SBML2SHACL

Edoardo De Matteis
1746561

20 agosto 2020

Indice

1 Introduzione

1.1 SBML

Systems Biology Markup Language (SBML) 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**. La specifica del *Core* che offre le funzionalità base di SBML, in questo documento lo si indica come "SBML 3".
- **Hierarchical Model Composition**. Noto come "comp" (namespace del package e quindi parola chiave necessaria in un file XML) questo package offre la possibilità di include modelli e sottomodelli uno dentro l'altro permettendo allo sviluppatore e aventuali 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".

1.2 RDF

Il **Resource Description Framework (RDF)** 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 desctrittive 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" avremo

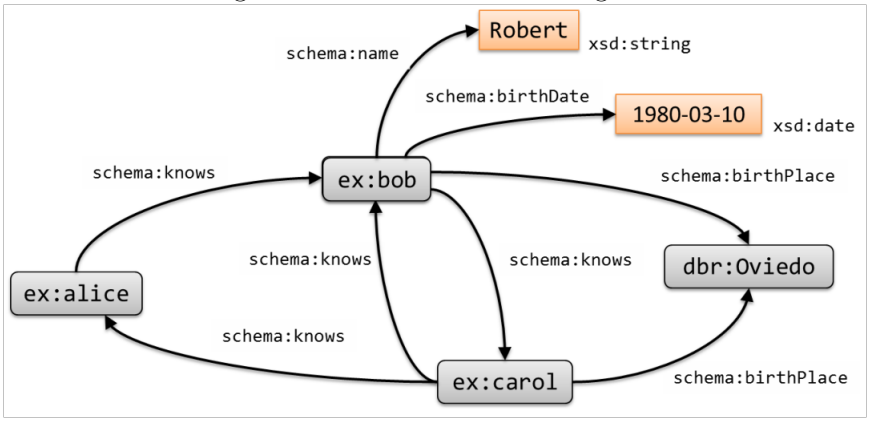
soggetto: "il cielo"

predicato: "è di colore"

oggetto: "blu"

Contrariamente ad 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 un modello RDF è mantenuto come database relazionale (chiamato Triplestores o Quad stores).

Figura 1: Un modello RDF è un grafo.



Esistono vari formati per esprimere il modello astratto RDF, in questo progetto ho scelto di utilizzare solo file Turtle (`.ttl`).

1.3 SHACL

Shapes Constraint Language (SHACL) è una specifica del W3C per la verifica di informazioni sotto forma di grafi dato un insieme di vincoli. 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 qulsiasi formato RDF, in questo progetto si userà Turtle.

SHACL include anche il linguaggio di query **SPARQL SPARQL Protocol and RDF Query Language (SPARQL)**.

1.4 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 dei 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.

2 Modellazione

In questa prima fase si è scelto un sottoinsieme di SBML 3.2 e i costrutti supplementari in Extended SBML e ne è stato formalizzato un modello. Nella tabella 1 vengono descritti i principali costrutti, sono state volontariamente omesse entità 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`.

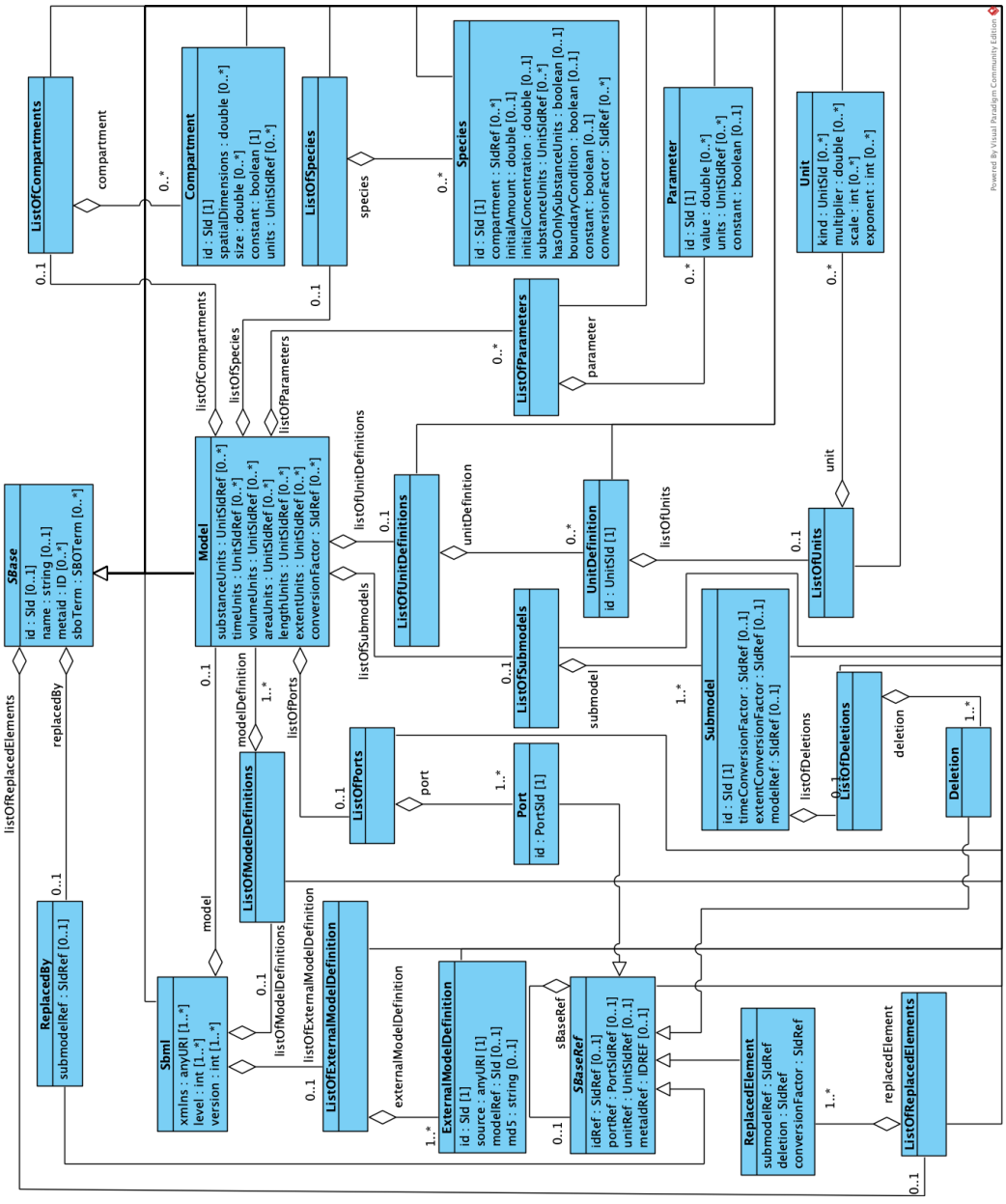
Entità	**Descrizione**
SBML 3.2	
SBase	Classe astratta superclasse di ogni altra classe. Per quanto in SBML sia considerata a tutti gli effetti un tipo, poiché non esistono attributi di tipo SBase a questa classe si riserva un trattamento differente rispetto ai tipi composti quali ID, SId, etc . . . 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 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 e associare un Replacement ad un componente risulta estremamente macchinoso se non impossibile, in `extended_parser.py` questo problema non si presenta perché il file XML viene esplorato 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 1: Modellazione SHACL

Figura 2: Modellazione SBML



3 Parsing da SBML a SHACL

In questa fase sono stati scritti due parser, di questi `extended_parser.py` è la versione finale e corretta. In `parser.py` si riescono a tradurre senza problemi file in SBML 3.2 ma siccome 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 nel secondo parser dato che la struttura XML viene esplorata come un albero, questa seconda versione è inoltre più breve (444 righe anziché 932), più veloce come si può vedere in tabella 3 ed è più facilmente estendibile. Questo parser infatti prende qualsiasi tag e vi associa i rispettivi attributi - che devono essere noti - al proprio valore; ho comunque imposto dei controlli per evitare questo comportamento dato che avrebbe solo rallentato la fase di verifica.

Il calcolo delle performance è basato sull'esecuzione di `test.sh` e si fa uso solo di file SBML 3.2 scaricati da Biomodels. Nella sottocartella `custom`) vi sono file d'esempio forniti direttamente dal W3C in extended SBML e potranno essere usati per il file `extended_parser.py`.

File	**System (s)**	**User (s)**	**Total**	**CPU**
parser.py	52.90	2445.38	42:54.54	97 %
extended_parser.py	38.07	1003.51	17:35.19	98 %

Tabella 3: Performance

Per il corretto funzionamento del codice è necessario il download del package *rdflib*.

4 Parsing da SHACL a SBML

In questa 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. 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 l'argomento della funzione print varia 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 traducendo una tripla `<subject> <predicate> <object>` in etichette `<subject> <predicate> <object/> </predicate> </subject>`.

Nella scrittura del parser è risultato necessario poter riottenere la struttura annidata dell'XML che si era persa nella scrittura delle istanze SHACL, per

farlo ho definito un albero e dei nodi, nella fase di lettura del file si costruisce gradualmente l'albero. Una volta ottenuto l'albero in questione per scrivere il testo XML basta esplorarlo ricorsivamente ricordandosi di chiudere le etichette aperte 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.

Come detto prima non ho usato esempi presi da Biomodels perché i file trovati con extended SBML implementavano la gerarchia con un attributo `outer` piuttosto che con i `submodel` come da documentazione ufficiale.

5 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.

6 Fonti

- The systems biology markup language.
- Sparql query language for rdf, 2008.
- Sparql query results xml format (second edition), 2013.
- Rdf 1.1 concepts and abstract syntax, 2014.
- Shapes constraint language (shacl), 2017.
- Michael Hucka, Frank T. Bergmann, Claudine Chaouiya, Andreas Dräger, Stefan Hoops, Sarah M. Keating, Matthias König, Nicolas Le Novère,Chris J. Myers, Brett G. Olivier, Sven Sahle, James C. Schaff, RahumanSheriff, Lucian P. Smith, Dagmar Waltemath, Darren J. Wilkinson, andFengkai Zhang. The systems biology markup language (sbml): Languagespecification for level 3 version 2 core, 2019.
- Lucian P. Smith, Stefan Hoops, Martin Ginkel, Ion Moraru, Michael Hucka,Andrew Finney, Chris J. Myers, and Wolfram Liebermeister. Hierarchicalmodel composition, 2013.