



Sviluppo di un contatore elettrico intelligente

Stefano Antonio Labianca

22 Gennaio 2024
a.a 2023/2024

matricola: 758364

email: s.labianca10@studenti.uniba.it

*Università degli studi di Bari Aldo Moro
Caso di Studio per l'esame di Ingegneria della Conoscenza.*

<https://github.com/Stefano-Labianca/smart-energy-controller>

Tabella dei contenuti

1	Introduzione	4
1.1	Dispositivo salvavita	4
1.2	Obiettivo del progetto	4
2	Il progetto	5
2.1	Struttura del progetto	5
2.2	Scelte progettuali generali	6
3	Argomenti trattati	7
3.1	CSP	7
3.1.1	Descrizione generale	7
3.1.2	Scelte progettuali	7
3.1.3	Algoritmi utilizzati	8
3.1.4	Analisi delle performance	9
3.1.5	Conclusioni	14
3.2	Ontologie	14
3.2.1	Descrizione generale	14
3.2.2	Scelte progettuali	14
3.2.3	Protégé	15
3.2.4	Lavorare sull'ontologia	17
3.2.5	Conclusioni	18
3.3	Sistema esperto	19
3.3.1	Descrizione generale	19
3.3.2	Scelte progettuali	19
3.3.3	Realizzazione del sistema esperto	19
3.3.4	Funzionamento	20
3.3.5	Conclusioni	23
4	Conclusioni e Sviluppi Futuri	23
5	Riferimenti Bibliografici	24

Appendice	25
.1 Inizializzare il progetto	25
.1.1 Scaricare da GitHub	25
.1.2 Impostare l'ambiente virtuale	25
.1.3 Avviare il progetto	27
.1.4 Avviare un test	27
.2 Troubleshooting	27
.2.1 Risolvere il problema di esecuzione con la PowerShell .	27
.2.2 Errore di esecuzione del programma Python	29

1 Introduzione

Nell'arco della nostra giornata, usiamo diversi dispositivi elettronici e, alle volte, anche per diverse ore della giornata o addirittura per tutto il giorno.

Per chi abita nelle zone di campagna, o in abitazioni singole, usare molti dispositivi elettronici contemporaneamente, specialmente se hanno alti consumi o possiedono una classe energetica bassa, fa scattare il salvavita.

1.1 Dispositivo salvavita

Il "salvavita", o più propriamente detto interruttore differenziale, è un dispositivo che arresta il flusso di energia elettrica dal contatore di un'abitazione, proteggendo persone e animali.

Questi interruttori, monitorano la differenza di corrente in entrata e in uscita dal dispositivo e, quando la differenza di corrente in entrata e in uscita supera una certa soglia, allora l'interruttore scatta togliendo l'alimentazione al circuito.



Figura 1: Esempio di contatore differenziale

1.2 Obiettivo del progetto

Il progetto si pone l'obiettivo di sviluppare un programma in grado di svolgere i seguenti task:

1. Determinare da una lista di dispositivi, quali possono tenere accesi contemporaneamente senza che salti il salvavita;
2. Tenere traccia dei dispositivi elettronici e del loro consumo in Watt;
3. Ottenere tutti quei dispositivi che rispettano certi vincoli di consumo energetico;

2 Il progetto

2.1 Struttura del progetto

All'intero del progetto, possiamo trovare le seguenti cartelle:

- `/.venv`: Cartella contenente tutto il necessario per lavorare con l'ambiente virtuale;
- `/appliance`: Qui è possibile trovare la classe `Appliance`, che definisce un elettrodomestico, insieme ad una serie di metodi di supporto, contenuti in `appliances_controller.py`. Insieme ad essi, si trovano dei file csv usati per i test del programma;
- `/cli`: Troviamo una classe che incapsula tutta la logica legata agli input e all'output del terminale;
- `/csp_problem`: Questa cartella contiene tutti i file legati all'argomento del CSP.
Infatti è possibile trovare la rappresentazione delle variabili e dei vincoli, fatta rispettivamente usando le classi `Variable` e `Constraint`.
Inoltre è presente anche la classe `CSP` usata come wrapper per rappresentare un generico problema di questa categoria.
Infine è presente la cartella `/algorithm` che contiene le realizzazioni degli algoritmi DFS e GAC usati per risolvere i problemi legati al CSP;
- `/docs`: In questa cartella è possibile trovare la relazione, in formato PDF, del progetto;
- `/knowledge_base`: Contiene una classe usata per rappresentare il sistema esperto realizzato;

- `/ontology`: Questa cartella contiene una classe che permette di manipolare l'ontologia contenuta all'intero del file `appliance_ontology.rdf`;
- `/project_test`: Contiene tutti quei file contenente vari test fatti al programma. Nella cartella `/project_test/performance` sono contenuti i file usati per misurare le performance degli algoritmi del CSP utilizzati;
- `/utils`: Contiene file di utilità che facilitano alcune operazioni interne al programma. Il file `pagination.py` viene usato per impaginare l'output del programma, mentre il file `printer.py` contiene una serie di funzioni utili per la stampa di liste di variabili, di vincoli e per gli assegnamenti, che siano parziali o totali;

2.2 Scelte progettuali generali

Il progetto è stato svolto usando le versioni 3.11.5 e 3.12.0 di Python in quanto offrono diversi miglioramenti e un supporto maggiore alla tipizzazione delle variabili e delle costanti. Usare i tipi, infatti, si è rivelato molto utile per rendere la codebase più resiliente, evitando di assegnare, erroneamente, valori con tipo non corretto per una variabile.

L'interazione con il progetto avviene attraverso l'uso del seguente menù:

```
Premi 1 per avviare il CSP
Premi 2 per lavorare sull'ontologia
Premi 3 per avviare il sistema esperto

Per uscire premi 4

> 
```

Figura 2: Menù di avvio del programma

In questo modo, è possibile utilizzare le parti del programma principale in modo più rapido.

Il programma lavora intorno all'uso delle ontologie in quanto permettono di salvare, sottoforma di triple, le informazioni sugli apparecchi elettronici che saranno usate dal sistema esperto e dagli algoritmi di CSP.

3 Argomenti trattati

3.1 CSP

3.1.1 Descrizione generale

Un Constraint Satisfaction Problem, o CSP, si pone l'obiettivo di trovare un insieme di assegnamenti totali che rispettano dei vincoli. Ogni assegnamento totale, che rispetta i vincoli dati dal problema, viene considerato come una soluzione al problema.

Un CSP è formato dalle seguenti componenti:

- Insieme finito di variabili;
- Ogni variabile ha un dominio;
- Insieme di vincoli;

I problemi che voglio risolvere sono: trovare quali dispositivi possono accendere contemporaneamente senza superare una certa soglia di consumi oppure limitare i dispositivi da usare.

3.1.2 Scelte progettuali

Per l'argomento del CSP [2], sono state usate delle versioni leggermente modificate degli algoritmi DFS e GAC fornite dal libro di testo AIPython, in quando si è scelto di adattare al problema imposto dal progetto. Non solo, lo stesso vale per le classi `Variable` e `Constraint` dove sono stati presi in considerazione solamente le funzionalità essenziali.

Un'altra scelta è stata quella di impaginare [3] i risultati dati dagli algoritmi del CSP. Grazie ad essa, è possibile dividere grandi quantità di dati in blocchi di dimensioni più piccoli e maneggevoli, permettendo anche di scorrere una pagina alla volta.

La paginazione è possibile grazie all'uso della class `Pagination` che permette di spezzare le soluzioni date dagli algoritmi usati, in blocchi più piccoli.

L'algoritmo del Generalized Arc Consistency, per aiutarlo maggiormente a ridurre lo spazio di ricerca e a trovare le soluzioni, è stato scelto di usare la tecnica del *Domain Splitting*. In questo modo, il dominio di una variabile viene diviso in due sottodomini per poi eliminare, in entrambi i domini, quei valori che rendono inconsistente un arco. Non è stata scelta la variante in cui avviene la *Variable Elimination*, in quanto l'efficienza di questo approccio dipendono molto dall'ordine con cui le variabili vengono eliminate.

Con il *Domain Splitting*, inoltre, possiamo migliorarlo ulteriormente quando vogliamo contare il numero di soluzioni. Ogni componente disconnessa dal grafo, può essere risolta separatamente rendendo così l'algoritmo più efficiente e, il numero di soluzioni al problema completo è uguale al prodotto del numero di soluzioni trovate nelle componenti separate.

Infine sono stati scelti gli algoritmi di ricerca sistematici in quando vogliamo la garanzia di trovare delle soluzioni ottimali.

3.1.3 Algoritmi utilizzati

Sono stati presi in considerazione i seguenti algoritmi:

- DFS;
- Generalized Arc Consistency;

I due algoritmi, prevedono i seguenti approcci:

- DFS: va a rappresentare ogni assegnamento come un albero, dove la sua altezza è pari al numero di variabili che si stanno considerando e, ogni nodo, viene rappresentato come un assegnamento ad una variabile. Un assegnamento totale, quindi, viene visto come il cammino che va dal nodo radice ad un nodo foglia.
- Generalized Arc Consistency: prima di trovare delle soluzioni, crea una rete di archi consistenti. Un arco $\langle X, c \rangle$, con X una variabile e c un

vincolo che ha scope $\{X, Y_1, Y_2, \dots, Y_n\}$, si dice consistente se per ogni valore x appartenente al dominio della variabile X , esistono dei valori y_1, y_2, \dots, y_n , che appartengono ai domini delle loro corrispettive variabili Y_i , tali che, l'assegnamento $\{X = x, Y_1 = y_1, Y_2 = y_2, \dots, Y_n = y_n\}$ mi va a soddisfare il vincolo c .

Per creare archi consistenti, l'algoritmo andrà ad eliminare, per ogni dominio delle variabili, tutti quei valori che non soddisfano il vincolo c . In altri termini va a ridurre lo spazio di ricerca, prima di andare a trovare una soluzione.

3.1.4 Analisi delle performance

I due algoritmi sono in grado di trovare le soluzioni ad una serie di problemi dati. Adesso vogliamo sapere quale dei due riesce a trovare delle soluzioni in tempi minori medi.

I test sono stati svolti usando un Ryzen 5 3400G, sistema operativo Windows 10 e versione di Python 3.12.0.

Test 1

Voglio trovare degli assegnamenti in cui i dispositivi, della categoria Multimedia, non vanno a consumare più di 450W.

Le informazioni sul test sono contenute nel file `/appliance/appliance.csv` e l'esempio mostrato si trova nel file nel `/project_test/performance/test_1.py`.

monitor	laptop	tv	sound_system	phone_charger	internet_router	computer	3D_printer	printer
70	300	150	137	6	30	400	430	30
70	300	150	137	6	30	400	430	70
70	300	150	137	6	50	400	430	30
70	300	150	137	6	50	400	430	70
70	200	150	137	4	10	400	430	30
70	200	150	137	4	10	400	430	70
70	200	150	137	4	30	400	430	30
70	200	150	137	4	30	400	430	70
70	200	150	137	4	50	400	430	30
70	200	150	137	4	50	400	430	70

Figura 3: Esempio di assegnamenti trovati dal DFS

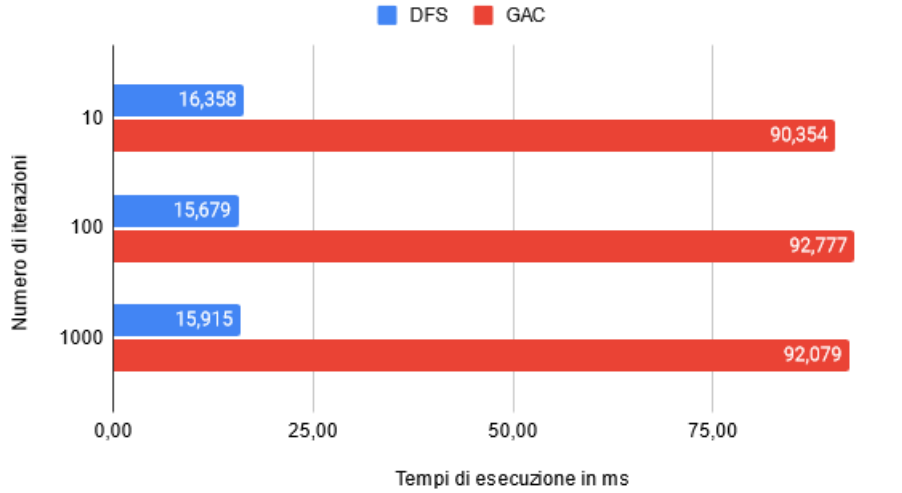
computer	phone_charger	monitor	laptop	internet_router	printer	sound_system	3D_printer	tv
400	4	70	105	10	30	137	430	150
400	4	70	105	10	70	137	430	150
400	4	70	105	30	30	137	430	150
400	4	70	105	30	70	137	430	150
400	4	70	105	50	30	137	430	150
400	4	70	105	50	70	137	430	150
400	4	70	300	10	30	137	430	150
400	4	70	300	10	70	137	430	150
400	4	70	300	30	30	137	430	150
400	4	70	300	30	70	137	430	150

Figura 4: Esempio di assegnamenti trovati dal GAC

In entrambe le figure, possiamo osservare delle tabelle dove una riga di una tabella, rappresenta un assegnamento di variabili e, le variabili presenti nell'assegnamento, sono indicate con i nomi delle colonne. Nella prima riga della tabella in figura 3, per esempio, alla variabile *computer* andiamo ad assegnare il valore 400.

Adesso diamo uno sguardo ai loro tempi di esecuzione. Per farlo vado ad eseguire entrambi gli algoritmi 10, 100 e 1000 volte, per poi salvare i tempi medi di esecuzione, tramite la funzione `perf_counter_ns()` [4].

Test 1 - Tempi medi di esecuzione in ms



Algoritmo	μ	σ^2
DFS	15.984ms	0.119ms
GAC	91.737ms	1.556ms

Table 1: Media (μ) e Varianza (σ^2) dei tempi di esecuzione per il test 1

Possiamo notare come, nel primo caso di test, l'uso della DFS impiega, in media, molto meno tempo rispetto al Generalized Arc Consistency. Inoltre, per il GAC i tempi di esecuzione medi possono variare maggiormente, rispetto ai tempi medi della DFS.

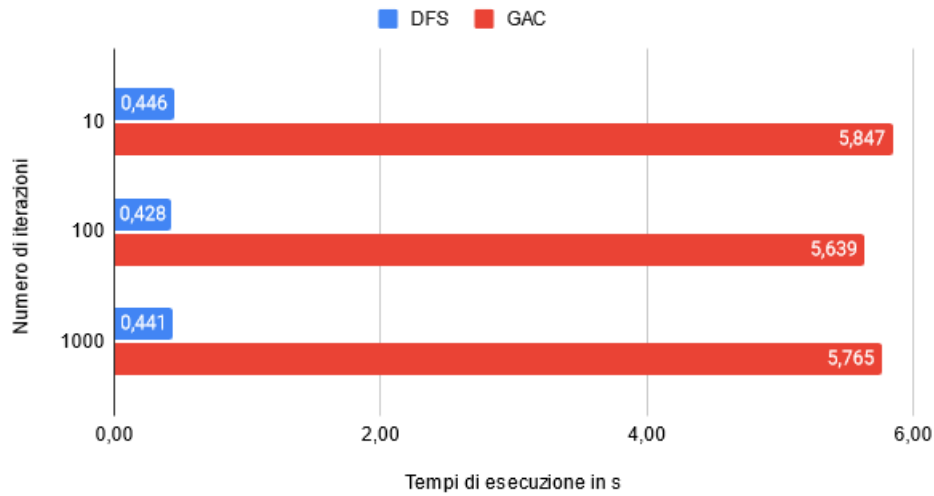
Test 2

Prendiamo adesso una situazione di un ambiente casalingo, dove possiamo trovare accesa una friggitrice, il forno, la televisione, insieme al suo impianto audio, il router, un computer fisso con il suo monitor, un caricatore del tele-

fono, un frigo, il freezer e l'aspirapolvere. In questa situazione, non possiamo andare oltre i $3kW$.

Le informazioni sul test sono contenute nel file `/appliance/home.csv` e il test dentro `/project_test/performance/test_2.py`.

Test 2 - Tempi medi di esecuzione in s



Algoritmo	μ	σ^2
DFS	0.438s	$8.633 \cdot 10^{-5} s$
GAC	5.750s	0.011s

Table 2: Media (μ) e Varianza (σ^2) dei tempi di esecuzione per il test 2

Anche in questo esempio, possiamo vedere come la DFS sia più costante e la variazione dei tempi medi è nettamente inferiore a quella del GAC.

3.1.5 Conclusioni

Si può evidenziare da questi due test, come l'uso del GAC sia meno conveniente, in termini di tempo di esecuzione media, rispetto alla DFS. Le motivazioni sono legate alle modalità di ricerca delle soluzioni.

La DFS crea un albero con profondità massima pari al numero delle variabili del problema e, in caso dovesse trovare un assegnamento parziale che non soddisfa uno o più vincoli, allora usa il backtracking e inizia a creare una nuova diramazione.

Il Generalized Arc Consistency, invece, va a rimuovere dai domini di ogni variabile, tutti quei valori che rendono la rete inconsistente. Questo viene fatto andando a dividere il dominio di una variabile in due suoi sottodomini e, per entrambi, cerca quali valori rendono inconsistente l'arco. Infine, dopo aver eliminato i valori indesiderati, mi determina tutti gli assegnamenti che risolvono il problema.

A causa della lentezza del Generalized Arc Consistency non si è potuto condurre altri test più sofisticati. Come nel caso del secondo test, per ottenere il tempo di esecuzione media su 1000 iterazioni, ci ha impiegato ~ 96 minuti.

3.2 Ontologie

3.2.1 Descrizione generale

In informatica, un'ontologia è una rappresentazione formale, condivisa che esplicita una concettualizzazione di un dominio di interesse. In particolare, nell'ambito dell'AI, mi specifica dei significati e dei simboli di un sistema informativo.

Quindi specifica quali individui e relazioni esistono, e quale terminologia è stata usata per descriverli.

3.2.2 Scelte progettuali

L'uso del design pattern del Singleton si è rivelato utile in quando permette l'uso di una sola istanza globale della classe `ApplianceOntology`. Questa classe è stata usata per rappresentare le informazioni dell'ontologia usata e

di manipolarla, tramite operazioni di lettura e scrittura.

L'istanza creata, infatti, viene usata in più parti del programma e, creare istanze differenti fra i vari moduli, rischieremmo di avere istanze non aggiornate ai cambiamenti fatti in altre parti del programma.

Per progettare e strutturare l'ontologia, è stato scelto il software Protégé, in quanto consigliato dal libro di testo.

Inoltre, per permettere a Python di lavorare con l'ontologia, è stata usata la libreria `rdflib` in quanto fornisce un'organizzazione e un supporto migliore rispetto alla libreria `owlready2`.

Infine, per rappresentare i dispositivi ho scelto di considerare solamente le categorie dedicate al multimedia, alla cucina, alla pulizia e al raffreddamento della casa, in quanto sono tra i dispositivi più utilizzati. Tutti quei dispositivi che non ricadono in queste categorie, allora andranno nella categoria *altro*.

3.2.3 Protégé

Lo strumento utilizzato per creare l'ontologia è Protégé [5] con la versione 5.6.3.

L'ontologia è organizzata prendendo in considerazione la classe principale **Appliance**, usata per rappresentare un generico dispositivo elettronico. All'interno di essa, sono presenti le seguenti data property:

- **appliance_name**: Nome del dispositivo;
- **energy_consumption**: insieme di possibili consumi elettrici per uno stesso dispositivo;
- **size**: Dimensioni del dispositivo. Queste possono essere: **small**, **medium** oppure **large**;

Le data property sono utilizzate per contenere valori primitivi, come stringhe o interi.

La classe **Appliance** possiede le seguenti sottoclassi, che andranno ad ereditare i data property della classe genitore.

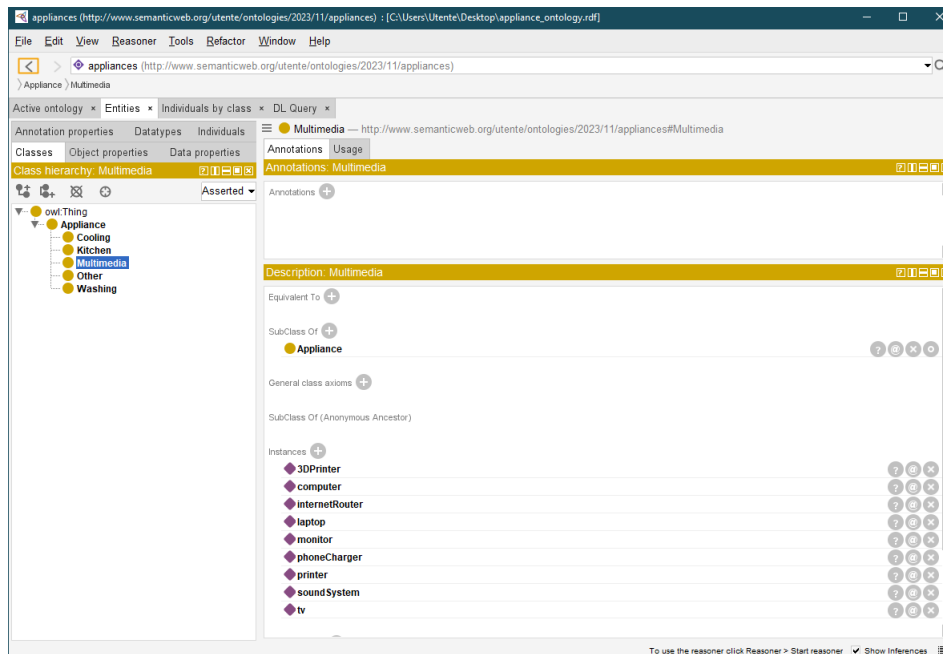


Figura 5: Schermata del tool Protégé

- **Multimedia:** Contiene tutti gli individui che fanno parte della categoria dei dispositivi multimediali.
- **Kitchen:** Contiene tutti gli individui che fanno parte della categoria dei dispositivi per la cucina.
- **Cooling:** Contiene tutti gli individui che fanno parte della categoria dei dispositivi legati a rinfrescare le strutture abitative.
- **Washing:** Contiene tutti gli individui che fanno parte della categoria dei dispositivi per la pulizia.
- **Other:** Contiene tutti gli individui che non fanno parte delle sottoclassi precedenti.

L'uso di queste sottoclassi, ci permette di distinguere meglio la categoria degli individui sfruttando la proprietà `rdfs:subClassOf(C1,C2)`, questa infatti mi dice che la classe `C1` è sottoclasse di `C2`.

Tra gli individui, infatti, oltre ad assegnare dei valori alle data property, assegno come tipo la sua sottoclasse di appartenenza.

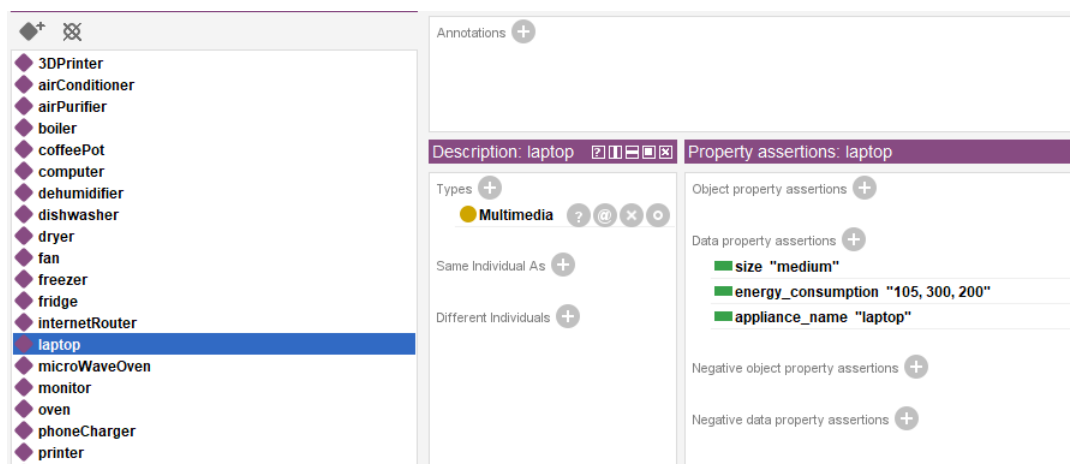


Figura 6: Schermata degli individui, con assegnamento dei valori alle sue proprietà e tipo.

3.2.4 Lavorare sull'ontologia

Tutti i dati contenuti nell'ontologia, vengono usati per strutturare i dispositivi elettronici e, una volta letti, vi è possibile applicare gli algoritmi di CSP o usare il sistema esperto.

Per lavorare sull'ontologia, ho usato la libreria `rdflib`. Grazie ad essa, è stato possibile caricare il file `/ontology/appliance_ontology.rdf` e di applicare su di esso una serie di operazioni, come:

- Lettura dell'ontologia;
- Ricerca di un individuo;
- Verificare la presenza di un individuo;
- Rimuovere un individuo;
- Aggiungere un individuo;

Quando si avvia il programma, all'utente sarà chiesto se vuole avviare il CSP, il sistema esperto, o l'ontologia. Se sceglie di avviare quest'ultima, li apparirà a schermo il seguente menù:

```
Premi 1 mostrare il contenuto dell'ontologia
Premi 2 per aggiungere un nuovo individuo
Premi 3 per eliminare un individuo
Premi 4 per salvare permanentemente gli individui aggiunti
Premi 5 per cercare un individuo dal nome
Premi 6 per verificare la presenza di un individuo dal nome

Per uscire premi 7

> █
```

Figura 7: Menù di scelta delle operazioni dell'ontologia

Tramite questo menù, potrà scegliere se aggiungere, rimuovere, cercare degli individui o visualizzarli tutti quanti. In questo modo, non sarà limitato ai soli dispositivi elettronici già presenti al suo interno.

3.2.5 Conclusioni

L'uso dell'ontologia si è rivelato utile e soprattutto è stato uno strumento importante per la rappresentazione delle informazioni sui relativi dispositivi elettronici.

La libreria `rdflib` ha accelerato di molto lo sviluppo in quanto, rispetto a `owlready2`, fornisce funzionalità molto più intuitive e in particolare, il motivo che mi ha spinto spostarmi da `owlready2` verso `rdflib` è stato il maggiore supporto da parte della community della libreria e, infine, una documentazione molto più esplicativa e intuitiva.

3.3 Sistema esperto

3.3.1 Descrizione generale

Un sistema esperto è un programma che cerca di riprodurre le prestazioni di una o più persone esperte in un determinato campo.

Un sistema esperto si basa sull'uso di una base di conoscenza, ovvero un insieme di assiomi veri. Ogni assioma della base di conoscenza è una clausola definite del tipo:

$$h \leftarrow a_1 \wedge a_2 \wedge \dots \wedge a_m$$

dove sia h che ogni a_i sono atomi.

h viene detta testa della clausola, mentre $a_1 \wedge a_2 \wedge \dots \wedge a_m$ è il corpo della clausola.

In base al numero di atomi presenti nel corpo della clausola, possiamo dividerle in due categorie:

- Se $m > 0$, allora la clausola viene chiamata *regola*;
- Se $m = 0$, allora la clausola viene chiamata *fatto*;

Il sistema esperto sviluppato, si pone l'obiettivo di segnalare all'utente se i dispositivi che vuole usare, possono portare a far scattare il salvavita.

3.3.2 Scelte progettuali

Il sistema esperto è stato realizzato tramite l'uso della libreria Python **experta**, in quanto permette di creare un sistema basato su regole in modo rapido e semplice.

3.3.3 Realizzazione del sistema esperto

Il sistema esperto possiede una serie di fatti, ovvero il consumo elettrico di tutti dei dispositivi presenti nell'ontologia. Per realizzarli, è stata usata la libreria *Experta*, una libreria Python, dove le regole si basano su due idee:

- LHS (Left Hand Side): Definisce le condizioni che devono essere verificate affinché la regola sia verificata;

- **RHS (Right Hand Side):** Le operazioni eseguite quando la regola viene applicata; sia verificata;

Un esempio di regola è la seguente:

```
@Rule( Fact( action=" start" )) # LHS
def start_system( self ):
    # RHS
```

In altri termini: **LHS** è il decoratore `@Rule()`, che si trova sopra la definizione di un metodo, mentre l'**RHS** sono le istruzioni che quel metodo eseguirà quando la regola sarà applicata.

Le regole utilizzate sono le seguenti, che si differenziano in base allo stato del sistema:

- **status_up:** Il salvavita non salterà;
- **status_warning:** Il salvavita potrebbe saltare a causa di improvvisi picchi nell'uso del dispositivo elettronico che consuma di più;
- **status_down:** Il salvavita scatterà;

Ad ognuno di esse è stata assegnata un metodo Python della classe `ExpertSystem` e, in caso si ritrovi nello stato **status_warning** o **status_down**, il sistema esperto avviserà l'utente dello stato, andandoli a consigliare quale dispositivo deve essere spento oppure quale va limitato l'uso.

3.3.4 Funzionamento

Per avviare il sistema esperto, viene richiamata la funzione `run_expert_system(max_usage: float)`, dove il parametro richiesto è il limite del salvavita espresso in *kWh*.

Il sistema esperto, una volta avviato, andrà a chiedere all'utente quali dispositivi elettronici vuole accendere.

Dispositivi elettronici disponibili				
Indice	Nome	Categoria	Dimensioni	Consumi Energetici
1	micro_wave_oven	Kitchen	medium	[207]
2	3D_printer	Multimedia	medium	[430, 780]
3	air_conditioner	Cooling	large	[300, 500]
4	freezer	Kitchen	large	[250]
5	phone_charger	Multimedia	small	[4, 6]
6	monitor	Multimedia	medium	[70]
7	fridge	Kitchen	large	[400]
8	fan	Cooling	small	[30, 50]
9	vacuum_cleaner	Other	medium	[40, 60]
10	boiler	Kitchen	small	[200]
11	laptop	Multimedia	medium	[105, 300, 200]
12	air_purifier	Cooling	small	[130]
13	oven	Kitchen	medium	[248]
14	radio	Multimedia	small	[50, 100]
15	sound_system	Multimedia	large	[137]
16	washing_machine	Washing	medium	[120, 300]
17	dryer	Washing	large	[201]

Figura 8: Alcuni dispositivi disponibili.

Una volta selezionati i dispositivi da controllare, se un dispositivo ha più consumi energetici, verrà chiesto quale, o quali, vuole prendere in considerazione.

Per ogni tipo di dispositivo elettronico, scegli quale prendere in base al suo consumo

printer

Indice	Consumo energetico
1	30
2	70

Figura 9: Esempio di scelta di un dispositivo

Infine, una volta scelti, il sistema esperto determina se non faranno saltare il contatore, andando a simulare per un'ora l'uso dei dispositivi scelti.

Mostriamo alcuni esempi, realizzati considerando come soglia massima $1.7kWh$.

Esempio 1

Seleziono solamente una stampante da $70W$

Per ogni tipo di dispositivo elettronico, scegli quale prendere in base al suo consumo

printer

Indice	Consumo energetico
1	30
2	70

Seleziona l'indice di uno o piu' consumi energetici, separati da uno spazio (Es. 1 5):
2

Tutto sotto controllo!

Figura 10: Primo esempio di uso del sistema esperto

Esempio 2

Adesso scegliamo una TV da $150W$, il suo impianto audio da $137W$, condizionatore da $500W$ e una stampante 3D da $780W$.

La soglia per cui viene mostrato questo messaggio è di $150W$, infatti se la differenza del consumo massimo consentito e la somma dei dispositivi scelti, non supera i $150W$ allora il salvavita potrebbe saltare.

```
Seleziona l'indice di uno o piu' consumi energetici, separati da uno spazio (Es. 1 5):  
2  
  
Un uso prolungato dell seguente dispositivo: 3D_printer, con consumo di 780 Wh, potrebbe  
far scattare il salvavita, causato da un improvviso aumento dei suoi consumi.
```

Figura 11: Secondo esempio di uso del sistema esperto

Esempio 3

Adesso scegliamo un condizionatore da 500W, una stampante 3D da 780W, una lavatrice da 300 e il frigo da 400W.

```
Seleziona l'indice di uno o piu' consumi energetici, separati da uno spazio (Es. 1 5):  
2  
  
Attenzione!, accendendo i dispositivi scelti rischi di far saltare il salvavita!  
Consiglio di spegnere il seguente dispositivo: 3D_printer, in quando consuma 780 Wh
```

Figura 12: Terzo esempio di uso del sistema esperto

3.3.5 Conclusioni

Grazie all'uso della libreria Python **experta**, è stato possibile sviluppare facilmente.

Naturalmente, si può estendere ulteriormente mettendo in considerazione anche situazioni in cui l'impianto elettrico sia datato o meno e di prendere in considerazione anche eventuali fattori atmosferici, come la presenza di venti forti o di ondate di calore, che possono compromettere l'efficienza del impianto.

4 Conclusioni e Sviluppi Futuri

L'applicativo realizzato, è riuscito nell'intento di poter tenere traccia degli elettrodomestici presenti, di poter fornire feedback all'utente su quale dispositivo rischia di far saltare il salvavita e, infine, di poter determinare quali dispositivi rispettino i vincoli forniti.

Possiamo estendere e migliorare l'applicativo in vari modi:

- Migliorare l'interazione con l'utente tramite una GUI;
- Avviare una simulazione dell'uso di un sottogruppo di elettrodomestici nell'arco di una giornata per un individuo;
- Poter considerare, insieme al consumo energetico, anche i costi della bolletta;
- Uso di dataset che prendano in considerazione consumi e tempi di utilizzo reali;
- Espandere la base di conoscenza, con il costo energetico di ogni dispositivo elettronico;

5 Riferimenti Bibliografici

- [1] freeCodeCamp (2022): [Creazione di un ambiente virtuale in Python](#).
- [2] AIPython (2023): [Ragionamento con i vincoli, capitolo 4 pagina 69](#).
- [3] Paginazione: [Funzionamento della paginazione](#).
- [4] `perf_counter_ns()`: [Documentazione ufficiale per la funzione `perf_counter_ns\(\)`](#).
- [5] Protégé: [Sito ufficiale di Protégé](#).
- [6] PowerShell Documentation (2022): [Documentazione sul funzionamento delle `execution policy` della PowerShell Windows](#).
- [7] Attribute Error (2023): [Risoluzione problema legato alle dipendenze `datate`](#).

Appendices

.1 Inizializzare il progetto

.1.1 Scaricare da GitHub

Il primo passaggio è quello di clonare la repository cliccando al seguente link: <https://github.com/Stefano-Labianca/smart-energy-controller>.

.1.2 Impostare l'ambiente virtuale

Va creato successivamente un ambiente virtuale [1]. In questo modo l'interprete Python, le librerie e gli script installati al suo interno, saranno isolati dagli altri ambienti virtuali e da qualsiasi libreria installata sul proprio sistema.

Per creare l'ambiente virtuale, entrate nella cartella del progetto e digitate il seguente comando:

```
python -m venv .venv
```

Grazie a questo comando, verrà creata una cartella `/.venv` che conterrà tutto il necessario per lavorare con l'ambiente virtuale.

Una volta creato, è necessario attivare l'ambiente virtuale. Se siete in ambiente MacOS o Linux, digitate il comando

```
source .venv/bin/activate
```

Il file `activate` serve per "accendere" l'ambiente virtuale.

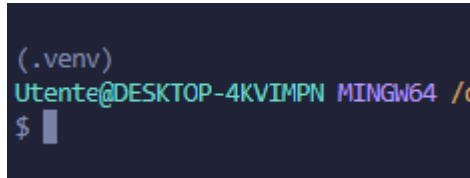
Se invece siete in ambiente Windows, allora posizionatevi prima dentro la cartella `./venv/Scripts/` per poi digitare uno dei seguenti comandi, in base al tipo di terminale in uso:

```
activate.bat // Se usi il CMD  
.\Activate.ps1 // Se usi la PowerShell
```

Se invece si sta usando il GitBash, in ambiente Windows, allora il comando da applicare è il seguente:

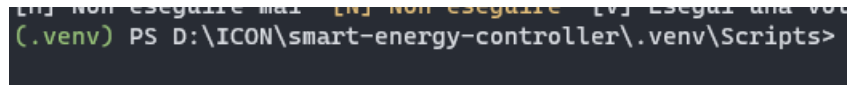
```
source .venv/Scripts/activate
```

L'ambiente virtuale sarà attivato con successo quando sul vostro terminale avrete qualcosa di simile alla figura 2.



```
(.venv)  
Utente@DESKTOP-4KVIMPX MINGW64 /d  
$
```

(a) Uso di GitBash



```
[n] Non eseguire mai [n] Non eseguire [v] Esegui una volta  
(.venv) PS D:\ICON\smart-energy-controller\.venv\Scripts>
```

(b) Uso della PowerShell

Figura 13: Risultato dell'attivazione dell'ambiente virtuale. In figura (a) abbiamo l'uso del GitBash, in figura (b) abbiamo l'uso della PowerShell.

In caso di problemi nell'uso della PowerShell, è possibile andare alla sezione: [Risolvere il problema di esecuzione con la PowerShell](#).

.1.3 Avviare il progetto

Per avviare il progetto, bisogna tornare alla directory principale del progetto e installare le dipendenze con il comando:

```
pip install -r requirements.txt
```

Una volta installate, digitare il seguente comando per avviare il programma

```
python main.py
```

In caso di errore, vedere la sezione: [Errore di esecuzione del programma Python](#).

.1.4 Avviare un test

I test svolti sul programma si trovano tutti nella cartella `/project_test`. Se si vuole eseguire un test, basta aprire un file, contenuto in una delle sotto-cartelle, prendere il suo contenuto e spostarlo dentro il file `test_main.py`, presente nella root del progetto, dove si potrà eseguire il testo scelto.

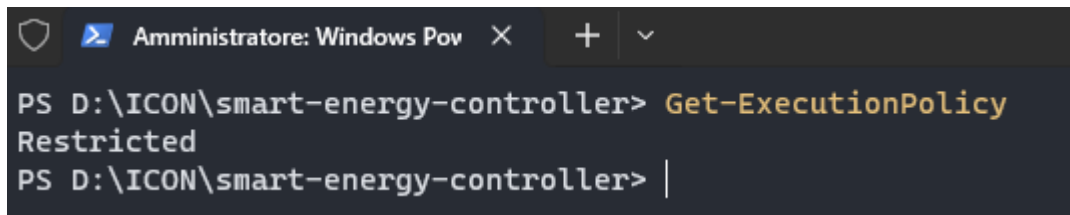
.2 Troubleshooting

.2.1 Risolvere il problema di esecuzione con la PowerShell

In caso non si riesca ad eseguire con la PowerShell [6], l'attivazione dell'ambiente virtuale, provate a svolgere i seguenti passi.

Aprire innanzitutto la PowerShell come amministratore nella cartella del progetto. Una volta aperta la PowerShell, digitare il comando:

```
Get-ExecutionPolicy
```

A screenshot of a Windows PowerShell terminal window. The title bar at the top reads "Amministratore: Windows Pov" with a shield icon on the left and window control buttons on the right. The terminal content shows a PowerShell prompt "PS D:\ICON\smart-energy-controller>" followed by the command "Get-ExecutionPolicy" in orange. The output is "Restricted". Below that, the prompt "PS D:\ICON\smart-energy-controller>" is followed by a vertical bar "|".

```
PS D:\ICON\smart-energy-controller> Get-ExecutionPolicy
Restricted
PS D:\ICON\smart-energy-controller> |
```

Figura 14: Possibile execution policy

Grazie a questo comando, possiamo sapere quale execution policy è impostata per la PowerShell. Nella figura seguente, è mostrato una possibile execution policy impostata nella PowerShell.

Per permettere l'esecuzione degli script `.ps1`, allora bisogna impostare su `AllSigned` la execution policy.

Per farlo si usa il comando:

```
Set-ExecutionPolicy -ExecutionPolicy AllSigned
```

Una volta eseguito questo comando, andare nella cartella `/.venv/Scripts/` e digitare il comando:

```
.\Activate.ps1
```

Apparirà sul terminale il seguente output:

```
PS D:\ICON\smart-energy-controller\.venv\Scripts> .\Activate.ps1

Eseguire software di questo autore non attendibile?
Il file D:\ICON\smart-energy-controller\.venv\Scripts\Activate.ps1 è pubblicato
da CN=Python Software Foundation, O=Python Software Foundation, L=Beaverton,
S=Oregon, C=US e non è considerato attendibile nel sistema in uso. Eseguire solo
script creati da autori attendibili.
[M] Non eseguire mai [N] Non eseguire [V] Esegui una volta
[S] Esegui sempre[?] Guida (il valore predefinito è "N"): |
```

Figura 15: Messaggio di conferma

Per eseguire lo script di attivazione, allora inserite V e premete invio.
Una volta che avete finito l'esecuzione del programma, potete anche reimpostare la execution policy al suo stato precedente, usando il comando:

```
Set-ExecutionPolicy -ExecutionPolicy <PolicyNamePrecedente>
```

.2.2 Errore di esecuzione del programma Python

E' possibile che, durante l'esecuzione del programma, possa apparire il seguente messaggio di errore [7]:

```
Utente@DESKTOP-4KQVMPN MINGW64 /d/ICON/smart-energy-controller (main)
$ python main.py
Traceback (most recent call last):
  File "D:\ICON\smart-energy-controller\main.py", line 1, in <module>
    from knowledge_base.expert_system import run_expert_system
  File "D:\ICON\smart-energy-controller\knowledge_base\expert_system.py", line 1, in <module>
    from experta import DefFacts, Fact, KnowledgeEngine, Rule
  File "D:\ICON\smart-energy-controller\.venv\Lib\site-packages\experta\__init__.py", line 5, in <module>
    from .engine import KnowledgeEngine
  File "D:\ICON\smart-energy-controller\.venv\Lib\site-packages\experta\engine.py", line 13, in <module>
    from experta.fact import InitialFact
  File "D:\ICON\smart-energy-controller\.venv\Lib\site-packages\experta\fact.py", line 9, in <module>
    from experta.utils import freeze, unfreeze
  File "D:\ICON\smart-energy-controller\.venv\Lib\site-packages\experta\utils.py", line 4, in <module>
    from frozendict import frozendict
  File "D:\ICON\smart-energy-controller\.venv\Lib\site-packages\frozendict\__init__.py", line 16, in <module>
    class frozendict(collections.Mapping):
    ~~~~~
AttributeError: module 'collections' has no attribute 'Mapping'
(.venv)
```

Figura 16: Errore di esecuzione

Questo errore è dovuto ad una versione datata della libreria **frozendict** usata come dipendenza della libreria **experta**.

Fare l'upgrade della libreria **frozendict** alla versione più recente, andrebbe a creare dei conflitti di dipendenza tra le due versioni della librerie.

Per risolvere questo problema, bisogna andare nella cartella `/.venv/Lib/site-packages/frozendict` e aprire il file `__init__.py`.

Una volta aperto, bisogna cambiare la seguente linea di codice:

```
class frozendict(collections.Mapping):  
    ...
```

Nella seguente:

```
class frozendict(collections.abc.Mapping):  
    ...
```