Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

---

Tesi di Laurea Triennale in Ingegneria Informatica

Development of an ontology for IOT industry plant elements and relationships modeling and its execution for SVG graph production

Candidato
Mattia Bacci

Relatore
Prof. Paolo Nesi

Correlatori
Prof. Nicola Mitolo
Prof. Pierfrancesco Bellini

Anno Accademico 2019-2020

Indice

Introduzione

Il Freamwork Snap4City fornisce metodi e soluzioni flessibili per creare velocemente una grande varietà di applicazioni per Smart City, sfruttando dati eterogenei e attivando servizi per aziende o enti pubblici tramite applicazioni IOT/IOE, data analytics e tecnologie big data. Una di queste applicazioni permette di usare un file SVG per rappresentare variabili salvate su database o inviate da applicazioni IOT, permettendo quindi di visualizzare lo stato di sistemi in real-time e di eseguire azioni su di essi. Quest'applicazione trova il suo massimo utilizzo per la visualizzazione e gestione degli impianti industriali. Tali impianti però, nella maggior parte dei casi, risultano di alta complessità e la creazione di un file SVG che ne dia una rappresentazione accurata è molto onerosa dal punto di vista temporale. Inoltre una qualsiasi modifica all'impianto comporterebbe una modifica del SVG il che rende, in pratica, l'applicazione poco utilizzabile sopra una certa complessità.

Questo lavoro si pone dunque l'obiettivo di creare un modello per la rappresentazione di impianti industriali e di utilizzarlo all'interno di un applicazione Java per automatizzare la costruzione del file SVG partendo da un semplice file CSV nel quale viene descritto l'impianto.

Capitolo 1

Ontologia

Il dover rappresentare sistemi ad alta complessità, come quelli degli impianti industriali, porta obbligatoriamente a dover creare un modello per la rappresentazione dell'entità e delle relazioni che le legano.

1.1 Una definizione formale

Una conoscenza formalmente rappresentata è basata sul concetto di concettualizzazione: gli oggetti, i concetti, le entità, che si presume esistano in qualche area di interesse e le relazioni che le caratterizzano (Genesereth & Nilsson, 1987). Una concettualizzazione è una vista astratta e semplificata del mondo che vogliamo rappresentare per qualche scopo. Un'Ontologia è un'esplicita specificazione di una concettualizzazione. [1]

1.2 Requisiti

Lo studio ha avuto inizio con la ricerca di ontologie adatte a realizzare l'intento con il rispetto dei seguenti requisiti:

1. Rappresentare una grande varietà di apparati industriali quali pompe, valvole, bruciatori ecc..
2. Permettere la localizzazione di tali apparati negli spazio come aree, settori, sotto-settori ecc..
3. Rappresentare Flussi che si spostano da un apparato all'altro, sia di oggetti che di sostanze.
4. Avere la possibilità di collegare agli apparati dei sensori, ognuno specializzato nella misurazione di una determinata grandezza.
5. Poter associare ad ogni dispositivo delle caratteristiche proprie.

Tutte le caratteristiche, ad eccezione dei Flussi, sono state riscontrate nell'ontologia Open-Source Saref4bldg.

1.3 Stato dell'arte

1.3.1 SAREF

I requisiti elencati rendono chiara l'idea che un'ontologia adatta deve mettere al primo posto il concetto di dispositivo, nel nostro caso un qualsiasi apparato industriale. L'ontologia SAREF sviluppata da ETSI ha come punto di partenza proprio il concetto di Device con il quale si intente un oggetto tangibile atto a svolgere una o più funzioni. L'ontologia offre delle funzioni di base che possono eventualmente essere combinate in modo da vere funzioni più complesse in un singolo Device. Un Device inoltre offre un Servizio, il quale è la rappresentazione di una Funzione.

Come da descrizione SAREF fornisce una soluzione molto limitata al problema che vogliamo affrontare, d'altra parte ETSI ha rilasciato molte ontolo-

gie che vanno ad estendere la SAREF in domini differenti. Le due candidate per questo progetto risultano SAREF4INMA e SAREF4BLDG.

SAREF4INMA estende l'ontologia SAREF sul dominio industriale e manifatturiero. L'ontologia non risulta però adatta al nostro caso in quanto essa si focalizza sulla produzione industriale partendo da Batch di oggetti, i dispositivi vengono intesi come macchine orientate alla produzione, nella quali entrano certi oggetti e ne escono altri. L'ontologia descrive quindi l'industria con una relazione tra queste macchine e lascia indefinito come, ad esempio, un oggetto passi da una macchina all'altra.

1.3.2 SAREF4BLDG

Nella gestione delle strutture è richiesta un'interazione e integrazione più efficiente di attori, metodi e strumenti durante le diverse fasi del ciclo costruttivo, nella quali, multipli strumenti interagiscono con modelli per estrarre informazioni per diversi fini. D'altra parte, sono richiesti meccanismi per facilitare lo scambio di dati tra attori durante tutte le diverse fasi del ciclo costruttivo per provvedere all'interoperabilità tra strumenti.

Visto che il modello standard, ISO Industry Foundation Classes(IFC), supporta l'interoperabilità tra dati e strumenti, è stato deciso di estendere l'ontologia SAREF con un sottoinsieme di questo standard relativo ai dispositivi ed elettrodomestici.

SAREF4BLDG è pensata per permettere l'interoperabilità tra le diverse parti(architetti, ingegneri, consulenti ecc.) e le applicazioni che gestiscono informazioni nelle diverse fasi nel ciclo costruttivo(Studio e Design, Costruzione, Demolizione ecc.).

Usando SAREF4BLDG, elettrodomestici smart da produttori che supportano il modello IFC potranno comunicare facilmente tra loro.

SAREF4BLDG è un ontologia OWL-DL che si focalizza sull'estendere l'ontologia SAREF per includere quei dispositivi definiti nella versione 4 dell'IFC e, in questo modo, permettere la rappresentazione di tali dispositivi e altri oggetti fisici in determinati luoghi. [2]

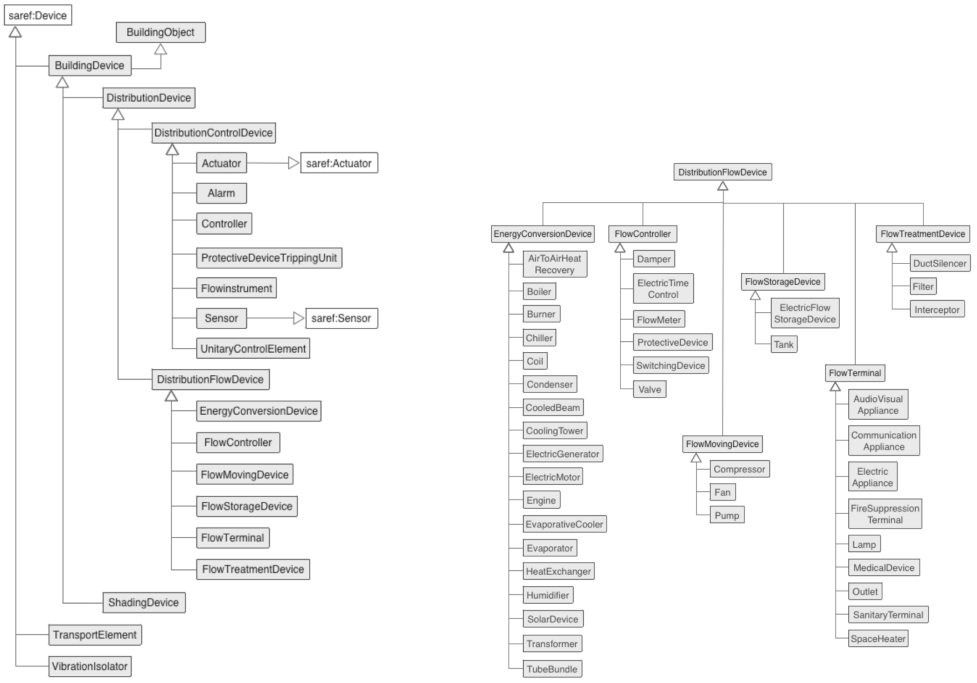
1.3.3 Analisi e Rispetto dei requisiti

Requisito 1: *Rappresentare una gran varietà di apparati industriali quali pompe, valvole, bruciatori ecc..*
L'ontologia usata ha già la disponibilità di una grande classificazione di dispositivi come mostrato nella Fig.1.1. Non si sente quindi la necessità di espanderla.

Requisito 2 *Permettere la localizzazione di tali apparati negli spazio come aree, settori, sotto-settori ecc..*
L'ontologia permette una localizzazione spaziale accurata attraverso le classi Building e BuildingSpace, utilizzando anche la classe geo:SpatialThing per permettere la localizzazione dei dispositivi attraverso latitudine e longitudine. Un esempio è mostrato nella Fig. 1.2.

Requisito 3 *Rappresentare Flussi che si spostano da un apparato all'altro, sia di oggetti che di sostanze.*
L'ontologia non permette nessun scambio di oggetti o di sostanze tra due dispositivi, la classi che rappresentano tali relazioni devo essere sviluppate.

Requisito 4 *Avere la possibilità di collegare agli apparati dei sensori, ognuno specializzato nella misurazione di una determinata grandezza.*



(a) Gerarchia Device (b) Gerarchia DistributionFlowDevice

Figura 1.1: Gerarchia dispositivi Saref4bldg

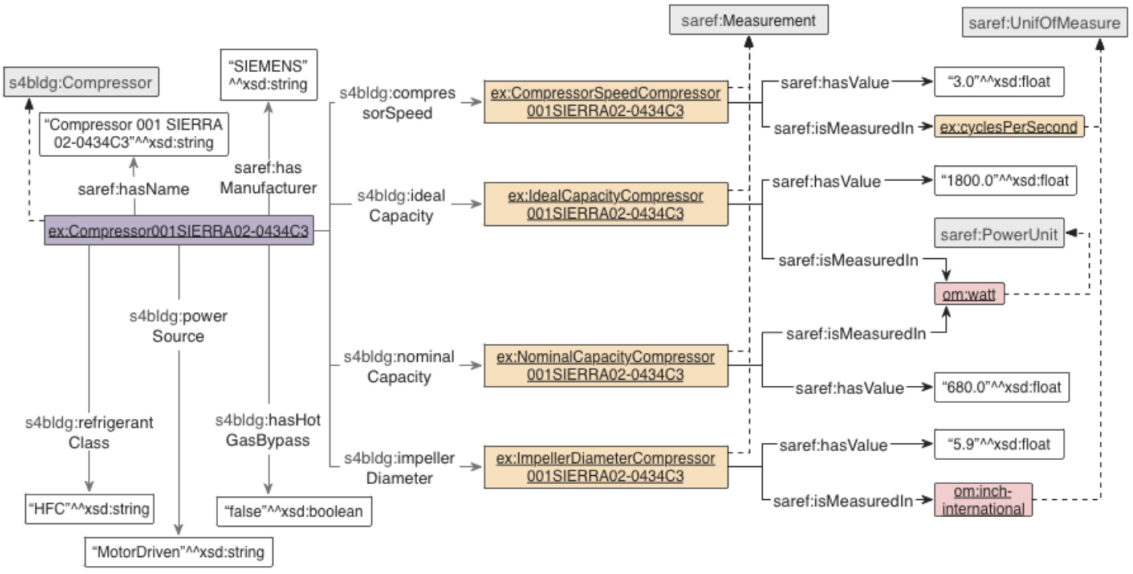


Figura 1.2: Esempio Saref4bldg

L'ontologia mette a disposizione una classe Sensor per raggiungere l'intento. Da notare però che non esiste una proprietà che collega la classe Sensor alla classe Unit of Measure, requisito indispensabile, in un impianto industriale.

Requisito 5 *Poter associare a ogni dispositivo delle caratteristiche proprie.*
L'ontologia fornisce, per ogni dispositivo, le proprietà che descrivono le sue caratteristiche principali. Il numero di proprietà di un dispositivo però può essere, in teoria, anche molto grande, in questo caso l'ontologia dovrà essere modificata per adattarsi a questo caso particolare. Per il nostro sviluppo sia le proprietà che le classi dei dispositivi vengono lasciate invariate in quanto si adatta perfettamente alla maggior parte dei casi.

1.4 Sviluppo

Il non rispetto dei requisiti numero 3 e 4 ci porta a dover estendere l'ontologia. La nuova Ontologia prende il nome di saref4bldg-ext il cui IRI risulta *http://www.disit.org/saref4bldg-ext/*. Per farlo si utilizza il software Protégé.

1.4.1 Protégé

Protégé è un editor di ontologie gratuito e open source e un framework per basi di conoscenza.

La piattaforma Protégé rende disponibile due strade per modellare le ontologie tramite gli editor Protégé-Frames e Protégé-OWL.

Le ontologie protégé possono essere esportate in una varietà di formati tra cui RDF, RDFS, OWL e XML Schema.

Protégé è basato su Java, fornisce un ambiente plug-and-play che lo rende flessibile per la prototipazione rapida e lo sviluppo di applicazioni. Esempi sono un editor visuale per OWL (chiamato OWLViz), back-end di archiviazione per Jena e Sesame, nonché un plug-in OWL-S, che fornisce alcune funzionalità specializzate per la modifica delle descrizioni OWL-S dei servizi Web. [3]

1.4.2 I Flussi

Il requisito numero 3 porta a dover definire della classi che permettano di rappresentare il passaggio di oggetti o sostanze tra due dispositivi. La classi sviluppate sono mostrate in Fig. 1.3. Per ogni classe è stato definito un rdfs:comment per identificarne lo scopo e il dominio di riferimento.

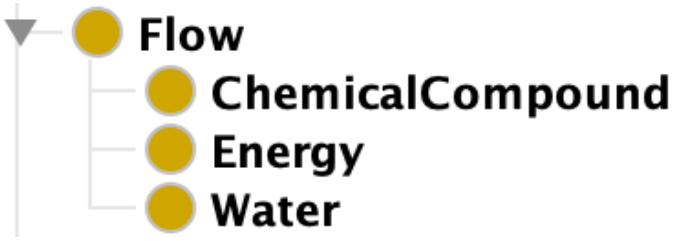


Figura 1.3: Sottoclassi Flow

Si riportano gli rdfs:comment(in inglese) delle classi sviluppate.

- **Flow:** A generic Flow from and to a Physical Object, usually achived with pipes.
- **ChemicalCompound:** Any sort of Chemical Compounds that flow between Physical Object.
- **Energy:** Any sort of Energy used or producted by a Phisycal Object.
- **Water:** Water Flow from and to a Physical Object.

1.4.3 Le proprietà

Avendo introdotto nuove classi è necessario definirne le seguenti proprietà:

- **startIn:** The Physical Object from which the Flow start.
- **endIn:** The Physical Object into which the Flow end.

A conseguenza di questo sono state introdotte restrizioni nella classe Flow, mostrate in Fig. 1.4.

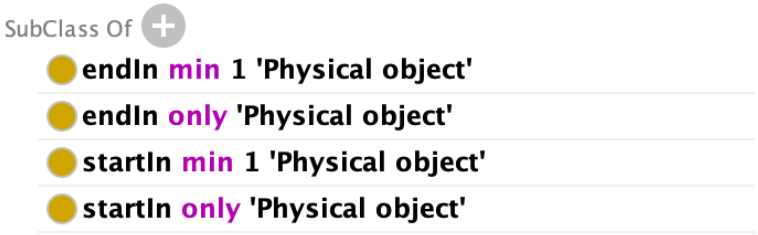


Figura 1.4: Restrizioni Flow

Per poter soddisfare il requisito 5 invece è sufficiente aggiungere la seguente proprietà:

canMeasureIn: A relation between a sensor and the unit of measurement in which the sensor measurement is expressed.

Capitolo 2

Snap4City

2.1 Panoramica

Il Freamwork Snap4City fornisce metodi e soluzioni flessibili per creare velocemente una grande varietà di applicazioni per Smart City sfruttando dati eterogenei e attivando servizi per aziende o enti pubblici tramite applicazioni IOT/IOE, data analytics e tecnicologie big data.

Snap4City è capace di tenere sotto controllo l'evoluzione in real-time della città: leggere sensori, analizzare e controllare indicatori di performance, KPI, individuare evoluzioni inaspettate, analisi dei dati. Snap4City supporta la città nel processo di continua evoluzione dei servizi, infrastutture, con controllo e supervisione, applicazioni per business intelligence, predizioni, individuazione di anomalie, allerta precoce, valutazione del rischio, analisi whatif, la messa a punto di strategie per aumentare la resilienza della città rispetto a eventi inaspettati e sconosciuti.

Grazie al supporto della knowledge base, Snap4City fornisce soluzioni flessibili per ottenere intuizioni e deduzioni immediate dello stato e dell'evoluzione della città, sfruttando l'intelligenza artificiale, analisi dei dati e tec-

nologie dei big data, attivando la raccolta di soluzioni senzienti e sfruttando dati eterogenei di qualsiasi tipo, da qualsiasi fonte di dati. [4]

2.2 L'applicazione

L'applicazione ha lo scopo di mappare delle variabili, presenti in un database o fornite direttamente da una IOT App, su un File SVG per la loro visualizzazione real-time. Una volta istanziato, viene mostrato il file SVG dato in input e, nelle aree disegnate, i valori delle variabili. Tale applicazione risulta perfetta quindi per la visualizzazione di un impianto industriale con i valori dei sensori in esso presenti. Nella Fig. 2.1 si mostra Il file SVG di una sezione per la produzione di soda dell'impianto chimico Altair di Prato.

Dalla Fig. 2.1 possiamo dedurre i seguenti fatti:

- La visualizzazione ha una bassa complessità, si tende a generalizzare le parti dell'impianto.
- Essendo stato creato a mano la sua costruzione ha richiesto un tempo non indifferente.

L'obiettivo di questa tesi è quello di risolvere queste due problematiche.

2.3 Requisiti

Il dover mappare delle variabili direttamente sul file SVG porta tale file a dover essere costruito secondo uno schema predefinito. Per il nostro obiettivo è sufficiente visualizzare le variabili e non interagire con l'SVG. Come da specifica, per raggiungere tale obiettivo all'interno di un nodo <**text**></**text**>, all'interno del quale deve essere visualizzata la variabile, deve essere incluso il seguente attributo:

```
<text data-siow = "[{  "event":"s4csvg_NAME", "originator":"
    server", "actions": [{ "input":"$.lastValue", "target":"
    textContent"}]}]"></text>
```

Dove al posto di *NAME* si sostituisce il nome della variabile.

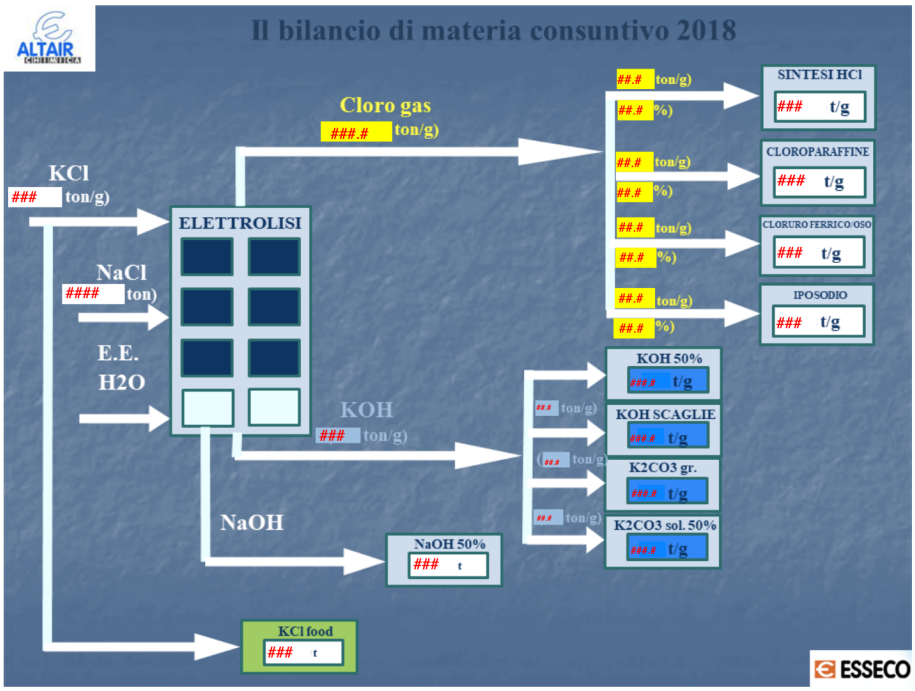


Figura 2.1: SVG Impianto Altair

Capitolo 3

SVG Generator

3.1 Caratteristiche generali

L'applicazione Java si pone l'obiettivo di fornire all'utente un interfaccia grafica attraverso la quale poter inserire il file CSV(dove è descritto l'impianto)e il file OWL(l'ontologia) per aver come risultato un file SVG che potrà poi essere caricato su Snap4City.

3.2 Input e Output

3.2.1 Il File CSV

Il compito del file CSV è quello di descrivere l'impianto con tutti i dispositivi presenti e le sue proprietà. Per realizzare l'intendo è stata stabilita una struttura interna del file stesso per permettere all'applicazione Java di leggerlo senza errori.
Come da acronimo CSV(Comma Separated Values) i valori all'interno del file dovrebbero essere separati da virgole. Excel d'altronde prevede per tale

file(nella versione italiana) la separazione con punto e virgola quindi, non volendo alterarne i parametri o creare confusione, il programma funzionerà soltanto con file che rispettino tale carattere separatore.
Attraverso il file è possibile creare istanze di Classi, ObjectProperty e DataProperty.

- Per le classi è stata adottata una struttura a triple:

ENTITY	Class	CLASS

  Dove *ENTITY* sarà il nome dell'entità e *CLASS* sarà il nome della classe.

- Per gli ObjectProperty è stata adottata una struttura a quadruple:

ENTITY_1	ObjectProperty	OBJECT_PROPERTY	ENTITY_2

  Dove *ENTITY_1* e *ENTITY_2* saranno il nome dell'entità che partecipano alla relazione e *OBJECT_PROPERTY* sarà la proprietà a cui si riferisce.

- Per i DataProperty è stata adottata una struttura a quadruple:

ENTITY_1	DataProperty	DATA_PROPERTY	VALUE

  Dove *ENTITY* sarà il nome dell'entità, *DATA_PROPERTY* sarà la proprietà a cui si riferisce e *VALUE* il valore della proprietà espresso sotto forma di Double, String o Boolean.

Nella Fig. 3.1 viene mostrato un esempio di un file CSV completo.

KCL	Class	Tank	
R-2003	Class	PhysicalObject	
R-2001	Class	PhysicalObject	
Valve_1	Class	Valve	
Bruciatore	Class	Burner	
Valve_2	Class	Valve	
HCL	Class	Tank	
Flusso_1	Class	Flow	
Flusso_1	ObjectProperty	startIn	KCL
Flusso_1	ObjectProperty	endIn	R-2003
Flusso_1	ObjectProperty	endIn	R-2001
Flusso_2	Class	Flow	
Flusso_2	ObjectProperty	startIn	R-2003
Flusso_2	ObjectProperty	startIn	R-2001
Flusso_2	ObjectProperty	endIn	Valve_1
Flusso_3	Class	Flow	
Flusso_3	ObjectProperty	startIn	R-2003
Flusso_3	ObjectProperty	startIn	R-2001
Flusso_3	ObjectProperty	endIn	Bruciatore
Flusso_4	Class	Flow	
Flusso_4	ObjectProperty	startIn	Valve_1
Flusso_4	ObjectProperty	endIn	Bruciatore
Flusso_5	Class	Flow	
Flusso_5	ObjectProperty	startIn	Bruciatore
Flusso_5	ObjectProperty	endIn	Valve_2
Flusso_6	Class	Flow	
Flusso_6	ObjectProperty	startIn	Valve_2
Flusso_6	ObjectProperty	endIn	HCL
HC-R2003-4	Class	Sensor	
HC-R2003-4	ObjectProperty	isContainedIn	R-2003

(a) CSV Tabella

```
KCL;Class;Tank;
R-2003;Class;PhysicalObject;
R-2001;Class;PhysicalObject;
Valve_1;Class;Valve;
Bruciatore;Class;Burner;
Valve_2;Class;Valve;
HCL;Class;Tank;
Flusso_1;Class;Flow;
Flusso_1;ObjectProperty;startIn;KCL
Flusso_1;ObjectProperty;endIn;R-2003
Flusso_1;ObjectProperty;endIn;R-2001
Flusso_2;Class;Flow;
Flusso_2;ObjectProperty;startIn;R-2003
Flusso_2;ObjectProperty;startIn;R-2001
Flusso_2;ObjectProperty;endIn;Valve_1
Flusso_3;Class;Flow;
Flusso_3;ObjectProperty;startIn;R-2003
Flusso_3;ObjectProperty;startIn;R-2001
Flusso_3;ObjectProperty;endIn;Bruciatore
Flusso_4;Class;Flow;
Flusso_4;ObjectProperty;startIn;Valve_1
Flusso_4;ObjectProperty;endIn;Bruciatore
Flusso_5;Class;Flow;
Flusso_5;ObjectProperty;startIn;Bruciatore
Flusso_5;ObjectProperty;endIn;Valve_2
Flusso_6;Class;Flow;
Flusso_6;ObjectProperty;startIn;Valve_2
Flusso_6;ObjectProperty;endIn;HCL
HC-R2003-4;Class;Sensor;
HC-R2003-4;ObjectProperty;isContainedIn;R-2003
```

(b) CSV Plain

Figura 3.1: Esempio File CSV

3.2.2 Il File OWL

Il file OWL contiene la nostra ontologia. Non mi dilungo sulla spiegazione della sintassi in quanto il file è generato direttamente da Protégé. Nel nostro caso è stata usata la sintassi *Turtle*.

3.2.3 Il File SVG

SVG è un linguaggio di markup per descrivere applicazioni grafiche ed immagini in due dimensioni. La versione usata è la più recente al momento della scrittura, SVG 1.1 seconda edizione, in accordo con W3C. Il file SVG ci permette quindi di poter codificare la rappresentazione grafica dell'impianto direttamente in formato XML. Non è stato fatto utilizzo di nessuna libreria per la creazione del file, i nodi vengono semplicemente aggiunti come un qualsiasi file XML.

All'interno del file è presente una piccola parte di codice Javascript atto a permettere determinate animazioni.

3.3 UI

La UI(User Interface) è stata creata attraverso l'ausilio di Swing, un Freamwork Java orientato allo sviluppo di interfacce grafiche. Nella Fig. 3.2 si mostra l'interfaccia, se ne descrive brevemente i componenti:

1. **File di Input** Si inserisce, tramite un file picker, i File CSV e OWL richiesti.

2. **Opzioni di Salvataggio** E' possibile scegliere tra due impostazioni:

   - Utilizzare l'ontologia solo come modello e quindi non apportare modifiche al file OWL.
   - Utilizzare l'ontologia come modello e salvare le entità aggiunte all'interno del file OWL.

3. **Generate SVG** Da il via alla generazione dell'SVG.

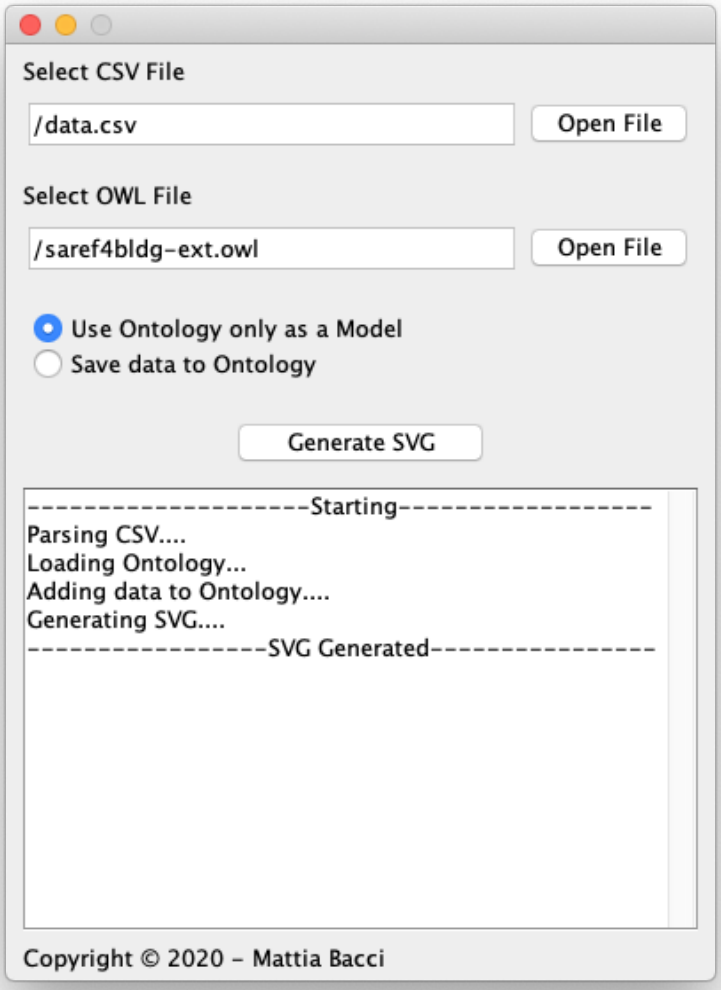


Figura 3.2: UI

4. **Pannello di Stato** Viene visualizzato lo stato corrente del programma durante l'esecuzione e vi vengono riportati eventuali errori.

3.4 Funzionamento

3.4.1 Librerie Usate

Dover interagire con l'ontologia ha reso necessario l'uso di una libreria. La libreria in questione deve avere i seguenti requisiti:

- Essere compatibile con OWL-2, essendo la nostra ontologia sviluppata con tale specifica.
- Avere la possibilità di poter utilizzare Query SPARQL per interrogare l'ontologia.

Il freamwork Apache Jena rende possibile lavorare direttamente con il grafo RDF, permettendo in questo modo l'utilizzo di Query SPARQL ma il suo supporto per OWL è limitato alla specifica OWL-1, ormai obsoleta.
La libreria OWL-API, d'altra parte, supporta completamente la specifica OWL-2 ma non permette di lavorare con il grafo RDF.

Per unire i punti di forza di Apache Jena e OWL-API è presente una libreria denominata ONT-API, un'implementazione delle interfacce di OWL-API su Apache Jena, in questo modo in aggiunta alla rappresentazione strutturale degli assiomi con specifica OWL-2, prevede la possibilità di lavorare direttamente con il grafo RDF.

ONT-API è l'unica libreria esterna utilizzata all'interno di questo progetto.

3.4.2 Stuttura del software - UML

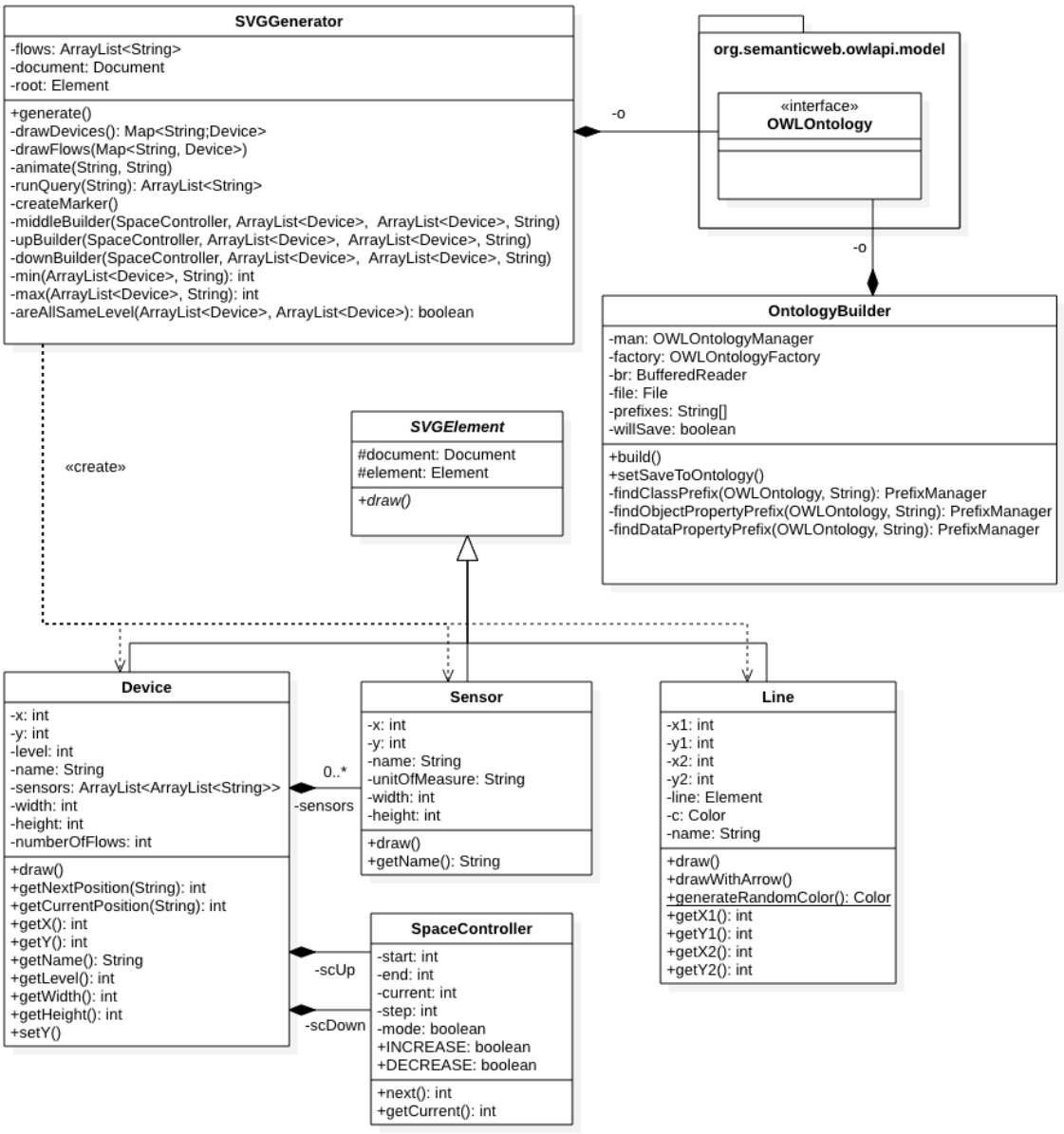


Figura 3.3: UML

3.4.3 Dal CSV all'ontologia

La Classe Ontology Builder è responsabile delle lettura del file CSV e dell'aggiunta delle entità, Object Property e DataProperty, all'ontologia. Il programma esegue ciclicamente i seguenti passaggi.

1. Legge il contenuto di una riga del file CSV.
2. Analizza la riga appena letta per capire se si vuole aggiungere un entità o una proprietà.
3. Visto che il file CSV non include davanti alle classi/proprietà gli IRI dell'ontologia viene ricercato all'interno dell'ontologia la classe/proprietà per identificarne l'IRI e successivamente creare l'entità.

3.4.4 Query SPARQL

La possibilità di poter lavorare con il grafo RDF fornita dalla libreria ONT-API ci permette di eseguire e di interrogare l'ontologia attraverso Query SPARQL.

Viene riportata la funzione che esegue delle query generiche:

```
1  private ArrayList<String> runQuery(String query) {
2    ArrayList<String> arrayList = new ArrayList<>();
3    try (QueryExecution qexec = QueryExecutionFactory.create(QueryFactory.
4    create("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
5                    "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
6                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
7                    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
8                    "PREFIX : <http://www.disit.org/saref4bldg-ext/>\n" +
9                    "PREFIX s: <https://saref.etsi.org/saref4bldg/>\n" +
10                   "PREFIX c: <https://saref.etsi.org/core/>\n" +
11                   "SELECT ?subject WHERE {" + query + "}"),
12                   ((Ontology) o).asGraphModel())) {
13       ResultSet res = qexec.execSelect();
14       while (res.hasNext()) {
```

```
15              QuerySolution qs = res.nextSolution();
16              RDFNode x = qs.get("subject");
17              arrayList.add(x.asNode().getLocalName());
18          }
19      }
20      return arrayList;
21  }
```

Tale funzione prende in ingresso una query generica e restituisce un Arraylist<String> contenente i nomi delle entità/proprietà.

Si utilizza la funzione per richiedere all'ontologia:

- Dispositivi
- Flussi
- Dispositivi di partenza e di arrivo dei flussi
- Sensori e loro unità di misura

3.4.5 Generazione dell'SVG

Una volta che l'OntologyBuilder ha terminato l'aggiunta delle entità all'ontologia, essa viene passata come riferimento OWLOntology alla classe SVGGenerator, responsabile della generazione dell'SVG.

Una volta che tutti i dispositivi, flussi e sensori sono stati recuperati, attraverso Query SPARQL, si passa all'aggiunta dei nodi al file SVG secondo una schema ben definito qui sotto illustrato:

Dispositivi e sensori Per i dispositivi si utilizza una struttura rettangolare in modo da poter collegare facilmente i flussi alla due basi maggiori dei rettangoli. Essi vengono disposti in due file orizzontali e separati da uno spazio variabile in modo da poter usare tale spazio per i flussi, che dovranno collegare dispositivi posti su file diverse.

I sensori sono inseriti all'interno dei rispettivi dispositivi indicandone nome, valore e unità di misura.

Un esempio è visibile nella Fig. 3.4

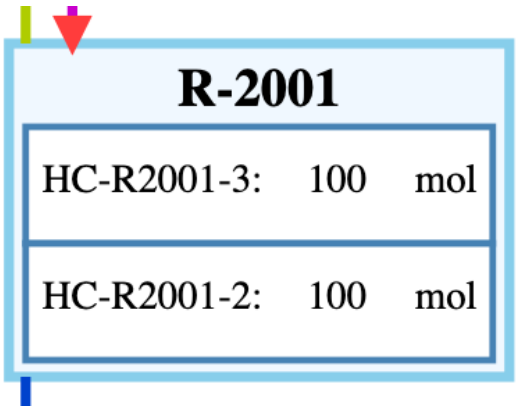


Figura 3.4: Esempio Dispositivo e Sensori

Flussi Per i Flussi sono state scelte delle linee il cui colore è anch'esso indicato nel file CSV, la distinzione del dispositivo di arrivo a quello di partenza è data da una freccia rossa che segue la direzione del flusso.

Gli spazi di partenza, arrivo e passaggio dei flussi sono tutti determinati da degli oggetti di tipo SpaceController. Ogni dispositivo ha due SpaceController, uno superiore e uno inferiore, l'arrivo e la partenza di un flusso è permessa da entrambi i lati. Il software gestisce dinamicamente lo spazio presente sopra, tra e sotto i dispositivi in modo da adattarsi al numero di flussi presenti in quel caso, ognuna di queste aree è gestita da uno SpaceController specifico. Se è presente un flusso che collega due o più dispositivi dello stesso livello esso verrà riportato nello spazio superiore o inferiore ai dispositivi e non in quello centrale, riservato alla connessione tra dispositivi di livelli diversi.

Un esempio di SVG completo è visibile nella Fig. 3.5

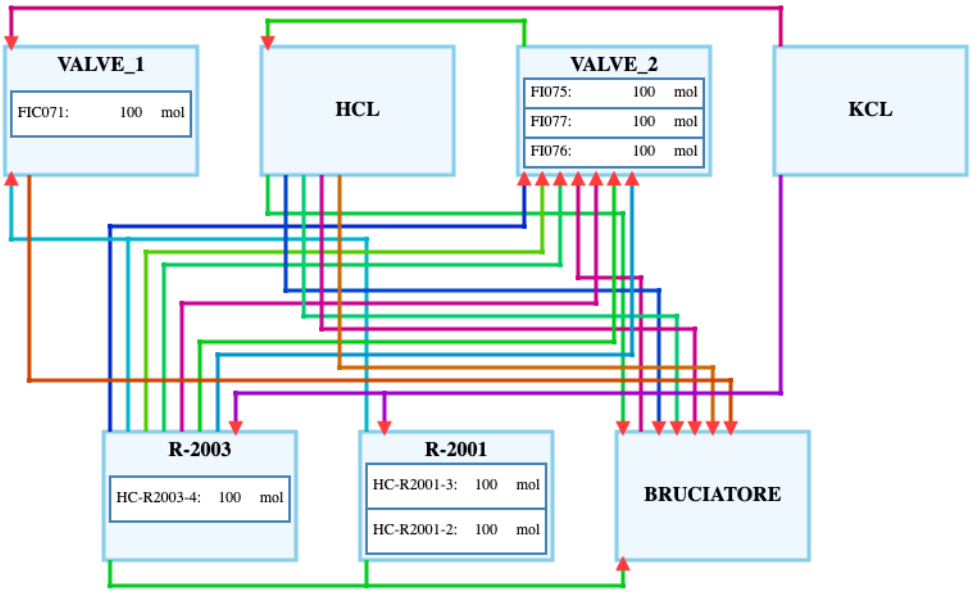


Figura 3.5: Esempio SVG

3.4.6 Animazioni

Al crescere del numero di flussi e dispositivi la visualizzazione dell'SVG può risultare difficoltosa, non si riesce ad identificare velocemente le partenze e gli arrivi di determinati flussi. Introducendo delle animazioni riusciamo ad ovviare a questo problema.

L'animazione finale risulta come da Fig. 3.6.

Una volta che ci spostiamo con il mouse sul flusso interessato esso aumenta di spessore, mostrerà il suo nome e i dispositivi di partenza ed arrivo. In questo modo si riesce a visualizzare la relazione diretta tra i flussi e i dispositivi a cui sono connessi.

Le animazioni sono state scritte in JavaScript direttamente all'interno dell'SVG, questo implica che, se il file è aperto da un lettore SVG generico che non supporta JavaScript o da un browser dove è stato disabilitato, le animazioni non saranno disponibili.

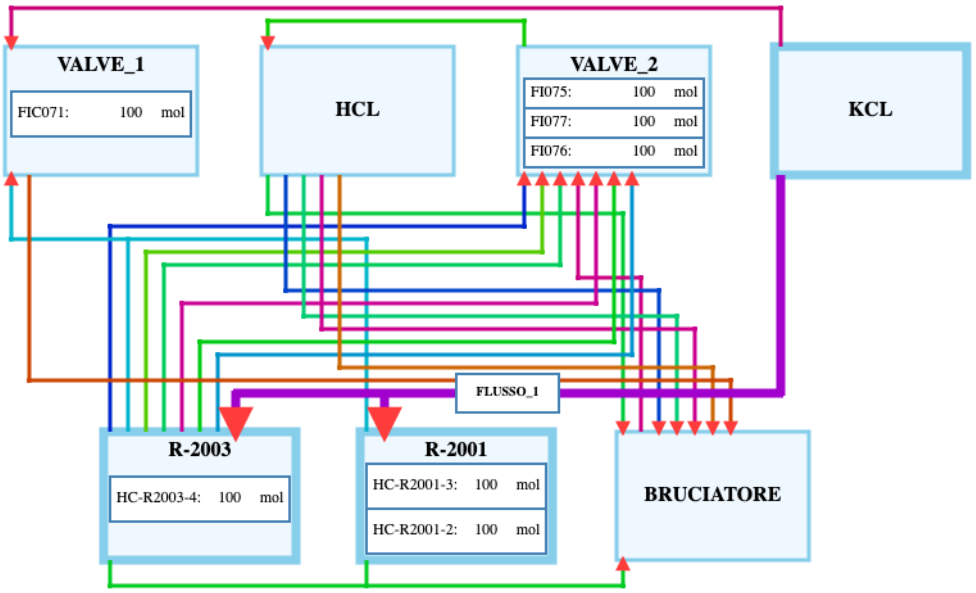


Figura 3.6: Esempio Animazione

Capitolo 4

Conclusioni

Il Software Java, sviluppato durante questo lavoro di tesi, permette di rappresentare impianti industriali ad alta complessità ed utilizzare tale rappresentazione all'interno di un'applicazione Snap4City che abilita la mappatura di variabili da database o applicazione IOT, avendo come obiettivo finale una rappresentazione grafica dell'impianto con la possibilità di visualizzare su di essa i valori dei sensori in Real-Time.

Il software, usando un'ontologia come modello di rappresentazione e un file CSV come descrizione dell'impianto, diventa uno strumento flessibile atto in primo luogo alla rappresentazione di impianti industriali ma esso non vieta l'utilizzo in altri ambiti dove sono presenti relazioni tra dispositivi.

Infine, la possibilità di poter salvare all'interno dell'ontologia le entità che sono descritte all'interno del file CSV, non solo ci permette di raggiungere l'obiettivo che questo lavoro di tesi si è preposto, ma diventa uno strumento di analisi potendo utilizzare l'ontologia per altri scopi quali, ad esempio, effettuare inferenze o generare le triple per essere caricate su un RDF Store.

Bibliografia

[1] Thomas R. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. Stanford Knowledge Systems Laboratory , Palo Alto, 1993.

[2] SAREF extension for building devices.

`https://mariapoveda.github.io/saref-ext/OnToology/SAREF4BLD/ontology/saref4bldg.ttl/documentation/index-en.html`

[3] Protégé

`https://www.w3.org/2001/sw/wiki/Protege`

[4] Snap4City.

`https://www.snap4city.org`