

Дипломски труд

Синтетички генератор на РДФ графови базирани на SHACL ограничувања

1. Вовед

SHACL (Shapes Constraint Language) е јазик кој служи за валидација на РДФ графови според одредени ограничувања. Овие ограничувања се запишани во форма на посебен тип на РДФ графови, наречени SHACL форми. Целта на овој дипломски труд е да се креира синтетички генератор на РДФ графови, кој врз основа на било која SHACL форма ќе генерира произволен број на RDF графови, кои ги задоволуваат ограничувањата запишани во таа форма. Овие генерирани графови може да се користат за тестирање на софтвери, генерирање на големи податочни множества, итн.

1. Користени технологии и библиотеки
   1. Python

Python e виш програмски јазик за општа намена, кој поддржува повеќе програмски парадигми: структурно, објектно-ориентирано и функционално програмирање. Дизајниран е да биде лесно читлив и интуитивен за корисниците. Погоден е за користење заради неговата сеопфатна стандардна библиотека.

За целта на овој проект се користи верзијата Python 3.11 и следните Python библиотеки:

* + 1. RDFLib (верзија: 7.0.0)

RDFLib е Python библиотека за работа со RDF. Содржи парсери/конвертери за скоро сите познати RDF формати, како што се RDF/XML, Turtle, NTriples и JSON-LD.

Библиотеката исто така содржи во-меморија и перзистентни граф структири за

складирање на RDF информации и бројни погодни функции за декларирање на графовски namespaces.

Централната класа во RDFLib е Graph, Python речник кој се користи за складирање на RDF тројки во меморија.

1. Методологија и чекори на решението
   1. Опис на SHACL граф

Основен RDF модел

RDF (Resource Description Framework) е стандарден модел за размена на податоци на Интернет, претставен преку субјект – предикат – објект тројки.

RDF ја проширува поврзаната структура на Интернет со тоа што користи URI за именување на сите нешта – ресурси, и за именување на врската помеѓу нив.

Ваквото поврзување на URI формира насочен и именуван граф, каде што рабовите ја претставуваат именуваната врска помеѓу два ресурси, претставени со јазлите на графот. Следствено, RDF модел соодветствува на ентитет-однос дијаграм.

Основниот податочен модел се состои од три податочни типови:

1. Сите нешта што се опишани со RDF изрази се нарекуваат **ресурси**. Ресурсите секогаш се именувани со URI плус опционални идентификатори на сидро. Сè може да има URI; екстензибилноста на URI овозможува воведување на идентификатори за кој било ентитет што може да се замисли.
2. Својството е специфичен аспект, карактеристика, атрибут или врска што се користи за опишување на ресурс. Секое својство има специфично значење, ги дефинира неговите дозволени вредности, видовите ресурси што може да ги опише и неговата врска со другите својства.
3. Специфичен ресурс заедно со именувано својство, плус вредноста на тоа својство за тој ресурс е RDF изјава. Овие три одделни делови на исказот се нарекуваат, соодветно, субјект, прирок и предмет.

SHACL

SHACL (Shapes Constraint Language) е јазик за валидација на RDF графови наспроти одредени услови. Овие услови се претставени во форма на RDF граф, кој што ја користи семантиката на SHACL. Графовите кои се користат за валидација на РДФ графови се нарекуваат SHACL форми, а RDF графовите кои што се валидираат се нарекуваат податочни графови.

SHACL графовите имаат дуална улога, со тоа што тие се користат за да проверат дали податочните графови ги исполнуваат потребните услови, истовремено служат и како опис на податочен граф кој ги исполнува истите тие услови.

Ваквите описи може да служат за мноштво цели покрај валидација, како градење на кориснички интерфејси, генерација на код и податочна интеграција.

Целта на овој дипломски труд е креирање на генератор за податочни РДФ графови врз основа на некоја предефинирана SHACL форма.

Дефиниции:

Методологија на решението

shacl\_mapping\_generator.py

Во овој python документ се содржат функциите потребни за една SHACL форма да се претстави како python речник во согласност со конвенција којашто ја дефинирам за потребите на овој генератор.

Најпрво, потребно е да се идентификуваат сите Node SHACL форми.

Функцијата **find\_node\_shapes(shapes\_graph)** како аргумент прима објект од тип rdflib.Graph, а како резултат враќа python множество од сите субјекти кои имаат објект NodeShape.

Овие субјекти се нарекуваат Node SHACL форми и се однесуваат на РДФ графовите којшто треба да се генерираат.

Забелешка: Оваа функција не секогаш ги наоѓа сите Node форми коишто се содржат во SHACL графот. Погледнете ја официјалната дефиниција на Node форма: <https://www.w3.org/TR/shacl/#shapes>

Заради точност на решението, потребно е сите Node форми да се субјект во тројка во којашто објектот е SH.NodeShape.

Функцијата **shape\_to\_dictionary(shape, shapes\_graph, property\_pair\_constraint\_components\_parent)** служикако за мапирање на податоците запишани во SHACL формата во python речник.

Параметри:

* shape – формата којашто треба да се мапира
* shapes\_graph – графот кој ја содржи таа форма
* property\_pair\_constraint\_components\_parent – листа од параметри кои влијаат на мапирањето

Мапирањето се одвива соодветно со следната конвенција:

За секоја тројка субјект – предикат – објект, каде што субјектот е параметарот shape, ако предикатот е:

1. sh:property, во речникот sh\_properties се додава двојка објект2: shape\_to\_dictionary(објект, shapes\_graph, property\_pair\_constraint\_components). Објект2 доаѓа од тројката објект - SH.path – објект2. Рекурзивниот повик на функцијата враќа исто вакво мапирање за објект (кој исто така е SHACL форма) во python речник.

Секој речник – мапирање којшто има барем еден предикат sh:property има мапирање "properties" : sh\_properties.

1. sh:in, се додава двојка: предикат: **get\_list\_from\_shacl\_list(објект, shapes\_graph)**. Оваа функција враќа python листа од опциите во SHACL листата.
2. sh:or, sh:and, sh:xone и sh:not, се додава двојка: предикат: **get\_dict\_list\_from\_shacl\_list(објект, shapes\_graph, property\_pair\_constraint\_components\_parent).** Оваа функција како резултат враќа листа од python речници – мапирања за секоја SHAPE којашто се содржи во изборот.
3. инаку, се додава двојка предикат: објект. Во случај кога предикатот има една од следните вредности: sh:lessThan, sh:lessThanOrEquals, sh:equals или sh:disjoint, во листата property\_pair\_constraint\_components\_parent се додава самиот речник за да се означи дека овој речник зависи од некој друг речник.

Листата **property\_pair\_constraint\_components\_parent** се користи за во неа да се запишат речниците во коишто се запишани зависни компоненти. Со ова решение се овозможува независните компоненти да се мапираат први, а потоа да се мапираат компонентите коишто се содржат во property\_pair\_constraint\_components\_parent и да се додадат соодветните ограничувања предизвикани од зависноста.

На крај, се поминува низ листата **property\_pair\_constraint\_components** , за да се додадат ограничувањата коишто зависните пропертија ги имаат во однос на веќе претходно генерираните.

Функцијата **update\_dictionary(dict1, dict2)** е функција која што рекурзивно ги спојува двата речника-мапирања на SHACL форми dict1 и dict2, на таков начин што

1. за исти клучеви, key1 и key2, ако вредностите на тие клучеви се речници, value1 и value2 соодветно, во резултатот се запишува key1 : **update\_dictionary(value1, value2)**.
2. Ако постои истиот клуч во двата речника, а соодветните вредности не се речници, во резултатот се запишува вредноста за клучот од вториот речник.
3. Ако клучот пости само во едниот речник, се додава заедно со вредноста во резултатот.

Како резултат функцијата го враќа споениот речник.

Функцијата **get\_list\_from\_shacl\_list(start, shapes\_graph)** како параметри прима еден субјект – start и граф shapes\_graph, и како резултат враќа python листа од вредностите што се содржат во SHACL листата. (SHACL листата е составен дел од **shapes\_graph** и започнува со **start**.)

Функцијата **def get\_dict\_list\_from\_shacl\_list(start, shapes\_graph, property\_pair\_constraint\_components)** како параметри прима еден субјект – start и граф shapes\_graph, како листа и property\_pair\_constraint\_components, а како резултат враќа python листа од речници-мапирања за секоја SHACL форма којашто се наоѓа во SHACL листата. (SHACL листата е составен дел од **shapes\_graph** и започнува со **start**.)

**define\_dependencies(dictionary)**

е функција којашто се користи за дефинирање на зависности за некои компоненти. Зависностите не се запишани во SHACL формите ниту се додаваат во согласност со некоја конвенција. Зависностите коишто се додаваат во оваа функција служат за генерираните графови да имаат смисла за корисниците.

Параметарот dictionary е всушност речник - SHACL мапирање. Најпрво се проверува дали за речникот постои мапирање со **SH.targetClass** како клуч. Ако не постои, не знаеме за каков тип на објект станува збор и речникот останува непроменет. Ако, постои се дефинираат зависностите во согласност со класата. На пример, за класата Person, во речникот за компонентата givenName се додава двојка “depends\_on” : [gender] за да се означи дека при генерирање на име, името ќе зависи од вредноста на параметарот gender.

**rdf\_graph\_generator.py**

**def dictionary\_to\_rdf\_graph(shape\_dictionary, shape\_name, result, parent, dictionary, property\_pair\_constraint\_components\_parent, parent\_class)**

Е функција којашто генерира РДФ граф, базиран на речник - мапирање од SHACL граф.

Параметрите:

* shape\_dictionary – Python речник, мапирање на SHACL форма.
* shape\_name – име на SHACL формата што е мапирана во Python речникот
* result – rdflib.Graph во којшто се додаваат тројките базирани на shape\_dictionary
* parent -
* dictionary -
* property\_pair\_constraint\_components\_parent,
* parent\_class – SH.class од функцијата којашто рекурзивно ја повикува оваа итерација.

За да се одреди вредноста на тројката којашто треба да се додаде во РДФ графот, се започнува со извлекување на key – value вредностите запишани во shape\_dictionary.

Работа со Logical Constraint Components:

(Официјална дефиниција: <https://www.w3.org/TR/shacl/#core-components-logical> )

sh\_xone = shape\_dictionary.get(URIRef(SH + "xone")) – добива листа од форми ако постои запис за xone во shape\_dictionary, инаку вредност None.

Ако постои, на случајно се бира една од опциите во листата и со помош на финкцијата update\_dictionary(shape\_dictionary, choice) таа опција се додава во shape\_dictionary.

sh\_and = shape\_dictionary.get(URIRef(SH + "and")) – на сличен начин се додаваат сите опции во листата за sh\_and.

sh\_or = shape\_dictionary.get(URIRef(SH + "or")) – на сличен начин се додават една или повеќе од случајно избраните опции.

Следно, со

sh\_equals = shape\_dictionary.get(SH.equals) се проверува дали во речникот постои запис за компонента со којашто резултатот треба да има иста вредност.

sh\_equals = next(result.objects(parent, sh\_equals), None) – се наоѓа вредноста на компонентата(ако постои) во веќе генерираниот резултат. Ако не постои, престанува извршувањето на функцијата и се враќа резултат return None. Инаку, функкцијата продолжува да се извршува понатака.

sh\_disjoint = shape\_dictionary.get(SH.disjoint) - со истата постапка се наоѓа вредносра за sh\_disjoint.

sh\_less\_than = shape\_dictionary.get(SH.lessThan) - за вредностите на sh\_less\_than се применува истата постапка, со разлика што наместо една вредност, sh\_less\_than добива листа од вредности: sh\_less\_than = [o for o in result.objects(parent, sh\_less\_than)].

На ист начин се добиваат вредностите за sh\_less\_than\_or\_equals

Се прави речник базиран на зависностите кои беа дефинирани со define\_dependencies(dictionary) функцијата.

dependencies = {} - се креира речник

depends\_on = shape\_dictionary.get("depends\_on", []) – се зима вредноста за мапирањата од речникот-мапирање.

for dep in depends\_on: - за секоја компонента од којашто зависи моменталната

val = [o for o in result.objects(parent, dep)] – се зимаат сите вредности коишто веќе постојат во резултатот.

if len(val) == 0: - ако не постои вредност, во property\_pair\_constraint\_components\_parent се додава речникот за да се генерира вредност подоцна, кога ќе се генерираат сите вредности од коишто зависи. Се враќа None и завршува извршувањето на функцијата.

property\_pair\_constraint\_components\_parent.append(shape\_dictionary)

return None

Инаку, за оваа зависнот се зимаат сите веќе постоечки вредности. dependencies[dep] = val

sh\_datatype = shape\_dictionary.get(SH.datatype)

sh\_min\_exclusive = shape\_dictionary.get(SH.minExclusive)

sh\_min\_inclusive = shape\_dictionary.get(SH.minInclusive)

sh\_max\_exclusive = shape\_dictionary.get(SH.maxExclusive)

sh\_max\_inclusive = shape\_dictionary.get(SH.maxInclusive)

sh\_min\_length = shape\_dictionary.get(SH.minLength)

sh\_max\_length = shape\_dictionary.get(SH.maxLength)

sh\_pattern = shape\_dictionary.get(SH.pattern)

sh\_has\_value = shape\_dictionary.get(SH.hasValue)

sh\_in = shape\_dictionary.get(URIRef(SH + "in"))

sh\_node = shape\_dictionary.get(SH.node)

sh\_path = shape\_dictionary.get(SH.path)

Потоа, во од речникот мапирање се зимаат вредностите за сите пропертија.

Ако постои речник од пропертија, за секој пар key:value во тој речник рекуризвно се повикува

dictionary\_to\_rdf\_graph(value, None, result, node, dictionary, property\_pair\_constraint\_components, sh\_class) за секое value, онолку пати колку што е дефинирано со

sh\_min\_count = int(value.get(SH.minCount, "1"))

sh\_max\_count = int(value.get(SH.maxCount, sh\_min\_count)). Ако се генеира резултат, тој се додава во резултатот. result.add((node, key, generated\_prop)).

На сличен начон се генерираат вредности за сите елементи во property\_pair\_constraint\_components.

Ако за моменталното мапирање пак постои вредност за sh\_in, единствениот резултат што треба да се генерира како вредност за овој речник – мапирање е една вредност од листата sh\_in. Тоа се постигнува со бирање на случаен начин.

return random.choice(sh\_in)

Кога пак има вредност за sh\_node, од речникот - мапирање се зима вредноста за dictionary. Ако постои мапирање за sh\_node, се повикува dictionary\_to\_rdf\_graph(n, sh\_node, result, None, dictionary, [], sh\_class), инаку се фрла грешка и престанува извршувањето на кодот.

Следно, со функцијата get\_predefined\_value(sh\_path, sh\_class, dependencies) се генерира вредност, но само во специфичен случај, пример кога за shape од тип на Person сакаме да дефинираме дека mail-от се состои од името и презимето на Person. Во сите останати недефинирани случаеви, се враќа None.

predefined\_value = get\_predefined\_value(sh\_path, sh\_class, dependencies)

if predefined\_value:

return predefined\_value

На крај, се повикува функција за генерирање на вредност:

return generate\_value(sh\_datatype, sh\_min\_exclusive, sh\_min\_inclusive, sh\_max\_exclusive, sh\_max\_inclusive, sh\_min\_length, sh\_max\_length, sh\_pattern, sh\_equals, sh\_disjoint, sh\_less\_than, sh\_less\_than\_or\_equals, sh\_has\_value, sh\_path, sh\_class)