=4)) -> (D = DataElementEvent(event_type=6
and timestamp_start < A.timestamp_start + 3600000) and not
(DataElementEvent(event_type=2) or DataElementEvent(event_type=3))))]
```

Quella scritta rappresenta una sequenza di azioni (detto pattern), in linguaggio naturale possiamo descriverla come: l'utente va in bagno e tira lo sciacquone, poi va in cucina ed apre il frigorifero o il cassetto e successivamente pesa una pietanza tramite la bilancia, il tutto deve essere stato eseguito (e rilevato dai sensori) entro un'ora dall'inizio della prima azione costituita dall'andare in bagno.

c. Presentazione risultati

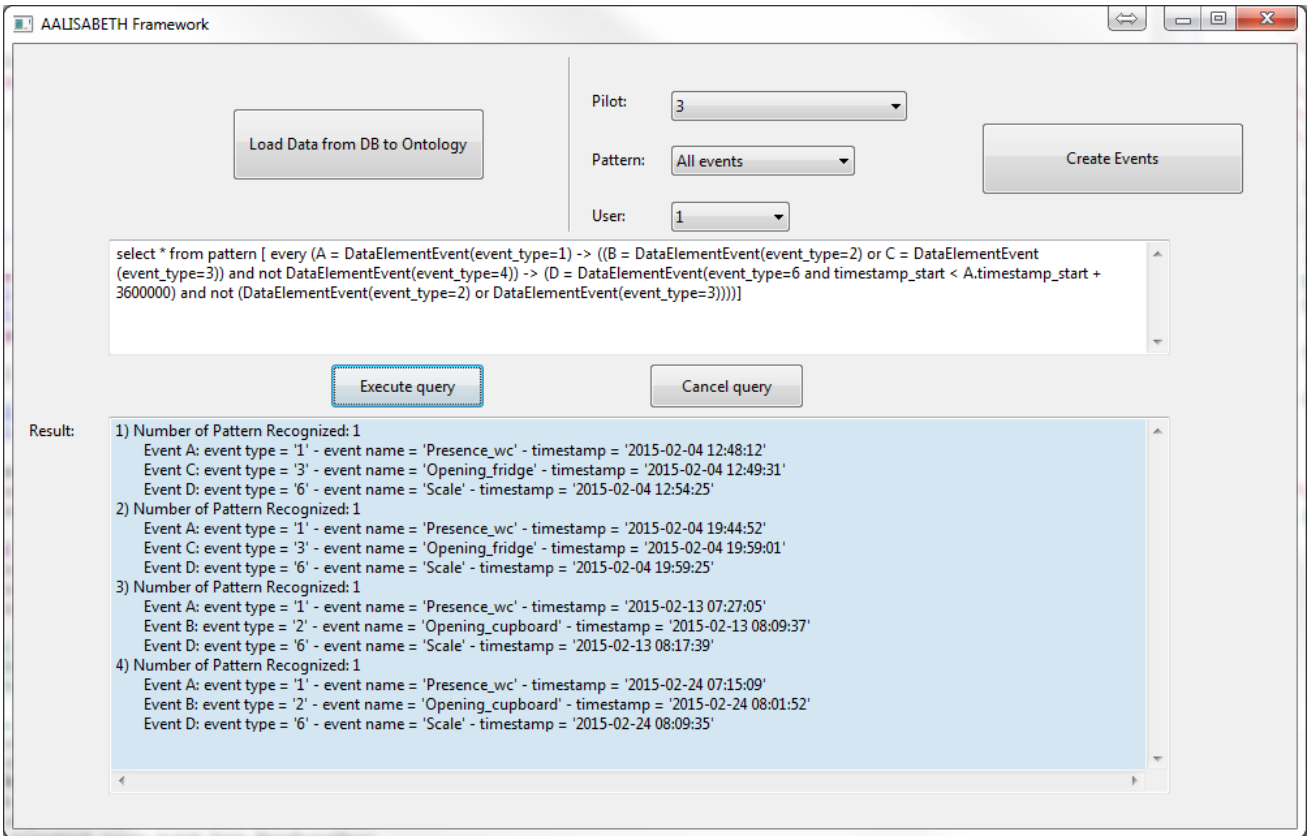
Nelle figure di seguito sono presentati i risultati, nel box in basso sono visualizzati pattern riconosciuti con i relativi timestamp.
Da notare come la creazione degli eventi (basati sui records del database precedentemente caricati nell'ontologia) può essere vincolata secondo certi parametri:

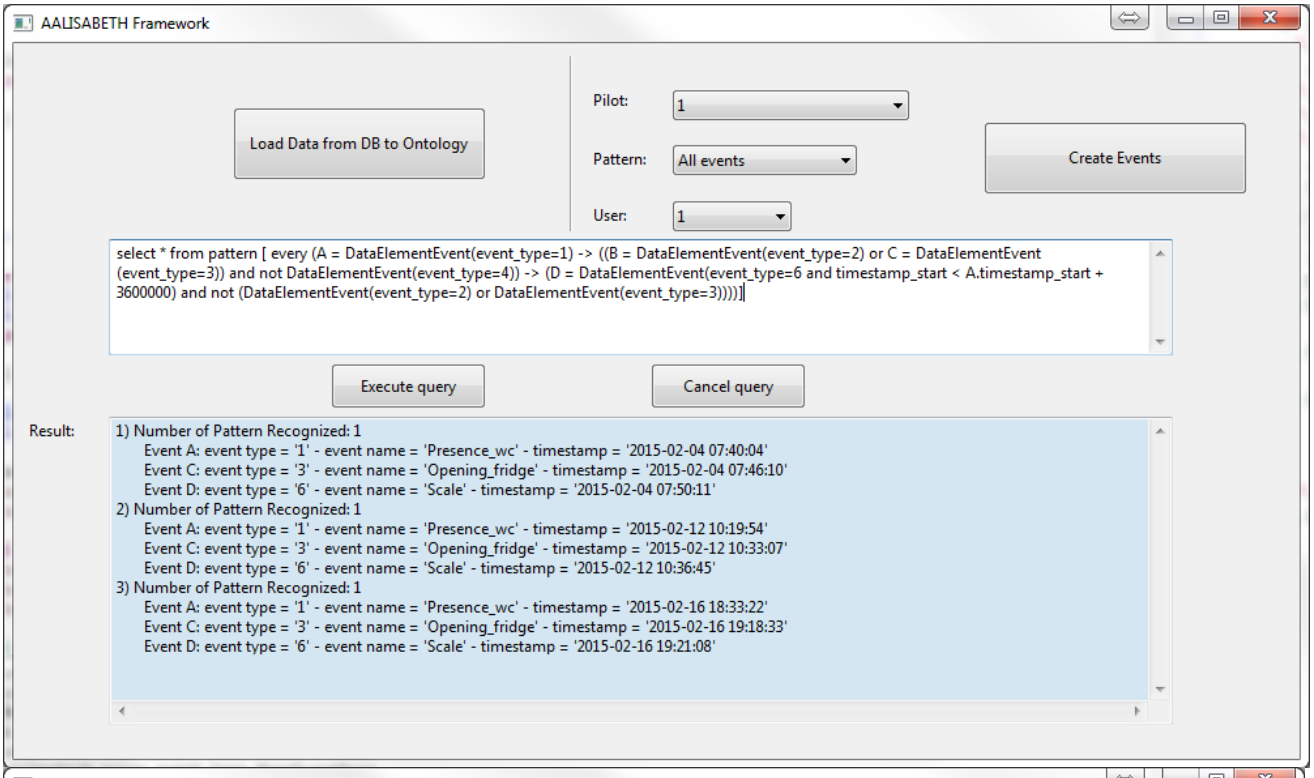
- Il pilot che si vuole analizzare, nel nostro caso sono tre diversi appartamenti;
- Gli eventi associati ad un particolare pattern, infatti è possibile, tramite l'utilizzo delle rule descritte precedentemente, filtrare gli eventi secondo la loro tipologia o altre caratteristiche;
- Il framework è predisposto per la multi-utenza, quindi è possibile filtrare gli eventi in base all'utente che si vuole monitorare.

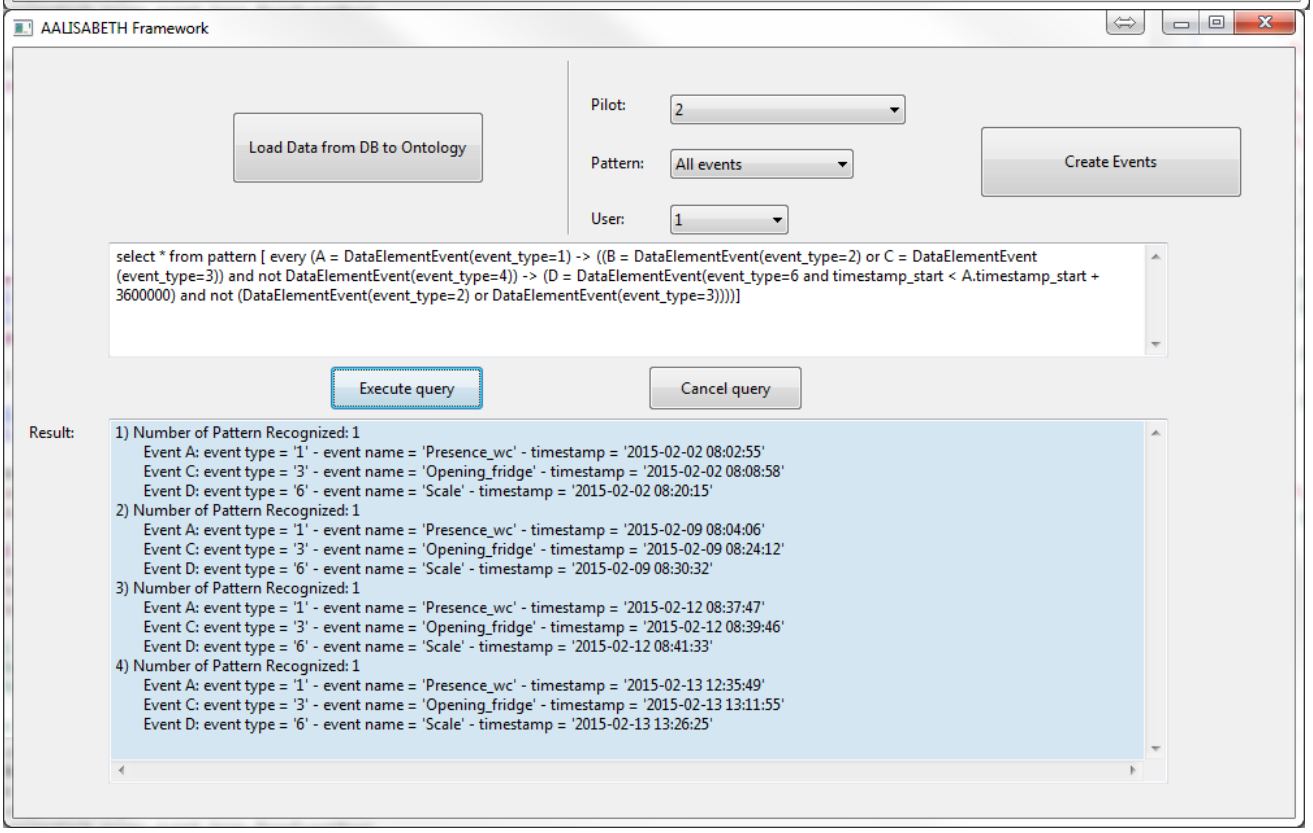
Ovviamente il pattern che abbiamo scelto di mostrare è solo a scopo dimostrativo delle capacità del framework. E' un pattern molto particolare in quanto esige che le azioni coinvolte siano eseguite in sequenza ed in un lasso di tempo ristretto, per questo motivo le ricorrenze nell'arco di un mese non sono numerose.

L'esempio mostra anche che tramite il linguaggio di query posso essere ricercati varie tipologie di pattern. Infatti una volta studiata la sintassi e struttura del linguaggio di query di Esper sarà possibile eseguire una notevole varietà di richieste diverse.

Nel dettaglio sono stati trovati 4 pattern per il pilot numero 1, 3 pattern per il pilot numero 2 e 4 pattern per il numero 3.




AALISABETH Framework
Load Data from DB to Ontology
Pilot: 1
Pattern: All events
User: 1
Create Events
select * from pattern [every (A = DataElementEvent(event_type=1) -> ((B = DataElementEvent(event_type=2) or C = DataElementEvent (event_type=3)) and not DataElementEvent(event_type=4)) -> (D = DataElementEvent(event_type=6 and timestamp_start < A.timestamp_start + 3600000) and not (DataElementEvent(event_type=2) or DataElementEvent(event_type=3)))))
Execute query
Cancel query
Result:
1) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-04 07:40:04'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-04 07:46:10'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-04 07:50:11'
2) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-12 10:19:54'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-12 10:33:07'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-12 10:36:45'
3) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-16 18:33:22'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-16 19:18:33'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-16 19:21:08'


AALISABETH Framework
Load Data from DB to Ontology
Pilot: 2
Pattern: All events
User: 1
Create Events
select * from pattern [every (A = DataElementEvent(event_type=1) -> ((B = DataElementEvent(event_type=2) or C = DataElementEvent (event_type=3)) and not DataElementEvent(event_type=4)) -> (D = DataElementEvent(event_type=6 and timestamp_start < A.timestamp_start + 3600000) and not (DataElementEvent(event_type=2) or DataElementEvent(event_type=3)))))]
Execute query
Cancel query
Result:
1) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-02 08:02:55'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-02 08:08:58'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-02 08:20:15'
2) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-09 08:04:06'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-09 08:24:12'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-09 08:30:32'
3) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-12 08:37:47'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-12 08:39:46'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-12 08:41:33'
4) Number of Pattern Recognized: 1
Event A: event type = '1' - event name = 'Presence_wc' - timestamp = '2015-02-13 12:35:49'
Event C: event type = '3' - event name = 'Opening_fridge' - timestamp = '2015-02-13 13:11:55'
Event D: event type = '6' - event name = 'Scale' - timestamp = '2015-02-13 13:26:25'

Conclusioni

In generale, scoprire lo stile di vita di una persona monitorando le sue attività all'interno della propria abitazione non risulta essere un semplice obiettivo, poiché una stessa azione può essere svolta in modi differenti da ogni essere umano e, in aggiunta, si possono eseguire più attività nello stesso tempo. Per esempio, è possibile guardare la televisione mentre si prepara la cena. Poi, alcune attività assumono diversi significati in base allo momento della giornata in cui essa avviene. Per esempio, in riferimento all'attività "mangiare", essa assume significati diversi se viene svolta in prima mattinata, in tarda mattinata o in tardo pomeriggio. Quindi, le restrizioni temporali giocano un ruolo centrale all'interno dell'identificazione delle attività umane. In aggiunta, nel monitoraggio dello stile di vita dell'anziano non risulta essere importante solo la quantità del cibo e la relativa classe di appartenenza, ma anche quando l'utente mangia ogni giorno. Pertanto, se si conosce lo stile di vita di una persona, è possibile prevedere quale attività svolgerà in un particolare momento. Per esempio, se l'utente è solito fare la doccia due volte al giorno, cioè appena sveglio e prima di andare a letto, il sistema può prevedere l'attività di "fare la doccia", se l'utente si alza dal letto e va in bagno in prima mattinata.

Il nostro lavoro presenta un nuovo approccio per risolvere il problema del riconoscimento delle attività umane. La novità della nostra metodologia è rappresentata dalla combinazione di un'ontologia di dominio con il motore CEP. Essa è in grado di riconoscere particolari sequenze di eventi. Inoltre, grazie alla sintassi intuita di Esper e all'interfaccia utente opportunamente creata, le query sul sistema possono essere effettuate anche da un utente non esperto.

Tale metodologia può effettuare un'analisi in tempo reale implementando semplicemente opportuni *trigger* nel Database. Tale caratteristica permette, insieme all'interfaccia di semplice utilizzo, di utilizzare il framework anche per il monitoraggio e l'assistenza remota dell'anziano. Per esempio, è possibile integrare tale metodologia con un sistema che invia un *allert*, per esempio ad un database, nel quale si specifica l'attività che l'utente sta svolgendo in quel momento. In questo contesto, assume un importante rilevanza la comunicazione di eventuali cadute dell'anziano o di un cambiamento improvviso di abitudini.

Inoltre, tale metodologia può essere sfruttata anche al fine di favorire il risparmio energetico dell'abitazione. Se, per esempio, dalle abitudini degli abitanti di una casa emerge che nessuno frequenta durante la mattina una particolare stanza, è possibile pensare di tenere la temperatura di quella stanza più bassa rispetto alle altre.

Bibliografia

[1] The ambient assisted living (aal)joint programme. [Online]. Available: http://www.aal-europe.eu/

[2] http://www.aalisabeth.it/

[3] T. R. Gruber. (retrieved: March, 2014) What is an Ontology? [Online]. Available: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[4] Progetto AALISABETH: Risultato D2.2, Specifica funzionale del sistema AALISABETH

[5] http://esper.codehaus.org/

[6] http://d2rq.org/d2rq-language

[7] D. Bonino, E. Castellina, and F. Corno, "The dog gateway: enabling ontology-based intelligent domotic environments." IEEE Trans. Consumer Electronics, no. 4, pp. 1656–1664.

[8] Pellet reasoner plug-in for protégé 4. [Online]. Available: http://clarkparsia.com/pellet/protege/

[9] Apache jena. [Online]. Available: http://jena.sourceforge.net/

[10] Protégé. [Online]. Available: http://protege.stanford.edu/