Consiglio Nazionale delle Ricerche

Istituto di Scienza e Tecnologie dell'Informazione “A.Faedo” (ISTI)

Laboratorio di Domotica

La casa apprende automaticamente le abitudini dell'utente

Salvatore Leone, Dario Russo, Vittorio Miori

Indice generale

1 Sommario

La relazione illustra gli obiettivi, i risultati e il processo di sviluppo seguiti durante la realizzazione del progetto, il quale è stato realizzato nel Laboratorio di Domotica presso l'Istituto di Scienza e Tecnologie dell'Informazione ISTI "Alessandro Faedo" del CNR di Pisa.

Lo scopo del progetto è stato quello di sviluppare un sistema di apprendimento automatico che fosse in grado di imparare le abitudini di un utente di un sistema domotico al fine di configurare automaticamente i dispositivi facenti parte dell'impianto.

Sono stati analizzati diversi meccanismi di apprendimento e si è poi optato per un classificatore bayesiano che, eseguendo un'analisi dei dati di utilizzo dei dipositivi, fosse in grado di apprendere le azioni ricorrenti e di offrire la possibilità di sostituirsi all'utente nell'esecuzione delle medesime.

Allo scopo di illustrare i risultati teorici è stato realizzato un prototipo con il quale sono stati eseguiti dei test su un insieme di dati che simulavano quelli raccolti durante l'utilizzo del sistema. Il programma individua correttamente le azioni più comuni, a partire da queste produce opportunamente delle regole e queste vengono infine schedulate ed eseguite al momento opportuno.

2 Domonet

2.1 Introduzione

Una delle grosse limitazioni alla diffusione di sistemi domotici è l'impossibilità di cooperazione tra tecnologie appartenenti a diversi standard e/o produttori. Quindi, una volta che l'utente investirà su un tipo di sistema, sarà vincolato in futuro all'acquisto di sole tecnologie compatibili con esso. E ciò si traduce in una dipendenza commerciale dell'utente nei confronti di un certo produttore.

Emerge quindi l'esigenza di riuscire a mettere in contatto queste tecnologie e di permetterne l'interoperabilità. L'obiettivo che si prefigge DomoNet, come verrà descritto più in dettaglio nel seguito, è proprio quello di studiare un'architettura che permetta la collaborazione tra dispositivi eterogenei dal punto di vista tecnologico e di fornire la possibilità di implementare applicazioni su di essi.

Questo diventa possibile creando un livello di astrazione che permetta di descrivere i dispositivi domotici sia dal punto di vista fisico che comportamentale in modo da avere una visione omogenea indipendentemente dalle tecnologie sottostanti. L'idea è quella di avere un gruppo di *web service* ognuno dei quali è capace di interagire con i dispositivi domotici da lui fisicamente raggiungibili,

tipicamente in un'area geografica ben precisa e circoscritta intorno sua locazione fisica. Ogni area geografica, è rappresentata logicamente dal *web service*, il quale espone come servizi l'insieme delle funzionalità offerte dai dispositivi presenti. Ogni *web service* deve essere in grado di interpretare e agire di conseguenza allo stato del livello astratto usando dei moduli capaci di interfacciarsi con i *middleware* dei dispositivi.

2.2 Scenario

Per comprendere la necessità di un framework che garantisse l'interoperabilità tra i diversi sistemi domotici presenti sul mercato, siamo partiti da uno scenario generico. In un ipotetico ambiente in cui coesistono alcuni tra i principali *middleware* domotici, è possibile identificare ogni sistema come una particolare "sotto-rete", pur senza soffermarsi sui dettagli e sulle infrastrutture necessarie.

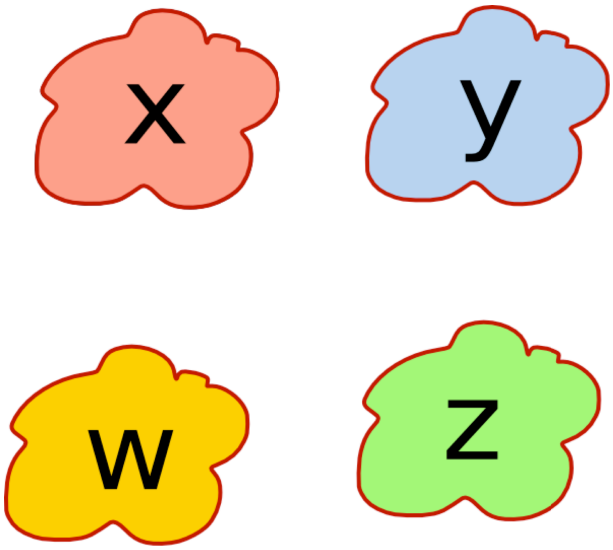

Figura 2.1: Rappresentazione dei middleware domotici.

Nella figura precedente ogni “nuvola” rappresenta un *middleware*, ovvero una sotto-rete all’interno della quale sono presenti sia le infrastrutture *hardware* e *software* per il supporto, sia i dispositivi conformi. Tutti i dispositivi all’interno della stessa sotto-rete colloquiano e cooperano per costituire un sistema funzionante ed indipendente, ma chiuso. Di fatto è impossibile controllare un dispositivo che si trova in una nuvola tramite un altro presente in una nuvola diversa, poiché non esiste nessun tipo di collegamento tra le due sotto-reti. Per collegamento si intende un’infrastruttura logica che consenta ai dispositivi di qualsiasi sotto-rete di conoscere tutti i dispositivi presenti nelle altre sotto-reti.

Infine, anche qualora fosse possibile avere una visione completa dell’intera rete, sarebbe comunque necessario un meccanismo comune per lo scambio di informazioni tra i dispositivi. Vi sono quindi due aspetti da risolvere: da un lato riuscire a costruire un’infrastruttura logica di collegamento tra i vari *middleware*, dall’altro mettere a punto un linguaggio di comunicazione universale veicolato da quest’infrastruttura. Una possibile soluzione a questi due problemi è quella di introdurre tra ogni coppia di sotto-reti un modulo *hardware* e/o *software* che sia provvisto di un’interfaccia verso entrambe le nuvole. Ogni modulo, che nella terminologia generale è detto *gateway*, deve non solo fornire funzionalità di traduzione tra i due protocolli delle sotto-reti che collega, ma anche conciliare le differenze tra i paradigmi di comunicazione su cui si basano i due sistemi.

Quest’ultima necessità risulta evidente quando i sistemi domotici che si

desidera collegare presentano caratteristiche notevolmente differenti (ad es. *middleware* a configurazione manuale vs. *plug and play*). L'implementazione di un particolare *gateway* non può dunque prescindere da una conoscenza approfondita di entrambi i sistemi domotici da collegare. Questa soluzione, tuttavia, risulterebbe poco scalabile a causa del numero di *gateway* necessari al crescere dei sottosistemi da collegare: infatti, all'aumentare del numero delle sotto-reti, il numero di *gateway* da sviluppare cresce sensibilmente. A questa prima soluzione se ne è preferita una seconda, che interviene ad un livello superiore. La validità e l'applicabilità dell'idea che si trova dietro tale approccio è stata derivata direttamente dal mondo delle reti di calcolatori. In passato diversi costruttori hanno sviluppato diverse infrastrutture per la costruzione di reti tra piattaforme *hardware* e *software* dello stesso produttore (ad esempio AppleTalk dei sistemi Macintosh, NFS per i sistemi Unix, NetBeui per i PC Windows oppure alle reti geografiche IBM SNA, NOVELL/IPX e altre ancora). Questa filosofia creava notevoli problemi di scalabilità delle reti poiché risultava impossibile mettere in comunicazione macchine appartenenti ad infrastrutture diverse. Per risolvere questo problema le diverse aziende in gioco hanno cercato di imporre il proprio sistema come unico standard. Nessuna di esse è riuscita in questo obiettivo, e il motivo di tale fallimento è dovuto soprattutto allo sviluppo di un'infrastruttura, ispirata alla pila ISO/OSI, in grado di superare il vincolo dell'unicità della piattaforma. Si tratta del noto *stack* di protocolli TCP/IP.

Sulla base delle stesse considerazioni si è pensato di introdurre un'ulteriore nuvola in grado di gestire i diversi sistemi domotici e che fosse in grado di consentire la comunicazione tra nodi appartenente a sotto-reti diverse. La nuova infrastruttura sviluppata prende il nome di DomoNet, basa il suo funzionamento sui *web services* e permette di ottenere una visione completa dell'intera topologia della rete secondo un modello SOA.

Al fine di permettere l'interoperabilità tra i vari nodi, è necessario sviluppare degli opportuni *gateway* detti *tech manager* ed utilizzare una grammatica standard basata su XML chiamata D*omoML*. Sarà necessario avere un *gateway* per ognuna delle nuvole presenti nel sistema e ognuno di essi dovrà avere un'interfaccia verso il sistema domotico al quale siconnette e un'altra interfaccia rivolta verso DomoNet.

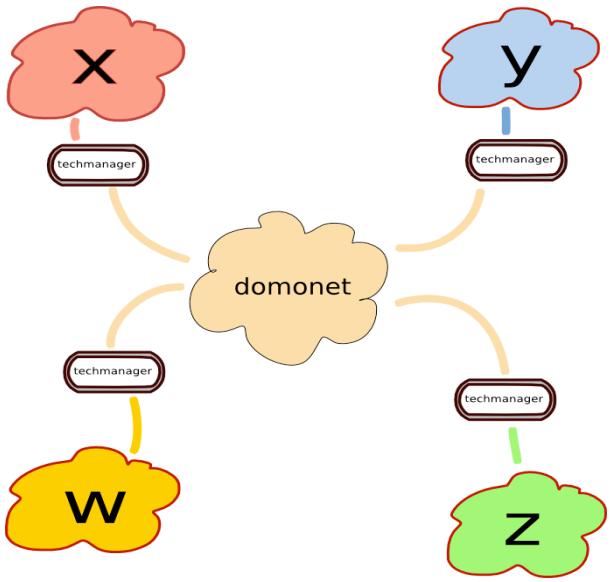

Figura 2.2:framework domonet.

3 Machine Learning

3.1 Apprendimento

L'apprendimento è un processo *induttivo*, che permette di analizzare un insieme di esempi e di inerirne un insieme di regole che verranno poi utilizzate per lo svolgimento di determinati compiti. L'utilizzo "standard" di un calcolatore comporta tipicamente un processo inverso, *deduttivo*, che passa dalle regole (i programmi) ai risultati. Le regole provengono quindi dall'esterno e ciò esclude quindi la presenza di una qualsiasi forma di apprendimento.

Possiamo quindi interpretare l'apprendimento come un processo di estrazione di informazione da un insieme di esempi. Proseguendo su questa strada possiamo osservare come l'apprendimento spesso consista nell'effettuare una *ricerca* in uno spazio di possibili ipotesi e selezionare fra queste quella maggiormente appropriata, rispetto all'insieme di allenamento e ad eventuali vincoli o conoscenze note a priori[1].

L'estrazione di informazione dagli esempi è quindi il problema centrale del

1 (Mitchell, Tom M, 1997)

machine learning o dell'apprendimento, che definiamo automatico per distinguerlo da quello naturale dell'uomo.

3.2 Il problema dell'apprendimento automatico

Fin da quando i calcolatori furono inventati, una delle prime sfide è stata quella di costruire programmi le cui prestazioni fossero in grado di migliorare nel tempo in base all'esperienza acquisita.

Vediamo una definizione più precisa di questo concetto:

> *si dice che un programma apprende dall'esperienza E rispetto ad una classe di compiti T e ad una misura delle performance P, se la sua performance nell'eseguire i compiti in T migliora con l'esperienza E.*[2]

Quindi, in generale, per poter definire correttamente un problema di apprendimento è necessario individuare queste tre entità:

- la classe T dei compiti che il *software* dovrà svolgere;
- una misura delle performance P;
- un'esperienza di allenamento E.

Nel corso degli anni sono stati sviluppati diversi algoritmi utili all'apprendimento e questi hanno mostrato la loro efficacia e utilità in un vasto insieme di applicazioni:

- *Data mining* , in cui vi sono vasti database contenenti informazioni che

2 *Ibidem*

contengono delle regolarità implicite che possono venire scoperte automaticamente. Ad esempio, dall'analisi di cartelle cliniche è possibile estrarre conoscenza medica.

- *Software* "difficili" da programmare. Questo può succedere quando si ha a che fare con domini poco conosciuti in cui non si hanno le conoscenze adatte a definire algoritmi efficaci ovvero non vi sono esperti umani, come ad esempio nel caso dell'analisi del DNA. Un altra possibile applicazione è data da tutta una classe di compiti che gli esseri umani sanno svolgere facilmente ma dei quali non sono in grado di esprimere la metodologia adottata. Tra questi abbiamo il problema del riconoscimento di immagini o della lingua parlata.
- Domini dove i programmi devono dinamicamente adeguarsi ai cambiamenti delle condizioni in cui devono operare. In quest'ultimo caso rientrano tutte quelle applicazioni che devono adattarsi alle preferenze dell'utente. Ne sono un esempio le applicazioni che suggeriscono gusti musicali, pagine internet o prodotti, in base al comportamento passato dell'utilizzatore di tali servizi.

3.3 Definizione di un sistema di apprendimento

Per procedere allo sviluppo di un sistema di apprendimento automatico occorre

effettuare precise scelte progettuali.

Per prima cosa si deve decidere la **modalità di *training*.** Le due opzioni possibili individuano due classi di apprendimento automatico: *non supervisionato* e *supervisionato*. Nel primo caso non è possibile individuare ingressi al sistema. Per cui, per far sì che il *software* "comprenda" di aver fatto una scelta giusta o sbagliata, ci deve essere qualche metodo di rinforzo implicito. Cioè si deve "premiare" o "punire" il sistema in seguito alle scelte compiute. Quando si parla di apprendimento supervisionato invece il sistema riceve uno o più *input* dall'esterno e, in base a questi input e all'insieme di allenamento, produce un certo risultato.

Occorre poi decidere la **rappresentazione degli oggetti del dominio**. Un sistema di apprendimento deve tipicamente classificare oggetti del mondo reale oppure eseguire dei compiti in un certo ambiente. Nel primo caso si devono rappresentare gli oggetti da raggruppare e di solito, a tale proposito, si utilizzano vettori o grafi. Invece, per rappresentare l'ambiente, si utilizzano matrici o rappresentazioni grafiche più complesse quali contorni o *textures*.

Un altro passo importante è la scelta della **funzione obiettivo**. Questa è un'espressione formale della conoscenza appresa. Viene usata per determinare le prestazioni del programma di apprendimento. Esempi di tale funzione sono polinomi, alberi, reti neurali e insiemi di regole. È da notare che in casi reali è difficile ottenere un sistema in grado di apprendere perfettamente la funzione

obiettivo. In questi casi si sceglie un'approssimazione che abbia la stessa forma prescelta (polinomio, regole etc) e che approssimi il meglio possibile la funzione reale.

L'ultimo passo è la **scelta dell'algoritmo** in conseguenza della funzione obiettivo. Se questa è una funzione probabilistica, allora si sceglierà una metodo statistico come ad esempio l'apprendimento bayesiano. Se invece la funzione è algebrica, ad esempio è un polinomio, allora si utilizzerà una metodologia algebrica come l'algoritmo di discesa del gradiente. Infine, se la funzione obiettivo è composta da espressioni logiche allora si utilizzerà un algoritmo basato sulla conoscenza, per esempio uno che sfrutti gli alberi di decisione.

3.4 Approcci all'apprendimento

Abbiamo visto che il problema principale attorno a cui si sviluppa l'apprendimento automatico è quello di indurre funzioni generali a partire da un insieme di esempi particolari. Al variare della funzione in esame corrispondono diversi modi di affrontare il problema del *machine learning*. Di seguito descriveremo brevemente alcuni di questi approcci.

3.4.1 Concept learning.

Consiste nell'inferire una funzione booleana a partire da un insieme di allenamento composto da campioni di esempi positivi e negativi. Il problema del

concept learning può anche essere formulato come una ricerca in uno spazio di possibili ipotesi, dell'ipotesi che meglio si adatta all'insieme di allenamento. Questa metodologia di lavoro si basa su l'assunzione che l'ipotesi che meglio approssima la funzione su un insieme sufficientemente largo di esempi, sia anche la migliore approssimazione per la funzione obiettivo, anche per ogni altro esempio non ancora osservato.

3.4.2 Decision tree learning.

E' un metodo di induzione per l'approssimazione di funzioni a valori discreti in cui la funzione appresa può essere rappresentata come un albero di decisione o, in favore di una migliore leggibilità, come un insieme di regole *if-then-else*. E' uno dei metodi che, anche grazie alla sua robustezza al rumore dei dati, è stato applicato ad una vasto range di compiti. Tipici problemi affrontabili con gli alberi di decisione sono quelli in cui:

- le istanze sono rappresentata da attributi chiave-valore;
- possono richiedere descrizioni disgiuntive;
- l'insieme di *training* può contenere errori;
- nell'insieme di allenamento possono mancare dei valori per alcuni attributi.

Problemi che rientrano in queste caratteristiche sono tipicamente quelli di

classificazione, in cui lo scopo è classificare gli esempi in uno dei possibili (e discreti) insiemi di categorie.

3.4.3 Artifical neural networks.

Le reti di neuroni artificiali sono uno dei metodi più robusti per l'apprendimento di funzioni obiettivo i cui valori possono essere reali, discreti o vettoriali. I problemi che meglio si adattano ad essere trattati da una rete neurale sono quelli in cui gli esempi di allenamento sono composti da dati che sono complessi e affetti da rumore. Tipicamente si tratta di informazioni provenienti da sensori quali possono essere telecamere o microfoni. È anche possibile utilizzare questo approccio per i problemi affrontabili attraverso gli alberi di decisione, e in questo caso i due sistemi forniscono risultati la cui accuratezza è comparabile[3]. Riassumendo, i problemi trattabili con questa metodologia sono quelli in cui:

- le istanze sono composte da coppie attributo-valore;
- l'output della funzione obiettivo è composta da valori reali, discreti o da vettori;
- gli esempi di *training*possono contenere errori;
- sono accettabili lunghi tempi per l'allenamento;

3 (Shavlik Jude W.; Mooney Raymond J.; Towell Geoffrey G. , 1991)

- è richiesta una valutazione veloce della funzione obiettivo;
- non è importante per gli esseri umani riuscire a interpretare la funzione obiettivo.

3.4.4 Statistical learning.

Il ragionamento statistico fornisce un approccio probabilistico all'inferenza. Si basa sull'assunzione che le quantità di interesse siano governate da una certa probabilità di distribuzione e che una decisione ottimale possa essere fatta ragionando su queste probabilità e sui dati osservati.

3.5 Bayesian learning

L 'approccio bayesiano fornisce un metodo probabilistico alla soluzione del problema dell'apprendimento automatico. Partendo da una situazione in cui si hanno diverse ipotesi tra le quali scegliere, gli algoritmi bayesiani calcolano le probabilità esplicite di ognuna di queste ipotesi. Questo risultato viene poi utilizzato per selezionare quella più adatta.

Vi sono alcune caratteristiche comuni a tutti i metodi di apprendimento bayesino:

- ognuno degli esempi contenuti all'interno del *training* set contribuisce

ad incrementare o decrementare la probabilità di una certa ipotesi;

- la conoscenza a priori viene utilizzata in combinazione con i dati osservati per il calcolo della probabilità finale;

la conoscenza a priori viene ricavata fornendo, per ognuna delle ipotesi candidate, una probabilità a priori e una distribuzione di probabilità sui dati osservati;

vengono favorite ipotesi che effettuano predizioni probabilistiche;

si possono classificare nuove istanze combinando le predizioni di diverse ipotesi, pesate con le loro probabilità.

È importante osservare che viene sempre richiesta una cognizione iniziale delle probabilità delle ipotesi in esame. Quando tale conoscenza non è però disponibile, è comunque possibile fare delle stime probabilistiche su di essa basandosi sull'esperienza o su precise assunzioni riguardo la distribuzione delle probabilità.

3.5.1 Teorema di Bayes

Il teorema di Bayes ci fornisce un metodo per il calcolo della probabilità di un'ipotesi conoscendo la sua probabilità a priori e la probabilità di osservare un

certo insieme di dati, avendo a disposizione l'ipotesi e i dati osservati.

Per enunciare il teorema verrà utilizzata un'opportuna notazione:

- P(h): rappresenta la probabilità iniziale dell'ipotesi h prima di aver esaminato l'insieme di allenamento. È anche detta *probabilità a priori* e rispecchia le conoscenze iniziali su di h. Talvolta è possibile che tale conoscenza non sia presente, per indicare questo fatto si assegna ad ogni ipotesi la stessa probabilità a priori.
- P(D): è la probabilità che l'insieme D venga osservato.

P(D|h): delinea la probabilità di osservare D in un mondo in cui si è verificata l'ipotesi h.

P(h|D): esprime la probabilità che che si verifichi l'ipotesi h, avendo osservato D. E' chiamata anche *probabilità a posteriori*, e rappresenta la nostra confidenza che h si possa verificare in seguito all'osservazione di D.

Possiamo adesso enunciare il **teorema di Bayes**:

$$P(h|D)=\frac{P(D|h)P(h)}{P(D)} \quad (3.1)$$

Innanzitutto possiamo osservare che il risultato è direttamente proporzionale al prodotto tra P(D|h) e P(h). Viceversa, esso diminuisce all'aumentare del valore P(D), il che è abbastanza intuitivo ed esprime il fatto che maggiore è la

probabilità di osservare D indipendentemente da h, minori sono le prove che D può fornire a sostegno di h.

Se con H indichiamo l'insieme delle possibili ipotesi, il teorema ci fornice un modo per selezionare una di queste. Vediamo ora un paio di assunzioni che permettono di fare questa scelta.

In molti scenari di apprendimento, data l'osservazione D, siamo interessati alla scoperta delle ipotesi più probabili. Ognuna di queste, che chiameremo ipotesi di probabilità massima, è detta **ipotesi MAP** (Maximum A Posteriori).

Formalmente:

$$h_{MAP} \equiv \underset{h \in H}{argmax} P(h|D) \equiv$$

$$\underset{h \in H}{argmax} \frac{P(D|h)P(h)}{P(D)} \equiv \quad (3.2)$$

$$\underset{h \in H}{argmax} P(D|h)P(h)$$

Notare che nel passo finale il termine P(D) è stato eliminato dal denominatore in quanto è una costante indipendente da h.

In qualche caso possiamo poi assumere che ogni ipotesi h abbia la stessa probabilità a priori, cioè:

$$P(h_i) = P(h_j) \quad \forall h_i, h_j \in H$$

Questa osservazione ci permette di applicare un'ulteriore semplificazione prendendo

in considerazione soltanto P(D|h):

$$h_{ML} \equiv \underset{h \in H}{argmax} \; P(D|h) \quad (3.3)$$

Ogni ipotesi che massimizza P(D|h) è detta **ipotesi ML** (Maximum Likelyhood).

Di fatto il teorema di Bayes ha un'applicazione molto più generale rispetto a quanto suggerito in questa discussione. Esso può essere infatti applicato ad un qualunque insieme H di proposizioni mutualmente esclusive ("il gatto è nero" e "il gatto non è nero"). Comunque per chiarire la connessione esistente con il problema dell'apprendimento automatico, possiamo vedere l'insieme D degli esempi come l'insieme di training di qualche funzione obiettivo e ci possiamo riferire ad H come lo spazio delle funzioni obiettivo candidate.

È possibile utilizzare il teorema per la costruzione di un algoritmo che calcoli direttamente le probabilità di *ognuna* delle ipotesi in esame:

Brute-Force MAP Learning:

1. foreach (h in H):
   calculate the posteriori probability P(h|D);
2. return h_{MAP} hypothesis

3.5.2 Classificatore bayesiano ottimale

Fin'ora abbiamo cercato la risposta alla domanda "Conoscendo i dati di allenamento, qual'è l'ipotesi più probabile?". Però spesso è più significativo individuare la più probabile *classificazione* di ogni nuova istanza osservata. Apparentemente potrebbe sembrare che, anche alla seconda questione, sia possibile dare una risposta applicando l'ipotesi MAP alla nuova istanza. In realtà questo non è generalmente vero.

In generale la classificazione più probabile la si ottiene combinando fra di loro le predizioni ottenute da tutte le ipotesi, pesate con le rispettive probabilità a posteriori. Se la possibile classificazione di un nuovo esempio può determinare valori v_j appartenenti a qualche insieme V di possibili classi, allora la probabilità $P(v_j|D)$ che rappresenta la classificazione corretta per v_j è:

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

E la **classificazione ottimale bayesiana** è il valore v_j per cui $P(v_j|D)$ è massimo:

$$\underset{v_j \in V}{argmax} = P(v_j|D) \quad (3.4)$$

Di seguito, vediamo un esempio che mostra un'applicazione del classificatore ottimale. Supponiamo che l'insieme delle possibili classificazioni sia V = {+,-} e

che:

- $P(h_1 \mid D) = 0.4$, $P(- \mid h_1) = 0$, $P(+ \mid h_1) = 1$
- $P(h_2 \mid D) = 0.3$, $P(- \mid h_2) = 1$, $P(+ \mid h_2) = 0$
- $P(h_3 \mid D) = 0.3$, $P(- \mid h_3) = 1$, $P(+ \mid h_3) = 0$

allora:

$$\sum_{h_i \in H} P(+|h_i) P(h_i|D) = 0.4$$

$$\sum_{h_i \in H} P(-|h_i) P(h_i|D) = 0.6$$

e quindi:

$$\underset{v_j \in (+,-)}{argmax} P(v_j|D) = -$$

Osserviamo che se avessimo utilizzato l'ipotesi MAP avremmo scelto h_1, ottenendo quindi una classificazione positiva. Tale risultato non sarebbe stato corretto. Infatti, come evidenziato dalla combinazione di tutte le ipotesi, la classificazione più corretta è quella negativa.

3.5.3 Classificatore bayesiano “naive”

Tra le più usate implementazioni per un classificatore bayesiano abbiamo il

cosiddetto *classificatore bayesiano naive*.

Si applica a problemi di apprendimento in cui ogni istanza x da classificare è definita da una tupla di valori di attributi e in cui la funzione obiettivo può prendere valori solo da un insieme finito V.

Siano $<a_1,a_2,...,a_n>$ i valori degli attributi della nuova istanza, lo scopo di questa metodologia è quello di assegnare a nuovi esempi il valore obiettivo più probabile v_{MAP}:

$$v_{MAP} = \underset{v_i \in V}{argmax} \; P(v_j | a_1, a_2, \ldots, a_n)$$

Possiamo utilizzare il teorema di Bayes per riscrivere l'espressione precedente:

$$v_{MAP} = \underset{v_j \in V}{argmax} \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots, a_n)}$$

$$= \underset{v_j \in V}{argmax} P(a_1, a_2, \ldots, a_n | v_j) P(v_j) \quad (3.5)$$

Adesso, basandoci sui dati di training, dobbiamo stimare i due termini dell'equazione precedente. Per esprimere una valutazione di $P(v_j)$ è sufficiente contare la frequenza con cui ogni valore obiettivo v_j appare negli esempi di allenamento. Non è altrettanto semplice stimare i diversi termini $P(a_1, a_2,...,a_n | v_j)$ allo stesso modo; infatti il numero di questi termini è uguale al numero delle

possibili istanze per il numero dei possibili valori obiettivo. Quindi per ottenere una stima affidabile, avremmo bisogno di vedere ogni istanza più di una volta.

Per ovviare a questa difficoltà, il classificatore bayesiano naive (cioè "ingenuo") assume che, dato il valore obiettivo, i valori degli attributi siano condizionatamente indipendenti. Detto altrimenti, assunto un certo valore obiettivo per l'istanza, la probabilità di osservare la congiunzione $a_1,a_2,...a_n$ è data dal prodotto delle probabilità individuali degli attributi. Viene dunque fatta la seguente *assunzione naive*:

$$P(a_1, a_2, ..., a_n|v_j) = \prod_i P(a_i|v_j) \quad (3.6)$$

Sostituendo all'interno dell'equazione (3.5) otteniamo la **classificazione bayesiana naive:**

$$P(a_1, a_2, ..., a_n|v_j) = \prod_i P(a_i|v_j) \quad (3.7)$$

Concludendo, possiamo affermare che l'approccio bayesiano naive richiede per l'apprendimento un passo iniziale in cui vengono calcolati i vari $P(v_j)$ e $P(a_i|v_j)$. Per misurare tali probabilità ci si basa sulla loro frequenza all'interno dell'insieme di allenamento. L'insieme di tutte queste stime corrisponderà all'insieme delle ipotesi apprese. Esse sono poi utilizzate per classificare ogni nuova istanza applicando l'equazione (3.7).

Lo spazio delle ipotesi è quindi lo spazio dei possibili valori che possono essere assegnati alle probabilità $P(v_j)$ e $P(a_i|v_j)$. È importante osservare come, a differenza degli altri metodi di apprendimento automatico, non vi sia una ricerca esplicita nello spazio delle ipotesi. Queste vengono invece costruite contando le occorrenze delle varie combinazioni dei dati dell'insieme di training.

3.6 Conclusione

In diversi campi, è stato misurato che le performance di un sistema di apprendimento bayesiano, sono paragonabili a quelle delle reti neurali e degli alberi di decisione. Inoltre è importante osservare che a partire dallo stesso spazio delle ipotesi e dalla stessa conoscenza a priori, nessun altro metodo di classificazione conosciuto si comporta, in media, meglio del classificatore bayesiano naive[4].

4 DomoBrain

Si è discusso di DomoNet e delle sue funzionalità. È emerso come lo scopo

4 (Mitchell, Tom M, 1997)

principale del sistema fosse quello di implementare una sorta di “telecomando universale” in grado di permettere l'interoperabilità tra dispositivi appartenenti a diverse tecnologie domotiche.

L'obiettivo del progetto di tesi è stato quello di realizzare DomoBrain, un sistema da integrare all'interno di DomoNet, che fosse in grado di apprendere le abitudini di un utente di un sistema domotico. In questo capitolo vedremo un esempio di quello che potrebbe essere uno scenario rappresentante un ambiente domotico e introdurremo brevemente l'architettura di DomoBrain.

4.1 Scenario

Un tipico scenario di utilizzo di un ambiente domestico presuppone che l'utente abbia e mantenga nel tempo un certo insieme di comportamenti abituali. Per capire meglio possiamo immaginare che, giorno dopo giorno e più o meno agli stessi orari, l'utente compia queste attività:

- la mattina si alza ed esce di casa (ad esempio per recarsi sul posto di lavoro);
- la sera rientra in casa;
- la notte si corica.

È implicito che in ognuno di questi momenti egli compia delle azioni ben precise:

- ogni mattina solleva le avvolgibili, poi accende il riscaldamento del bagno per farsi la doccia;
- al suo rientro la sera, appena fa buio, accende le luci e il riscaldamento;
- prima di andare a letto, spegne tutti i dispositivi.

Lo scopo del progetto è stato quello di raccogliere informazioni su questi comportamenti abituali e, attraverso un'analisi statistica di questi dati, realizzare un sistema di regole che automatizzi le azioni più comuni. Così facendo è possibile regolare il funzionamento dei dispositivi domotici in modo da rendere la casa a misura di utente.

I vantaggi derivanti da una simile applicazione sono molteplici. I più evidenti si riscontrano a livello di comfort. Una obiezione che potrebbe essere fatta è che, anche senza un sistema di apprendimento, sarebbe possibile sfruttare le tecnologie domotiche attualmente disponibili per ottenere una dimora ritagliata sulle proprie abitudini. Questo è vero, ma sarebbe comunque compito dell'utente impostare i vari dispositivi , uno ad uno, secondo le proprie esigenze. Invece, con l'adozione di un sistema in grado di apprendere da solo tali costumi, tutto ciò che si richiede agli abitanti della casa è di viverci, continuando ad utilizzare i vari

servizi come di consueto. L'utente non deve più preoccuparsi di configurazioni e impostazioni che potrebbero, in alcuni casi, risultare complesse. Sarà il sistema stesso che, dopo aver raccolto dati a sufficienza, proporrà all'utente una nuova regola la quale avrà lo scopo di automatizzare una certa funzione. Inoltre, se l'utente dovesse cambiare le proprie abitudini (ad esempio perché è cambiata la stagione) il sistema si adatta automaticamente alle nuove esigenze e non vi è quindi la necessità di riconfigurare manualmente tutti i dispositivi.

Altre conseguenze positive si riscontrano sul piano della sicurezza e il risparmio energetico. Sarà infatti impossibile dimenticarsi le “buone abitudini”, dato che queste verranno eseguite automaticamente dal sistema. Quindi se per una sera l'utente dimentica di spegnere particolari luci, di chiudere i rubinetti del gas o la porta d'ingresso, queste azioni verranno comunque eseguite alla solita ora dal sistema.

Anche dal punto di vista del prolungamento e del mantenimento dell'autonomia è possibile riscontrare diversi benefici. Ciò risulta essere di particolare interesse per tutte quelle categorie di persone con esigenze particolari. Ne sono un esempio gli anziani o coloro i quali attraversano, per diversi motivi, periodi di difficoltà momentanea (donne in gravidanza, persone con fratture e altre invalidità temporanee). In questi casi, anche i gesti quotidiani più elementari possono andare a costituire ostacoli anche insormontabili. È quindi facile immaginare come il sistema venga loro in aiuto eseguendo

automaticamente e per tutto il tempo necessario, l'insieme di azioni abituali che sono state acquisite nel tempo.

4.2 Architettura

Come abbiamo visto, Domonet si compone di un lato *server* e di un lato *client*. DomoBrain è sostanzialmente un client parallelo a quello "ufficiale" di DomoNet. Da questo differisce nello scopo, che non è quello di interagire con i dispositivi dell'abitazione, ma di monitorare l'utilizzo che, da parte dell'utente, viene fatto di tali dispositivi. Per permettere ciò, il server mette a disposizione una serie di *Web Services*, mentre sul lato del client troviamo un *socket* al quale DomoNet invierà i cambiamenti di stato dei dispositivi.

Principalmente, il lavoro svolto da DomoBrain si compone di quattro attività principali:

- raccolta di informazioni sull'utilizzo dei dispositivi;
- analisi di questi dati;
- creazione di regole in base ai risultati di tale analisi;
- esecuzione delle regole.

Questi compiti vengono eseguiti da diverse entità *software*. Di queste,

vediamo ora una breve descrizione, riservando al seguito una trattazione più approfondita.

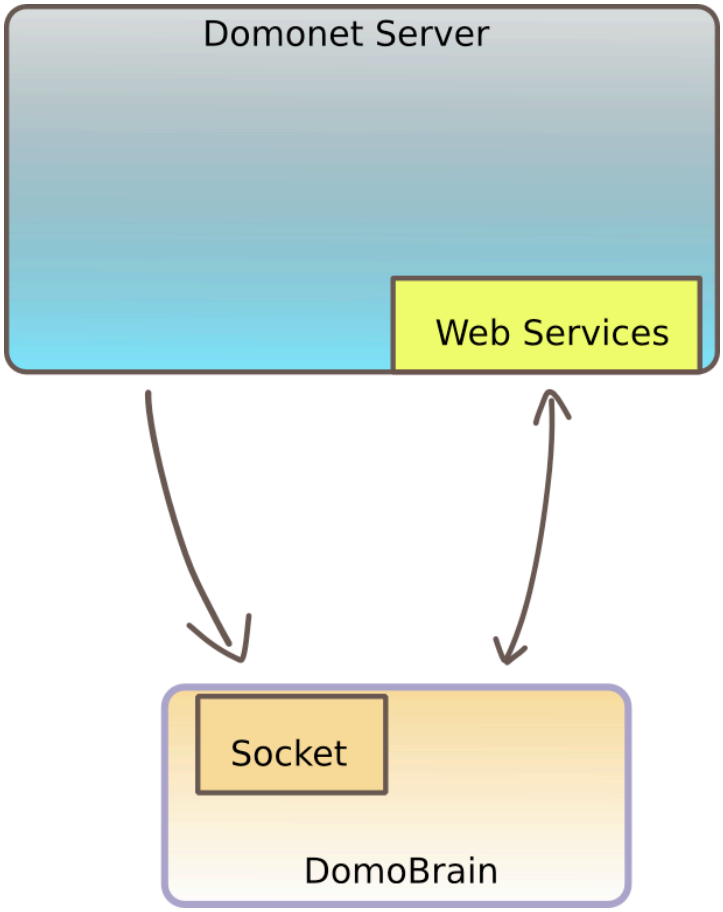

Figura 4.1: schema delle connessioni esistenti tra DomoNet e DomoBrain

Il monitoraggio dei dati di uso dei dispositivi della rete viene svolta dal **DomoBrainSniffer**. Ogni volta che un utente utilizza uno dei dispositivi della rete, modificandone lo stato, viene generato da DomoNet un particolare messaggio che contiene informazioni sul cambiamento di stato subito dal dispositivo. Questi dati sono inviati in *multicast* a tutti i *client* che dichiarano il

loro interesse alla ricezione di tali informazioni. Lo scopo del DomoBrainSniffer è proprio quello di esprimere il proprio interesse alla ricezione di questi messaggi. Deve quindi "sniffare" questi pacchetti inviando quelli di interesse su di un file di log.

L'analisi dei dati raccolti e la creazione delle regole è un'attività svolta dal **DomoBrainLearner.** Si tratta sostanzialmente di un'implementazione di un classificatore bayesyano naive. Il suo compito è quindi quello di accedere al file di log e di analizzarne il contenuto allo scopo di inferire conoscenza sull'utilizzo dei dispositivi. Essenzialmente calcola la probabilità a posteriori di un certo evento, se tale probabilità è superiore ad una data soglia, allora a partire da tale evento verrà creata una nuova regola. Ad esempio, se la probabilità che l'utente accenda la luce della cucina alle ore 20:00:00 è superiore alla soglia di 0.30, allora da tale evento deve essere creata una nuova regola.

Le regole sono sostanzialmente dei messaggi che verranno inviati al *server* ad una certa ora. Questi messaggi conterranno delle istruzioni per i dispositivi presenti nella rete. Facendo riferimento all'esempio precedente, una possibile regola conterrà istruzioni per inviare alle ore 20:00:00 un messaggio recante il comando di accensione della luce della cucina.

È importante sottolineare che queste entità non rappresentano singole unità di software, ma a loro volta si compongono di più parti i cui dettagli verranno chiariti in seguito.

Una rappresentazione grafica che riassume il lavoro svolto da DomoBrain è data nella seguente figura:

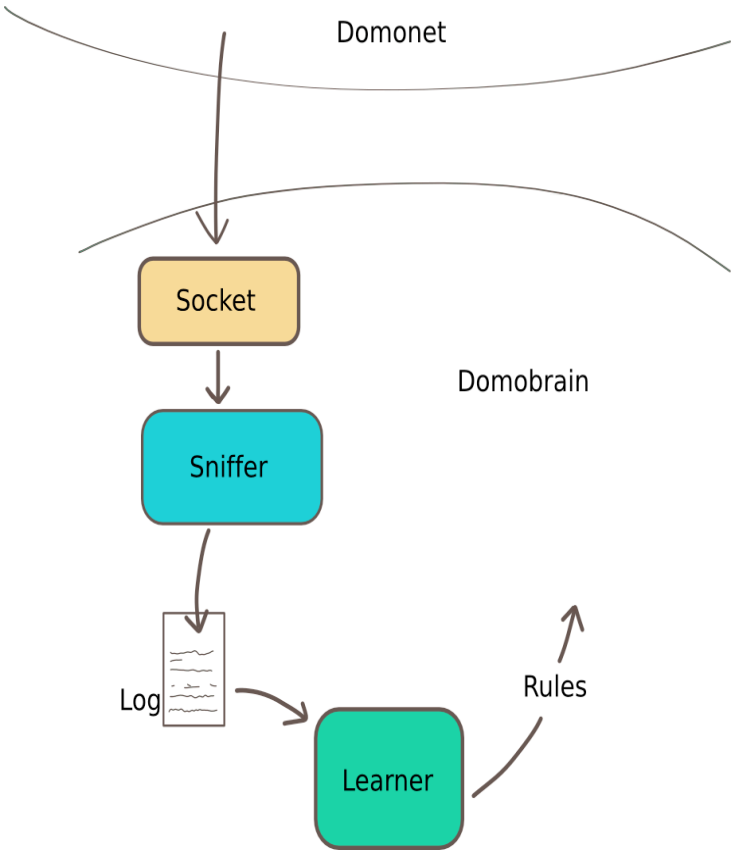

Figura 4.2: schema di funzionamento di DomoBrain

5 Sviluppo del progetto

5.1 Design del learner

Abbiamo visto cosa si intende per *machine learning* e quali sono le scelte progettuali che devono essere fatte in fase di sviluppo di un sistema di apprendimento automatico. Vediamo ora in dettaglio le decisioni prese per l'implementazione di DomoBrain.

5.1.1 Compito e misura delle performance

Abbiamo visto, che il problema del *machine learning* è quello di realizzare un programma che attraverso l'esperienza migliori la sua abilità nell'eseguire una certa attività.

Il primo passo nella definizione di un sistema di apprendimento automatico è quindi quello di definire quale sia il compito per il quale si richiede tale crescita. Nel nostro caso, DomoBrain dovrà imparare le abitudini di un utente di un sistema domotico.

Per poter osservare un qualsivoglia miglioramento occorre però definire una qualche metrica che possa essere utilizzata al fine di giudicare il lavoro svolto dal software. Per DomoBrain una tale misura è data dalla percentuale di abitudini apprese correttamente.

5.1.2 Scelta dell'insieme di allenamento

La scelta del tipo di esperienza sulla quale il nostro *learner* effettuerà la procedura di apprendimento è una decisione che può avere un grosso impatto sulla scelta degli algoritmi di apprendimento utilizzati e sull'esito di questo processo di *learning*.

Nel caso di DomoBrain si è scelto di utilizzare come insieme di allenamento un file di log, il quale contiene le informazioni sull'utilizzo di un certo dispositivo.

Un altro possibile approccio per la definizione di un insieme di *training* sarebbe potuto essere quello di fornire al sistema una serie di esempi preconfezionati, indicando per ognuno di essi il risultato atteso. Ma la natura stessa dei file di log permette di evitare un tale artificio. Infatti questi file contengono le informazioni sul reale ed effettivo utilizzo del sistema domotico, e si assume quindi che le informazioni in essi contenute siano essenzialmente corrette e che il risultato atteso sia esattamente quello riportato sul log. Ad esempio, se l'utente accende la lampadina della camera da letto ogni sera alle ore 22:00:00, si può essere certi che quello descritto sia esattamente il comportamento da apprendere.

Un'altra conseguenza di questa scelta è che vi è una totale identità tra l'ambiente di allenamento e quello finale nel quale il sistema verrà utilizzato. Questo risolve uno dei principali problemi nella scelta dell'insieme di esempi di

addestramento, cioè quello stimare quanto esso sia in grado di rappresentare l'ambiente operativo "reale". In questo caso i due mondi coincidono e quindi ci sarà la massima rappresentatività possibile. Se proprio si vuole dare una qualche forma di separazione tra i due insiemi, essa va cercata dal punto di vista "temporale" piuttosto che da quello "spaziale" (da una parte gli esempi e dall'altra le nuove istanze). Infatti fino ad un certo momento, i dati verranno raccolti al solo scopo di accumulare abbastanza informazione per poter far partire il processo di apprendimento. Dopo tale istante, ogni ulteriore evento innescherà il meccanismo di *learning*.

5.1.3 Scelta della funzione obiettivo

Il prossimo passo è quello di decidere il tipo di conoscenza che deve essere appresa. Nel caso di DomoBrain lo scopo è apprendere un'insieme di abitudini a partire dalle informazioni raccolte nel file di log. Per capire come scegliere la funzione obiettivo è necessario prima approfondire quale sia la natura dei dati memorizzati in questi file e che cosa si intenda esattamente con il termine "abitudini".

Abbiamo visto che DomoNet emette un messaggio informativo ogni volta che l'utente utilizza un dispositivo cambiandone lo stato. Se con il termine "evento" definiamo una di queste variazioni di stato dei dispositivi, allora possiamo dire che la funzione cercata avrà come dominio l'insieme degli eventi.

Inoltre osserviamo che con l'espressione “abitudini” ci riferiamo a tutte quelle azioni che l'utente esegue ogni giorno alla stessa ora. DomoBrain ha il compito di imparare queste abitudini e di eseguirle al posto dell'utente. Quindi, possiamo vedere un'abitudine come un comando che DomoBrain *deve* inviare al sistema ogni giorno alla stessa ora. Definiamo queste azioni imperative come “regole”, e allora possiamo affermare che il codominio della nostra funzione obiettivo sarà composto dall'insieme delle regole apprese.

La nostra funzione obiettivo f sarà quindi della seguente forma:

$f : E \rightarrow R$ Dove indichiamo con E l'insieme degli eventi e con R l'insieme delle regole,

5.1.4 Rappresentazione della funzione obiettivo

Dopo aver scelto una forma per la funzione obiettivo, occorre decidere la rappresentazione che il programma utilizzerà per esprimere la conoscenza acquisita.

Le opzioni per fissare questa rappresentazione sono legate al tipo di funzione obiettivo. Nel nostro caso, abbiamo detto che una regola altro non è che un comando che deve essere eseguito ad una certa ora. Le regole con cui abbiamo a che fare sono perciò coppie del tipo <comando, ora di esecuzione>.

Un file contenente una lista di queste coppie è proprio la rappresentazione cercata. A run-time le regole saranno mantenute in una struttura a lista e,

eventuali operazioni su di essa verranno eseguite anche sul file. Per cui la lista in memoria non sarà altro che una copia, sempre aggiornata, del file.

5.1.5 Scelta dell'algoritmo di apprendimento

Viste le argomentazioni discusse, il sistema di apprendimento automatico è stato realizzato implementando un classificatore bayesiano.

L'algoritmo prende in ingresso due input:

- un evento:

  che rappresenta il comando del quale interessa calcolare la probabilità a posteriori;

- una soglia:

  che è una numero con il quale andrà confrontato il valore della probabilità a posteriori dell'evento.

Lo scopo è decidere se, a partire dall'evento in esame, si dovrà o meno creare una regola. A tale proposito, ogni volta che viene eseguito un comando, si calcola la sua probabilità e la si confronta con il valore di soglia. Se la probabilità è maggiore o uguale alla soglia allora si dovrà procedere alla creazione di una nuova regola.

Pseudocodice dell'algoritmo implementato:

```
checkBayes(event E, threshold T ):
  P ← posterioriProb(E);
  if (P >= T):
    createRule(E);
```

5.1.6 Linguaggio di programmazione

DomoNet è implementato interamente in Java e come descritto, è sviluppato attorno ad un modello SOA. Dal solo punto di vista dell'interazione sarebbe quindi stato possibile, utilizzando un qualunque linguaggio di programmazione, realizzare un *client* che fosse in grado di comunicare con i servizi disponibili.

Ma, per meglio gestire l'integrazione con DomoNet e poter così usufruire del *framework* messo a disposizione, anche DomoBrain è stato sviluppato interamente in Java.

5.1.7 XML e XMLSchema

All'interno dell'applicazione vengono utilizzati diversi file che sono di ausilio allo svolgimento delle funzioni principali del programma. Ad esempio troviamo i file di log che tengono traccia dell'utilizzo che viene fatto di un certo dispositivo; abbiamo il file delle regole che conserva le regole apprese; vi sono poi i file che memorizzano le proprietà e le impostazioni dei componenti del programma e

infine, troviamo i file che descrivono i dispositivi domotici presenti nell'ambiente.

Questi file, pur venendo utilizzati in momenti diversi e da differenti parti di codice, vengono descritti utilizzando lo stesso formato. Si tratta infatti di file XML le cui specifiche vengono date attraverso il formalismo degli XML Schema. Vediamo una breve descrizione di queste due specifiche:

- XML (eXtensible Markup Language):[5]

  è una specifica *general-purpose* per la definizione di linguaggi di *markup*. È un linguaggio, flessibile, dinamico e pronto per Internet.

- XMLSchema[6]:

  è una descrizione di un documento XML. Tipicamente tale descrizione è data attraverso una serie di vincoli sulla struttura e sul contenuto dei documenti che descrive.

In DomoBrain ogni documento XML deve essere *ben formato* e *valido* rispetto ad uno schema. Per "ben formato" si intende che il file deve essere un documento XML sintatticamente corretto. La validazione invece implica che il documento rispetti i requisiti contenuti nello schema.

5.1.8 Progetto finale

Unendo quanto detto fin'ora possiamo tracciare lo schema del nostro progetto:

5 http://www.w3.org/XML/
6 http://www.w3.org/XML/Schema

Compito:Imparare abitudiniMisura delle performance:Percentuale di abitudini apprese correttamenteInsieme di addestramento:File di logFunzione obiettivo:

Compito:	Imparare abitudini
Misura delle performance:	Percentuale di abitudini apprese correttamente
Insieme di addestramento:	File di log
Funzione obiettivo:	$f : E \rightarrow R$
Rappresentazione della funzione obiettivo:	File di regole

Tabella 1: progetto finale di DomoBrain

5.2 Interazione e integrazione con DomoNet

5.2.1 Visione di DomoNet

Abbiamo già osservato come DomoNet si componga di due entità, da una parte troviamo il lato *server* e dall'altro il lato *client*.

Il lato *client* ha il compito di collegarsi ai servizi messi a disposizione dal *server* e di comunicare con esso per permettere il controllo remoto e l'interoperabilità dei dispositivi della rete. Anche DomoBrain presenta una parte che necessita di connettersi al *server* in modo da interagire con i vari *device*. Tale interazione si snoda su due livelli, nel primo lo scopo è quello di monitorare l'uso che si fa dei *device* e nel secondo il fine è quello di realizzare l'esecuzione delle regole apprese.

Tutto il codice necessario a questo tipo di comunicazione era già presente nel *client* di DomoNet e quindi si è preferito sfruttare tale sorgente piuttosto che

procedere alla scrittura di nuovo codice che sarebbe stato comunque speculare a quello già esistente. È stato quindi creato DomoBrianClient, il quale è sostanzialmente un *wrapper* per DomoNetClient. Lo scopo di questo "nuovo" *client* è quello di collegarsi al *server* e di aprire un *socket* le cui specifiche,indirizzo ip e porta, verranno comunicate a DomoNetServer.

Abbiamo visto che i messaggi a cui DomoBrain è interessato sono quelli in cui viene riportato l'aggiornamento dello stato di un dispositivo. Tali messaggi, detti di UPDATE, vengono inviati dal *server* al *socket* che è stato aperto sul *client.* Nella sua versione iniziale, DomoNetClient utilizzava i messaggi di UPDATE a solo scopo informativo, mentre DomoBrain ha la necessità di analizzare e processare questi pacchetti. È stato perciò necessario apportare alcune modifiche al DomoNetClient.

Per quanto riguarda il *server*, è stato lasciato inalterato e di esso si è mantenuta sostanzialmente una visione *"blackbox",* in cui tutto ciò che di esso veniva osservato erano le interfacce necessarie allo scambio di dati.

Abbiamo detto che DomoNet permette l'interoperabilità tra dispositivi appartenenti a tecnologie domotiche diverse. Per rendere possibile questa forma di interconnessione DomoNet fornisce un'unica astrazione, detta DomoDevice, che permette di rappresentare tutti i dispositivi del sistema domotico a prescindere dalla loro tecnologia. Di seguito si farà riferimento al termine "DomoDevice" sia per indicare il formalismo e sia per indicare l'astrazione di un

generico dispositivo della rete. Le caratteristiche, intese come insieme di operazioni supportate, dei vari DomoDevice della rete domotica, sono poi specificate all'interno di un file XML.

Per gestire l'interazione con i dispositivi o, per meglio dire, con la loro astrazione, DomoNet mette a disposizione un formato specifico per i messaggi che possono essere gestiti dal sistema. Questo standard prende il nome di DomoMessage.

I messaggi che DomoBrain deve monitorare sono per l'appunto espressi in questo formato e per la loro manipolazione sono state utilizzate funzioni di libreria messe a disposizione dal *framework* di DomoNet.

Sia i DomoDevice che i DomoMessage sono espressi nello stesso linguaggio: DomoML.

Nei prossimi due paragrafi verranno introdotti questi formati.

5.2.2 DomoML[7]

DomoML è un un linguaggio, basato su XML, che ha lo scopo di rappresentare in maniera compatta i dispositivi domotici, i servizi offerti e le relative interazioni, astraendo dalla tecnologia di appartenenza. DomoML, è quindi un *middle-language* da e verso il quale tradurre le rappresentazioni dei dispositivi e dei pacchetti. Tutta la parte logica dell'architettura di DomoNet usa domoML.

7 Rif: (Russo, 2006)

Solamente le interazioni fisiche con i dispositivi necessarie alla costruzione dei corrispondenti DomoDevice e per l'esecuzione dei servizi, all'interno dei moduli (tech manager), usano il linguaggio specifico proprio di ogni tecnologia.

5.2.3 DomoDevice[8]

Un DomoDevice ha lo scopo di creare un meccanismo semplice, compatto ed efficace per rappresentare i dispositivi astraendo dalla tecnologie.

Un DomoDevice può essere rappresentato attraverso un albero, largo e poco profondo, che si sviluppa su un massimo di quattro livelli. Vediamo di seguito una breve[9] descrizione dei tag che compongono la rappresentazione di un DomoDevice:

- `device`:

  è il tag di apertura della descrizione di uno specifico dispositivo, alcuni dei suoi attributi sono:

  - `description`: una descrizione in lingua naturale del DomoDevice;
  - `id`: identifica il DomoDevice all'interno del *web service*;
  - `URL`: identifica il *web service* che gestisce il dispositivo.

- `service:`

8 *Ibidem.*
9 Per una trattazione più approfondita si veda (Russo, 2006)

descrive un singolo servizio offerto dal DomoDevice; ha diversi campi:

- `name`: un identificatore che è utile quando il DomoMessage viene generato;
- `output`: se in seguito all'esecuzione del DomoMessage è atteso un valore di ritorno, il suo tipo sarà specificato da questo attributo.

- `input:`

  indica che il servizio ha bisogno di un valore di input per poter essere eseguito. Questo tag è opzionale e ogni servizio può avere più input. I campi per questo tag sono:

  - `name`: un identificativo dell'input;
  - `description`: una descrizione dell'input;
  - `type`: il tipo atteso.

- `allowed`:

  rappresenta un possibile valore che può assumere l'input. Questo tag ha un solo attributo:

  - `value`: il valore ammesso, in formato stringa.

Di seguito possiamo osservare un descrizione per un semplice DomoDevice rappresentante una lampadina:

```
<device description="Luce" id="18" manufacturer="philips" position=""
 positionDescription="Soggiorno" serialNumber="1.1.9" tech="KNX"
 type="Luce" url="http://www.esempio.it/services/">

 <service description=""  name="0/0/12" output="BOOLEAN"
  outputDescription="" prettyName="Stato luce"/>

 <service description="Dimming, 2 x Inputs 301901" name="0/0/12"
  prettyName="On/Off luce">
        <input description="" name="value" type="BOOLEAN">
          <allowed value="0"/>
          <allowed value="1"/>
        </input>
 </service>

</device>
```

Dall'esempio si può osservare che il dispositivo in oggetto espone due servizi. Il primo è di carattere informativo e alla sua invocazione restituisce un valore booleano che rappresenta lo stato della lampadina, 1 se è accesa, 0 se è spenta. L'altro servizio ha invece lo scopo di cambiare lo stato del *device,* in questo caso quindi serve per accendere o spegnere la lampadina fornendogli in ingresso il rispettivo input booleano.

In questa versione di DomoDevice non è però possibile distinguere tra le due tipologie di servizi in quanto, in generale, un servizio informativo potrebbe anch'esso ricevere degli input e viceversa, un servizio potrebbe cambiare lo stato di un *device* senza aspettarsi un input.

DomoBrain, per motivi che verranno chiariti in seguito, ha l'esigenza di

distinguere tra i tipi di servizio. Perciò è stato necessario estendere il tag `<service>` con l'attributo "`serviceType`" il quale può assumere uno tra i seguenti due valori:

- `query`:

  se il servizio a cui si riferisce è di tipo informativo;

- `input`:

  se invece è un servizio che cambia lo stato del dispositivo.

Perciò in seguito a tali modifiche, il DomoDevice dell'esempio precedente diventa:

```
<device description="Luce" id="18" manufacturer="philips" position=""
 positionDescription="Soggiorno" serialNumber="1.1.9" tech="KNX"
 type="Luce" url="http://www.esempio.it/services/">

 <service description=""  name="0/0/12" output="BOOLEAN"
  outputDescription="" prettyName="Stato luce" serviceType="query"/>

 <service description="Dimming, 2 x Inputs 301901" name="0/0/12"
  prettyName="On/Off luce" serviceType="input">
       <input description="" name="value" type="BOOLEAN">
          <allowed value="0"/>
          <allowed value="1"/>
       </input>
 </service>

</device>
```

5.2.4 DomoMessage

Lo scopo del DomoMessage è quello di creare un meccanismo semplice, compatto ed efficace per rappresentare le interazioni tra i DomoDevice astraendo

dalle tecnologie. Anche il DomoMessage può essere rappresentato attraverso un albero largo e poco profondo, in questo caso abbiamo un massimo di due livelli. Anche in questo caso vediamo ora una breve[10] descrizione di alcuni dei tag:

- `message`:

  è il tag di apertura della descrizione del dispositivo. Ha diversi attributi:

  - `message`:

    rappresenta il corpo del messaggio, tipicamente è uguale al valore dell'attributo `name` del servizio da invocare;

  - `messageType`:

    indica il tipo di messaggio in questione. Può assumere diversi valori, per la realizzazione di DomoBrain si sono utilizzati principalmente due tipi di messagi[11]:

    - `COMMAND`: utilizzato per invocare un servizio;
    - `UPDATE`: utilizzato per trasportare informazioni sui cambiamenti di stato del dispositivo.

- `input`:

  i valori di input da associare al *message*. Questo tag è opzionale e per

10 Per una trattazione più approfondita si veda (Russo, 2006)
11 *Ibidem.*

ogni messaggio possiamo ritrovare più *input*. Gli attributi per questo tag sono:

- `name`: il nome dell'input;
- `type`: il tipo dell'input;
- `value`: il valore in formato stringa dell'input.

Facendo riferimento al DomoDevice rappresentante la lampadina dell'esempio precedente, vediamo ora un DomoMessage per l'accensione di tale lampadina. Invocheremo quindi il servizio "0/0/12" sul DomoDevice con web service "http://www.esempio.it/services/" inviandogli un input di tipo booleano e valore 1.

```
<message message="0/0/12" messageType="COMMAND" receiverId="18"
 receiverURL="http://http://www.esempio.it/services/" >
 <input name="value" type="BOOLEAN" value="1"/>
</message>
```

6 Implementazione

In questo paragrafo verrà descritta l'implementazione di DomoBrain, delle parti che lo compongono e dell'interconnessione esistente tra queste. Di seguito, in Figura 6.1, vediamo uno schema che rappresenta i componenti, le loro connessioni e un esempio di comunicazione con DomoNet.

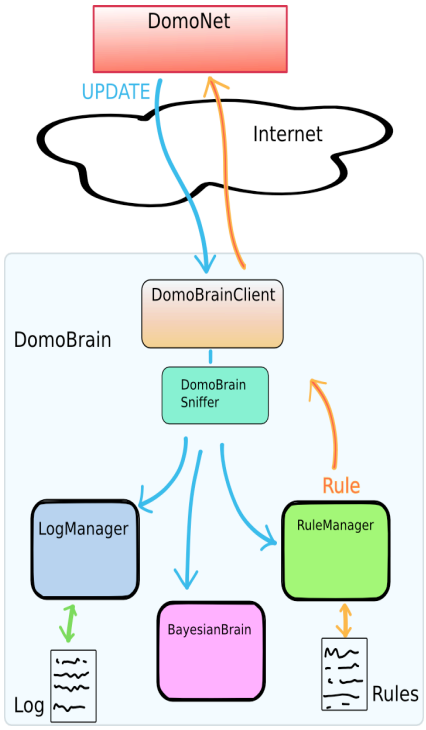

Figura 6.1: Interazione con DomoNet

6.1 DomoBrainClient

È il componente di DomoBrain che si occupa di gestire la comunicazione con il lato *server* di DomoNet.

Per organizzare lo scambio di dati con DomoNet, e in particolare per connettersi ai *web services* ed inviare dei messaggi, vengono utilizzate alcune funzionalità del *client* di DomoNet. Il lavoro svolto da tale *client* è composto di due passi distinti:

1. stabilisce un collegamento con un servizio del server attraverso l'invocazione del metodo `connectToWebServices`;
2. crea un oggetto TCPServer che fornisce un *socket* per la ricezione dei messaggi di UPDATE.

Per la realizzazione dei suoi compiti DomoBrainClient ha l'esigenza di manipolare i messaggi in arrivo sul *socket*. Per questo motivo il DomoNetClient è stato modificato in maniera tale che, la chiamata a `connectToWebServices` restituisse un puntatore all'oggetto TCPServer. In questo modo, invocando tale metodo il *client* di DomoBrain è in grado di leggere i messaggi che vengono spediti sul *socket*.

Anche per il TCPServer è stato necessario apportare alcune modifiche. Ad esso è stato infatti aggiunto un vettore di oggetti di tipo Sniffer. Per ora è sufficiente

sapere che il suo compito è quello di intercettare e processare i messaggi di UPDATE relativi ad un certo dispositivo. I dettagli riguardanti questo tipo di oggetto e le sue funzionalità verranno dati nel prossimo paragrafo.

All'interno del DomoBrainClient possiamo idealmente individuare due interfacce. Una, è rivolta verso il *server* ed è essenzialmente costituita da un oggetto DomoNetClient. L'altra è costituita dai metodi `connect` e `sendMessage`, è rivolta verso l'interno e consente agli altri componenti di DomoBrain di interagire con la prima interfaccia e, di conseguenza, con il *server*. Di seguito vediamo uno schema di tale interfaccia:

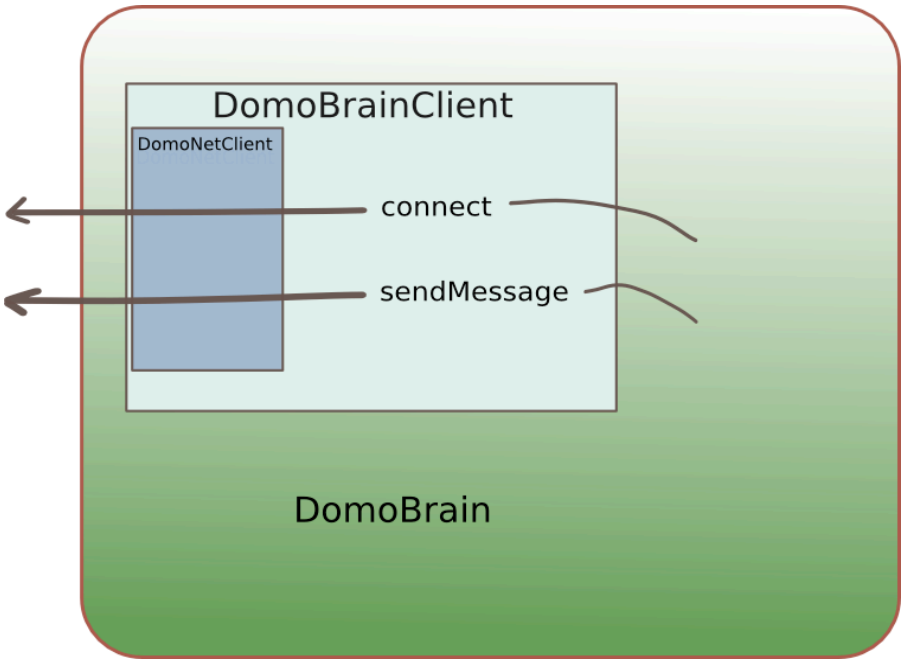

Figura 6.2: Schema di DomoBrainClient e delle sue interfacce

6.2 DomoBrainSniffer

Si occupa di processare tutti i messaggi di UPDATE relativi ad un certo

DomoDevice della rete domotica. Quindi è molto probabile che, contemporaneamente, più oggetti di questo tipo siano in ascolto sul canale di UPDATE. Ad esempio, potremmo quindi ritrovarci in una situazione in cui uno *sniffer* si occupa di restare in ascolto dei pacchetti riguardanti una lampadina, un altro sulla lavatrice, e così via per ogni altro dispositivo del quale ci interessa raccogliere i dati di utilizzo.

Le considerazioni appena fatte giustificano quanto detto nel precedente paragrafo, dove abbiamo visto come il TCPServer mantenga un vettore di oggetti DomoBrainSniffer. Infatti, all'arrivo di un nuovo messaggio di aggiornamento, il TCPServer controlla l'id del dispositivo al quale il messaggio si riferisce e di conseguenza lo inoltra allo *sniffer* opportuno.

Il lavoro di analisi dei DomoMessage in ingresso è svolto dalla procedura `parseUpdateMessage(DomoMessage update)`. Per prima cosa, all'arrivo di un messaggio di aggiornamento, viene inizializzato un timer. Allo scadere di tale contatore verranno invocati due metodi. Il primo si occuperà di scrivere una nuova linea nel file di log mentre il secondo darà inizio al calcolo bayesiano per l'evento a cui il DomoMessage di UPDATE si riferisce. Il risultato di quest'ultima operazione determinerà o meno, la creazione di una nuova regola.

Per capire la necessità di tale timer, è necessario fare alcune considerazioni sul processo di apprendimento. Abbiamo visto che un classificatore bayesiano lavora su un insieme di valori di attributi e sulle loro probabilità. Nel caso di

DomoBrain gli attributi sono rappresentati dalle funzionalità dei dispositivi e, in aggiunta a questi, dall'orario nel quale viene eseguito il comando. L'esecuzione di una operazione su di un *device* comporta tipicamente la modifica dello stato una proprietà del dispositivo. Ma, per il processo di apprendimento, è necessario avere a disposizioni il valore di ognuno di questi attributi. La durata del timer è l'intervallo di tempo nel quale ci si aspetta di osservare la ricezione dei messaggi di UPDATE di tutti i servizi di un dispositivo. Supponiamo, per esempio, di avere a che fare con una lampadina avente due funzioni, la prima che gestisce l'accensione e lo spegnimento e la seconda che ne permette il *dimming*. Se alle ore 21:00:00 l'utente accende questa lampadina il sistema emetterà due messaggi di UPDATE; uno relativo al cambiamento di stato da OFF a ON e l'altro relativo al valore del *dimmer*. Se lo *sniffer* eseguisse il suo lavoro senza il timer, ma direttamente alla ricezione del primo UPDATE, allora si perderebbero le informazioni relative al secondo messaggio di aggiornamento.

Ma il timer ha anche un altra importante funzione. Esso serve infatti a modellare il comportamento di un utente che esegue un'operazione sbagliata. Restando sempre sul semplice esempio della lampadina, immaginiamo di avere nella stessa parete più interruttori uno di fianco all'altro. In questo contesto sarebbe facile per chiunque premerne uno invece di un altro. Immaginiamo ora che venga premuto il pulsante errato. È logico pensare che, in seguito a tale errore, l'utente premerà ancora una volta questo interruttore e, a seguire, quello

corretto. In un caso come questo il sistema genera due messaggi di UPDATE. Il primo contiene l'azione sbagliata mentre il secondo contiene l'informazione corretta, rispettivamente possiamo immaginare che i due messaggi contengano “luce ON” (errore) e “luce OFF” (correzione dell'errore). Quindi, il sistema vedrà arrivare per lo stesso servizio dello stesso dispositivo due o più messaggi, aventi contenuto diverso. Di questi, solo l'ultimo in ordine cronologico viene mantenuto, mentre tutti gli altri sono scartati. Nel nostro esempio, il messaggio di “luce ON” verrà quindi gettato via mentre quello di “luce OFF” verrà conservato. Risulta quindi evidente la funzionalità del timer, senza il quale l'utente non avrebbe il tempo materiale di tornare sui suoi passi.

Abbiamo detto che DomoBrainSniffer, allo scattare del timer “scrive” una nuova *entry* nel file di log, e poi “calcola” la probabilità bayesiana per l'evento contenuto nel messaggio ed, eventualmente, “costruisce” una nuova regola per tale evento. In realtà queste operazioni non sono svolte direttamente dallo *sniffer* ma egli delega tali compiti ad altrettanti oggetti distinti:

- il LogManager: che si occupa della gestione del file di log;
- il BayesianBrain: che calcola le probabilità a posteriori;
- il RuleManager: che gestisce il file delle regole.

Nella seguente immagine, possiamo osservare uno schema che mostra il

DomoBrainSniffer, le sue connessioni con il DomoBrainClient e con gli oggetti sopraelencati. Possiamo inoltre osservare il tipico cammino percorso da un DomoMessage di UPDATE.

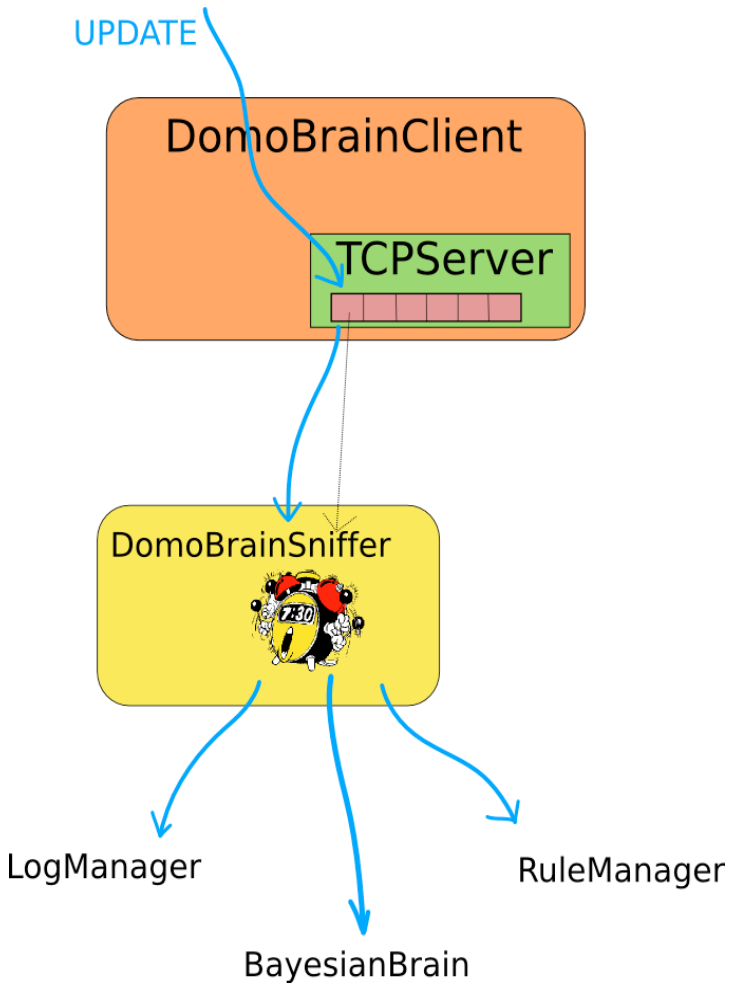

Figura 6.3: interazione tra componenti di DomoBrain

6.3 LogManager

Ogni DomoBrainSniffer contiene un LogManager che si occupa di gestire il file di log per il dispositivo di cui lo *sniffer* processa i messaggi.

Questo componente, di per se, non risulta particolarmente interessante ai fini di questa trattazione. Infatti il suo compito è sostanzialmente quello di creare un file di log e di fornire una serie di metodi che permettono le operazioni di *input/output* su di esso.

Di maggiore interesse è il file di log e per esso riserveremo una trattazione separata.

6.3.1 File di log

In questa implementazione il log è realizzato attraverso un file XML che può essere rappresentato come un albero largo e poco profondo. Il compito di questo file è conservare al suo interno le informazioni sull'utilizzo dei dispositivi del sistema domotico.

Nel log è possibile individuare due parti distinte. Nella prima sono memorizzate delle meta-informazioni sul file medesimo e sul tipo di dati in esso contenuti. A seguire, troviamo la parte con i dati riguardanti l'utilizzo di un DomoDevice. Nel caso di DomoBrain le informazioni contenute nel log sono composte da tag di tipo `“line”` all'interno delle quali troviamo un imprecisato numero di tag `“variable”` associate ai rispettivi valori. Queste rappresentano

gli attributi che verranno utilizzati dal classificatore bayesiano durante il processo di apprendimento.

Alcune di queste variabili possono essere considerate di *input*. Diciamo che una variabile è di *input* se ad essa corrisponde un servizio del dispositivo che è possibile invocare con un DomoMessage di tipo COMMAND. Supponiamo, al solito, di avere a che fare con un DomoDevice rappresentante una semplice lampadina senza dimmer e di inviare il comando di accensione alle ore 20:30:00. Allora in una linea del log troveremo due variabili. La prima che indica l'ora alla quale è stato eseguito l'accensione e l'altra contenente il valore inviato dal comando che, in questo, caso sarà "ON". Quindi, quest'ultima variabile sarà considerata di input, mentre non possiamo fare altrettanto per la variabile indicante l'orario.

Vediamo ora in dettaglio i tag che compongono il documento:

- `log`:

  è il tag che apre la struttura XML, ha un unico attributo:

  - `device_id`:

    è l'identificativo del dispositivo a cui si riferisce il file.

Al suo interno troviamo due tag:

- `log_meta_info`:

  rappresenta le meta informazioni sul file di log. Non ha attributi e

racchiude un numero variabile di tag di tipo:

- `variable_meta_info`:

  questi rappresentano le meta-informazioni sulle variabili memorizzate nelle linee del file di log. Hanno un solo attributo:

  - `is_input`: è un dato booleano che indica se la variabile a cui fanno riferimento le meta-informazioni è di *input*.

Conterrà poi i seguenti tag non facoltativi:

- `id:` è un identificativo della variabile;
- `variable_name`: è una stringa rappresentante il nome della variabile;
- `input_values`: se la variabile attuale è di *input*, allora `input_values` rappresenta il numero di valori ammessi per essa, altrimenti conterrà il valore "-1". Ad esempio, per la lampadina dell'esempio precedente sono ammessi due valori: "ON" e "OFF".
- `variable_type`: indica il tipo degli `input_values`;
- `service_name`: per le variabili di *input*, contiene il nome del servizio da invocare sul server mentre, per le altre variabili conterrà la stringa "`no-service`".

- `line`:

  è il tag di apertura di una nuova linea di log. Non ha attributi e contiene al suo interno una lista di tag di tipo:

  - `variable`:

    rappresenta una variabile e contiene due attributi:

    - `id`: è l'identificativo della variabile. Viene utilizzato per accedere alle meta informazioni;
    - `value`: è il valore della variabile.

Di seguito vediamo un esempio di come potrebbe essere strutturato il file di log che debba contenere i dati di utilizzo di un DomoDevice rappresentante una lampadina con dimmer. Si può osservare come il log riguardi i dati di utilizzo del *device* il cui `device_id` è uguale a “18”. Inoltre le informazioni raccolte per tale dispositivo verranno organizzate su tre variabili. La prima si chiama “Stato Luce” e sarà un *input* che potrà assumere due valori di tipo booleano e che corrisponderà all'invocazione del servizio “0/0/12”. La seconda prende il nome di “Stato Dimmer” e anch'essa una variabile di *input* e potrà assumere valori interi compresi nell'intervallo 0-255, questa volta il servizio invocato è il “0/0/14”. L'ultima variabile del dispositivo rappresenta l'ora in cui è stato richiesto un servizio della lampadina e, come è facile intuire, non si tratta di una variabile di input.

Possiamo inoltre osservare una linea di log che riporta lo stato della lampadina alle ore 18:07:58, vediamo quindi che la luce era accesa e il *dimmer* era impostato a 150.

```
<log device_id=”18”>
    <log_meta_info>
        <variable_meta_info is_input="true">
            <id>0</id>
            <variable_name>Stato luce</variable_name>
            <input_values>2</input_values>
            <variable_type>BOOLEAN</variable_type>
            <service_name>0/0/12</service_name>
        </variable_meta_info>
        <variable_meta_info is_input="true">
            <id>1</id>
            <variable_name>Stato dimmer</variable_name>
            <input_values>255</input_values>
            <variable_type>ONEBYTE</variable_type>
            <service_name>0/0/14</service_name>
        </variable_meta_info>
        <variable_meta_info is_input="false">
            <id>2</id>
            <variable_name>time</variable_name>
            <input_values>-1</input_values>
            <variable_type>time</variable_type>
            <service_name>no-service</service_name>
        </variable_meta_info>
```

```
    </log_meta_info>
    <line>
        <variable id="2" variable_value="18:07:58"/>
        <variable id="1" variable_value="150"/>
        <variable id="0" variable_value="1"/>
    </line>
<log>
```

6.4 BayesianBrain

Costituisce il vero e proprio "cervello" di tutto il sistema, è questo componente infatti che implementa il classificatore bayesiano.

6.4.1 Instanziazione del classificatore

Abbiamo detto che l'utilizzo di un classificatore bayesiano naive, ben si adatta a problemi in cui le istanze da classificare sono formate da tuple di valori. Inoltre, assumendo che la funzione obiettivo potesse prendere valori solo da un insieme finito V, abbiamo visto che la classificazione naive è la seguente:

$$v_{NB} = \underset{v_j \in V}{argmax} \; P(v_j) \prod_i P(a_i|v_j)$$

Nel caso di DomoBrain le tuple vengono estratte dal file di log e si tratta quindi dei valori delle variabili memorizzate su tale documento. Nel paragrafo precedente, descrivendo il formato del file di log, abbiamo osservato che le variabili in questione rappresentano lo stato dei servizi del dispositivo o

dell'ambiente. Abbiamo inoltre detto che, tra le variabili, distinguiamo quelle di *input* che sono quelle su cui possiamo eseguire la procedura di apprendimento.

È importante osservare che noi siamo interessati a calcolare, di volta in volta, la probabilità di uno di questi attributi in relazione a tutti gli altri. Un'importante conseguenza di quanto appena detto è che, nel nostro caso, l'insieme V e quello degli attributi coincidono.

Per capire meglio questo concetto riprendiamo l'esempio del DomoDevice rappresentante una lampadina con *dimmer*. Supponiamo che un giorno l'utente accenda la luce alle ore 21:30:00. Sul file di log verrà salvata una riga di questo tipo:

```
<line>
  <variable id="2" variable_value="21:30:00"/> <!-- ora -->
  <variable id="1" variable_value="255"/> <!-- dimmer -->
  <variable id="0" variable_value="1"/> <!-- luce -->
</line>
```

La questione da capire è se questa azione rappresenti un comportamento abituale oppure no. Quindi andiamo a calcolare la probabilità a posteriori che la luce sia accesa dati l'ora e il valore del *dimmer*.

Supponiamo invece che alla stessa ora la lampadina sia già accesa e che

l'utente decida di abbassare la luminosità dell'ambiente impostando il *dimmer* al valore 150. In questo caso avremmo una linea di log simile alla precedente in cui l'unica differenza è data dall'attributo `variable_value` della variabile di `id` 1. Questa volta però l'apprendimento riguarderà il comando eseguito sul *dimmer*. Perciò la probabilità che andrà calcolata sarà quella di trovare il *dimmer* impostato al valore 150 dati l'ora e lo stato della luce.

Perciò, se con a_j indichiamo il generico valore di un attributo, cioè il valore di una delle variabili di input, allora la probabilità da calcolare diventa:

$$P(a_j)\prod_{i\neq j} P(a_i|a_j)$$

6.4.2 Counter

Per stimare le probabilità $P(a_j)$ e $P(a_i|a_j)$ occorre contare la frequenza degli attributi e delle relazioni tra di essi all'interno dell'insieme di allenamento. Nel nostro caso si tratta quindi di contare le occorrenze delle variabili memorizzate all'interno del file di log.

Ricordiamo che i vari $<a_1, a_2, ..., a_n>$ sono valori di variabili distinte. Quindi i generici a_i apparterranno probabilmente ad insieme diversi. Nell'esempio della lampadina quindi avremmo tuple di tre attributi $<a_1, a_2, a_3>$ che corrisponderanno a triplette del tipo <stato luce, stato dimmer, orario>.

Per il processo di apprendimento il BayesianBrain ha bisogno di stimare le probabilità $P(a_j)$ e $P(a_i|a_j)$. Tali valori sono calcolati creando all'occorrenza oggetti di tipo Counter.

Un Counter viene inizializzato passando al suo costruttore un LogManager e un identificatore di variabile. Utilizzando il LogManager il contatore analizza il file di log e crea una struttura ad albero, i cui nodi sono composti da tabelle hash. La radice di tale albero è data da una tabella relativa alla variabile di id uguale all'identificativo passato per parametro al costruttore.

L'albero in questione ha una profondità che è esattamente uguale a 2, ed è strutturato come segue:

- Al livello 0 troviamo ovviamente la radice.
- È una *Hashtable* che, come abbiamo visto, fa riferimento ad una certa variabile. Le chiavi di questa tabella sono i possibili valori della variabile e i valori della tabella sono dei puntatori ad altre tabelle le quali andranno a formare il secondo livello.
- Al secondo livello abbiamo delle *Hashtable* le cui chiavi sono composte dagli identificatori di variabili. Tra questi non risulta ovviamente l'id della variabile a cui si riferisce la radice dell'albero. I valori di queste tabelle sono puntatori ad altre *Hashtable*, quelle che andranno a formare il livello 3.

- L'ultimo livello è composto da tabelle le cui chiavi sono i valori assunti dalle variabili del livello 2 e i valori sono degli interi che contano le occorrenze tra le chiavi e il valore della variabile della radice.

Quindi, le foglie di questo albero, sono delle tabelle *hash* che contano le occorrenze tra i valori della variabile in radice e i valori di tutte le altre variabili.

Vediamo ora come l'oggetto Counter viene utilizzato per stimare le probabilità d'interesse.

Per il calcolo di $P(a_j)$ viene invocato il metodo `getVariableValueCount(String inputValue)`. Utilizzando il LogManager, questa procedura conta il numero di volte che la variabile a cui il Counter si riferisce occorre, nel file di log, con valore uguale a `inputValue`.

Il calcolo di $P(a_i|a_j)$ giustifica la presenza della struttura ad albero vista precedentemente. Esaminiamo la chiamata al metodo che restituisce le occorrenze cercate. Tale procedura prende il nome di `getOccurence` e prende in input tre parametri:

- `inputValue`:

  è una stringa che rappresenta il valore assunto dalla variabile rappresentata dalla radice dell'albero. Esso verrà utilizzato come

chiave per ottenere dalla tabella di livello 0 un puntatore ad una tabella del livello 1.

- `variableId`:

  è un intero che rappresenta l'identificatore della variabile di cui vogliamo conoscere la relazione tra essa e la variabile in radice. Con questo intero si accede alla tabella del livello 1 per ottenere un puntatore ad una delle *Hashtable* residenti nelle foglie.

- `VariableValue` :

  è una stringa che rappresenta il valore della variabile di id uguale a `variableId.` Questa stringa verrà usata per accedere ai valori dell'ultima tabella che saranno proprio le occorrenze cercate.

Per comprendere a fondo questo metodo, riprendiamo l'esempio della lampadina. Supponiamo di essere interessati al conoscere il numero di volte il cui la luce risulta accesa alle ore 18:00:00. Assumiamo inoltre che la variabile di id 0 rappresenti lo stato della luce e quella di id 1 l'orario. Creiamo quindi un nuovo Counter per la variabile di id uguale ad 1. A questo punto abbiamo a disposizione l'albero di tabelle hash. Per conoscere la quantità cercata invochiamo quindi il metodo getOccurrences nel seguente modo:

```
getOccurences(“on”, 1, “18:00:00);
```

In Figura 7.4.1 vediamo uno schema che dovrebbe chiarire sia la struttura ad albero e sia il meccanismo con il quale si accede al valore cercato.

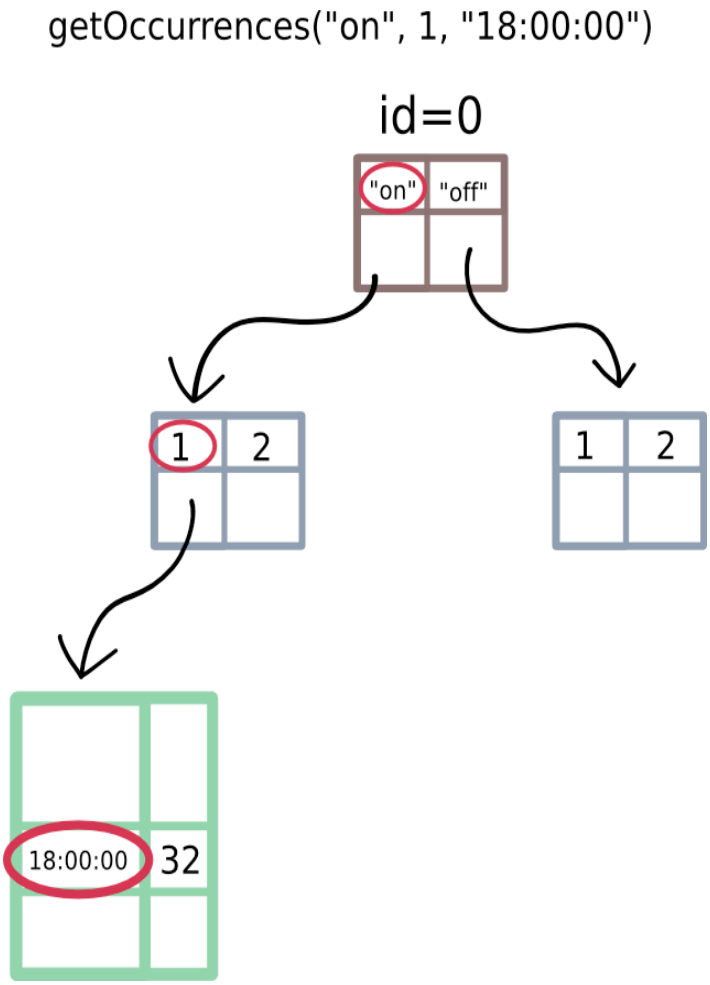

Figura 7.4.1: esempio della struttura ad albero di tabelle hash e di chiamata del metodo getOccurences

6.5 DeviceManager

Abbiamo visto che i DomoDevice offrono due tipi di servizi, di *query* e di *input*, e che per distinguere tra di essi è stato necessario estendere il formalismo utilizzato con un nuovo attributo che indicasse il tipo di servizio.

Allargare la grammatica utilizzata con i nuovi attributi avrebbe significato dover modificare sia il lato *client* che il lato *server* di DomoNet. Per il *client* non ci sarebbero stati problemi dato che il codice a disposizione è stato ampiamente studiato ed utilizzato anche per la stessa realizzazione di DomoBrainClient. Diversamente, abbiamo visto che l'approccio al DomoNetServer è stato di tipo *blackbox* e quindi non era possibile accedere al codice e apportare le modifiche necessarie.

Per ovviare a questo problema è stata creata un'interfaccia tra la "vecchia" e la "nuova" versione della grammatica. Questo punto di contatto è rappresentato da dal DeviceManager. Questo componente si occupa di gestire l'accesso ad un nuovo file contenente le descrizioni dei *device*.

Il file in questione è sostanzialmente una copia del file XML originale, al quale è stato aggiunto, per ogni servizio di ogni dispositivo, il nuovo attributo `serviceType`.

Ogni altro componente del sistema, che abbia la necessità di conoscere il tipo di uno dei servizi offerti da uno specifico DomoDevice, può quindi fare una richiesta al DeviceManager fornendogli in ingresso l'identificativo del dispositivo e il nome del servizio del quale si vuole conoscere il tipo.

6.6 RuleManager

È il componente che si occupa della gestione del file delle regole apprese. Nei

capitoli precedenti abbiamo visto che, ogni volta che l'utente esegue una certa azione, per essa viene innescato il meccanismo di apprendimento il quale ha lo scopo di calcolare la probabilità a posteriori per l'azione svolta. Se tale probabilità supera una certa soglia, allora dall'evento in questione verrà creata una nuova regola.

Abbiamo inoltre visto che una regola corrisponde ad un comando che deve essere inviato ad un dispositivo ad una certa ora.

Fino a questo momento abbiamo utilizzato il termine “regola” in maniera generica. In realtà è possibile individuare due insiemi di regole:

- *pending*:

  subito dopo la loro creazione, le regole non vengono subito schedulate per l'esecuzione, ma restano in attesa del giudizio dell'utente. Egli potrà in qualunque momento decidere se la regola costruita è corretta e pronta per lo *scheduling* oppure se debba essere eliminata dal sistema.

- *confirmed*:

  contiene tutte le regole apprese la cui esecuzione è stata confermata dall'utente. Poiché si assume che, nel tempo, una certa regola possa perdere la sua validità (ad esempio, nel caso in cui l'utente debba partire per periodi di vacanza o al variare delle stagioni), per l'utente

sarà sempre possibile cancellarla o rimetterle nell'insieme di *pending*.

Il RuleManager fornisce sostanzialmente un'interfaccia per interagire con il file delle regole. Esso mette infatti a disposizione una serie di metodi che permettono l'esecuzione di operazioni di lettura/scrittura su tale file. Per questo motivo non verranno descritti i dettagli della sua implementazione.

Più interessate è la descrizione del file di delle regole e ad esso verrà dedicato il prossimo paragrafo.

6.6.1 File di regole

Si tratta di un file XML che memorizza al suo interno le regole generate durante il processo di apprendimento. Abbiamo già visto che una regola è una coppia composta da un messaggio e da un orario di invio. Il messaggio altro non è che un DomoMessage il cui attributo `messageType` ha valore `COMMAND`.

Al file è possibile associare una struttura ad albero avente al massimo quattro livelli. Vediamo in dettaglio i tag che compongono questo file:

- `rules`:

  è il tag che apre il file, non ha attributi e al suo interno contiene una lista di elementi di tipo:

- `rule`:

  rappresenta la regola vera e propria. Ha due attributi:

  - `time`: indica l'ora alla quale deve essere eseguito il comando;
  - type: specifica il tipo della regola, può quindi assumere due soli valori:
    - `pending`;
    - `confirmed`;

  Inoltre, all'interno di ogni tag `rule` troviamo un tag `message` che rappresenta il DomoMessage da inviare.

Di seguito diamo un esempio di un semplice file di log[12].

12 Per i dettagli sul formato dei DomoMessage vedere il Capitolo 6 e (Russo, 2006)

```
<rules>
    <rule time="17:30:00" type="confirmed">
        <message
            senderURL="http://http://www.esempio.it/service
s/"
            senderId="18"
            receiverURL="http://www.esempio.it/services/"
            receiverId="18"
            messageType="COMMAND"
            message="0/0/12">
             <input value="1" type="BOOLEAN" name="value"/>
        </message>
    </rule>

    <rule time="17:38:00" type="pending">
        <message
           senderURL="http://www.esempio.it/services/"
           senderId="18"
           receiverURL="http://www.esempio.it/services/"
           receiverId="18"
           messageType="COMMAND"
           message="0/0/12">
            <input value="0" type="BOOLEAN" name="value"/>
```

```
        </message>
    </rule>

</rules>
```

In questo semplice esempio possiamo osservare due regole. La prima è di tipo `confirmed`, quindi essa appartiene all'insieme di regole schedulate per l'esecuzione periodica. Dall'attributo `time` vediamo che ogni giorno, alle ore 17:30:00, verrà inviato al servizio di indirizzo *http://www.esempio.it/services/* il DomoMessage contenuto all'interno del tag `message` che, in questo caso, comanda l'accensione di una lampadina. La seconda regola è in attesa di essere confermata dall'utente e quindi la sua esecuzione non è programmata. Se dovesse venire confermata invierebbe alle ore 17:38:00 messaggio per comandare lo spegnimento della luce.

7 Conclusioni

Uno dei problemi della domotica è legato alle difficoltà che è talvolta possibile incontrare nella configurazione dei dispositivi di un sistema domotico.

Pensiamo ad esempio ad una persona anziana che non ha dimestichezza con dispositivi elettronici e che quindi non ha la capacità tecnica di configurare l'impianto domotico in relazione alle sue necessità. Oppure vi sono situazioni in cui l'utente non riesce a prevedere quelli che saranno i suoi bisogni e non sa quindi in che modo di modificare le impostazioni dei dispositivi domotici.

DomoBrain cerca di risolvere questi problemi proponendo una soluzione in grado di liberare l'utente dalla necessità di impostare manualmente i *device* dell'impianto e, attraverso l'apprendimento automatico delle sue abitudini, è in grado di suggerirgli nuove configurazioni per il sistema.

Vi sono comunque diverse strade lasciate aperte ad ulteriori sviluppi futuri. Ad esempio le informazioni raccolte potrebbero essere utilizzate per la creazione di particolari scenari domotici, dove con il termine “scenario” si intende una combinazione di stati e regolazioni dei diversi impianti della casa. In DomoBrain, sia le informazioni raccolte nei file di log e sia le regole apprese possono essere utilizzate per la creazione di nuovi scenari. Ad esempio, analizzando le regole, si potrebbero raccogliere tutte quelle che corrispondono a

comandi che vengono inviati a diversi dispositivi ad una certa ora del giorno. Così facendo potremmo creare un nuovo tipo di regole riguardanti l'invio di combinazioni di comandi. Oppure si potrebbe studiare la probabilità che alcuni servizi vengano invocati in successione ad altri e creare di conseguenza delle regole corrispondenti a una sequenza ben precisa di comandi.

Altre modifiche potrebbero riguardare la struttura del programma. Ad esempio si potrebbe decidere di spostare direttamente sul *server* la logica di DomoBrain che, in questo modo, non sarebbe più un *client* ma diventerebbe un componente di DomoNetServer.

Di particolare interesse sarebbe sviluppare altre implementazioni per il sistema di apprendimento automatico. In questo modo si sarebbe in grado di affrontare lo stesso problema sfruttando tecnologie diverse. Ad esempio si potrebbe realizzare una rete neurale che lavori in parallelo al classificatore bayesiano al fine di poter eseguire un confronto tra i due sistemi. Questo potrebbe servire a capire se vi siano situazioni in cui un sistema risulta più efficace dell'altro e, dall'analisi di tale risultato, si potrebbero studiare le motivazioni che stanno dietro alla differenza di *performance.*

Attualmente DomoBrain è stato stato progettato per essere eseguito su di un normale *computer,* ma poiché in domotica si ha spesso a che fare con i cosiddetti dispositivi *embedded*, si potrebbe effettuare il *porting* di DomoBrain in alcuni di questi *device* e osservare come esso si comporti avendo a disposizione risorse

hardware differenti da quelle di un calcolatore.

Appendice 1: XML Schema

DomoDevice.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>


<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="allowed">
  <xs:complexType>
   <xs:attribute name="value" type="xs:NMTOKEN"
use="required" />
  </xs:complexType>
 </xs:element>


 <xs:element name="device">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="service" maxOccurs="unbounded" />
   </xs:sequence>
   <xs:attribute name="url" type="xs:string"
use="required" />
   <xs:attribute name="id" type="xs:NMTOKEN"
use="required" />
```

```
   <xs:attribute name="serialNumber" type="xs:NMTOKEN"
use="required" />
   <xs:attribute name="position" type="xs:string"
use="required" />
   <xs:attribute name="type" type="xs:NMTOKEN"
use="required" />
   <xs:attribute name="positionDescription"
type="xs:NMTOKEN" use="required"/>
   <xs:attribute name="tech" type="xs:NMTOKEN"
use="required" />
   <xs:attribute name="manufacturer" type="xs:string"
use="required" />
   <xs:attribute name="description" type="xs:string"
use="required"/>
  </xs:complexType>
 </xs:element>

 <xs:element name="devices">
  <xs:complexType>
   <xs:sequence>
```

```
    <xs:element ref="device" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="input">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="allowed" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="xs:NMTOKEN"
use="required" />
    <xs:attribute name="type" type="xs:NMTOKEN"
use="required" />
    <xs:attribute name="description" type="xs:string"
use="required" />
  </xs:complexType>
 </xs:element>


 <xs:simpleType name="serviceType">
  <xs:restriction base="xs:string">
```

```
    <xs:enumeration value="query" />
    <xs:enumeration value="input" />
   </xs:restriction>
  </xs:simpleType>

  <xs:element name="service">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="input" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="output" type="xs:NMTOKEN"
use="optional" />
    <xs:attribute name="prettyName" type="xs:string"
use="required" />
    <xs:attribute name="name" type="xs:string"
use="required" />
    <xs:attribute name="description" type="xs:string"
use="required" />
    <xs:attribute name="outputDescription" type="xs:string"
use="optional" />
```

```
   <xs:attribute name="serviceType" type="serviceType"
use="required" />
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Log.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:simpleType name="variable_value_type">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="int"/>
   <xsd:pattern value="bool"/>
   <xsd:pattern value="time"/>
```

```
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:element name="variable_meta_info">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="id" type="xsd:int" />
    <xsd:element name="variable_name" type="xsd:string" />
    <xsd:element name="input_values" type="xsd:int" />
    <xsd:element name="variable_type"
type="variable_value_type" />
    <xsd:element name="service_name" type="xsd:string" />
   </xsd:sequence>
   <xsd:attribute name="is_input" type="xsd:boolean" />
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="log_meta_info">
  <xsd:complexType>
```

```
<xsd:sequence>
```

```
    <xsd:element ref="variable_meta_info"
maxOccurs="unbounded"></xsd:element>
   </xsd:sequence>
   <xsd:attribute name="num_inputs"  type="xsd:int" />
   <xsd:attribute name="num_variables" type="xsd:int" />
   <xsd:attribute name="device_name" type="xsd:string" />
  </xsd:complexType>
```

```
 </xsd:element>



 <xsd:element name="variable" >
  <xsd:complexType>
   <xsd:attribute name="variable_value" type="xsd:string" /
>
    <xsd:attribute name="id" type="xsd:int" />
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="line">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="variable"  maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="id" type="xsd:int"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="log">
  <xsd:complexType >
```

```
   <xsd:sequence>
    <xsd:element ref="log_meta_info" maxOccurs="1"
minOccurs="1" />
    <xsd:element ref="line"  maxOccurs="unbounded"/>
   </xsd:sequence>
   </xsd:complexType>
 </xsd:element>

</xsd:schema>
```

Rules.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="input">
  <xsd:complexType>
   <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="value" type="xsd:string" />
```

```
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="message">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="input" maxOccurs="unbounded" />
   </xsd:sequence>

   <xsd:attribute name="message" type="xsd:string" />
   <xsd:attribute name="messageType" type="xsd:string" />
   <xsd:attribute name="output" type="xsd:string" />
   <xsd:attribute name="receiverId" type="xsd:int" />
   <xsd:attribute name="receiverURL" type="xsd:string" />
   <xsd:attribute name="senderId" type="xsd:int" />
   <xsd:attribute name="senderURL" type="xsd:string" />
   <xsd:attribute name="value" type="xsd:string" />
  </xsd:complexType>
 </xsd:element>


 <xsd:simpleType name="ruleType">
```

```
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="pending" />
   <xsd:enumeration value="confirmed" />
  </xsd:restriction>
 </xsd:simpleType>

 <xsd:element name="rule">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="message"  maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="type" type="ruleType"/>
   <xsd:attribute name="time" type="xsd:string"/>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="rules">
  <xsd:complexType >
   <xsd:sequence>
    <xsd:element ref="rule"  maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
```

```
 </xsd:element>


</xsd:schema>
```

Appendice 2: Esempio di previsione bayesiana

Vediamo ora un esempio di previsione bayesiana su di un file che raccoglie i dati di utilizzo di una lampadina con *dimmer*.

Supponiamo che ci siano due servizi associati al dispositivo:

- *statoLuce* : che è un valore booleano e rappresenta lo stato della luce;
- *statoDimmer* : che rappresenta l'impostazione del *dimmer*. È di tipo ONEBYTE[13] e può quindi assumere un valore compreso nell'intervallo [0-255].

Ad ogni operazione sulla lampadina verrà quindi salvata nel file di log una nuova linea con tre variabili, una per ogni servizio più una variabile `"ora"` che rappresenta l'orario nel quale è stato invocato uno dei servizi della lampadina.

Supponiamo che sul file via siano un totale di trenta linee di log, e che esse siano suddivise in questo modo:

- 15 righe con:
  - `luce = 1;`
  - `dimmer = 123ora = 05:00:00`
- 3 righe con:
  - `luce = 1`

13 Per i tipi di dato degli input dei dispositivi vedere (Russo, 2006)

- dimmer = 123
- ora = 18:00:00

- 7 righe con:
  - luce = 1
  - dimmer = 222
  - ora = 18:00:00
- 5 righe con:
  - luce = 0
  - dimmer = 222
  - ora = 18:00:00

Assumiamo, in seguito all'accensione della lampadina, di osservare il seguente stato del dispositivo:

```
< luce=1; dimmer=123; ora=18:00:00>
```

Poiché l'azione che ha dato origine a questa osservazione è stata l'accensione della luce da parte dell'utente, allora ci interessa effettuare una previsione proprio sulla probabilità che ha la luce di essere accesa dati il valore del dimmer e dell'ora, cioè:

```
P(luce = 1 | dimmer=123, ora=18:00:00) =
  P(dimmer=123 | luce=1) *
  P(ora=18:00:00 | luce = 1) *
```

```
P(luce = 1)
```

Per stimare le probabilità della formula, bisogna contare le occorrenze cercate sul file di log. Per comodità rappresentiamo il tutto con una tabella:

StatoDimmer			**Ora**			**Totale statoLuce**	
	luce			*luce*		*luce*	
	0	*1*		*0*	*1*	*0*	*1*
123	0/5	18/25	*5:00:00*	0/5	15/25	25/30	5/30
222	5/5	7/25	*18:00:00*	5/5	10/25		

Tabella 2: dati presenti sul file di log

Decidiamo che la soglia di probabilità oltre la quale creiamo una nuova regola a partire da un certo evento sia di 0,20. Calcoliamo ora la probabilità a posteriori cercata e vediamo se supera tale valore. Sfruttando la tabella è facile vedere che:

```
P(dimmer=123 | luce=1) = 18/25

P(ora=18:00:00 | luce=1) = 10/25

P(luce=1) = 25/30
```

Perciò:

```
P(luce = 1 | dimmer=123, ora=18:00:00) =

 18/25 * 10/25 * 25/30 = 0,24
```

Poiché il valore ottenuto è maggiore della soglia, allora verrà creata una nuova regola che conterrà il comando necessario ad accendere automaticamente la luce alle ore 18:00:00.

Bibliografia

Dragoni, G	2004	*Il problema degli standard nella domotica, Conferenza Spazi tecnologici per l'uomo – Le sfide della domotica,* Fondazione Politecnico di Milano,
Alpadym E.	2004	*Introduction to Machine Learning*
Augusto Juan .C.; Nugent Chris D	2006	*Designing Smart Homes: The Role of Artificial Intelligence,* Springer
Bellintani S.	2004	*Manuale della domotica. Tecnologie ed evoluzione dell'abitare,* Il Sole 24 Ore
Bianchi Bandinelli, R.	2007	*Lucidi del corso di Domotica*
Cerami E	2002	*Web Services Essential,* O'Reilly
Deitel Harvey M.; Deitel Paul J.	2006	*Java. Tecniche avanzate di programmazione,* Apogeo
Ferri M.; Perrone M.; Traversi M.	2006	*Lo Stato dell'arte della Domotica: scenari futuribili e promesse non mantenute*
Furfari, F., Sommaruga, L.,	2004	*DomoML: the definition of a standard*

Soria, C., Fresco, R.		*markup for interoperability of Human Home Interactions*
Grudic, G.	2004	*Bayesian Book*
Harold E. R.; Means W. S.	2004	*XML in a Nutshell: A Desktop Quick Reference,* O'Reilly
Kononenko I. ; Kukar M.	2007	*Machine Learning and Data Mining: Introduction to Principles and Algorithms,* Horwood Publishing
Korb Kevin B.; Nicholson Ann E.	2004	*Bayesian Artificial Intelligence,* Chapman & all/CRC
Kurose James F.; Ross Keith W.	2005	*Reti di calcolatori ed Internet: Un approccio top-down,* Pearson Education
Manca Maurizio M.	2004	*DomoNET: Un framework e un prototipo per l'interoperabilità dei middleware domotici basato su XML/Web-Services*
McLaughlin B	2002	*Java & XML 2nd Edition: Solutions to Real-World Problems,* O'Reilly
Mitchell, Tom M.	1997	*Machine Learning,* McGraw-Hill
Nilsson Nils J.	1996	*Introduction to Machine Learning*
Norvig P.; Russel S.	2003	*Artificial Intelligence: A Modern Approach,* Prentice Hall
Quaranta Giuseppe G.; Mongiovì P.	2004	*L'abc della domotica*
Russo D.	2006	*La domotica e Internet. Una soluzione per l'interoperabilià*
Shavlik Jude W.; Mooney Raymond J.; Towell Geoffrey G.	1991	*Symbolic and Neural Learning Algorithms:An Experimental Comparison*

Topley K.	2003	*Java Web Services in a nutshell,* O'Reilly
Trisciuoglio D.	2002	*Introduzione alla domotica,*Tecniche Nuove