SBML2SHACL

Edoardo De Matteis

16 settembre 2020

Indice

1 Logiche Descrittive

La logica del primo ordine (FOL) aggiunge espressività alla logica proposizionale introducendo predicati, funzioni e quantificatori e permettendo di modellare eventi dinamici nel tempo al prezzo di un'inferenza semidecidibile, data infatti una knowledge base KB e una formula α è sempre possibile scrivere un algoritmo che risponda affermativamente se $KB \models \alpha$ ma se $KB \not\models \alpha$ l'esecuzione potrebbe non terminare.

Le **description logics** (DL) [6] sono una famiglia di linguaggi formali per rappresentazione della conoscenza spesso più espressivi della logica proposizionale, priva di quantificatori e che fa uso di proposizioni dichiarative, ma meno della logica del primo ordine, nella quale si fa uso di quantificatori, predicati, funzioni e variabili; per alcuni di questi linguaggi esistono algoritmi d'inferenza corretti e completi, più il linguaggio diventa espressivo più è difficile che l'inferenza sia decidibile.

La sintassi delle DL è fatta in modo tale da rendere facile la modellazione di definizioni e proprietà di categorie (collezioni di "oggetti" utilizzate per la rappresentazione di conoscenza su larga scala) tramite operazioni insiemistiche piuttosto che con una caratterizzazione logica, tra FOL e DL cambia dunque la sintassi: ciò che in FOL è **costante**, **predicato unario** e **predicato binario** nelle DL diventa rispettivamente **individuo**, **concetto** e **ruolo**. Se in FOL si parla di statement nelle DL si parla di descrizioni, descrizioni elementari sono descrizioni composte da concetti e ruoli atomici e combinandole tra loro si ottenegono descrizioni complesse; le descrizioni di concetti nella logica descrittiva $\mathcal{AL}$ si ottengono secondo la seguente sintassi:

$$
\begin{array}{lcll}
C, D & \rightarrow & A & \text{(concetto atomico)} \\
 & & \mid \top & \text{(concetto universale)} \\
 & & \mid \bot & \text{(concetto nullo)} \\
 & & \mid \neg A & \text{(negazione atomica)} \\
 & & \mid C \sqcap D & \text{(intersezione)} \\
 & & \mid \forall R.C & \text{(restrizione di valore)} \\
 & & \mid \exists R.\top & \text{(quantificazione esistenziale limitata)}
\end{array}
$$

Come in FOL un'interpretazione I consiste di un insieme non vuoto Δ^I, il dominio d'interpretazione, e di una funzione di interpretazione che assegna ad ogni concetto atomico A un insieme $A^I \subseteq \Delta^I$ e ad ogni ruolo atomico R una relazione binaria $R^I \subseteq \Delta^I \times \Delta^I$.

$$
\begin{array}{rcc}
\top^I & = & \Delta^I \\
\bot^I & = & \emptyset \\
\neg A^I & = & \Delta^I \setminus A^I \\
(C \sqcap D)^I & = & C^I \cap D^I \\
(\forall R.C)^I & = & \{a \in \Delta^I | \forall b.(a,b) \in R^I \rightarrow b \in C^I\} \\
(\exists R.\top)^I & = & \{a \in \Delta^I | \exists b.(a,b) \in R^I\}
\end{array}
$$

Dal momento che le DL sono una famiglia possiamo definire una logica descrittiva [6, Appendix 1] componendo vari frammenti che ne indicano l'espressività, ad esempio possiamo estendere $\mathcal{AL}$ con i costrutti in tabella 1, nella colonna **Simbolo** è rappresentato il nome del frammento cui è proprio un costrutto. Se ne vedono altri esempi nella tabella 1.

Tabella 1: Paronamica di frammenti di DL che ci permettono di avere la stessa espressività della FOL.

Costrutto	**Sintassi**	**Semantica**
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Intersezione	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Unione	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negazione	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Restrizione	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
Esistenziale	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$

Per comprendere meglio le DL è utile vedere una corrispondenza tra alcuni costrutti in FOL e nelle DL (tabella 2) e vedere alcune descrizioni d'esempio.

Tabella 2: Corrispondenza DL e FOL.

FOL	**DL**
$A(x)$	A
$C(a)$	$C(a)$, $a : C$
$\neg C(x)$	$\neg C$
$C(x) \wedge D(x)$	$C \sqcap D$
$C(x) \vee D(x)$	$C \sqcup D$
$\forall x.C(x) \rightarrow D(x)$	$C \sqsubseteq D$
$R(a,b)$	$R(a,b)$, $(a,b) : R$
$\forall x,y.R(x,y) \rightarrow S(x,y)$	$R \sqsubseteq S$
$\exists y.R(x,y) \wedge C(y)$	$\exists R.C$
$\forall x,y,z.R(x,y) \rightarrow R(y,z) \rightarrow R(x,z)$	$R \circ R \sqsubseteq R$
$A \leftrightarrow B$	$A \equiv B$

Siano **Uomo**, **Mortale** concetti atomici, allievoDi un ruolo atomico e *Socrate*, *Platone*, *Aristotele* individui con significati ovvi.

- Socrate è un uomo.

$$\mathbf{Uomo}(Socrate)$$

- Ogni uomo è mortale.

$$\mathbf{Uomo} \sqsubseteq \mathbf{Mortale}$$

- Se ogni uomo è mortale e Socrate è un uomo allora ogni uomo è mortale.

$$[(\mathbf{Uomo} \sqsubseteq \mathbf{Mortale}) \sqcap \mathbf{Uomo}(Socrate)] \sqsubseteq \mathbf{Mortale}(Socrate) \qquad (1)$$

- Platone è allievo di Socrate.

$$\underline{\text{allievoDi}}(Platone, Socrate)$$

Tabella 3: Alcuni simboli di DL con associati i costrutti caratteristici.

Simbolo	**Costrutti**	**Esempio**
$\mathcal{AL}$	Negazione atomica, intersezione di concetti, restrizione universale, quantificazione esistenziale limtiata	**Gatto** $\equiv$ **Domestico** $\sqcap$ **Felino**
$\mathcal{C}$	Classe di espressioni complesse ottenute combinato operatori come quali la relazione di sottoclasse, equivalenza, congiunzione, disgiunzione, negazione, restrizioni, tautologie e contraddizioni	**Gatto** $\equiv$ **Felino** $\sqcap \neg$(**Leone** $\sqcup$ **Tigre**)
$\mathcal{S}$	$\mathcal{ALC}$ con ruoli transitivi	$\underline{\text{antenatoDi}} \circ \underline{\text{antenatoDi}} \sqsubseteq \underline{\text{antenatoDi}}$
$\mathcal{E}$	Quantificazione esistenziale completa	$\exists \underline{\text{mangiatoDa}}.$**Insetto**
$\mathcal{U}$	Unione di concetti	**Animale** $\equiv$ **Cane** $\sqcup$ **Gatto**
$\mathcal{H}$	Gerarchia di ruoli	$\underline{\text{mangiatoDa}} \sqsubseteq \underline{\text{predaDi}}$
$\mathcal{O}$	Classi enumerate di oggetti	**Italia** $\equiv \{$**Nord**, **Centro**, **Sud**$\}$
$\mathcal{I}$	Ruoli inversi	$\underline{\text{predatoreDi}} \equiv \underline{\text{predaDi}}^{-}$
$\mathcal{Q}$	Restrizione di cardinalità qualificata	**Gatto** $\sqsubseteq = 4\underline{\text{haGambe}}$
$(\mathcal{D})$	Uso di tipi di dati, sue proprietà o valori	

Dal momento che nelle DL si eseguono operazioni insiemistiche non si parla propriamente di inferenza ma di sussunzione (subsumption): $C \sqsubseteq D$ equivale in FOL a $C \rightarrow D$, similmente a quanto avviene nel model checking in logica proposizionale $\alpha \models \beta \longleftrightarrow M(\alpha) \subseteq M(\beta)$ - la funzione $M(\phi)$ data una formula ϕ ne restituisce l'insieme dei modelli. Per il teorema di deduzione $\alpha \models \beta \longleftrightarrow \alpha \rightarrow \beta$. L'equazione 1 è un esempio di sussunzione.

Le DL si basano su un'interfaccia "tell and ask" e si fanno sussunzioni analizzando TBox e ABox della KB: nella TBox si danno le definizioni dei concetti e si forma una conoscenza intensionale, ad esempio è qui che deve sarà presente

Gatto $\equiv$ **Domestico** $\sqcap$ **Felino**; quando si costruisce una TBox l'operazione base è la *classification*, consiste nel posizionare il concetto appena definito nella giusta posizione gerarchira rispetto ai concetti già presenti nella TBox. Questo viene fatto verificando la sussunzione tra il concetto preso in esame e ogni altro concetto nella TBox. In questo progetto la TBox corrisponde al file `shapes.ttl`, la definizione di un concetto corrisponde infatti alla definizione di una shape.

Nell'ABox invece sono presenti le affermazioni su individui (membership assertions), si definsice quindi una conoscenza estensionale del dominio di interesse: è qui che va ad esempio la descrizione **Gatto**($Silvestro$). La principale operazione di ragionamento in un'ABox è l'*instance checking* nel quale si verifica che un individuo sia un'istanza di un dato concetto; altre operazioni svolte sono le seguenti che comunque non sono altro che casi particolari di *instance checking*.

- *knowledge base consinstency* con la quale si verifica che per ogni concetto esista almeno un individuo.
- *realization* con la quale si trova il concetto più specifico del quale un individuo è istanza.
- *retrieval* col il quale dato un concetto si trovano tutti gli individui che ne sono istanza.

Il file *output.ttl* generato dai parser `parser.py` e `extended_parser.py` corrisponde all'ABox, quando si verifica la sua correttezza rispetto alla TBox `shapes.ttl` con `shacl_verifier.py` si eseguono le sussunzioni alla base delle operazioni di ragionamento sopra descritte .

1.1 RDF

Il **Resource Description Framework (RDF)** [4] è una famiglia di specifiche definite dal World Wide Web Consortium (W3C) utilizzate per la modellazione di informazioni implementate come risorse web. RDF è simile ad altri approcci classici come database relazionali o logiche descrittive nei quali si hanno affermazioni sulle risorse sotto forma di triple soggetto-predicato-oggetto; il soggetto è la risorsa stessa e il predicato esprime una relazione tra soggetto e oggetto.

Se ad esempio volessimo esprimere il concetto "il cielo è blu" avremmo:

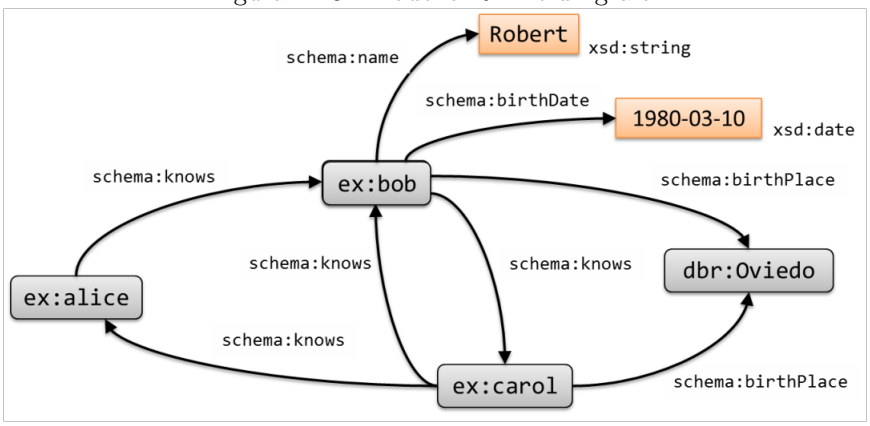
soggetto: "il cielo"

predicato: "è di colore"

oggetto: "blu"

Contrariamente all'approccio object oriented entità-attributo-valore in cui avremmo un oggetto "cielo" con attributo "colore" di valore "blu". Come si può vedere nell'immagine 1 un insieme di triple rappresenta intrinsecamente un grafo: una rappresentazione decisamente conveniente ma spesso nella pratica

Figura 1: Un modello RDF è un grafo.



un modello RDF è mantenuto come database relazionale (chiamato Triplestore o Quad store).

L'ABox è definita in RDF (nello specifico Turtle in quanto è il linguaggio RDF scelto) che è meno espressivo delle DL [7, Logic for the semantic web, 2.4], nella tabella 5 si vede come sia possibile esprimere alcune espressioni RDF con le logiche descrittive:

Tabella 5: Traduzione da RDF a DL.

RDF	DL
x `rdfs:type` A	$A(x)$
xPy	$P(x, y)$
A `rdfs:subClassOf` B	$A \sqsubseteq B$
P `rdfs:domain` A	$\exists P \sqsubseteq A$
A `rdfs:range` B	$\exists P^- \sqsubseteq A$
P `rdfs:subPropertyOf` Q	$P \sqsubseteq Q$

1.2 SHACL

Shapes Constraint Language (SHACL) [5] è una specifica del W3C per la verifica di informazioni sotto forma di grafi rispetto ad un insieme di vincoli, include anche il linguaggio di query **SPARQL Protocol and RDF Query Language (SPARQL)** [2]. Il processo di verifica prende in input un grafo contenente le dichiarazioni delle shape ovvero le definizioni cui i nodi, le istanze, dovranno essere verificati; in input si può avere qualsiasi formato RDF e in questo progetto si userà Turtle; SHACL è diretto discendente di OWL [1] il quale è stato progettato basandosi su RDF e sulle description logics, è composto infatti da *RDF-Based semantics* e *direct semantics* rispettivamente e la seconda è più espressiva della prima [9, 7] [7, Logic for the semantic web, 3.2]. La

TBox come l'ABox è definita in Turtle quindi RDF, il ruolo di SHACL è quello di verificare tramite inferenza la consistenza di TBox e ABox: è qui che avviene la vera fase di ragionamento, in questo progetto è svolta dall'agente razionale `shacl_verifier.py`.

SHACL corrisponde ad una logica descrittiva molto estesa - $\mathcal{ALCOIQ}(\circ)$ - che è corretta, non completa, senza garanzie di terminazione e NEXPTIME-hard; le premesse non sembrano ottimistiche ma è possibile rendere meno espressivo il linguaggio per ottenerere inferenze decidibili. Escludendo la composizione di ruoli $\circ$ si ha $\mathcal{SROIQ}$ per la quale esiste un algoritmo d'inferenza corretto, non completo, decidibile e NEXPTIME-hard; escludendo anche la possibilità di esprimere ruoli inversi si ottiene la DL $\mathcal{ALCOQ}$ che è sound, completa, decidibile e PSPACE-complete e dal momento che NP $\subseteq$ PSPACE $\subseteq$ EXPTIME si ha un netto miglioramento.

2 Caso di studio

2.1 SBML

Systems Biology Markup Language (SBML) [3] è un formato di rappresentazione basato su XML per la modellazione di processi biologici e chimici. Non è un linguaggio bensì una *lingua franca* tra tool che utilizzano formati di rappresentazione differenti e ad oggi è *de facto* uno standard per i modelli computazionali in biologia.

Ai fini di questo progetto distinguiamo due differenti SBML:

- **SBML Level 3** [8]. La specifica del *Core* che offre le funzionalità base di SBML, in questo documento lo si indica come "SBML 3".
- **Hierarchical Model Composition** [10]. Noto come "comp" (namespace del package e quindi parola chiave in XML) questo package offre la possibilità di include modelli annidati permettendo allo sviluppatore ed eventuali tool di:

  1. Scomporre modelli troppo grandi in modelli più piccoli per ridurne la complessità.
  2. Avere istanze multiple di un modello - precedentemente definito - all'interno di altri modelli, evitando ripetizione di codice.
  3. Creare librerie di codice riusabile e modelli considerati corretti.

  In questo documento lo si indica come "Extended SBML".

2.2 Obiettivi

Lo scopo di questo progetto, sotto la guida del Professor Tronci, è quello di convertire automaticamente una specifica SBML in SHACL. A tal fine il problema è stato diviso in tre fasi:

1. Modellare in SHACL un sottoinsieme di costrutti che definiscono SBML .
2. Scrivere un parser che traduca una specifica SBML in SHACL.
3. Scrivere un parser che traduca una specifica SHACL in SBML.

Si assume a priori che i file SBML in input siano corretti, la verifica è eseguibile online. Per il corretto funzionamento del codice inoltre è necessario il download del package Python *rdflib*.

3 Modellazione

In questa prima fase si è scelto un sottoinsieme di SBML 3.2 più i costrutti in Extended SBML e ne è stato formalizzato un modello. Nella tabella 6 vengono descritti i principali costrutti, sono state volontariamente omesse shape quali `listOf*` il cui significato è intuitivo e avrebbero solo reso la tabella meno leggibile, in ogni caso è possibile consultare ulteriormente il diagramma `diagram.png`, mostrato anche in figura 2, e il file delle shape `shapes.ttl`. È possibile verificare la consistenza di un file di nodi rispetto al modello tramite lo script `shacl_verifier.py`.

Entità	**Descrizione**
SBML 3.2	
SBase	Classe astratta superclasse di ogni altra classe. Per quanto in SBML sia considerata a tutti gli effetti un tipo, poiché non esistono attributi di tipo SBase a questa classe si riserva un trattamento differente rispetto ai tipi composti quali ID, SId, etc . . . che sono state omesse dal diagramma ma sono comunque presenti nel file `shapes.ttl`.
Sbml	Ogni file SBML ha un'etichetta con tag `sbml`, data la sua obbligatorietà ed unicità è possibile costruire grafi SHACL composti da multipli modelli SBML ed esplorarli radicando l'albero in Sbml, è importante specificare che questo nuovo modello composto non sarà legale in SBML e non si dà alcuna garanzia nella fase di riconversione in SBML. Il costrutto Sbml ammette altri attributi oltre quelli noti, siccome in SHACL ho necessità di conoscerne il tipo è stata persa questa possibilità.
Model	Rappresenta il modello, se ne può avere più di uno come detto sopra nonostante SBML non lo permetta.

Unit	Definisce un'unità di misura definita dall'utente, in SBML sono definite delle unità base (i.e. le unità del SI e altre scelte dagli sviluppatori di SBML) e combinandole opportunamente tra loro è possibile definire nuove unità di misura (e.g l'accelerazione $\frac{m}{s^2}$ che altro non è se non $m^1 s^{-2}$). Nel modello in `shapes.ttl` le unità base (kind) sono trattate come normalissime unità di misura, in SBML non è concesso avere come attributo kind un'unità di misura che non sia base e questa libertà non genera problemi.
Compartment	Rappresenta un insieme di entità biologiche.
Species	Rappresenta un'entità biologica.
Parameter	In SBML si possono definire parametri sia locali che globali, il modello presentato in queso progetto non implementa la località quindi i parametri sono sempre globali e in Model l'attributo `ListOfParameters` ha moltiplicità $[0, n]$ piuttosto che $[0, 1]$.
Extended SBML	
ExternalModelDefinition	Necessario per importare modelli esterni.
ModelDefinition	Questo costrutto fornisce la definizione di un modello.
Submodel	L'istanza di una ModelDefinition è rappresentata da un Submodel, si ha quindi un modello dentro ad un altro modello. Durante la fase di test per extended SBML non ho usato esempi scaricati da Biomodels perché la gerarchia non veniva rappresentata tramite il costrutto Submodel ma con un attributo `outer` in Compartment, con il Professor Tronci si è ritenuto fosse meglio attenersi allo standard W3C.

Port	Una port permette allo sviluppatore di definire come ci si deve interfacciare con un modello, per quanto non siano vincolanti di norma è preferibile seguire le indicazioni dello sviluppatore.
Deletion	Non è detto che i modelli importati abbiano solo ed esclusivamente componenti desiderabili e con una deletion è possibile ignorare quelli indesiderati.
Replacement	Tramite questo costrutto è possibile sostituire una componente con un'altra, ogni riferimento alla prima ora punta alla seconda. A causa di Replacement sono presenti più parser: il primo `parser.py` esplora il file XML come una lista ed `extended_parser.py` come un albero.
SBaseRef	Port, Deletion, Replacement e Submodel utilizzano dei riferimenti, SBaseRef similmente a SBase offre alle proprie sottoclassi degli attributi comuni a tutte.

Tabella 6: Modellazione SHACL

Prendiamo d'esempio il file SBML dato in input al parser `example.xml`, l'output in Turtle generato `output.ttl` e il diagramma `diagram.png` e vediamo come viene popolata l'ABox in concordanza al modello, prendiamo come riferimento la shape Model. Nel listato 1 è stata isolata la parte che riguarda il modello, gli attributi presenti nell'etichetta sono `id`, `metaid` e `name` e quindi attributo e valore sono tradotti in Turtle in predicato e oggetto rispettivamente come si può vedere nel listato 2; lo stesso viene fatto anche per le relazioni con altre entità sia che Model sia soggetto o oggetto.

```
3     <model id="YeastWTmodel2" metaid="_044070" name="
      Proctor2006_telomere">
4       <listOfUnitDefinitions>
5         <unitDefinition id="substance" metaid="metaid_0000112">
6           <listOfUnits>
7             <unit kind="item" metaid="_582757"/>
8           </listOfUnits>
9         </unitDefinition>
10      </listOfUnitDefinitions>
11      <listOfCompartments>
12        <compartment id="cellMembrane" metaid="_044088" size="1"/>
13        <compartment id="nucleus" metaid="_044071" size="1"/>
14      </listOfCompartments>
15      <listOfSpecies>
```

```
16          <species compartment="nucleus" hasOnlySubstanceUnits="true"
        id="Ctelo" initialAmount="64" metaid="_044108"/>
17        </listOfSpecies>
18      </model>
```

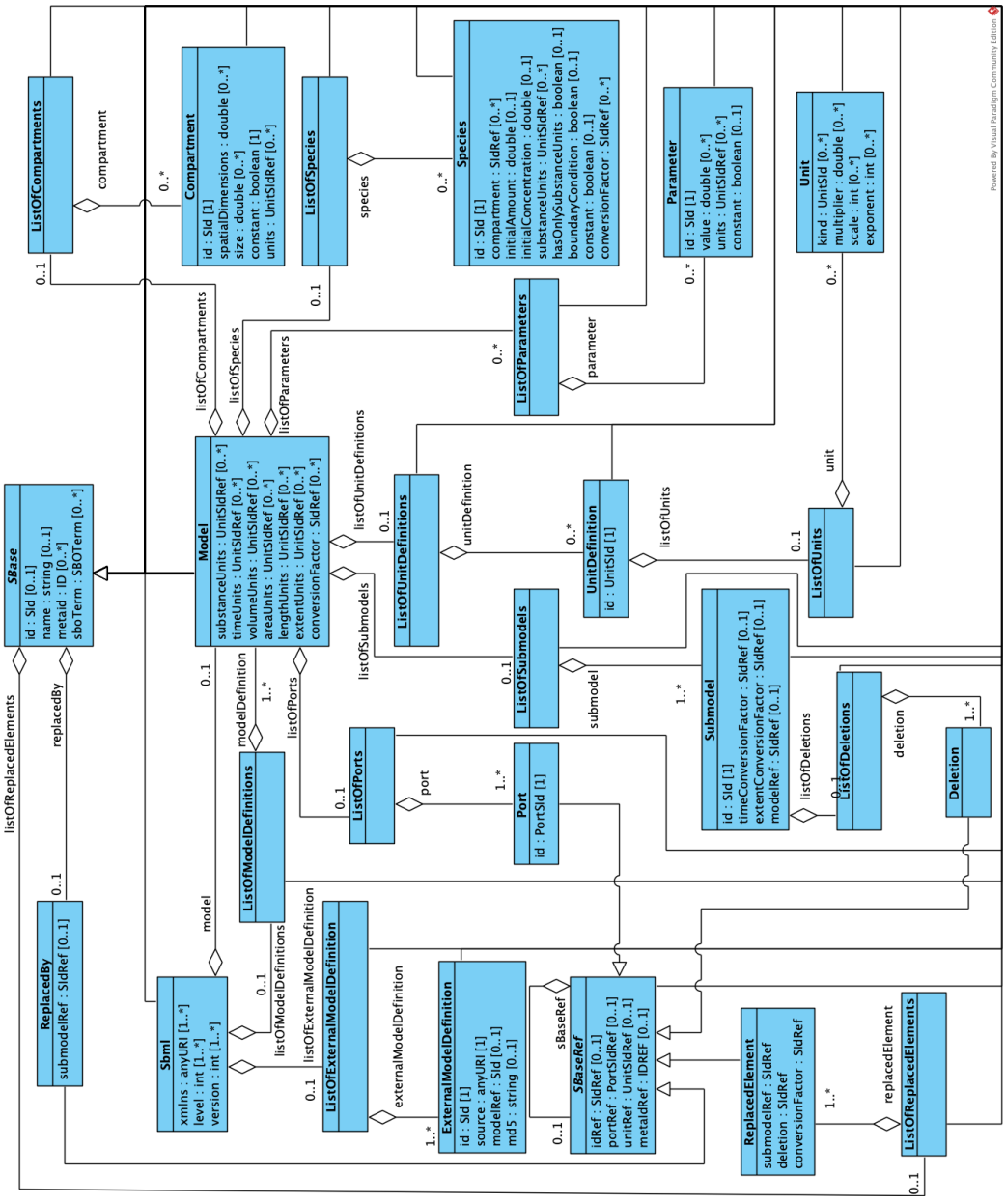
Listing 1: Sezione del modello isolata dal file di input `example.org`

```
ex:sbml_1 schema:model ex:model_2 .
ex:model_2 a schema:Model .
ex:model_2 schema:id sid:YeastWTmodel2 .
ex:model_2 schema:metaid id:_044070 .
ex:model_2 schema:name "Proctor2006_telomere"^^xsd:string .

ex:model_2 schema:listOfUnitDefinitions ex:listOfUnitDefinitions_3
    .
ex:model_2 schema:listOfCompartments ex:listOfCompartments_7 .
ex:model_2 schema:listOfSpecies ex:listOfSpecies_10 .
```

Listing 2: Sezione del modello isolata nel file di output `output.ttl`

Figura 2: Modellazione SBML



4 Parsing da SBML a SHACL

In questa seconda fase dato uno o più file SBML in input li si traduce in SHACL tramite un parser e si verifica la correttezza dell'output tramite il file `shacl_verifier.py`. Come già detto si hanno due parser: `parser.py` è la prima versione e non supporta Extended SBML contrariamente a `extended_parser.py` che è la versione finale.

In `parser.py` si riescono a tradurre senza problemi file in SBML 3.2 ma poiché il file XML viene letto come una lista di etichette risulta macchinoso (ammesso e non concesso sia possibile) fare riferimento all'etichetta padre di una sostituzione o una cancellazione. Il problema non si presenta in `extended_parser.py` dato che il file XML viene esplorato come un albero, questa seconda versione è inoltre più breve (444 righe anziché 932) ed è più facilmente estendibile. Questo parser infatti legge qualsiasi tag del file XML e vi associa i propri attributi - noti - al rispettivo valore, ho comunque imposto dei controlli per evitare questo comportamento dato che avrebbe solo rallentato la fase di verifica. Inoltre è più veloce come si può vedere in tabella 8: il calcolo delle performance è basato sull'esecuzione di `test.sh`, sono stati scaricati da Biomodels i 18 modelli SBML presenti nella cartella `examples/input/biomodel` e di seguito elencati, in `test.sh` i parser prendono in input due file così da verificare la correttezza della costruzione di un grafo a partire da più modelli e allo stesso tempo si ha un numero più consistente di test ovvero 324.

1. BIOMD0000000087.xml
2. BIOMD0000000105.xml
3. BIOMD0000000399.xml
4. BIOMD0000000474.xml
5. BIOMD0000000476.xml
6. BIOMD0000000559.xml
7. BIOMD0000000562.xml
8. BIOMD0000000619.xml
9. BIOMD0000000624.xml
10. BIOMD0000000705.xml
11. BIOMD0000000706.xml
12. MODEL1012110001.xml
13. MODEL1012220002.xml
14. MODEL1012220003.xml

15. MODEL1012220004.xml
16. MODEL1112260002.xml
17. MODEL1812100001.xml
18. MODEL3632127506.xml

File	**Time (m)**
parser.py	42:54.54
extended_parser.py	17:35.19

Tabella 8: Risultati ottenuti eseguendo `time ./test.sh` su macOS Catalina 10.15.6 con processore Intel Core i5 1.6 GHz Dual-Core.

Nella sottocartella `custom` invece vi sono file d'esempio forniti direttamente dal W3C per Extended SBML, potranno essere usati solo per verificare `extended_parser.py`. In Biomodels per esprimere una gerarchia tra compartment si fa uso di un attributo `outer` e non di sottomodelli come nella specifica del W3C.

5 Parsing da SHACL a SBML

Nella terza e ultima fase si esegue la controprova della precedente ovvero si traduce il file di istanze SHACL ottenuto in un file SBML e ci si attende che venga modellata la stessa conoscenza, il file non sarà uguale perché privo di eventuali costrutti non modellati. Nella cartella `query` sono presenti tre file:

- `query.py`: Permette di interrogare il modello con query SPARQL, la query deve essere scritta direttamente nel codice perché gli argomenti della print variano in base alla query.
- `ttl2sbml.py`: Il parser da SHACL a SBML.
- `ttl2xml`: Converte un file di istanze Turtle in XML, una query e un file XML possono esprimere la stessa conoscenza: una tripla `<subject> <predicate> <object>` si può tradurre in XML nelle etichette `<subject> <predicate> <object/> </predicate> </subject>`.

Nella scrittura del parser è risultato necessario poter riottenere la struttura annidata dell'XML che era andata persa nella scrittura delle istanze SHACL, per farlo ho definito delle classi Tree e Node, nella fase di lettura del file l'albero viene costruito gradualmente e una volta ottenuto basta esplorarlo ricorsivamente per costruire il testo XML, ricordandosi chiaramente di chiudere le etichette una volta terminate le chiamate ai figli.

L'algoritmo genera la stessa conoscenza dei file di partenza, per SBML 3.2 non ci sono problemi invece per i file nella cartella `custom` tutti tranne `MANUAL_1.xml` non superano la verifica SBML perché trattandosi di file d'esempio i riferimenti non sono reali; chiaramente anche l'output di `ttl2sbml.py` dà risultato negativo, per `MANUAL_1.txt` invece si ha lo stesso esito ovvero numerosi warning.

6 Esempio

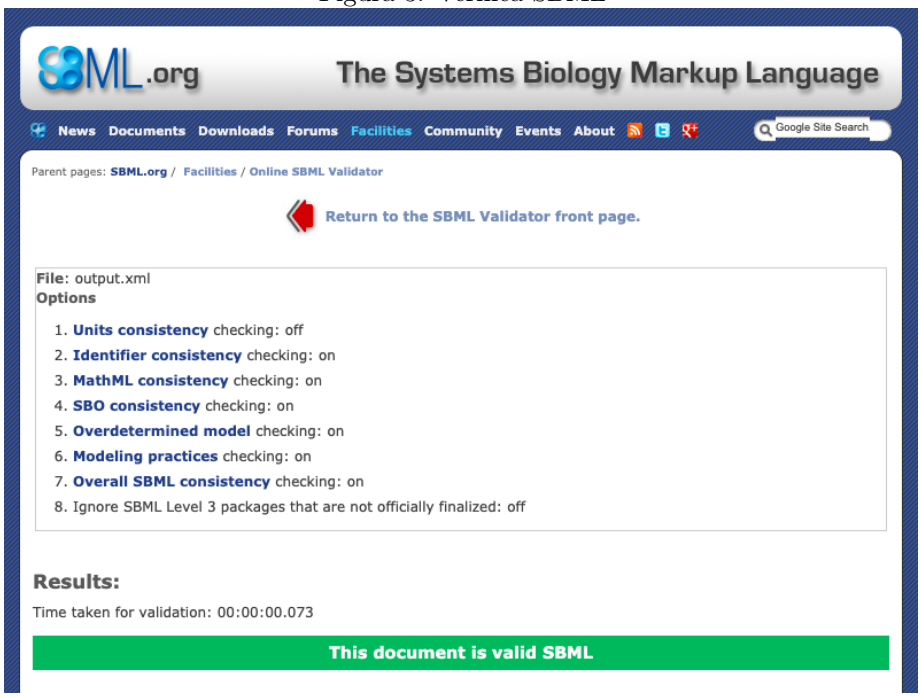
Un esempio di esecuzione è quello nella cartella `examples/output`, è stato ottenuto eseguendo i comandi nel listato 3 e in figura 3 si può vedere come SBML verifichi la traduzione da SHACL a SBML, mettendo a confronto i listati 4 e 5 si può vedere come i file SBML di input e output corrispondano.

```
python ./project/parser/extended_parser.py -f ./examples/input/
    example.xml -o ./examples/output/output.ttl
python ./project/model/shacl_verifier.py -s ./project/model/shapes.
    ttl -d ./examples/output/output.ttl
True
Validation Report
Conforms: True

python ./project/query/ttl2sbml.py -f ./examples/output/output.ttl
    -o ./examples/output/output.xml
python ./project/query/query.py -f ./examples/output/output.ttl
http://example.org/ns#compartment_9 a schema:Compartment
http://example.org/ns#compartment_8 a schema:Compartment
```

Listing 3: Esecuzione dei comandi

Figura 3: Verifica SBML



7 Commenti e critiche

Durante lo sviluppo del progetto si è tenuto conto dei seguenti commenti e critiche da parte del Professor Tronci:

Data	**Descrizione**
22/07/20	Videochiamata con specifica del problema da parte del Professor Tronci.
27/07/20	Prima stesura di una modellazione dei costrutti in SBML, avendo dimenticato di modellare extended SBML mi è stato fatto notare dal Professore e ho risolto, mi è stato anche consegnato il materiale su extended SBML da consultare.
28/07/20	Corretto il punto precedente non avevo implementato deletion e replacement, il Professore ha inoltre consigliato la strategia di testing.
31/07/20	Dopo aver aggiunto replacement e deletion e dopo aver eseguito dei test con risultati positivi è seguita una videochiamata con il Professor Tronci in cui mi è stato indicato di modificare il parser in maniera tale da poter ricevere in input più modelli SBML e di capire se il risultato di una query SPARQL rappresenti lo stesso tipo di conoscenza di un file XML/SBML, così da poter sfruttare questa corrispondenza per eseguire dei controtest. Questa corrispondenza esiste ed è usata spesso sotto il nome di "SPARQL query results xml format".
06/08/2020	Videochiamata con il Professor Tronci in cui si è deciso di comprendere se fosse possibile scrivere un parser da SHACL a SBML e, in caso di risposta affermativa, farlo.
19/08/2020	Correzioni su questa relazione ampliandola affinché i concetti risultino più chiari durante l'esposizione.
08/09/2020	Videochiamata con il Professor Mancini nella quale si è esposto come la relazione mancasse di una componente didattica significativa.
15/09/2020	Ulteriore videochiamata con il Professor Mancini perché l'estensione della relazione era ancora poco chiara.

8 Appendice

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <sbml xmlns="http://www.sbml.org/sbml/level2" level="2" metaid="
    metaid_0000001" version="1">
    <model id="YeastWTmodel2" metaid="_044070" name="
    Proctor2006_telomere">
      <listOfUnitDefinitions>
        <unitDefinition id="substance" metaid="metaid_0000112">
          <listOfUnits>
            <unit kind="item" metaid="_582757"/>
          </listOfUnits>
        </unitDefinition>
      </listOfUnitDefinitions>
      <listOfCompartments>
        <compartment id="cellMembrane" metaid="_044088" size="1"/>
        <compartment id="nucleus" metaid="_044071" size="1"/>
      </listOfCompartments>
      <listOfSpecies>
        <species compartment="nucleus" hasOnlySubstanceUnits="true"
     id="Ctelo" initialAmount="64" metaid="_044108"/>
      </listOfSpecies>
    </model>
  </sbml>
```

Listing 4: File SBML in input `example.xml`

captioncaption

```
caption
ex:sbml_1 schema:xmlns "http://www.sbml.org/sbml/level2"^^xsd:
    anyURI .
ex:sbml_1 a schema:Sbml .
ex:sbml_1 schema:level "2"^^xsd:integer .
ex:sbml_1 schema:metaid id:metaid_0000001 .
ex:sbml_1 schema:version "1"^^xsd:integer .

ex:sbml_1 schema:model ex:model_2 .
ex:model_2 a schema:Model .
ex:model_2 schema:id sid:YeastWTmodel2 .
ex:model_2 schema:metaid id:_044070 .
ex:model_2 schema:name "Proctor2006_telomere"^^xsd:string .

ex:model_2 schema:listOfUnitDefinitions ex:listOfUnitDefinitions_3
    .
ex:listOfUnitDefinitions_3 a schema:ListOfUnitDefinitions .

ex:listOfUnitDefinitions_3 schema:unitDefinition ex:
    unitDefinition_4 .
ex:unitDefinition_4 a schema:UnitDefinition .
ex:unitDefinition_4 schema:id usid:substance .
ex:unitDefinition_4 schema:metaid id:metaid_0000112 .

ex:unitDefinition_4 schema:listOfUnits ex:listOfUnits_5 .
ex:listOfUnits_5 a schema:ListOfUnits .

ex:listOfUnits_5 schema:unit ex:unit_6 .
ex:unit_6 a schema:Unit .
```

```
ex:unit_6 schema:kind usid:item .
ex:unit_6 schema:metaid id:_582757 .

ex:model_2 schema:listOfCompartments ex:listOfCompartments_7 .
ex:listOfCompartments_7 a schema:ListOfCompartments .

ex:listOfCompartments_7 schema:compartment ex:compartment_8 .
ex:compartment_8 a schema:Compartment .
ex:compartment_8 schema:id sid:cellMembrane .
ex:compartment_8 schema:metaid id:_044088 .
ex:compartment_8 schema:size "1"^^xsd:decimal .

ex:listOfCompartments_7 schema:compartment ex:compartment_9 .
ex:compartment_9 a schema:Compartment .
ex:compartment_9 schema:id sid:nucleus .
ex:compartment_9 schema:metaid id:_044071 .
ex:compartment_9 schema:size "1"^^xsd:decimal .

ex:model_2 schema:listOfSpecies ex:listOfSpecies_10 .
ex:listOfSpecies_10 a schema:ListOfSpecies .

ex:listOfSpecies_10 schema:species ex:species_11 .
ex:species_11 a schema:Species .
ex:species_11 schema:compartment sidref:nucleus .
ex:species_11 schema:hasOnlySubstanceUnits "true"^^xsd:boolean .
ex:species_11 schema:id sid:Ctelo .
ex:species_11 schema:initialAmount "64"^^xsd:decimal .
ex:species_11 schema:metaid id:_044108 .
```

```
<?xml version="1.0" encoding="UTF-8"?>
  <sbml xmlns="http://www.sbml.org/sbml/level2" level="2" metaid="
    metaid_0000001" version="1">
    <model id="YeastWTmodel2" metaid="_044070" name="
    Proctor2006_telomere">
      <listOfUnitDefinitions>
        <unitDefinition id="substance" metaid="metaid_0000112">
          <listOfUnits>
            <unit kind="item" metaid="_582757"/>
          </listOfUnits>
        </unitDefinition>
      </listOfUnitDefinitions>
      <listOfCompartments>
        <compartment id="cellMembrane" metaid="_044088" size="1"/>
        <compartment id="nucleus" metaid="_044071" size="1"/>
      </listOfCompartments>
      <listOfSpecies>
        <species compartment="nucleus" hasOnlySubstanceUnits="true"
     id="Ctelo" initialAmount="64" metaid="_044108"/>
      </listOfSpecies>
    </model>
  </sbml>
```

Listing 5: File SBML `output.xml` generato da `shacl2sbml.py`

Bibliografia

[1] Shacl and owl compared. URL: https://spinrdf.org/shacl-and-owl.html.

[2] Sparql query language for rdf, 2008. URL: https://www.w3.org/TR/rdf-sparql-query/.

[3] The systems biology markup language. URL: http://sbml.org/Main_Page.

[4] Rdf 1.1 concepts and abstract syntax, 2014. URL: https://www.w3.org/TR/rdf11-concepts/#dfn-datatype.

[5] Shapes constraint language (shacl), 2017. URL: https://www.w3.org/TR/shacl/.

[6] Franz Baader, Diego Calvanese, Deborah L Mcguinness, and Daniele Nardi. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2 edition, January 2007. URL: https://www.researchgate.net/publication/230745455_The_Description_Logic_Handbook_Theory_Implementation_and_Applications.

[7] Jörg H. Siekmann (Eds.). *Computational Logic*. Handbook of the History of Logic 9. North Holland, 1 edition, 2014.

[8] Michael Hucka, Frank T. Bergmann, Claudine Chaouiya, Andreas Dräger, Stefan Hoops, Sarah M. Keating, Matthias König, Nicolas Le Novère andChris J. Myers, Brett G. Olivier, Sven Sahle, James C. Schaff, RahumanSheriff, Lucian P. Smith, Dagmar Waltemath, Darren J. Wilkinson, and andFengkai Zhang. The systems biology markup language (sbml): Languagespecification for level 3 version 2 core. 2019. URL: http://co.mbine.org/specifications/sbml.level-3.version-2.core.release-2.pdf.

[9] RDF-Based semantics. URL: https://www.w3.org/2007/OWL/wiki/RDF-Based_Semantics#Appendix:_Relationship_to_the_Direct_Semantics_.28Informative.29.

[10] Lucian P. Smith, Stefan Hoops, Martin Ginkel, Ion Moraru, Michael Hucka, Andrew Finney, Chris J. Myers, and Wolfram Liebermeister. Hierarchical-model composition. 2013. URL: https://authors.library.caltech.edu/50975/1/sbml-comp-version-1-release-3.pdf.