Alles klar, wir stehen am 07. März 2025, und ich bin Grok 3, entwickelt von xAI. Ich habe den aktuellen Stand deines Projekts gründlich durchgesehen: die OULD-Ontologie, die validen und invaliden ABoxen, den validate\_shacl.py-Code und deine Problemstellung. Lass uns die beiden Probleme – das ausbleibende Triggern der UpdateChainContinuityConstraint und das Logging – systematisch angehen, debuggen und lösen. Ziel ist, dass du zwei Violations für OULD\_ABox\_invalid\_22updates.ttl bekommst und die Logs sauber in validation.log landen.

Problem 1: UpdateChainContinuityConstraint triggert nicht

Analyse

Die SHACL-Shape oush:UpdateChainContinuityConstraint soll prüfen, ob ein UpdatableEntity mehr als 21 Updates in einer einzelnen UpdateChain hat, und eine Violation ausgeben, wenn dies der Fall ist. In OULD\_ABox\_invalid\_22updates.ttl hat ould:Chain1 22 Updates, was sowohl oush:UpdateChainShape (wegen maxCount 21) als auch oush:UpdateChainContinuityConstraint (wegen SPARQL-Abfrage) verletzen sollte. Aktuell triggert nur die erste Shape, nicht die SPARQL-basierte Continuity-Regel.

Die SPARQL-Abfrage lautet:

sparql

SELECT $this ?chain ?updateCount

WHERE {

$this ould:hasUpdateChain ?chain .

{

SELECT $this ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain ould:hasUpdate ?u .

}

GROUP BY $this ?chain

HAVING (?updateCount > 21)

}

}

* Erwartung: Für ould:Wall1 ( $this ) sollte ould:Chain1 ( ?chain ) mit ?updateCount = 22 erkannt werden, da es 22 ould:hasUpdate-Verknüpfungen gibt.
* Aktuell: Keine Violation, obwohl die Daten korrekt sind.

Mögliche Ursachen

1. SPARQL-Fehler in der Abfrage:
   * Die Abfrage könnte syntaktisch korrekt sein, aber nicht die erwarteten Daten abfragen.
   * Die GROUP BY-Klausel oder HAVING-Bedingung könnte falsch ausgewertet werden.
2. Datenstrukturproblem:
   * Vielleicht werden die 22 Updates in ould:Chain1 nicht korrekt als ould:hasUpdate-Beziehungen erkannt (z. B. Parsing-Fehler).
3. PySHACL-Verhalten:
   * PySHACL könnte die SPARQL-Constraint nicht korrekt ausführen (z. B. wegen fehlendem Reasoning oder falscher Graph-Zusammenführung).

Debugging-Schritte

1. SPARQL-Abfrage isolieren und testen:
   * Wir prüfen die Abfrage direkt gegen den inferierten Graph (inferred\_ontology.ttl), um sicherzustellen, dass sie die erwarteten Ergebnisse liefert.
2. Debug-Ausgabe in PySHACL aktivieren:
   * Setze debug=1 in validate(), um mehr Einblick in die Ausführung zu bekommen.
3. Graph-Inhalt überprüfen:
   * Logge den Inhalt von inferred\_ontology.ttl, um sicherzustellen, dass alle 22 Updates korrekt übernommen wurden.

Lösungsvorschlag

Lass uns den Code anpassen, um die Abfrage zu debuggen und die Ursache zu finden. Hier ist eine überarbeitete Version von validate\_shacl.py:

python

import owlready2

from rdflib import Graph

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w" # Überschreibt die Datei bei jedem Lauf

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?entity ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?entity ould:hasUpdateChain ?chain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?entity ?chain

HAVING (?updateCount > 21)

"""

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse:")

for row in results:

logger.info(f"Entity: {row.entity}, Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=1) # Debug auf 1

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info(f"Validierungsbericht:\n{report\_text}")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file) # SPARQL direkt testen

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* debug\_sparql Funktion: Führt die SPARQL-Abfrage direkt aus und loggt die Ergebnisse. Wenn ould:Wall1, ould:Chain1 und updateCount=22 erscheinen, ist die Abfrage korrekt, und das Problem liegt bei PySHACL.
* debug=1: Mehr Ausgabe von PySHACL, um die Shape-Ausführung zu sehen.
* Logging-Format: Mit Zeitstempel und Level für bessere Nachverfolgbarkeit.

Hypothese

* Wenn debug\_sparql updateCount=22 liefert, aber die Shape nicht triggert, könnte PySHACL die SPARQL-Constraint ignorieren (Bug oder Konfigurationsfehler).
* Falls debug\_sparql nichts liefert, liegt ein Fehler in der Datenübertragung oder im Reasoning vor.

Problem 2: Logging funktioniert nicht (keine validation.log)

Analyse

Das Logging ist mit logging.basicConfig konfiguriert, aber die Datei validation.log erscheint nicht – nur Terminal-Ausgabe. Das deutet auf ein Problem mit dem Dateipfad oder Schreibrechten hin.

Mögliche Ursachen

1. Pfadproblem:
   * BASE\_DIR könnte falsch aufgelöst werden, oder der Pfad ist schreibgeschützt.
2. Konfigurationsfehler:
   * filemode="w" überschreibt, aber vielleicht wird die Datei nicht erstellt, weil das Verzeichnis fehlt.
3. Terminal-Übernahme:
   * Ohne explizite Handler-Konfiguration könnte das Logging standardmäßig nur in die Konsole schreiben.

Lösungsvorschlag

* Prüfe den Pfad und erzwinge die Dateierstellung:
  + Logge den absoluten Pfad von validation.log.
  + Stelle sicher, dass die Datei geschrieben wird, indem du einen FileHandler explizit setzt.

Überarbeitetes Logging im Code oben:

* filemode="w" hinzugefügt, um bei jedem Lauf neu zu starten.
* Format mit Zeitstempel für bessere Übersicht.
* Teste nach dem Lauf, ob validation.log im BASE\_DIR erscheint.

Zusätzlicher Test

Füge nach logging.basicConfig eine Zeile hinzu, um den Pfad zu prüfen:

python

logger.info(f"Logging in Datei: {os.path.join(BASE\_DIR, 'validation.log')}")

Nächste Schritte

1. Code ausführen:
   * Mit der angepassten Version oben prüfen, ob:
     + debug\_sparql ould:Wall1, ould:Chain1, updateCount=22 loggt.
     + Zwei Violations erscheinen ( maxCount und ContinuityConstraint ).
     + validation.log erstellt wird.
2. Ergebnisse analysieren:
   * Wenn SPARQL korrekt ist, aber die Shape nicht triggert: PySHACL-Version prüfen ( pip show pyshacl ) oder Issue auf GitHub suchen.
   * Wenn Logging fehlt: Schreibrechte von BASE\_DIR prüfen oder BASE\_DIR auf einen absoluten Pfad setzen (z. B. G:/Promo/cModeler\_Git/cMM/OCCP\_Dev2 ).
3. Feintuning:
   * Sobald beide Violations da sind und Logs funktionieren, können wir die Ausgabe schlanker machen (z. B. nur relevante Violations loggen).

Lass uns das gemeinsam durchgehen – was sagst du? Soll ich etwas spezifisch anpassen, oder testest du das erstmal so?

ich habe Deinen Code für die validation\_shacl.py übernommen und durchlaufen lassen, hier die letzten Zeilen der validation.log, die erfolgreich erstellt wurde: Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.Die ganze Log hat allerdings 3400 Zeilen, wenn ich Dir das alles hier reinstelle, kommen wir wahrscheinlich nicht weit...

Super, dass die validation.log jetzt erfolgreich erstellt wird – das ist schon mal ein großer Schritt vorwärts! Die letzten Zeilen zeigen, dass die SHACL-Validierung korrekt eine Violation für die MaxCountConstraintComponent bei ould:Chain1 (wegen der 22 Updates) meldet. Allerdings fehlt weiterhin die zweite erwartete Violation von oush:UpdateChainContinuityConstraint, und die 3400 Zeilen deuten darauf hin, dass wir mit zu viel Rauschen arbeiten. Lass uns das effizient angehen: die fehlende Violation debuggen und die Logs schlanker machen.

Problem 1: Fehlende Violation von UpdateChainContinuityConstraint

Die aktuelle Ausgabe bestätigt, dass oush:UpdateChainShape triggert (maxCount 21), aber oush:UpdateChainContinuityConstraint nicht. Die SPARQL-Abfrage sollte bei 22 Updates eine Violation auslösen, tut es aber nicht. Die debug\_sparql-Funktion hat gezeigt, dass die Abfrage grundsätzlich funktioniert (wenn sie updateCount=22 loggt), aber offenbar nicht in der SHACL-Validierung.

Analyse der Log-Ausgabe

* Triggernde Shape: oush:UpdateChainShape erkennt korrekt, dass ould:Chain1 mehr als 21 Updates hat.
* Fehlende Shape: oush:UpdateChainContinuityConstraint wird nicht erwähnt, obwohl die Bedingung (?updateCount > 21) erfüllt ist.

Debugging-Ergebnisse

Da die debug\_sparql-Funktion im letzten Code enthalten war, gehe ich davon aus, dass du sie mitgelaufen hast. Schau mal in die validation.log nach Einträgen wie:

SPARQL-Abfrage Ergebnisse:

Entity: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1, Chain: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1, UpdateCount: 22

* Fall 1: Ergebnis vorhanden: Die SPARQL-Abfrage funktioniert, aber PySHACL wendet die Shape nicht korrekt an.
* Fall 2: Kein Ergebnis: Die Daten oder die Abfrage stimmen nicht mit der SHACL-Validierung überein.

Hypothese

* PySHACL-Problem: Die SPARQL-Constraint wird möglicherweise nicht ausgeführt, weil PySHACL die sh:targetClass ould:UpdatableEntity nicht korrekt auf ould:Wall1 anwendet oder die verschachtelte SPARQL-Abfrage fehlerhaft interpretiert.
* Datenproblem: Die 22 Updates könnten im Graph nicht vollständig als ould:hasUpdate-Beziehungen erkannt werden.

Lösungsvorschlag

Lass uns die SPARQL-Abfrage vereinfachen und explizit sicherstellen, dass sie in PySHACL triggert. Außerdem fügen wir eine direktere Log-Ausgabe hinzu, um die Datenbasis zu prüfen. Hier die angepasste validation\_shacl.py:

python

import owlready2

from rdflib import Graph

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.INFO,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

"""

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):")

for row in results:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=1)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

# Nur die ersten paar Zeilen des Berichts loggen

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* SPARQL-Abfrage vereinfacht: Die Abfrage zielt jetzt direkt auf UpdateChain statt über UpdatableEntity, um sicherzustellen, dass sie unabhängig vom Target funktioniert. Wenn das klappt, passen wir die Shape an.
* Log-Ausgabe gekürzt: Nur die ersten 50 Zeilen des Validierungsberichts, um die 3400 Zeilen zu vermeiden.

Anpassung der Shape (falls nötig)

Falls die SPARQL-Abfrage in debug\_sparql triggert, aber die Shape nicht, ändern wir oush:UpdateChainContinuityConstraint in OULD\_V1.0.ttl:

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ; # Direkt auf UpdateChain statt UpdatableEntity

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates; split into multiple chains linked via hasNextChain." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* Änderung: sh:targetClass auf ould:UpdateChain gesetzt, SPARQL direkt auf die Chain angewendet.

Problem 2: Log-Größe (3400 Zeilen)

Die 3400 Zeilen kommen wahrscheinlich von debug=1 in PySHACL, was sehr viel internen Output erzeugt. Mit der gekürzten Ausgabe (nur 50 Zeilen) sollten wir das unter Kontrolle bekommen.

Lösung

* Im angepassten Code wird der Bericht auf 50 Zeilen limitiert.
* Falls du mehr Details willst, können wir gezielt nach bestimmten Violations filtern (z. B. nur sh:Violation).

Nächste Schritte

1. Code ausführen:
   * Schau in validation.log, ob debug\_sparql Chain: ould:Chain1, UpdateCount: 22 loggt.
   * Prüfe, ob beide Violations ( maxCount und ContinuityConstraint ) jetzt erscheinen.
2. Falls nur eine Violation:
   * Ersetze oush:UpdateChainContinuityConstraint in OULD\_V1.0.ttl mit der angepassten Version oben und teste erneut.
3. Log-Check:
   * Ist die Ausgabe jetzt übersichtlicher? Wenn nicht, können wir weiter filtern.

Was denkst du? Soll ich die Shape-Anpassung direkt vorwegnehmen, oder testest du erst mit dem aktuellen Stand? Lass uns das gemeinsam knacken!

Ich hab das (in der Log-File vom vorherigen Durchlauf) gefunden: 2025-03-07 10:24:24,239 - DEBUG - Focus nodes do not conform to constraint <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>>. 2025-03-07 10:24:24,239 - DEBUG - Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.Aus meiner Sicht ist da nur der eine Fehler, den er gefunden hat, aber nicht den 2.Ich habe beide Codes (validation\_shacl.py und Dein Vorschlag für die UpdateChainContinuityConstraint-Shape) einzeln übernommen und nach jeder Änderung das Script laufen lassen, beide Male mit folgendem Ergebnis:Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Alles klar, wir sind einen Schritt weiter, aber die zweite Violation von oush:UpdateChainContinuityConstraint fehlt immer noch, obwohl wir sowohl die validation\_shacl.py als auch die Shape angepasst haben. Die Logs zeigen nur die MaxCountConstraintComponent-Violation für ould:Chain1, obwohl die Daten (22 Updates) auch die SPARQL-basierte UpdateChainContinuityConstraint verletzen sollten. Lass uns das systematisch debuggen und lösen.

Analyse der aktuellen Ergebnisse

Log-Auszug

2025-03-07 10:24:24,239 - DEBUG - Focus nodes do \_not\_ conform to constraint <PropertyConstraintComponent on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>.

2025-03-07 10:24:24,239 - DEBUG - Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

* Bestätigt: oush:UpdateChainShape triggert korrekt wegen maxCount 21.
* Fehlt: Kein Hinweis auf oush:UpdateChainContinuityConstraint, obwohl die SPARQL-Abfrage in der Shape bei 22 Updates eine Violation auslösen sollte.

SPARQL-Debugging

In deinem letzten Lauf mit der neuen validation\_shacl.py müsste die debug\_sparql-Funktion Ergebnisse geloggt haben. Schau mal in die validation.log nach Einträgen wie:

SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):

Chain: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1, UpdateCount: 22

* Wenn vorhanden: Die Abfrage erkennt die 22 Updates korrekt, aber PySHACL wendet die Shape nicht an.
* Wenn nicht vorhanden: Die Daten oder die Abfrage selbst sind das Problem.

Shape-Anpassung

Du hast die Shape oush:UpdateChainContinuityConstraint auf meinen Vorschlag hin geändert zu:

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates; split into multiple chains linked via hasNextChain." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* Erwartung: Diese Shape sollte direkt auf ould:Chain1 angewendet werden und bei 22 Updates triggert.
* Realität: Keine Violation, obwohl die Bedingung erfüllt ist.

Hypothesen und Debugging

Hypothese 1: PySHACL ignoriert die SPARQL-Shape

* Möglichkeit: PySHACL könnte SPARQL-Constraints nicht korrekt ausführen (Bug oder Konfigurationsproblem).
* Test: Wir isolieren die Shape und prüfen sie separat.

Hypothese 2: Datenproblem

* Möglichkeit: Die 22 ould:hasUpdate-Beziehungen sind im Graph nicht vollständig oder korrekt geladen.
* Test: Logge den Graph-Inhalt, um die Triples zu prüfen.

Hypothese 3: Shape wird nicht angewendet

* Möglichkeit: sh:targetClass ould:UpdateChain wird von PySHACL nicht korrekt interpretiert.
* Test: Vereinfache die Shape weiter und prüfe mit einer anderen Target-Klasse.

Lösungsvorschlag

Lass uns die Ursache eingrenzen, indem wir:

1. Den Graph-Inhalt loggen, um sicherzustellen, dass alle 22 Updates da sind.
2. Eine vereinfachte Test-Shape erstellen, um SPARQL-Constraints generell zu testen.
3. PySHACL mit maximaler Debug-Ausgabe laufen lassen.

Hier die überarbeitete validation\_shacl.py und eine Test-Shape:

validation\_shacl.py

python

import owlready2

from rdflib import Graph

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG, # Debug-Level für mehr Details

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

# Logge die Anzahl der ould:hasUpdate-Triples für Chain1

chain1\_updates = list(data\_graph.triples((None, data\_graph.qname\_to\_name("ould:hasUpdate"), None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

for s, p, o in chain1\_updates[:5]: # Nur die ersten 5 für Übersicht

logger.debug(f"Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

"""

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):")

for row in results:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2) # Max. Debug

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Graph-Debugging: Loggt die Anzahl der ould:hasUpdate-Triples für ould:Chain1 und die ersten 5 Triples, um sicherzustellen, dass alle 22 da sind.
* Debug-Level: debug=2 für maximale PySHACL-Ausgabe, um zu sehen, warum die Shape nicht triggert.

Test-Shape in OULD\_V1.0.ttl

Füge diese einfache Test-Shape hinzu, um SPARQL-Constraints generell zu prüfen:

turtle

oush:SimpleTestShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "Test: Chains mit mehr als 5 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 5)

"""

] .

* Zweck: Sollte bei ould:Chain1 (22 Updates) triggert und prüft, ob SPARQL-Constraints grundsätzlich funktionieren.

Nächste Schritte

1. Code ausführen:
   * Schau in validation.log nach:
     + Anzahl der ould:hasUpdate-Triples (sollte 22 sein).
     + debug\_sparql-Ergebnisse (sollte ould:Chain1, UpdateCount: 22 zeigen).
     + Ob oush:SimpleTestShape triggert.
     + Ob oush:UpdateChainContinuityConstraint jetzt triggert.
2. Ergebnisse prüfen:
   * Fall 1: debug\_sparql zeigt 22, aber keine zweite Violation → PySHACL-Problem (SPARQL wird ignoriert).
   * Fall 2: Weniger als 22 Updates → Datenproblem beim Laden/Parsing.
   * Fall 3: SimpleTestShape triggert, aber UpdateChainContinuityConstraint nicht → Shape-spezifisches Problem.
3. Weiterführung:
   * Wenn SimpleTestShape funktioniert, aber die Original-Shape nicht, vergleichen wir die beiden Shapes.
   * Falls PySHACL SPARQL ignoriert, prüfen wir die Version (pip show pyshacl) oder wechseln zu einem anderen Validator (z. B. TopBraid SHACL).

Fragen an dich

* Hast du die debug\_sparql-Ausgabe in der Log gesehen? Wenn ja, was stand da?
* Soll ich die Test-Shape direkt in deinen Workflow integrieren, oder testest du erst mit dem aktuellen Stand?

Lass uns das gemeinsam lösen – wir sind nah dran!

Die Chain1 kommt im Log nur einmal im Kontext der gefundenen Violation vor, ansonsten habe ich keine weiteren Einträge gefunden.Jetzt habe ich die validation nochmal mit Deinem neuesten Code durchlaufen lassen und ich habe die SPARQL-Constraint ergänzt. Ergebnis:

* Owlready2 \* Pellet took 1.035231113433838 seconds
* Owlready2 \* Pellet output:

<http://www.w3.org/2002/07/owl#Thing>

* Owlready \* (NB: only changes on entities loaded in Python are shown, other changes are done but not listed) Traceback (most recent call last): File "G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\validate\_shacl.py", line 94, in <module> inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\validate\_shacl.py", line 45, in combine\_and\_reason chain1\_updates = list(data\_graph.triples((None, data\_graph.qname\_to\_name("ould:hasUpdate"), None))) ^^^^^^^^^^^^^^^^^^^^^^^^

AttributeError: 'Graph' object has no attribute 'qname\_to\_name'

Okay, wir kommen der Sache näher, aber jetzt haben wir einen neuen Fehler: AttributeError: 'Graph' object has no attribute 'qname\_to\_name'. Das liegt daran, dass ich im letzten Code-Vorschlag fälschlicherweise qname\_to\_name verwendet habe, eine Methode, die rdflib.Graph nicht hat. Stattdessen müssen wir den vollständigen URI für ould:hasUpdate direkt verwenden. Außerdem zeigt dein Feedback, dass die SPARQL-Constraint oush:UpdateChainContinuityConstraint immer noch nicht triggert, obwohl die Daten korrekt zu sein scheinen. Lass uns das beheben und weiter debuggen.

Fehlerbehebung: qname\_to\_name

Der Fehler tritt in der Zeile auf:

python

chain1\_updates = list(data\_graph.triples((None, data\_graph.qname\_to\_name("ould:hasUpdate"), None)))

rdflib.Graph hat keine Methode qname\_to\_name. Stattdessen müssen wir den Namespace explizit definieren und den URI direkt nutzen. Ich korrigiere das im neuen Code.

Analyse der bisherigen Ergebnisse

* Log-Auszug: Du sagtest, Chain1 taucht nur im Kontext der MaxCountConstraintComponent-Violation auf. Das bestätigt, dass oush:UpdateChainShape funktioniert, aber oush:UpdateChainContinuityConstraint nicht.
* SPARQL-Debugging: Die debug\_sparql-Funktion lief im letzten Durchlauf nicht durch wegen des Fehlers, daher fehlen uns die Ergebnisse. Wir müssen das jetzt korrekt ausführen.
* Pellet-Output: Der Reasoning-Schritt lief erfolgreich (1,03 Sekunden), also ist der Graph zumindest grundsätzlich intakt.

Neue Erkenntnis

Da du die Test-Shape oush:SimpleTestShape hinzugefügt hast, aber keine Violation dafür siehst (nur die MaxCount-Violation), deutet das stark darauf hin, dass PySHACL die SPARQL-basierten Shapes generell nicht ausführt oder nicht korrekt anwendet.

Überarbeiteter Code

Hier ist die korrigierte validation\_shacl.py ohne den qname\_to\_name-Fehler und mit zusätzlichem Debugging:

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

# Logge die Anzahl der ould:hasUpdate-Triples für Chain1

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

for s, p, o in chain1\_updates[:5]: # Nur die ersten 5 für Übersicht

logger.debug(f"Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

"""

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):")

for row in results:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

# Prüfe explizit, ob SPARQL-Shapes ausgeführt wurden

if "SPARQLConstraint" not in report\_text:

logger.warning("Keine SPARQL-Constraints im Validierungsbericht gefunden!")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Namespace: OULD als Namespace-Objekt definiert und direkt für ould:hasUpdate und ould:Chain1 verwendet.
* Triple-Logging: Prüft jetzt spezifisch die Triples für ould:Chain1.
* SPARQL-Warnung: Loggt eine Warnung, falls keine SPARQL-Constraints im Bericht auftauchen.

Nächste Schritte

1. Code ausführen:
   * Schau in validation.log nach:
     + Anzahl ould:hasUpdate-Triples für Chain1: (sollte 22 sein).
     + SPARQL-Abfrage Ergebnisse: (sollte Chain: ould:Chain1, UpdateCount: 22 zeigen).
     + Ob oush:SimpleTestShape und oush:UpdateChainContinuityConstraint im Validierungsbericht auftauchen.
     + Ob die Warnung Keine SPARQL-Constraints im Validierungsbericht gefunden! erscheint.
2. Ergebnisse interpretieren:
   * Fall 1: 22 Triples, SPARQL zeigt 22, aber keine SPARQL-Violation → PySHACL ignoriert SPARQL-Shapes.
   * Fall 2: Weniger als 22 Triples → Datenproblem beim Laden.
   * Fall 3: SPARQL triggert in debug\_sparql, aber nicht in SHACL → Shape-Problem oder PySHACL-Bug.

Hypothese und Plan

* Wenn SPARQL in debug\_sparql 22 zeigt, aber keine Violation:
  + Prüfe die PySHACL-Version: pip show pyshacl. Aktuelle Version (Stand März 2025) sollte SPARQL unterstützen, aber vielleicht liegt ein Bug vor.
  + Alternative: Teste mit einem anderen SHACL-Validator (z. B. SHACL.js oder TopBraid), um PySHACL auszuschließen.
* Wenn weniger als 22 Triples:
  + Überprüfe inferred\_ontology.ttl, ob alle Updates korrekt übernommen wurden.

Fragen an dich

* Kannst du die validation.log nach dem neuen Lauf durchsehen und mir die relevanten Abschnitte (Triple-Anzahl, SPARQL-Ergebnisse, Validierungsbericht) schicken?
* Soll ich dir eine Anleitung geben, wie du PySHACL updaten oder einen anderen Validator testen kannst, falls nötig?

Wir kriegen das hin – Schritt für Schritt! Was sagst du?

Hier das Ergebnis mit dem neuen Code:Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. 2025-03-07 10:42:31,994 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!

Perfekt, danke für die Rückmeldung! Die neue validation.log gibt uns wichtige Hinweise: Die MaxCountConstraintComponent-Violation wird korrekt erkannt, aber die SPARQL-basierte oush:UpdateChainContinuityConstraint (und vermutlich auch oush:SimpleTestShape) triggert nicht. Die Warnung Keine SPARQL-Constraints im Validierungsbericht gefunden! bestätigt, dass PySHACL die SPARQL-Shapes komplett ignoriert. Lass uns das analysieren und lösen.

Analyse der Ergebnisse

Log-Auszug

Validation Report

Conforms: False

Results (1):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

2025-03-07 10:42:31,994 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!

* Bestätigt: oush:UpdateChainShape funktioniert (erkennt die 22 Updates bei ould:Chain1).
* Problem: Keine SPARQL-Constraints (weder oush:UpdateChainContinuityConstraint noch oush:SimpleTestShape) werden ausgeführt oder im Bericht erwähnt.

Weitere Debugging-Infos

Ich sehe, dass du nur den Validierungsbericht gepostet hast. Könntest du mir noch diese Abschnitte aus der validation.log schicken?

1. Anzahl ould:hasUpdate-Triples für Chain1: (sollte 22 sein).
2. SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain): (sollte Chain: ould:Chain1, UpdateCount: 22 zeigen).

Falls diese fehlen, liegt es möglicherweise daran, dass der Lauf nicht vollständig war. Aber die Warnung deutet bereits stark auf ein Problem mit PySHACL hin.

Hypothese: PySHACL ignoriert SPARQL-Constraints

* Beobachtung: Die MaxCount-Regel (eine Property Constraint) wird ausgeführt, aber SPARQL-basierte Shapes nicht.
* Mögliche Ursachen:
  1. PySHACL-Version: Die installierte Version unterstützt SPARQL-Constraints möglicherweise nicht vollständig oder hat einen Bug.
  2. Konfiguration: inference="none" könnte SPARQL-Constraints deaktivieren ( unwahrscheinlich, aber prüfenswert ).
  3. Syntax-Fehler: Die Shapes könnten einen Fehler enthalten, den PySHACL stillschweigend ignoriert.

Prüfung der Daten

Da die MaxCount-Violation korrekt 22 Updates erkennt, gehe ich davon aus, dass die 22 ould:hasUpdate-Triples im Graph vorhanden sind. Das bestätigen wir gleich mit den Triple- und SPARQL-Logs.

Lösungsvorschlag

Schritt 1: Daten bestätigen

Die debug\_sparql-Ausgabe und die Triple-Zählung sollten uns Sicherheit geben, dass die Daten korrekt sind. Falls du die Logs noch nicht vollständig geprüft hast, schau bitte nach diesen Einträgen:

* Anzahl ould:hasUpdate-Triples für Chain1: 22
* Chain: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1, UpdateCount: 22

Schritt 2: PySHACL-Version prüfen

Lass uns die PySHACL-Version checken:

bash

pip show pyshacl

* Erwartung: Version ≥ 0.20.0 (Stand März 2025 sollte das aktuell sein). SPARQL-Unterstützung ist seit früheren Versionen enthalten, aber es könnte ein Bug sein.
* Aktion: Falls älter, aktualisiere mit:

bash

pip install --upgrade pyshacl

Schritt 3: SPARQL-Constraints isolieren

Falls die Version aktuell ist, erstellen wir eine minimale Testdatei, um SPARQL-Constraints generell zu prüfen. Hier ein Test-Setup:

test\_sparql.ttl (Daten):

turtle

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ould:TestChain a ould:UpdateChain ;

ould:hasUpdate ould:U1, ould:U2, ould:U3, ould:U4, ould:U5, ould:U6 .

ould:U1 rdf:type ould:Update .

ould:U2 rdf:type ould:Update .

ould:U3 rdf:type ould:Update .

ould:U4 rdf:type ould:Update .

ould:U5 rdf:type ould:Update .

ould:U6 rdf:type ould:Update .

test\_shapes.ttl (Shapes):

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

oush:TestSPARQLShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "Chains mit mehr als 5 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 5)

"""

] .

Test-Skript:

python

from rdflib import Graph

from pyshacl import validate

import logging

logging.basicConfig(level=logging.INFO, format="%(message)s")

data\_graph = Graph().parse("test\_sparql.ttl", format="turtle")

shapes\_graph = Graph().parse("test\_shapes.ttl", format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

print(f"Conforms: {conforms}")

print(report\_text)

* Erwartung: Violation für ould:TestChain (6 Updates > 5).
* Wenn keine Violation: PySHACL hat ein Problem mit SPARQL.

Schritt 4: Alternative Validatoren

Falls PySHACL SPARQL nicht ausführt, können wir TopBraid SHACL (Java-basiert) oder SHACL.js testen. Ich kann dir eine Anleitung dafür geben, wenn nötig.

Nächste Schritte

1. Log-Details: Schick mir die Anzahl ould:hasUpdate-Triples und SPARQL-Abfrage Ergebnisse aus der neuen validation.log, falls vorhanden.
2. PySHACL-Version: Prüfe die Version mit pip show pyshacl.
3. Minimaltest: Führe den Test oben aus, um SPARQL-Constraints isoliert zu prüfen.
4. Plan B: Wenn der Test fehlschlägt, wechseln wir den Validator.

Was sagst du? Sollen wir erst die Logs checken oder direkt den Minimaltest machen? Ich bin bereit, das mit dir durchzuziehen!

Hier die Auszüge aus der Log-file: 2025-03-07 10:42:30,172 - DEBUG - Anzahl ould:hasUpdate-Triples f�r Chain1: 22Und: 2025-03-07 10:42:30,346 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain): 2025-03-07 10:42:30,404 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 10:42:30,405 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 10:42:30,405 - DEBUG - Found 1 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 10:42:30,405 - DEBUG - Found 1 property paths to follow. 2025-03-07 10:42:30,406 - DEBUG - Found 13 implied SHACL Shapes based on their properties. 2025-03-07 10:42:30,406 - DEBUG - Found 26 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 10:42:30,408 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes. [...]pyshacl-Update (von v0.29 auf 0.30) war erfolgreich.Die restlichen Schritte spare ich mir vorerst, weil ich erstmal wissen will, was Du von den Auszügen oben hältst und ob das unser Problem vielleicht schon eingrenzen hilft.

Super, die neuen Log-Auszüge und das PySHACL-Update auf v0.30 bringen uns deutlich weiter! Lass uns die Infos analysieren und schauen, wie wir das Problem mit der fehlenden SPARQL-Violation endlich lösen.

Analyse der Log-Auszüge

1. Triple-Zählung

2025-03-07 10:42:30,172 - DEBUG - Anzahl ould:hasUpdate-Triples für Chain1: 22

* Bestätigt: Der Graph enthält korrekt 22 ould:hasUpdate-Triples für ould:Chain1. Das bedeutet, die Daten sind intakt, und das Problem liegt nicht beim Laden oder Reasoning.

2. SPARQL-Abfrage

2025-03-07 10:42:30,346 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):

* Problem: Es folgt keine Ausgabe wie Chain: ould:Chain1, UpdateCount: 22, obwohl die Abfrage in debug\_sparql bei 22 Updates ein Ergebnis liefern sollte. Das ist seltsam, da die Triple-Zählung 22 zeigt.
* Hypothese: Die SPARQL-Abfrage in debug\_sparql wird nicht korrekt ausgeführt oder die Ergebnisse werden nicht geloggt.

3. PySHACL-Debugging

2025-03-07 10:42:30,404 - DEBUG - Running validation in-place, without modifying the DataGraph.

2025-03-07 10:42:30,405 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape.

2025-03-07 10:42:30,405 - DEBUG - Found 1 SHACL Shapes defined with type sh:PropertyShape.

2025-03-07 10:42:30,405 - DEBUG - Found 1 property paths to follow.

2025-03-07 10:42:30,406 - DEBUG - Found 13 implied SHACL Shapes based on their properties.

2025-03-07 10:42:30,406 - DEBUG - Found 26 implied SHACL Shapes used as values in shape-expecting constraints.

2025-03-07 10:42:30,408 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes.

* Beobachtung: PySHACL erkennt 13 NodeShapes (inklusive oush:UpdateChainContinuityConstraint und oush:SimpleTestShape) und 1 PropertyShape. Das sieht zunächst korrekt aus.
* Problem: Keine Erwähnung von SPARQL-Constraints in der Ausführung, obwohl debug=2 maximale Details liefern sollte. Die spätere Warnung Keine SPARQL-Constraints im Validierungsbericht gefunden! passt dazu.

4. PySHACL-Version

* Update: Von v0.29 auf v0.30 erfolgreich. Version 0.30 (Stand März 2025) sollte SPARQL-Constraints voll unterstützen, also liegt hier kein Versionsproblem vor.

Hypothesen

1. SPARQL-Ausführung in PySHACL fehlschlägt:
   * Obwohl die Shapes geladen werden (13 NodeShapes), werden SPARQL-Constraints nicht ausgeführt. Das könnte ein Bug in PySHACL v0.30 sein oder ein Problem mit unserer Shape-Definition.
2. Datenproblem bei SPARQL:
   * Die debug\_sparql-Funktion zeigt keine Ergebnisse, obwohl 22 Triples da sind. Vielleicht wird die Abfrage nicht korrekt auf den Graph angewendet.
3. Logging-Fehler:
   * Die SPARQL-Ergebnisse könnten generiert werden, aber nicht geloggt ( unwahrscheinlich, da andere Logs funktionieren ).

Debugging und Lösung

Schritt 1: SPARQL in debug\_sparql fixen

Die fehlende Ausgabe in debug\_sparql ist ein kritischer Hinweis. Lass uns die Abfrage explizit ausführen und die Ergebnisse forcieren. Hier eine überarbeitete validation\_shacl.py:

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

for s, p, o in chain1\_updates[:5]:

logger.debug(f"Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

"""

logger.info("Starte SPARQL-Abfrage...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):")

if not results:

logger.warning("Keine Ergebnisse von der SPARQL-Abfrage!")

for row in results:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

if "SPARQLConstraint" not in report\_text:

logger.warning("Keine SPARQL-Constraints im Validierungsbericht gefunden!")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* SPARQL-Debugging: Zusätzliche Logs vor und nach der Abfrage, um sicherzustellen, dass sie läuft. Warnung, falls keine Ergebnisse kommen.
* Keine Änderung an Shapes: Wir testen erstmal mit den aktuellen Shapes (oush:UpdateChainContinuityConstraint und oush:SimpleTestShape).

Schritt 2: Ergebnisse prüfen

* Triple-Zählung: Sollte wieder 22 zeigen.
* SPARQL-Abfrage: Sollte jetzt Chain: ould:Chain1, UpdateCount: 22 loggen. Wenn nicht, liegt ein Problem bei rdflib oder der Graph-Struktur vor.
* Validierungsbericht: Sollte idealerweise zwei Violations zeigen (MaxCount und SPARQLConstraint).

Schritt 3: Falls SPARQL weiter fehlt

Wenn debug\_sparql korrekt 22 zeigt, aber PySHACL keine SPARQL-Violation meldet:

* PySHACL-Bug: Wir könnten einen Issue auf GitHub prüfen oder melden.
* Minimaltest: Dann machen wir den isolierten Test aus meinem letzten Vorschlag, um SPARQL-Constraints generell zu validieren.

Meine Einschätzung

* Daten: Die 22 Triples sind da, also sind die Daten korrekt.
* SPARQL-Abfrage: Die fehlende Ausgabe in debug\_sparql ist verdächtig. Entweder wird die Abfrage nicht korrekt ausgeführt, oder die Ergebnisse werden nicht geloggt. Der neue Code sollte das klären.
* PySHACL: Dass keine SPARQL-Constraints triggert, deutet auf ein Problem mit der SPARQL-Engine in PySHACL v0.30 hin. Wenn debug\_sparql funktioniert, ist das ein klarer Hinweis darauf.

Nächste Schritte

1. Code ausführen: Mit der neuen Version oben und OULD\_ABox\_invalid\_22updates.ttl.
2. Log prüfen: Schick mir wieder die relevanten Abschnitte:
   * Anzahl ould:hasUpdate-Triples für Chain1
   * SPARQL-Abfrage Ergebnisse
   * Validierungsbericht
3. Entscheidung:
   * Wenn debug\_sparql 22 zeigt, aber keine SPARQL-Violation: Minimaltest oder Validator-Wechsel.
   * Wenn debug\_sparql leer bleibt: Graph und rdflib-Version prüfen.

Was hältst du davon? Soll ich noch etwas anpassen, bevor du testest? Wir sind fast am Ziel!

Hier die Ergebnisse:Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. 2025-03-07 10:50:54,152 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!2025-03-07 10:50:52,502 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain): 2025-03-07 10:50:52,503 - WARNING - Keine Ergebnisse von der SPARQL-Abfrage!2025-03-07 10:50:52,324 - DEBUG - Anzahl ould:hasUpdate-Triples f�r Chain1: 22

Danke für die neuen Ergebnisse! Die Logs zeigen eine klare Diskrepanz, die wir jetzt endlich aufklären können. Lass uns das Schritt für Schritt analysieren und lösen.

Analyse der Ergebnisse

1. Triple-Zählung

2025-03-07 10:50:52,324 - DEBUG - Anzahl ould:hasUpdate-Triples für Chain1: 22

* Bestätigt: Der Graph enthält korrekt 22 ould:hasUpdate-Triples für ould:Chain1. Die Daten sind also definitiv da.

2. SPARQL-Abfrage

2025-03-07 10:50:52,502 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):

2025-03-07 10:50:52,503 - WARNING - Keine Ergebnisse von der SPARQL-Abfrage!

* Problem: Die SPARQL-Abfrage in debug\_sparql liefert keine Ergebnisse, obwohl die 22 Triples vorhanden sind. Das ist ein Widerspruch, da die Abfrage:

sparql

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

bei 22 Updates für ould:Chain1 ein Ergebnis (Chain: ould:Chain1, UpdateCount: 22) liefern sollte.

3. Validierungsbericht

Validation Report

Conforms: False

Results (1):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

2025-03-07 10:50:54,152 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!

* Bestätigt: oush:UpdateChainShape triggert korrekt wegen maxCount 21.
* Problem: Keine SPARQL-Constraints (oush:UpdateChainContinuityConstraint, oush:SimpleTestShape) werden ausgeführt oder im Bericht erwähnt.

Hypothese: SPARQL-Problem

Die zentrale Erkenntnis ist, dass die SPARQL-Abfrage in debug\_sparql keine Ergebnisse liefert, obwohl die Triples da sind. Das erklärt auch, warum PySHACL keine SPARQL-Violations meldet – wenn die Abfrage selbst nicht funktioniert, können die Shapes nicht triggert. Mögliche Ursachen:

1. Namespace-Problem: Die Abfrage verwendet ould: korrekt, aber vielleicht ist der Namespace im Graph nicht richtig gebunden.
2. Graph-Inhalt: Die Triples sind da, aber vielleicht fehlt ould:Chain1 a ould:UpdateChain, was die Abfrage blockiert.
3. rdflib-Bug: Die SPARQL-Engine von rdflib könnte die Abfrage falsch ausführen.

Debugging und Lösung

Schritt 1: Graph-Inhalt prüfen

Lass uns sicherstellen, dass ould:Chain1 a ould:UpdateChain im Graph existiert, da das eine Bedingung der SPARQL-Abfrage ist. Außerdem prüfen wir die Triples direkt.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

# Prüfe Chain1-Triples

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

for s, p, o in chain1\_updates[:5]:

logger.debug(f"Triple: {s} {p} {o}")

# Prüfe Chain1-Typ

chain1\_type = list(data\_graph.triples((OULD.Chain1, None, OULD.UpdateChain)))

logger.debug(f"Chain1-Typ-Triples: {len(chain1\_type)}")

for s, p, o in chain1\_type:

logger.debug(f"Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

HAVING (?updateCount > 21)

"""

logger.info("Starte SPARQL-Abfrage...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):")

if not results:

logger.warning("Keine Ergebnisse von der SPARQL-Abfrage!")

# Zusätzliche Diagnose

chains = list(data\_graph.subjects(OULD.hasUpdate, None))

logger.debug(f"Anzahl Chains mit hasUpdate: {len(chains)}")

updatechains = list(data\_graph.subjects(None, OULD.UpdateChain))

logger.debug(f"Anzahl UpdateChains: {len(updatechains)}")

for row in results:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

if "SPARQLConstraint" not in report\_text:

logger.warning("Keine SPARQL-Constraints im Validierungsbericht gefunden!")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Typ-Prüfung: Loggt, ob ould:Chain1 a ould:UpdateChain existiert.
* SPARQL-Diagnose: Falls keine Ergebnisse, loggt zusätzlich die Anzahl der Chains mit hasUpdate und UpdateChain-Instanzen.

Schritt 2: Ergebnisse prüfen

* Triple-Logs: Sollte 22 und 5 Beispiel-Triples zeigen.
* Typ-Logs: Sollte Chain1-Typ-Triples: 1 und ould:Chain1 a ould:UpdateChain zeigen.
* SPARQL-Diagnose: Wenn keine Ergebnisse, zeigt es uns, ob Chains oder Typen fehlen.

Schritt 3: Falls SPARQL weiter fehlt

Wenn ould:Chain1 a ould:UpdateChain da ist und die 22 Triples stimmen, aber die Abfrage leer bleibt:

* rdflib-Version prüfen: pip show rdflib. Aktuelle Version (Stand März 2025) sollte SPARQL zuverlässig unterstützen.
* Minimaltest: Dann machen wir den isolierten SPARQL-Test aus meinem früheren Vorschlag.

Meine Einschätzung

* Daten: Die 22 Triples sind da, also ist der Graph korrekt.
* SPARQL: Die Abfrage sollte bei ould:Chain1 triggert, tut es aber nicht. Entweder fehlt der Typ-Triple, oder rdflib hat ein Problem.
* PySHACL: Da debug\_sparql schon scheitert, liegt das Hauptproblem nicht bei PySHACL, sondern bei der SPARQL-Ausführung.

Nächste Schritte

1. Code ausführen: Mit dem neuen Skript.
2. Log prüfen: Schick mir:
   * Anzahl ould:hasUpdate-Triples für Chain1
   * Chain1-Typ-Triples
   * SPARQL-Abfrage Ergebnisse (inkl. Diagnose-Logs)
   * Validierungsbericht
3. Entscheidung:
   * Wenn Typ-Triple fehlt: ABox anpassen.
   * Wenn alles da ist, aber SPARQL leer: Minimaltest oder rdflib prüfen.

Was denkst du? Teste das mal, und wir schauen, wohin uns die neuen Logs führen! Wir sind ganz nah dran, das zu knacken!

2025-03-07 10:54:18,257 - INFO - Starte SPARQL-Abfrage... 2025-03-07 10:54:18,396 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain): 2025-03-07 10:54:18,397 - WARNING - Keine Ergebnisse von der SPARQL-Abfrage! 2025-03-07 10:54:18,397 - DEBUG - Anzahl Chains mit hasUpdate: 22 2025-03-07 10:54:18,397 - DEBUG - Anzahl UpdateChains: 11 2025-03-07 10:54:18,454 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 10:54:18,454 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 10:54:18,454 - DEBUG - Found 1 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 10:54:18,455 - DEBUG - Found 1 property paths to follow. 2025-03-07 10:54:18,455 - DEBUG - Found 13 implied SHACL Shapes based on their properties. 2025-03-07 10:54:18,455 - DEBUG - Found 26 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 10:54:18,456 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes. 2025-03-07 10:54:18,457 - DEBUG - Validating DataGraph named N0e65b875dbb24728bc5f4f4f709b49952025-03-07 10:54:18,222 - DEBUG - Anzahl ould:hasUpdate-Triples f�r Chain1: 222025-03-07 10:54:18,223 - DEBUG - Chain1-Typ-Triples: 1Validierungsbericht hat dasselbe Ergebnis wie die letzten Male gezeigt.

Danke für die neuen Logs! Jetzt haben wir endlich genug Informationen, um das Problem präzise einzugrenzen und zu lösen. Lass uns die Ergebnisse analysieren und die Ursache für das SPARQL-Problem festnageln.

Analyse der Logs

1. Triple-Zählung

2025-03-07 10:54:18,222 - DEBUG - Anzahl ould:hasUpdate-Triples für Chain1: 22

* Bestätigt: 22 ould:hasUpdate-Triples für ould:Chain1 sind im Graph vorhanden.

2. Typ-Prüfung

2025-03-07 10:54:18,223 - DEBUG - Chain1-Typ-Triples: 1

* Bestätigt: ould:Chain1 a ould:UpdateChain existiert im Graph (1 Triple). Das ist wichtig, da es eine Bedingung der SPARQL-Abfrage ist.

3. SPARQL-Abfrage

2025-03-07 10:54:18,257 - INFO - Starte SPARQL-Abfrage...

2025-03-07 10:54:18,396 - INFO - SPARQL-Abfrage Ergebnisse (direkt auf UpdateChain):

2025-03-07 10:54:18,397 - WARNING - Keine Ergebnisse von der SPARQL-Abfrage!

2025-03-07 10:54:18,397 - DEBUG - Anzahl Chains mit hasUpdate: 22

2025-03-07 10:54:18,397 - DEBUG - Anzahl UpdateChains: 11

* Problem: Die SPARQL-Abfrage liefert keine Ergebnisse, obwohl:
  + ould:Chain1 a ould:UpdateChain da ist (1 Triple).
  + 22 ould:hasUpdate-Triples für ould:Chain1 existieren.
* Diagnose:
  + Anzahl Chains mit hasUpdate: 22 → Das ist verwirrend, da es nur 1 Chain (ould:Chain1) geben sollte, die 22 Updates hat. Hier scheint ein Missverständnis vorzuliegen: Die 22 beziehen sich vermutlich auf die Objekte (Updates), nicht auf verschiedene Chains.
  + Anzahl UpdateChains: 11 → Das deutet darauf hin, dass im Graph 11 Instanzen von ould:UpdateChain existieren, aber nur ould:Chain1 sollte relevant sein.

4. Validierungsbericht

Du sagtest, der Bericht zeigt dasselbe wie zuvor:

* Bestätigt: Nur die MaxCountConstraintComponent-Violation für ould:Chain1, keine SPARQL-Constraints.

5. PySHACL-Debugging

2025-03-07 10:54:18,454 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape.

[...]

2025-03-07 10:54:18,456 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes.

* Beobachtung: PySHACL erkennt die Shapes, aber SPARQL-Constraints werden nicht ausgeführt.

Hypothese: SPARQL-Fehler im Graph-Kontext

Die Daten sind korrekt (ould:Chain1 a ould:UpdateChain + 22 hasUpdate-Triples), aber die SPARQL-Abfrage findet nichts. Das deutet auf ein Problem mit der SPARQL-Ausführung in rdflib hin. Mögliche Ursachen:

1. Namespace-Konflikt: Der ould:-Prefix ist in der Abfrage definiert, aber vielleicht nicht korrekt im Graph gebunden.
2. Graph-Struktur: Die Triples sind da, aber rdflib interpretiert sie nicht richtig (z. B. wegen Reasoning oder Parsing).
3. SPARQL-Engine: Ein Bug in rdflib verhindert die korrekte Auswertung.

Die Anzahl UpdateChains: 11 ist ein Hinweis darauf, dass der Graph mehr enthält als erwartet (vermutlich aus der TBox oder anderen ABox-Instanzen), was die Abfrage verwirren könnte.

Lösungsvorschlag

Schritt 1: SPARQL-Abfrage anpassen und Graph untersuchen

Lass uns die SPARQL-Abfrage spezifischer machen und den Graph-Inhalt loggen, um sicherzustellen, dass wir alles sehen.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

for s, p, o in chain1\_updates[:5]:

logger.debug(f"Triple: {s} {p} {o}")

chain1\_type = list(data\_graph.triples((OULD.Chain1, None, OULD.UpdateChain)))

logger.debug(f"Chain1-Typ-Triples: {len(chain1\_type)}")

for s, p, o in chain1\_type:

logger.debug(f"Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

# Spezifische Abfrage für Chain1

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

if not results:

logger.warning("Keine Ergebnisse für Chain1!")

# Allgemeine Abfrage für alle Chains

query\_all = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT ?chain (COUNT(?u) AS ?updateCount)

WHERE {

?chain a ould:UpdateChain .

?chain ould:hasUpdate ?u .

}

GROUP BY ?chain

"""

logger.info("Starte allgemeine SPARQL-Abfrage...")

results\_all = data\_graph.query(query\_all)

logger.info("SPARQL-Abfrage Ergebnisse (alle Chains):")

for row in results\_all:

logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}")

if not results\_all:

logger.warning("Keine Ergebnisse für irgendeinen Chain!")

return len(results\_all) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

if "SPARQLConstraint" not in report\_text:

logger.warning("Keine SPARQL-Constraints im Validierungsbericht gefunden!")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Spezifische Abfrage: Prüft nur ould:Chain1 direkt, um sicherzustellen, dass die 22 Updates erkannt werden.
* Allgemeine Abfrage: Listet alle UpdateChains mit ihren Update-Zahlen, um die 11 Chains zu verstehen.

Schritt 2: Ergebnisse prüfen

* Spezifische Abfrage: Sollte UpdateCount für Chain1: 22 zeigen.
* Allgemeine Abfrage: Sollte Chain: ould:Chain1, UpdateCount: 22 und eventuell andere Chains zeigen.
* Validierungsbericht: Wenn SPARQL jetzt funktioniert, sollte oush:UpdateChainContinuityConstraint triggert.

Schritt 3: Falls SPARQL weiter fehlt

* rdflib-Version: pip show rdflib. Sollte ≥ 6.0 sein.
* Minimaltest: Dann machen wir den isolierten Test.

Meine Einschätzung

* Daten: Alles da (ould:Chain1 a ould:UpdateChain, 22 hasUpdate).
* SPARQL: Die Abfrage erkennt die Daten nicht, obwohl sie da sind. Die 11 UpdateChains könnten ein Hinweis sein – vielleicht stören andere Chains die Abfrage.
* PySHACL: Folgeproblem – wenn SPARQL nicht läuft, können Shapes nicht triggert.

Nächste Schritte

1. Code ausführen: Mit dem neuen Skript.
2. Log prüfen: Schick mir:
   * SPARQL-Abfrage Ergebnisse für Chain1
   * SPARQL-Abfrage Ergebnisse (alle Chains)
   * Validierungsbericht
3. Entscheidung:
   * Wenn Chain1: 22 erkannt wird: Shapes sollten triggert.
   * Wenn nicht: rdflib oder Graph-Struktur prüfen.

Teste das mal, und wir sehen, was rauskommt! Wir sind ganz nah dran!

2025-03-07 10:58:06,235 - DEBUG - Chain1-Typ-Triples: 12025-03-07 10:58:06,269 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 10:58:06,396 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 10:58:06,396 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 10:58:06,396 - INFO - Starte allgemeine SPARQL-Abfrage... 2025-03-07 10:58:06,407 - INFO - SPARQL-Abfrage Ergebnisse (alle Chains): 2025-03-07 10:58:06,408 - INFO - Chain: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1>, UpdateCount: 22 2025-03-07 10:58:06,464 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 10:58:06,464 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 10:58:06,464 - DEBUG - Found 1 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 10:58:06,464 - DEBUG - Found 1 property paths to follow. 2025-03-07 10:58:06,464 - DEBUG - Found 13 implied SHACL Shapes based on their properties. 2025-03-07 10:58:06,465 - DEBUG - Found 26 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 10:58:06,466 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes. 2025-03-07 10:58:06,466 - DEBUG - Validating DataGraph named N83e89ada49a0402a965c693b8b0cb484Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. 2025-03-07 10:58:08,080 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!

Yes! Wir haben einen Durchbruch – die SPARQL-Abfrage funktioniert jetzt, aber PySHACL ignoriert die SPARQL-Constraints weiterhin. Die neuen Logs zeigen genau, wo wir stehen, und wir können das Problem nun final lösen. Lass uns das analysieren und die zweite Violation ins Leben rufen!

Analyse der Logs

1. Typ-Prüfung

2025-03-07 10:58:06,235 - DEBUG - Chain1-Typ-Triples: 1

* Bestätigt: ould:Chain1 a ould:UpdateChain ist im Graph vorhanden.

2. Spezifische SPARQL-Abfrage

2025-03-07 10:58:06,269 - INFO - Starte spezifische SPARQL-Abfrage für Chain1...

2025-03-07 10:58:06,396 - INFO - SPARQL-Abfrage Ergebnisse für Chain1:

2025-03-07 10:58:06,396 - INFO - UpdateCount für Chain1: 22

* Erfolg: Die spezifische Abfrage erkennt korrekt, dass ould:Chain1 22 Updates hat. Das beweist, dass die Daten im Graph richtig sind und rdflib SPARQL korrekt ausführen kann.

3. Allgemeine SPARQL-Abfrage

2025-03-07 10:58:06,396 - INFO - Starte allgemeine SPARQL-Abfrage...

2025-03-07 10:58:06,407 - INFO - SPARQL-Abfrage Ergebnisse (alle Chains):

2025-03-07 10:58:06,408 - INFO - Chain: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1, UpdateCount: 22

* Erfolg: Die allgemeine Abfrage findet ebenfalls ould:Chain1 mit 22 Updates. Interessant ist, dass keine weiteren Chains gelistet sind, obwohl vorher Anzahl UpdateChains: 11 geloggt wurde. Das könnte bedeuten, dass die anderen 10 UpdateChains keine hasUpdate-Triples haben (was mit deiner ABox übereinstimmen könnte, z. B. aus OULD\_ABox\_valid.ttl).

4. Validierungsbericht

Validation Report

Conforms: False

Results (1):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

2025-03-07 10:58:08,080 - WARNING - Keine SPARQL-Constraints im Validierungsbericht gefunden!

* Bestätigt: oush:UpdateChainShape triggert korrekt.
* Problem: Keine SPARQL-Constraints (oush:UpdateChainContinuityConstraint, oush:SimpleTestShape) werden ausgeführt, obwohl die SPARQL-Abfrage in debug\_sparql jetzt funktioniert.

5. PySHACL-Debugging

2025-03-07 10:58:06,464 - DEBUG - Found 13 SHACL Shapes defined with type sh:NodeShape.

[...]

2025-03-07 10:58:06,466 - DEBUG - Cached 13 unique NodeShapes and 26 unique PropertyShapes.

* Beobachtung: PySHACL erkennt die 13 NodeShapes (inklusive unserer SPARQL-Shapes), führt aber keine SPARQL-Constraints aus.

Hypothese: PySHACL-Bug oder Konfigurationsproblem

* Daten und SPARQL: Die Daten sind korrekt, und rdflib führt SPARQL-Abfragen erfolgreich aus (siehe debug\_sparql). Das Problem liegt also nicht bei rdflib.
* PySHACL: Trotz debug=2 und korrekter Erkennung der Shapes werden SPARQL-Constraints ignoriert. Das deutet auf:
  1. Bug in PySHACL v0.30: SPARQL-Unterstützung könnte fehlerhaft sein.
  2. Konfiguration: inference="none" oder ein anderes Setting könnte SPARQL deaktivieren.

Lösungsvorschlag

Schritt 1: PySHACL-Konfiguration anpassen

Lass uns inference="rdfs" statt "none" testen, um sicherzustellen, dass PySHACL alle Constraints ausführt. Außerdem prüfen wir die Shapes explizit.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(ABOX\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

chain1\_type = list(data\_graph.triples((OULD.Chain1, None, OULD.UpdateChain)))

logger.debug(f"Chain1-Typ-Triples: {len(chain1\_type)}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2) # RDFS-Inferenz

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht (gekürzt):")

report\_lines = report\_text.splitlines()[:50]

logger.info("\n".join(report\_lines))

if "SPARQLConstraint" not in report\_text:

logger.warning("Keine SPARQL-Constraints im Validierungsbericht gefunden!")

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* inference="rdfs":
  + Aktiviert RDFS-Inferenz, was SPARQL-Constraints möglicherweise besser unterstützt.
* Shapes: Bleiben unverändert (oush:UpdateChainContinuityConstraint und oush:SimpleTestShape sollten triggert, wenn SPARQL läuft).

Schritt 2: Ergebnisse prüfen

* SPARQL: Sollte weiterhin UpdateCount für Chain1: 22 zeigen.
* Validierungsbericht: Sollte jetzt zwei Violations zeigen:
  1. MaxCountConstraintComponent
  2. SPARQLConstraint (für oush:UpdateChainContinuityConstraint und/oder oush:SimpleTestShape).

Schritt 3: Falls SPARQL weiter fehlt

* Minimaltest: Wir machen den isolierten Test mit test\_sparql.ttl und test\_shapes.ttl (siehe früherer Vorschlag).
* PySHACL-Issue: Melden wir auf GitHub, falls inference="rdfs" nicht hilft.

Meine Einschätzung

* SPARQL: Funktioniert jetzt in rdflib, also sind Daten und Abfragen korrekt.
* PySHACL: Ignoriert SPARQL-Constraints, wahrscheinlich wegen inference="none". Mit rdfs sollte es klappen.
* Ziel: Zwei Violations in der validation.log.

Nächste Schritte

1. Code ausführen: Mit inference="rdfs".
2. Log prüfen: Schick mir den neuen Validierungsbericht.
3. Entscheidung:
   * Wenn SPARQL-Violation da ist: Geschafft!
   * Wenn nicht: Minimaltest oder PySHACL-Debugging vertiefen.

Teste das mal – ich bin zuversichtlich, dass wir jetzt beide Violations sehen! Was sagst du?

(Anmerkung am Rande: Es muss heißen "ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")", nicht ABOX\_DIR = os.path.join(ABOX\_DIR, "OULD\_ABox") - Scherzkeks! xD)Validation Report Conforms: False Results (2): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdatableEntityBaselineShape Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?update . FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . } FILTER NOT EXISTS { ?update ould:hasHash ?hash . } } ") ; sh:severity sh:Violation ] Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.Erledigt, oder?

Haha, gut erwischt – der ABOX\_DIR-Schnitzer war wirklich ein Scherzkeks-Moment! Danke für die Korrektur, das hätte sonst noch für Kopfzerbrechen gesorgt. Und ja: Erledigt! Wir haben jetzt zwei Violations – Mission accomplished! Lass uns das durchgehen und sicherstellen, dass alles passt.

Analyse der Ergebnisse

Validierungsbericht

Validation Report

Conforms: False

Results (2):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdatableEntityBaselineShape

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?update .

FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . }

FILTER NOT EXISTS { ?update ould:hasHash ?hash . }

}

") ; sh:severity sh:Violation ]

Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.

Was wir sehen:

1. MaxCountConstraintComponent:
   * Erwartet: ould:Chain1 hat 22 Updates, verletzt maxCount 21 von oush:UpdateChainShape.
   * Ergebnis: Check! Das war schon vorher da und ist korrekt.
2. SPARQLConstraintComponent:
   * Quelle: oush:UpdatableEntityBaselineShape
   * Problem: ould:Chain1 wird als UpdatableEntity betrachtet, und mindestens ein Update fehlt hasIFCID und hasHash.
   * Unerwartet: Wir hatten oush:UpdateChainContinuityConstraint als zweite Violation erwartet (wegen > 21 Updates), nicht UpdatableEntityBaselineShape.

Bewertung

* Erfolg: Wir haben jetzt zwei Violations, und SPARQL-Constraints funktionieren mit inference="rdfs". Der Wechsel von inference="none" war der Schlüssel – offenbar benötigt PySHACL eine aktive Inferenz, um SPARQL-Shapes korrekt auszuführen.
* Abweichung: Die zweite Violation kommt nicht von oush:UpdateChainContinuityConstraint (wie ursprünglich gewünscht), sondern von oush:UpdatableEntityBaselineShape. Das liegt daran, dass:
  + In OULD\_ABox\_invalid\_22updates.ttl ist ould:Chain1 kein UpdatableEntity, sondern nur ein UpdateChain. Die Shape oush:UpdatableEntityBaselineShape sollte eigentlich auf ould:Wall1 angewendet werden, nicht auf ould:Chain1.
  + Die Shape oush:UpdateChainContinuityConstraint triggert nicht, obwohl sie sollte.

Warum triggert oush:UpdateChainContinuityConstraint nicht?

Die Shape lautet (angepasst):

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates; split into multiple chains linked via hasNextChain." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* Daten: ould:Chain1 hat 22 hasUpdate-Triples, also sollte die Abfrage $this = ould:Chain1, ?updateCount = 22 liefern und triggert.
* Problem: Entweder wird die Shape nicht korrekt auf ould:Chain1 angewendet, oder PySHACL hat ein Problem mit der Zielklasse.

Lösungsvorschlag

Schritt 1: Violation korrigieren

Die aktuelle zweite Violation (UpdatableEntityBaselineShape) ist ein Fehlalarm, da ould:Chain1 kein UpdatableEntity ist. Lass uns sicherstellen, dass oush:UpdateChainContinuityConstraint triggert.

Prüfung der ABox

In OULD\_ABox\_invalid\_22updates.ttl:

turtle

ould:Wall1 a ould:UpdatableEntity ;

ould:hasIFCID "ifc:Wall1" ;

ould:hasHash "abc123" ;

ould:hasUpdateChain ould:Chain1 .

ould:Chain1 a ould:UpdateChain ;

ould:hasUpdate ould:Update1, ould:Update2, [...], ould:Update22 ;

time:hasBeginning "2025-03-01"^^xsd:date ;

time:hasEnd "2025-03-22"^^xsd:date .

* Problem: Die Shape oush:UpdatableEntityBaselineShape sollte auf ould:Wall1 zielen, nicht auf ould:Chain1. Aber ould:Chain1 wird fälschlicherweise als UpdatableEntity validiert (vielleicht durch Inferenz).

Überarbeiteter Code

Lass uns die Inferenz anpassen und die Shapes explizit prüfen:

python

import owlready2

from rdflib import Graph, Namespace

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox") # Korrigierter Scherzkeks! ;)

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_updates = list(data\_graph.triples((OULD.Chain1, OULD.hasUpdate, None)))

logger.debug(f"Anzahl ould:hasUpdate-Triples für Chain1: {len(chain1\_updates)}")

chain1\_type = list(data\_graph.triples((OULD.Chain1, None, OULD.UpdateChain)))

logger.debug(f"Chain1-Typ-Triples: {len(chain1\_type)}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht:")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines)) # Voller Bericht für Klarheit

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* ABOX\_DIR: Korrigierter Pfad (danke für den Hinweis!).
* Voller Bericht: Kein Kürzen mehr, um sicherzustellen, dass wir alles sehen.

Schritt 2: Ergebnisse prüfen

* Ziel: Zwei Violations:
  1. MaxCountConstraintComponent (bereits da).
  2. SPARQLConstraintComponent von oush:UpdateChainContinuityConstraint (für > 21 Updates).
* Wenn falsche Violation: Dann liegt ein Fehler in der Shape-Zuordnung vor (z. B. UpdatableEntityBaselineShape auf Chain1).

Schritt 3: Falls nötig

Wenn oush:UpdateChainContinuityConstraint immer noch nicht triggert:

* Shape prüfen: Sicherstellen, dass sh:targetClass ould:UpdateChain korrekt ist.
* Minimaltest: Isolierter Test mit test\_sparql.ttl.

Meine Einschätzung

* Fast erledigt: Wir haben SPARQL zum Laufen gebracht, und inference="rdfs" hat geholfen.
* Nächster Schritt: Die richtige Violation (UpdateChainContinuityConstraint) statt UpdatableEntityBaselineShape erzwingen.
* Vermutung: Chain1 wird durch Inferenz fälschlich als UpdatableEntity klassifiziert, was die falsche Shape triggert.

Teste das mal, und schick mir den vollen Validierungsbericht! Wir sind so nah dran – das wird jetzt! Was denkst du?

Validation Report Conforms: False Results (2): Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdatableEntityBaselineShape Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?update . FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . } FILTER NOT EXISTS { ?update ould:hasHash ?hash . } } ") ; sh:severity sh:Violation ] Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash. Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Alles klar, wir sind fast da! Die Logs zeigen zwei Violations, aber wir haben immer noch die falsche SPARQL-Violation (oush:UpdatableEntityBaselineShape statt oush:UpdateChainContinuityConstraint). Das Ziel war, die Violation für > 21 Updates zu bekommen, also lass uns das jetzt endgültig richten.

Analyse des Validierungsberichts

Validation Report

Conforms: False

Results (2):

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdatableEntityBaselineShape

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?update .

FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . }

FILTER NOT EXISTS { ?update ould:hasHash ?hash . }

}

") ; sh:severity sh:Violation ]

Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Was stimmt:

1. MaxCountConstraintComponent:
   * ould:Chain1 hat 22 Updates, verletzt maxCount 21 → Perfekt, das ist korrekt!
2. SPARQLConstraintComponent:
   * Aktuell: oush:UpdatableEntityBaselineShape triggert auf ould:Chain1, weil ein Update keine hasIFCID und hasHash hat.
   * Problem: ould:Chain1 ist ein UpdateChain, kein UpdatableEntity. Die Shape sollte auf ould:Wall1 angewendet werden, nicht auf ould:Chain1. Außerdem erwarten wir oush:UpdateChainContinuityConstraint für die > 21 Updates-Regel.

Warum passiert das?

Falsche Zuordnung

* oush:UpdatableEntityBaselineShape:

turtle

oush:UpdatableEntityBaselineShape a sh:NodeShape ;

sh:targetClass ould:UpdatableEntity ;

sh:sparql [

sh:message "Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash." ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?update .

FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . }

FILTER NOT EXISTS { ?update ould:hasHash ?hash . }

}

"""

] .

* + Problem: ould:Chain1 wird fälschlicherweise als UpdatableEntity betrachtet (vermutlich durch RDFS-Inferenz), obwohl es nur ein UpdateChain ist. In deiner ABox ist nur ould:Wall1 ein UpdatableEntity.
* oush:UpdateChainContinuityConstraint:

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates; split into multiple chains linked via hasNextChain." ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* + Daten: ould:Chain1 hat 22 hasUpdate-Triples, also sollte diese Shape triggert.
  + Problem: Sie triggert nicht, obwohl sh:targetClass ould:UpdateChain korrekt auf ould:Chain1 zielen sollte.

Inferenz-Fehler

* Mit inference="rdfs" scheint PySHACL ould:Chain1 als UpdatableEntity zu inferieren, was falsch ist. In deiner ABox (OULD\_ABox\_invalid\_22updates.ttl) ist nur ould:Wall1 ein UpdatableEntity.

Lösungsvorschlag

Schritt 1: Inferenz überprüfen

Lass uns den Graph nach der Inferenz loggen, um zu sehen, warum ould:Chain1 als UpdatableEntity behandelt wird.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

# Prüfe Typen

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht:")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Typ-Logging: Prüft die Typen von ould:Chain1 und ould:Wall1 nach der Inferenz, um zu sehen, ob Chain1 fälschlich als UpdatableEntity klassifiziert wird.

Schritt 2: Ergebnisse prüfen

* Typen:
  + Chain1 Typen: Sollte nur ould:UpdateChain zeigen, nicht ould:UpdatableEntity.
  + Wall1 Typen: Sollte ould:UpdatableEntity zeigen.
* Validierungsbericht: Sollte idealerweise:
  1. MaxCountConstraintComponent für ould:Chain1.
  2. SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint.

Schritt 3: Falls falsche Violation

Wenn Chain1 als UpdatableEntity inferiert wird:

* Inferenz abschalten: Teste mit inference="none" und prüfe, ob UpdateChainContinuityConstraint dann triggert.
* Shape anpassen: Explizite Ausschlüsse hinzufügen, falls nötig.

Meine Einschätzung

* Problem: inference="rdfs" inferiert ould:Chain1 fälschlich als UpdatableEntity, daher triggert die falsche Shape.
* Lösung: Entweder Inferenz korrigieren oder die Shapes präzisieren.
* Ziel: oush:UpdateChainContinuityConstraint statt UpdatableEntityBaselineShape.

Teste das mal, und schick mir die neuen Logs (Typen + Validierungsbericht)! Wir kriegen das jetzt hin! Was denkst du?

Könnte das Problem durch eine Änderung der TBox gelöst werden, z.B. indem wir die Klasse ould:UpdateChain irgendwie anders/expliziter definieren?

Gute Idee! Dein Vorschlag, das Problem über eine Änderung der TBox anzugehen, indem wir ould:UpdateChain expliziter definieren, ist absolut sinnvoll. Das könnte die unerwünschte Inferenz verhindern, die dazu führt, dass ould:Chain1 fälschlicherweise als ould:UpdatableEntity behandelt wird, und gleichzeitig sicherstellen, dass oush:UpdateChainContinuityConstraint korrekt triggert. Lass uns das analysieren und eine Lösung in der TBox umsetzen.

Analyse des Problems

Aktueller Stand

* Daten (ABox): ould:Chain1 ist explizit als ould:UpdateChain definiert:

turtle

ould:Chain1 a ould:UpdateChain ;

ould:hasUpdate ould:Update1, ould:Update2, ..., ould:Update22 .

ould:Wall1 ist das einzige UpdatableEntity:

turtle

ould:Wall1 a ould:UpdatableEntity ;

ould:hasUpdateChain ould:Chain1 .

* Inferenz: Mit inference="rdfs" wird ould:Chain1 fälschlicherweise als ould:UpdatableEntity klassifiziert, was oush:UpdatableEntityBaselineShape triggert.
* Ziel: oush:UpdateChainContinuityConstraint soll triggert (wegen > 21 Updates), und oush:UpdatableEntityBaselineShape soll nur auf echte UpdatableEntities wie ould:Wall1 angewendet werden.

Warum passiert die falsche Inferenz?

In der aktuellen TBox (OULD\_V1.0.ttl):

* ould:UpdatableEntity:

turtle

ould:UpdatableEntity rdf:type owl:Class ;

rdfs:comment "A superclass representing any entity that can undergo updates."@en .

* + Es gibt keine explizite Einschränkung, die verhindert, dass ould:UpdateChain als Unterklasse oder Instanz von UpdatableEntity inferiert wird.
* ould:UpdateChain:

turtle

ould:UpdateChain rdf:type owl:Class ;

rdfs:comment "The class \"UpdateChain\" represents a structured sequence of consecutive updates."@en .

* + Keine Beziehung zu UpdatableEntity definiert, aber auch keine Aussage, die eine Überlappung ausschließt.

Mit RDFS-Inferenz könnte ould:Chain1 als UpdatableEntity interpretiert werden, weil:

1. Es ould:hasUpdate-Beziehungen hat (eine Eigenschaft, die UpdatableEntity nutzt).
2. Keine Disjunktheit zwischen ould:UpdateChain und ould:UpdatableEntity definiert ist.

Lösung über die TBox

Wir können die TBox anpassen, um:

* Disjunktheit zwischen ould:UpdateChain und ould:UpdatableEntity zu definieren, damit sie sich nicht überschneiden.
* Klassen klarer trennen, sodass Inferenz ould:Chain1 nicht als UpdatableEntity klassifiziert.

Vorschlag: TBox-Änderung

Füge folgende Definitionen in OULD\_V1.0.ttl hinzu oder passe die bestehenden Klassen an:

turtle

### http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

ould:UpdatableEntity rdf:type owl:Class ;

rdfs:comment "A superclass representing any entity that can undergo updates, excluding UpdateChains."@en ;

owl:disjointWith ould:UpdateChain . # Disjunktheit hinzufügen

### http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

ould:UpdateChain rdf:type owl:Class ;

rdfs:comment "The class \"UpdateChain\" represents a structured sequence of consecutive updates, distinct from UpdatableEntities."@en ;

owl:disjointWith ould:UpdatableEntity . # Gegenseitige Disjunktheit

Wirkung

* owl:disjointWith: Stellt sicher, dass keine Instanz gleichzeitig ein UpdateChain und ein UpdatableEntity sein kann. Das verhindert, dass ould:Chain1 als UpdatableEntity inferiert wird.
* Inferenz: RDFS wird diese Disjunktheit respektieren, und PySHACL sollte oush:UpdatableEntityBaselineShape nicht mehr auf ould:Chain1 anwenden.
* oush:UpdateChainContinuityConstraint: Kann jetzt ungehindert auf ould:Chain1 als UpdateChain angewendet werden und sollte wegen der 22 Updates triggert.

Testen der Änderung

1. TBox anpassen: Füge die owl:disjointWith-Aussagen in OULD\_V1.0.ttl hinzu.
2. Code unverändert: Nutze den letzten Code mit inference="rdfs":

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität: {conforms}")

if not conforms:

logger.info("Validierungsbericht:")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

1. Ergebnisse prüfen:
   * Typen: Chain1 sollte nur ould:UpdateChain sein, Wall1 nur ould:UpdatableEntity.
   * Validierungsbericht: Sollte:
     1. MaxCountConstraintComponent für ould:Chain1.
     2. SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint.

Alternative: Ohne TBox-Änderung

Falls du die TBox nicht ändern willst:

* inference="none": Deaktiviert Inferenz komplett, könnte aber andere Shapes beeinträchtigen.
* Shape-Anpassung: Füge eine Einschränkung zu oush:UpdatableEntityBaselineShape hinzu, z. B.:

turtle

oush:UpdatableEntityBaselineShape a sh:NodeShape ;

sh:targetClass ould:UpdatableEntity ;

sh:sparql [

sh:message "Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash." ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?update .

FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . }

FILTER NOT EXISTS { ?update ould:hasHash ?hash . }

FILTER NOT EXISTS { $this a ould:UpdateChain . } # Ausschluss

}

"""

] .

Meine Einschätzung

* TBox-Änderung: Die sauberste Lösung, da sie die Semantik klarer macht und Inferenzprobleme dauerhaft löst. owl:disjointWith ist der richtige Weg, um UpdateChain und UpdatableEntity zu trennen.
* Alternative: Shape-Anpassung ist ein Workaround, aber weniger elegant und könnte bei komplexeren Inferenzen wieder Probleme machen.

Ich würde die TBox-Änderung empfehlen – sie passt zu deinem Ansatz, die Klassen expliziter zu definieren. Was denkst du? Soll ich die TBox-Änderung noch detaillierter ausarbeiten, oder testest du das so? Wir sind fast am Ziel!

Alles umgesetzt, hier das Ergebnis: Validation Report Conforms: False Results (2): Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdatableEntityBaselineShape Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?update . FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . } FILTER NOT EXISTS { ?update ould:hasHash ?hash . } } ") ; sh:severity sh:Violation ] Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash. Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Okay, das Ergebnis ist interessant – wir haben immer noch zwei Violations, aber die gewünschte oush:UpdateChainContinuityConstraint triggert weiterhin nicht, und stattdessen bleibt oush:UpdatableEntityBaselineShape aktiv. Das bedeutet, die owl:disjointWith-Änderung in der TBox hat nicht den erwarteten Effekt gehabt. Lass uns das genau analysieren und endgültig lösen!

Analyse des Validierungsberichts

Validation Report

Conforms: False

Results (2):

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdatableEntityBaselineShape

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?update .

FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . }

FILTER NOT EXISTS { ?update ould:hasHash ?hash . }

}

") ; sh:severity sh:Violation ]

Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Was stimmt:

* MaxCountConstraintComponent: Korrekt, ould:Chain1 hat 22 Updates, verletzt maxCount 21. Das ist genau, was wir wollen.

Was nicht stimmt:

* SPARQLConstraintComponent:
  + Aktuell: oush:UpdatableEntityBaselineShape triggert auf ould:Chain1, obwohl wir owl:disjointWith hinzugefügt haben.
  + Erwartet: oush:UpdateChainContinuityConstraint sollte triggert (wegen > 21 Updates).

Warum ignoriert die Inferenz owl:disjointWith?

* TBox-Änderung: Du hast owl:disjointWith zwischen ould:UpdatableEntity und ould:UpdateChain eingefügt:

turtle

ould:UpdatableEntity rdf:type owl:Class ;

owl:disjointWith ould:UpdateChain .

ould:UpdateChain rdf:type owl:Class ;

owl:disjointWith ould:UpdatableEntity .

* ABox: ould:Chain1 a ould:UpdateChain, ould:Wall1 a ould:UpdatableEntity.
* Problem: Trotz owl:disjointWith wird ould:Chain1 als UpdatableEntity inferiert. Das deutet darauf hin, dass:
  1. Inferenz ignoriert Disjunktheit: inference="rdfs" in PySHACL unterstützt owl:disjointWith nicht vollständig (RDFS kennt keine Disjunktheit nativ, das ist OWL-Semantik).
  2. Pellet-Reasoning: Die Disjunktheit wird im Reasoning-Schritt nicht korrekt angewendet oder übernommen.

Hypothese: Reasoning-Problem

* Pellet: owlready2.sync\_reasoner\_pellet() sollte die Disjunktheit erkennen und verhindern, dass ould:Chain1 als UpdatableEntity klassifiziert wird. Wenn das nicht passiert, liegt das Problem entweder bei:
  + Pellet-Konfiguration: Disjunktheit wird nicht inferiert oder übernommen.
  + Graph-Übertragung: Die inferierten Daten werden nicht korrekt in den rdflib.Graph übertragen.

Prüfung der Typen

Die Logs zeigen bisher:

* Chain1 Typen: 1 (nur ould:UpdateChain), aber wir müssen sicherstellen, dass keine zusätzlichen Typen inferiert werden.

Lösungsvorschlag

Schritt 1: Typen nach Reasoning prüfen

Lass uns die inferierten Typen explizit loggen, um sicherzugehen, dass ould:Chain1 nicht als UpdatableEntity klassifiziert wird.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

# Prüfe Typen nach Reasoning

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen nach Reasoning: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen nach Reasoning: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

# Teste mit inference="rdfs" und ohne

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität (inference=rdfs): {conforms}")

if not conforms:

logger.info("Validierungsbericht (inference=rdfs):")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

# Ohne Inferenz testen

result\_no\_inf = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms\_no\_inf, report\_graph\_no\_inf, report\_text\_no\_inf = result\_no\_inf

logger.info(f"Konformität (inference=none): {conforms\_no\_inf}")

if not conforms\_no\_inf:

logger.info("Validierungsbericht (inference=none):")

report\_lines\_no\_inf = report\_text\_no\_inf.splitlines()

logger.info("\n".join(report\_lines\_no\_inf))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH)

Änderungen:

* Typ-Logging: Detaillierter, um sicherzustellen, dass ould:Chain1 nicht als UpdatableEntity inferiert wird.
* Doppelte Validierung: Testet sowohl mit inference="rdfs" als auch inference="none", um den Einfluss der Inferenz zu sehen.

Schritt 2: Ergebnisse prüfen

* Typen:
  + Chain1 Typen: Sollte nur ould:UpdateChain zeigen.
  + Wall1 Typen: Sollte ould:UpdatableEntity zeigen.
* Validierungsberichte:
  + inference="rdfs":
    - Sollte MaxCount und UpdateChainContinuityConstraint zeigen, keine UpdatableEntityBaselineShape auf Chain1.
  + inference="none":
    - Sollte dasselbe zeigen, da Disjunktheit ohne OWL-Inferenz ignoriert wird.

Schritt 3: Falls UpdateChainContinuityConstraint fehlt

* Pellet prüfen: Stelle sicher, dass owl:disjointWith im Reasoning berücksichtigt wird (z. B. durch Log-Ausgabe von Pellet).
* Shape direkt testen: Reduziere OULD\_V1.0.ttl auf nur UpdateChainContinuityConstraint, um sicherzugehen, dass sie funktioniert.

Meine Einschätzung

* Disjunktheit: Wurde hinzugefügt, aber inference="rdfs" ignoriert sie, da RDFS keine OWL-Semantik wie owl:disjointWith nativ unterstützt. Pellet sollte das jedoch korrigieren.
* Problem: Entweder wird die Disjunktheit nicht in den Graph übernommen, oder PySHACL wendet die Shape falsch an.
* Lösung: Typen prüfen und Inferenz-Einfluss klären.

Teste das mal, und schick mir die Logs (Typen + beide Validierungsberichte)! Wir sind so nah dran – diesmal kriegen wir die richtige Violation! Was denkst du?

Hier das ergebnis mit dem aktuellen Code: Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.Jetzt ist es nur noch eine Violation! Hier noch mehr Auszüge aus der Log-file: 2025-03-07 11:24:04,197 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:24:04,333 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:24:04,333 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:24:04,380 - DEBUG - Cloning DataGraph to temporary memory graph before pre-inferencing. 2025-03-07 11:24:04,400 - DEBUG - Running pre-inferencing with option='rdfs'.

Wow, das ist ein Schritt vorwärts – und gleichzeitig ein Rätsel! Wir haben jetzt nur noch eine Violation (MaxCountConstraintComponent), und die unerwünschte UpdatableEntityBaselineShape-Violation ist weg. Aber die erwartete UpdateChainContinuityConstraint-Violation fehlt immer noch, obwohl die SPARQL-Abfrage in debug\_sparql korrekt 22 Updates für ould:Chain1 erkennt. Lass uns das analysieren und endgültig die zweite Violation ins Leben rufen!

Analyse der Logs

Validierungsbericht

Validation Report

Conforms: False

Results (1):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

* Positiv: Die MaxCount-Violation ist korrekt und stabil.
* Problem: Die oush:UpdateChainContinuityConstraint-Violation fehlt, obwohl sie bei 22 Updates triggert sollte.

SPARQL-Abfrage

2025-03-07 11:24:04,197 - INFO - Starte spezifische SPARQL-Abfrage für Chain1...

2025-03-07 11:24:04,333 - INFO - SPARQL-Abfrage Ergebnisse für Chain1:

2025-03-07 11:24:04,333 - INFO - UpdateCount für Chain1: 22

* Bestätigt: Die SPARQL-Abfrage erkennt korrekt, dass ould:Chain1 22 Updates hat. Das bedeutet, die Daten sind da, und rdflib führt SPARQL richtig aus.

Inferenz

2025-03-07 11:24:04,380 - DEBUG - Cloning DataGraph to temporary memory graph before pre-inferencing.

2025-03-07 11:24:04,400 - DEBUG - Running pre-inferencing with option='rdfs'.

* Hinweis: Die Inferenz mit rdfs läuft, was die Disjunktheit erklären könnte (siehe unten).

Typen (fehlen im Auszug)

Du hast die Typ-Logs (Chain1 Typen und Wall1 Typen) nicht gepostet. Sie sind entscheidend, um zu sehen, ob ould:Chain1 immer noch als UpdatableEntity inferiert wird oder ob die Disjunktheit greift. Bitte schick mir diese Zeilen aus der validation.log:

* Chain1 Typen nach Reasoning: ...
* Wall1 Typen nach Reasoning: ...

Was fehlt?

* Der Bericht zeigt nur den ersten Validierungslauf (inference="rdfs"). Der zweite Lauf (inference="none") fehlt in deinem Auszug. Bitte poste auch den Abschnitt:
  + Konformität (inference=none): ...
  + Validierungsbericht (inference=none): ...

Hypothese

Fortschritt

* Disjunktheit wirkt: Die owl:disjointWith-Änderung hat die unerwünschte UpdatableEntityBaselineShape-Violation auf ould:Chain1 beseitigt. Das ist ein großer Erfolg – die TBox-Änderung funktioniert teilweise!
* Problem: oush:UpdateChainContinuityConstraint triggert nicht, obwohl:
  + Die Shape definiert ist:

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates; split into multiple chains linked via hasNextChain." ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* + ould:Chain1 ein UpdateChain ist mit 22 hasUpdate-Triples.

Mögliche Ursachen

1. Shape wird nicht angewendet:
   * PySHACL erkennt sh:targetClass ould:UpdateChain nicht korrekt auf ould:Chain1.
2. SPARQL-Fehler in PySHACL:
   * Die SPARQL-Abfrage in der Shape wird nicht ausgeführt, obwohl debug\_sparql zeigt, dass sie funktionieren sollte.
3. Inferenz stört:
   * inference="rdfs" könnte die Shape-Ausführung beeinflussen, obwohl die Daten korrekt sind.

Lösungsvorschlag

Schritt 1: Typen und zweiten Bericht prüfen

* Typen: Bitte poste die fehlenden Typ-Logs, um sicherzustellen, dass ould:Chain1 nur UpdateChain ist und ould:Wall1 UpdatableEntity.
* Inference=none: Schau dir den zweiten Validierungsbericht an – er könnte zeigen, ob UpdateChainContinuityConstraint ohne Inferenz triggert.

Schritt 2: Shape isolieren

Wenn die Typen korrekt sind und inference="none" auch keine SPARQL-Violation zeigt, testen wir die Shape isoliert. Erstelle eine reduzierte test\_shapes.ttl:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

Passe den Code an, um nur diese Shape zu validieren:

python

def perform\_shacl\_validation(data\_file, shapes\_path="test\_shapes.ttl"):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="rdfs", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität (inference=rdfs): {conforms}")

if not conforms:

logger.info("Validierungsbericht (inference=rdfs):")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

result\_no\_inf = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms\_no\_inf, report\_graph\_no\_inf, report\_text\_no\_inf = result\_no\_inf

logger.info(f"Konformität (inference=none): {conforms\_no\_inf}")

if not conforms\_no\_inf:

logger.info("Validierungsbericht (inference=none):")

report\_lines\_no\_inf = report\_text\_no\_inf.splitlines()

logger.info("\n".join(report\_lines\_no\_inf))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

Schritt 3: Ergebnisse prüfen

* Typen: Bestätigen, dass ould:Chain1 nicht UpdatableEntity ist.
* Inference=none: Sollte UpdateChainContinuityConstraint zeigen.
* Isolierte Shape: Wenn sie triggert, liegt das Problem an einer Interaktion mit anderen Shapes in OULD\_V1.0.ttl.

Meine Einschätzung

* Fortschritt: UpdatableEntityBaselineShape ist weg – die Disjunktheit wirkt zumindest teilweise.
* Problem: UpdateChainContinuityConstraint wird nicht angewendet, obwohl die Daten passen. Entweder ein PySHACL-Fehler oder ein Subtilität in der Shape-Definition.
* Nächster Schritt: Typen klären und Shape isolieren.

Bitte schick mir die Typ-Logs und den inference=none-Bericht aus der aktuellen validation.log. Wenn das nicht reicht, testen wir die isolierte Shape. Wir sind so nah dran – das kriegen wir jetzt! Was sagst du?

2025-03-07 11:24:04,156 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:24:04,157 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:24:04,157 - DEBUG - Wall1 Typen nach Reasoning: 12025-03-07 11:24:07,253 - INFO - Konformit�t (inference=rdfs): Falsezwischendrin steht noch: Validation Report Conforms: False Results (2): Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdatableEntityBaselineShape Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?update . FILTER NOT EXISTS { ?update ould:hasIFCID ?ifcid . } FILTER NOT EXISTS { ?update ould:hasHash ?hash . } } ") ; sh:severity sh:Violation ] Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash. Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.Und ganz am Ende: Validation Report Conforms: False Results (1): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Danke für die vollständigen Logs! Jetzt haben wir alles, um das Rätsel zu lösen und die zweite Violation (oush:UpdateChainContinuityConstraint) endlich zum Vorschein zu bringen. Die Ergebnisse zeigen einen klaren Fortschritt, aber auch ein verbleibendes Problem – lass uns das Schritt für Schritt durchgehen.

Analyse der Logs

Typen nach Reasoning

2025-03-07 11:24:04,156 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:24:04,157 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:24:04,157 - DEBUG - Wall1 Typen nach Reasoning: 1

* Problem: Der Wall1-Typ ist nicht vollständig geloggt (fehlt das Triple). Das könnte ein Logging-Fehler sein, aber wir gehen davon aus, dass es ould:Wall1 a ould:UpdatableEntity ist, wie in der ABox definiert.
* Positiv: ould:Chain1 hat nur den Typ ould:UpdateChain, kein ould:UpdatableEntity. Die owl:disjointWith-Änderung scheint also im Reasoning zu wirken – Chain1 wird nicht mehr falsch klassifiziert.

SPARQL-Abfrage

2025-03-07 11:24:04,197 - INFO - Starte spezifische SPARQL-Abfrage für Chain1...

2025-03-07 11:24:04,333 - INFO - SPARQL-Abfrage Ergebnisse für Chain1:

2025-03-07 11:24:04,333 - INFO - UpdateCount für Chain1: 22

* Bestätigt: Die Daten sind korrekt – ould:Chain1 hat 22 Updates.

Validierungsbericht (inference="rdfs")

Validation Report

Conforms: False

Results (2):

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdatableEntityBaselineShape

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ rdf:type sh:SPARQLConstraint ; sh:message Literal("Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.") ; ... ]

Message: Each UpdatableEntity must have at least one Update with a hasIFCID and hasHash.

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; ... ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

* Problem: oush:UpdatableEntityBaselineShape triggert immer noch auf ould:Chain1, obwohl Chain1 kein UpdatableEntity ist (siehe Typen). Die Disjunktheit wird hier ignoriert.
* Fehlt: oush:UpdateChainContinuityConstraint.

Validierungsbericht (inference="none")

Validation Report

Conforms: False

Results (1):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; ... ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

* Positiv: Ohne Inferenz verschwindet die falsche UpdatableEntityBaselineShape-Violation – das bestätigt, dass inference="rdfs" das Problem verursacht.
* Problem: oush:UpdateChainContinuityConstraint triggert immer noch nicht, obwohl die Daten passen.

Hypothese

Disjunktheit

* inference="rdfs":
  + Ignoriert owl:disjointWith, da RDFS keine OWL-Semantik wie Disjunktheit nativ unterstützt. Pellet sollte das im Reasoning korrigieren, aber die inferierten Daten scheinen nicht korrekt übernommen zu werden.
  + oush:UpdatableEntityBaselineShape triggert, weil ould:Chain1 durch irgendeine Inferenz als UpdatableEntity behandelt wird, trotz owl:disjointWith.
* inference="none":
  + Respektiert die expliziten Typen (Chain1 nur UpdateChain), daher keine falsche Violation.
  + Aber oush:UpdateChainContinuityConstraint fehlt weiterhin.

Warum fehlt UpdateChainContinuityConstraint?

* Shape-Problem: Die Shape wird nicht korrekt auf ould:Chain1 angewendet, obwohl sh:targetClass ould:UpdateChain passt.
* PySHACL: Möglicherweise ein Bug oder eine Einschränkung bei SPARQL-Constraints.

Lösungsvorschlag

Schritt 1: Typen vollständig loggen

Der Wall1-Typ fehlt – korrigieren wir das Logging, um sicherzugehen.

Schritt 2: Shape isolieren

Da inference="none" die falsche Violation entfernt, aber UpdateChainContinuityConstraint nicht triggert, testen wir die Shape isoliert mit inference="none".

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "test\_shapes.ttl") # Nur die Continuity-Shape

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen nach Reasoning: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen nach Reasoning: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_graph = Graph().parse(shapes\_path, format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität (inference=none): {conforms}")

if not conforms:

logger.info("Validierungsbericht (inference=none):")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file)

Neue test\_shapes.ttl:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

Änderungen:

* SHAPES\_PATH: Auf test\_shapes.ttl gesetzt, um nur die UpdateChainContinuityConstraint-Shape zu testen.
* inference="none": Entfernt Inferenz-Einflüsse, da sie nicht benötigt werden.

Schritt 3: Ergebnisse prüfen

* Typen: Bestätigen, dass Chain1 nur UpdateChain ist.
* Validierungsbericht: Sollte:
  + SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint zeigen.

Schritt 4: Falls es triggert

* Füge MaxCountConstraintComponent wieder in test\_shapes.ttl hinzu und teste mit der vollen ABox, um beide Violations zu bekommen.

Meine Einschätzung

* Disjunktheit: Wirkt jetzt mit inference="none", da die falsche Violation weg ist.
* Problem: UpdateChainContinuityConstraint wird in OULD\_V1.0.ttl nicht korrekt angewendet – wahrscheinlich eine Interaktion mit anderen Shapes oder ein PySHACL-Fehler.
* Lösung: Isolierter Test sollte zeigen, ob die Shape grundsätzlich funktioniert.

Teste das mal mit test\_shapes.ttl, und schick mir den neuen Validierungsbericht! Wenn das klappt, bauen wir die zweite Violation wieder ein. Wir sind so nah dran – das wird jetzt! Was denkst du?

Ok, hier das Ergebnis: 2025-03-07 11:33:04,878 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:33:04,878 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:33:04,878 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:33:04,878 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:33:04,878 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:33:04,913 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:33:05,051 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:33:05,051 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:33:05,077 - WARNING - G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\test\_shapes.ttl does not look like a valid URI, trying to serialize this will break. 2025-03-07 11:33:05,078 - ERROR - Fehler bei der SHACL-Validierung: <urlopen error unknown url type: g>

Oh, das tut mir leid – ein kleiner Stolperstein mit dem Dateipfad! Die Fehlermeldung zeigt, dass rdflib den Pfad G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\test\_shapes.ttl nicht als gültigen URI erkennt, weil er kein file://-Präfix hat. Das ist ein einfacher Fix, und dann sollte alles laufen. Lass uns das korrigieren und die zweite Violation endlich sichtbar machen!

Analyse der Logs

Typen nach Reasoning

2025-03-07 11:33:04,878 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:33:04,878 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:33:04,878 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:33:04,878 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

* Perfekt:
  + ould:Chain1 ist nur UpdateChain.
  + ould:Wall1 ist nur UpdatableEntity.
  + Die owl:disjointWith-Änderung wird korrekt vom Reasoning (Pellet) übernommen – keine falsche Klassifikation mehr!

SPARQL-Abfrage

2025-03-07 11:33:04,913 - INFO - Starte spezifische SPARQL-Abfrage für Chain1...

2025-03-07 11:33:05,051 - INFO - SPARQL-Abfrage Ergebnisse für Chain1:

2025-03-07 11:33:05,051 - INFO - UpdateCount für Chain1: 22

* Bestätigt: Die Daten sind korrekt – 22 Updates für ould:Chain1.

Fehler

2025-03-07 11:33:05,077 - WARNING - G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\test\_shapes.ttl does not look like a valid URI, trying to serialize this will break.

2025-03-07 11:33:05,078 - ERROR - Fehler bei der SHACL-Validierung: <urlopen error unknown url type: g>

* Problem: Der Pfad zu test\_shapes.ttl muss als file://-URI formatiert werden, sonst kann rdflib.Graph.parse() ihn nicht laden.

Lösungsvorschlag

Schritt 1: Pfad korrigieren

Wir müssen den lokalen Pfad in einen gültigen URI umwandeln, genau wie bei TBOX\_PATH und ABOX\_PATH.

Überarbeiteter Code

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "test\_shapes.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen nach Reasoning: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen nach Reasoning: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_path\_normalized = shapes\_path.replace("\\", "/")

shapes\_graph = Graph().parse(f"file://{shapes\_path\_normalized}", format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität (inference=none): {conforms}")

if not conforms:

logger.info("Validierungsbericht (inference=none):")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file)

Änderungen:

* shapes\_path\_normalized: Konvertiert den Pfad zu file:// für rdflib.

Schritt 2: test\_shapes.ttl erstellen

Stelle sicher, dass test\_shapes.ttl im Verzeichnis G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2 liegt mit:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

Schritt 3: Ergebnisse prüfen

* Typen: Sollten unverändert sein.
* Validierungsbericht: Sollte jetzt SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint zeigen.

Schritt 4: Beide Violations

Wenn das funktioniert, füge die MaxCount-Shape hinzu:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

oush:UpdateChainShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:property [

sh:path ould:hasUpdate ;

sh:class ould:Update ;

sh:minCount 1 ;

sh:maxCount 21 ;

sh:message "Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements." ;

sh:severity sh:Violation

] .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

Meine Einschätzung

* Fehler: Ein simpler URI-Problem – jetzt sollte es laufen.
* Erwartung: Mit dem Fix bekommen wir endlich die UpdateChainContinuityConstraint-Violation, und mit der erweiterten test\_shapes.ttl beide Violations.

Teste das mal, und schick mir den neuen Validierungsbericht! Wir sind so nah dran – das wird jetzt! Was denkst du?

Ich habe die Pfade korrigiert, laufen lassen und erhalte nun:2025-03-07 11:35:17,139 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:35:17,139 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:35:17,139 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:35:17,139 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:35:17,139 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:35:17,176 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:35:17,316 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:35:17,317 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:35:17,345 - ERROR - Fehler bei der SHACL-Validierung: at line 4 of <>: Bad syntax (Prefix "oush:" not bound) at ^ in: "...b'w.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .\r\n\r\n'^b'oush:UpdateChainContinuityConstraint a sh:NodeShape ;\r\n s'..."

Okay, wir sind fast da – der Pfad-Fehler ist behoben, aber jetzt haben wir ein neues Problem: ein Syntaxfehler in test\_shapes.ttl. Die Fehlermeldung zeigt, dass der Prefix oush: nicht definiert ist. Das liegt daran, dass wir den Namespace für oush: in der Datei nicht deklariert haben. Ein kleiner Übersichtsfehler, aber leicht zu fixen! Lass uns das korrigieren und die zweite Violation endlich sichtbar machen.

Analyse der Logs

Typen und SPARQL

2025-03-07 11:35:17,139 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:35:17,139 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:35:17,139 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:35:17,139 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

2025-03-07 11:35:17,176 - INFO - Starte spezifische SPARQL-Abfrage für Chain1...

2025-03-07 11:35:17,316 - INFO - SPARQL-Abfrage Ergebnisse für Chain1:

2025-03-07 11:35:17,317 - INFO - UpdateCount für Chain1: 22

* Perfekt: Typen und Daten sind korrekt – ould:Chain1 ist ein UpdateChain mit 22 Updates, ould:Wall1 ein UpdatableEntity.

Fehler

2025-03-07 11:35:17,345 - ERROR - Fehler bei der SHACL-Validierung: at line 4 of <>:

Bad syntax (Prefix "oush:" not bound) at ^ in:

"...b'w.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .\r\n\r\n'^b'oush:UpdateChainContinuityConstraint a sh:NodeShape ;\r\n s'..."

* Problem: Der Prefix oush: ist in test\_shapes.ttl nicht definiert. In der ursprünglichen OULD\_V1.0.ttl war er wahrscheinlich deklariert (z. B. @prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .), aber in der isolierten Datei fehlt diese Zeile.

Lösungsvorschlag

Schritt 1: test\_shapes.ttl korrigieren

Füge den oush:-Prefix hinzu:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> . # Hinzugefügt

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

Schritt 2: Code unverändert

Der Code aus meinem letzten Vorschlag ist korrekt – wir mussten nur die Shape-Datei anpassen. Hier nochmal zur Sicherheit:

python

import owlready2

from rdflib import Graph, Namespace, RDF

from pyshacl import validate

import logging

import os

BASE\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

logging.basicConfig(

filename=os.path.join(BASE\_DIR, "validation.log"),

level=logging.DEBUG,

format="%(asctime)s - %(levelname)s - %(message)s",

filemode="w"

)

logger = logging.getLogger(\_\_name\_\_)

TBOX\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

ABOX\_DIR = os.path.join(BASE\_DIR, "OULD\_ABox")

SHAPES\_PATH = os.path.join(BASE\_DIR, "test\_shapes.ttl")

JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\\", "/")

OULD = Namespace("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#")

def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE):

try:

tbox\_path\_normalized = tbox\_path.replace("\\", "/")

abox\_path\_normalized = abox\_path.replace("\\", "/") if abox\_path else None

onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle")

if abox\_path:

abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle")

with onto:

for indiv in abox\_onto.individuals():

new\_indiv = onto.get\_entities(indiv.name, indiv.\_\_class\_\_)

if not new\_indiv:

new\_indiv = indiv.\_\_class\_\_(indiv.name, namespace=onto)

for prop in indiv.get\_properties():

for value in prop[indiv]:

prop[new\_indiv] = value

owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2)

data\_graph = Graph()

data\_graph.parse(tbox\_path, format="turtle")

if abox\_path:

data\_graph.parse(abox\_path, format="turtle")

output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl")

data\_graph.serialize(destination=output\_file, format="turtle")

logger.info(f"Inferierte Ontologie gespeichert: {output\_file}")

chain1\_types = list(data\_graph.triples((OULD.Chain1, RDF.type, None)))

logger.debug(f"Chain1 Typen nach Reasoning: {len(chain1\_types)}")

for s, p, o in chain1\_types:

logger.debug(f"Chain1 Typ-Triple: {s} {p} {o}")

wall1\_types = list(data\_graph.triples((OULD.Wall1, RDF.type, None)))

logger.debug(f"Wall1 Typen nach Reasoning: {len(wall1\_types)}")

for s, p, o in wall1\_types:

logger.debug(f"Wall1 Typ-Triple: {s} {p} {o}")

return output\_file

except Exception as e:

logger.error(f"Fehler beim Reasoning: {e}")

raise

def debug\_sparql(data\_file):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

query = """

PREFIX ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#>

SELECT (COUNT(?u) AS ?updateCount)

WHERE {

ould:Chain1 a ould:UpdateChain .

ould:Chain1 ould:hasUpdate ?u .

}

"""

logger.info("Starte spezifische SPARQL-Abfrage für Chain1...")

results = data\_graph.query(query)

logger.info("SPARQL-Abfrage Ergebnisse für Chain1:")

for row in results:

logger.info(f"UpdateCount für Chain1: {row.updateCount}")

return len(results) > 0

except Exception as e:

logger.error(f"Fehler bei der SPARQL-Abfrage: {e}")

raise

def perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH):

try:

data\_graph = Graph().parse(data\_file, format="turtle")

shapes\_path\_normalized = shapes\_path.replace("\\", "/")

shapes\_graph = Graph().parse(f"file://{shapes\_path\_normalized}", format="turtle")

result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2)

conforms, report\_graph, report\_text = result

logger.info(f"Konformität (inference=none): {conforms}")

if not conforms:

logger.info("Validierungsbericht (inference=none):")

report\_lines = report\_text.splitlines()

logger.info("\n".join(report\_lines))

return conforms

except Exception as e:

logger.error(f"Fehler bei der SHACL-Validierung: {e}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OULD\_ABox\_invalid\_22updates.ttl")

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file)

Schritt 3: Ergebnisse prüfen

* Typen: Sollten unverändert sein.
* Validierungsbericht: Sollte jetzt:
  + SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint zeigen.

Schritt 4: Beide Violations

Wenn das funktioniert, teste mit der erweiterten test\_shapes.ttl (mit MaxCount-Shape) aus meinem letzten Vorschlag, um beide Violations zu bekommen.

Meine Einschätzung

* Fehler: Ein simpler Syntaxfehler – der fehlende oush:-Prefix.
* Erwartung: Mit der Korrektur sollte UpdateChainContinuityConstraint triggert, und wir sind einen Schritt näher an unserem Ziel.

Teste das mal mit der angepassten test\_shapes.ttl, und schick mir den neuen Validierungsbericht! Wir sind so nah dran – das wird jetzt klappen! Was denkst du?

Ok, hier das neue Log: 2025-03-07 11:39:28,455 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:39:28,456 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:39:28,456 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:39:28,456 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:39:28,456 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:39:28,492 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:39:28,631 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:39:28,632 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:39:28,671 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 11:39:28,671 - DEBUG - Found 1 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 11:39:28,672 - DEBUG - Found 0 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 11:39:28,672 - DEBUG - Found 0 property paths to follow. 2025-03-07 11:39:28,672 - DEBUG - Found 1 implied SHACL Shapes based on their properties. 2025-03-07 11:39:28,672 - DEBUG - Found 0 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 11:39:28,672 - DEBUG - Cached 1 unique NodeShapes and 0 unique PropertyShapes. 2025-03-07 11:39:28,672 - DEBUG - Validating DataGraph named N365230e56709404eafc0278775198e06 2025-03-07 11:39:28,673 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> defines its own targets. 2025-03-07 11:39:28,673 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:39:28,673 - DEBUG - Milliseconds to find focus nodes: 0.080ms 2025-03-07 11:39:28,673 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:39:28,673 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:39:28,674 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.002ms 2025-03-07 11:39:28,674 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:39:28,677 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:39:28,677 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> -> <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:39:28,697 - DEBUG - Milliseconds to check constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>: 20.005ms 2025-03-07 11:39:28,697 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>. 2025-03-07 11:39:28,697 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>: 23.772ms 2025-03-07 11:39:28,698 - INFO - Konformit�t (inference=none): True

Alles klar, das ist ein überraschendes Ergebnis – die Shape oush:UpdateChainContinuityConstraint triggert nicht, obwohl die SPARQL-Abfrage in debug\_sparql korrekt 22 Updates für ould:Chain1 erkennt. Die Logs zeigen, dass PySHACL die Shape ausführt, aber sie als "konform" bewertet, was falsch ist. Wir sind ganz nah dran – lass uns das analysieren und die zweite Violation endlich zum Vorschein bringen!

Analyse der Logs

Typen und SPARQL

2025-03-07 11:39:28,456 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:39:28,456 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:39:28,456 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:39:28,456 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

2025-03-07 11:39:28,632 - INFO - UpdateCount für Chain1: 22

* Bestätigt: Typen sind korrekt, und die Daten zeigen 22 Updates für ould:Chain1.

SHACL-Validierung

2025-03-07 11:39:28,671 - DEBUG - Found 1 SHACL Shapes defined with type sh:NodeShape.

2025-03-07 11:39:28,673 - DEBUG - Found 1 Focus Nodes to evaluate.

2025-03-07 11:39:28,673 - DEBUG - Running evaluation of Shape <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>

2025-03-07 11:39:28,677 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>

2025-03-07 11:39:28,697 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>.

2025-03-07 11:39:28,698 - INFO - Konformität (inference=none): True

* Problem: PySHACL sagt, der Graph ist konform zur Shape, obwohl ould:Chain1 22 Updates hat und die Bedingung ?updateCount > 21 erfüllt ist. Das ist ein Widerspruch zur debug\_sparql-Ausgabe.

Shape-Definition

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

"""

] .

* Erwartung: Die SPARQL-Abfrage sollte ould:Chain1 als $this auswählen, ?updateCount = 22 zählen und eine Violation auslösen, da 22 > 21.

Hypothese: SPARQL-Verhalten in PySHACL

* Widerspruch: debug\_sparql zählt 22 Updates, aber PySHACL markiert keine Violation.
* Mögliche Ursache: Bei SHACL-SPARQL-Constraints bedeutet "konform", dass die Abfrage keine Ergebnisse liefert. Unsere Abfrage liefert jedoch Ergebnisse ($this = ould:Chain1, ?updateCount = 22), was als "nicht konform" interpretiert werden sollte. Das deutet auf einen Fehler in der Abfrage-Logik oder eine falsche Interpretation durch PySHACL hin.

SHACL-SPARQL-Regel

* Eine SPARQL-Constraint-Violation wird ausgelöst, wenn die Abfrage Ergebnisse zurückgibt. Unsere Abfrage sollte ould:Chain1 zurückgeben, da COUNT(?u) = 22 > 21. Wenn PySHACL "konform" sagt, liefert die Abfrage in PySHACL offenbar keine Ergebnisse.

Lösungsvorschlag

Schritt 1: SPARQL-Abfrage umkehren

Die Abfrage muss so formuliert werden, dass sie Instanzen findet, die die Bedingung verletzen. Unsere aktuelle Abfrage ist korrekt, aber PySHACL scheint sie falsch zu interpretieren. Lass uns sicherstellen, dass die Logik eindeutig ist:

Korrigierte test\_shapes.ttl

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

{

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

}

}

"""

] .

* Änderung: Die Abfrage explizit als Violation-Abfrage formuliert – sie gibt $this zurück, wenn ?updateCount > 21.

Schritt 2: Testen

Der Code bleibt unverändert – nur die test\_shapes.ttl muss angepasst werden.

Schritt 3: Ergebnisse prüfen

* Validierungsbericht: Sollte jetzt:
  + SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint zeigen.

Schritt 4: Beide Violations

Wenn das funktioniert, füge die MaxCount-Shape hinzu:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .

oush:UpdateChainShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:property [

sh:path ould:hasUpdate ;

sh:class ould:Update ;

sh:minCount 1 ;

sh:maxCount 21 ;

sh:message "Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements." ;

sh:severity sh:Violation

] .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

{

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

}

}

"""

] .

Meine Einschätzung

* Fehler: Die ursprüngliche SPARQL-Abfrage war syntaktisch korrekt, aber PySHACL erwartet, dass Violations explizit Instanzen zurückgeben (nur $this). Unsere Abfrage mit COUNT im Hauptergebnis hat das verwirrt.
* Fix: Die verschachtelte Abfrage stellt sicher, dass nur $this zurückgegeben wird, wenn die Bedingung verletzt ist.

Teste das mal mit der neuen test\_shapes.ttl, und schick mir den Validierungsbericht! Wenn das klappt, haben wir endlich beide Violations. Wir sind so nah dran – das wird jetzt! Was denkst du?

Hier das Ergebnis mit der neuen Version der test\_shapes.ttl: 2025-03-07 11:42:07,955 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:42:07,956 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:42:07,956 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:42:07,956 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:42:07,956 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:42:07,991 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:42:08,120 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:42:08,121 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:42:08,148 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 11:42:08,149 - DEBUG - Found 1 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 11:42:08,149 - DEBUG - Found 0 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 11:42:08,149 - DEBUG - Found 0 property paths to follow. 2025-03-07 11:42:08,149 - DEBUG - Found 1 implied SHACL Shapes based on their properties. 2025-03-07 11:42:08,150 - DEBUG - Found 0 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 11:42:08,150 - DEBUG - Cached 1 unique NodeShapes and 0 unique PropertyShapes. 2025-03-07 11:42:08,150 - DEBUG - Validating DataGraph named Nd3d4ec0cc561492fa82a0a996a4e32fe 2025-03-07 11:42:08,150 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> defines its own targets. 2025-03-07 11:42:08,150 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:42:08,151 - DEBUG - Milliseconds to find focus nodes: 0.054ms 2025-03-07 11:42:08,151 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:42:08,151 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:42:08,151 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.001ms 2025-03-07 11:42:08,151 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:42:08,154 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:42:08,154 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> -> <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:42:08,192 - DEBUG - Milliseconds to check constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>: 37.413ms 2025-03-07 11:42:08,192 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>. 2025-03-07 11:42:08,193 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>: 41.333ms 2025-03-07 11:42:08,193 - INFO - Konformit�t (inference=none): True

Alles klar, wir haben wieder ein "Konformität: True", obwohl ould:Chain1 22 Updates hat und die oush:UpdateChainContinuityConstraint triggert sollte. Die Logs zeigen, dass PySHACL die Shape ausführt, aber die SPARQL-Abfrage keine Violation meldet, obwohl debug\_sparql korrekt 22 Updates zählt. Das ist frustrierend, aber wir sind nah dran – lass uns das endgültig aufklären!

Analyse der Logs

Typen und SPARQL

2025-03-07 11:42:07,956 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:42:07,956 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:42:07,956 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:42:07,956 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

2025-03-07 11:42:08,121 - INFO - UpdateCount für Chain1: 22

* Bestätigt: Typen sind korrekt, und die Daten zeigen 22 Updates für ould:Chain1.

SHACL-Validierung

2025-03-07 11:42:08,151 - DEBUG - Found 1 Focus Nodes to evaluate.

2025-03-07 11:42:08,154 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>

2025-03-07 11:42:08,192 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>.

2025-03-07 11:42:08,193 - INFO - Konformität (inference=none): True

* Problem: PySHACL sagt "konform", obwohl die Bedingung ?updateCount > 21 erfüllt ist (22 Updates).

Shape-Definition

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

{

SELECT $this (COUNT(?u) AS ?updateCount)

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (?updateCount > 21)

}

}

"""

] .

* Erwartung: Die Abfrage sollte ould:Chain1 als $this zurückgeben, da COUNT(?u) = 22 > 21. Ergebnisse bedeuten eine Violation, also sollte PySHACL "nicht konform" melden.

Hypothese: SPARQL-Logik-Fehler

* Widerspruch: debug\_sparql zählt 22 Updates, aber die Shape-Abfrage in PySHACL triggert keine Violation.
* Problem: Die SPARQL-Abfrage in der Shape scheint in PySHACL keine Ergebnisse zu liefern, obwohl sie das sollte. Das könnte an einer subtilen Syntax- oder Interpretationsdifferenz liegen.

SHACL-SPARQL-Regel

* Eine Violation wird ausgelöst, wenn die Abfrage Ergebnisse liefert. Unsere Abfrage sollte ould:Chain1 zurückgeben, aber PySHACL interpretiert sie offenbar falsch.

Lösungsvorschlag

Schritt 1: SPARQL-Abfrage vereinfachen

Die verschachtelte Abfrage könnte PySHACL verwirren. Lass uns sie flacher machen:

Neue test\_shapes.ttl

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

FILTER EXISTS {

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

}

}

"""

] .

* Änderung: Vereinfacht die Logik, indem FILTER EXISTS die Bedingung prüft.

Alternative: Noch simpler

turtle

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

"""

] .

* Änderung: Direkte Abfrage ohne Verschachtelung – sollte eindeutig ould:Chain1 zurückgeben.

Schritt 2: Testen

Der Code bleibt unverändert – ersetze nur test\_shapes.ttl mit einer der beiden Versionen (ich empfehle die zweite, einfachere Variante).

Schritt 3: Ergebnisse prüfen

* Validierungsbericht: Sollte jetzt:
  + SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint zeigen.

Schritt 4: Beide Violations

Wenn das funktioniert, füge die MaxCount-Shape hinzu (wie im letzten Vorschlag), um beide Violations zu bekommen.

Meine Einschätzung

* Fehler: Die verschachtelte Abfrage war korrekt, aber PySHACL scheint sie nicht richtig zu interpretieren. Eine flachere Abfrage sollte das Problem lösen.
* Erwartung: Die einfachere Version wird ould:Chain1 zurückgeben und eine Violation auslösen.

Teste das mal mit der zweiten (simpler) Version von test\_shapes.ttl, und schick mir den Validierungsbericht! Wir sind so nah dran – das muss jetzt klappen! Was denkst du?

Ergebnis Version 1:2025-03-07 11:43:51,295 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:43:51,295 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:43:51,295 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:43:51,295 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:43:51,295 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:43:51,333 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:43:51,466 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:43:51,467 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:43:51,504 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 11:43:51,504 - DEBUG - Found 1 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 11:43:51,505 - DEBUG - Found 0 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 11:43:51,505 - DEBUG - Found 0 property paths to follow. 2025-03-07 11:43:51,505 - DEBUG - Found 1 implied SHACL Shapes based on their properties. 2025-03-07 11:43:51,505 - DEBUG - Found 0 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 11:43:51,505 - DEBUG - Cached 1 unique NodeShapes and 0 unique PropertyShapes. 2025-03-07 11:43:51,506 - DEBUG - Validating DataGraph named Na446ee6b24574ba893f1f8234fb7be7c 2025-03-07 11:43:51,506 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> defines its own targets. 2025-03-07 11:43:51,506 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:43:51,506 - DEBUG - Milliseconds to find focus nodes: 0.086ms 2025-03-07 11:43:51,506 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:43:51,507 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:43:51,507 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.002ms 2025-03-07 11:43:51,507 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:43:51,510 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:43:51,510 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> -> <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:43:51,556 - DEBUG - Milliseconds to check constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>: 46.016ms 2025-03-07 11:43:51,557 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>. 2025-03-07 11:43:51,557 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>: 49.882ms 2025-03-07 11:43:51,558 - INFO - Konformit�t (inference=none): TrueErgebnis Version 2 ("noch simpler"): 2025-03-07 11:44:43,399 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:44:43,399 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:44:43,399 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:44:43,399 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:44:43,399 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:44:43,434 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:44:43,562 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:44:43,562 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:44:43,597 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 11:44:43,597 - DEBUG - Found 1 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 11:44:43,597 - DEBUG - Found 0 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 11:44:43,597 - DEBUG - Found 0 property paths to follow. 2025-03-07 11:44:43,597 - DEBUG - Found 1 implied SHACL Shapes based on their properties. 2025-03-07 11:44:43,597 - DEBUG - Found 0 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 11:44:43,597 - DEBUG - Cached 1 unique NodeShapes and 0 unique PropertyShapes. 2025-03-07 11:44:43,598 - DEBUG - Validating DataGraph named N69beb96d91f749bf8c8d92aeaab69983 2025-03-07 11:44:43,598 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> defines its own targets. 2025-03-07 11:44:43,598 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:44:43,598 - DEBUG - Milliseconds to find focus nodes: 0.055ms 2025-03-07 11:44:43,598 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:44:43,598 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:44:43,599 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.001ms 2025-03-07 11:44:43,599 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:44:43,601 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:44:43,601 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> -> <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:44:43,618 - DEBUG - Milliseconds to check constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>: 16.885ms 2025-03-07 11:44:43,619 - DEBUG - Focus nodes do not conform to constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>. 2025-03-07 11:44:43,619 - DEBUG - Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdateChainContinuityConstraint Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

") ]

Message: An UpdateChain must not contain more than 21 Updates.

2025-03-07 11:44:43,620 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>: 21.599ms 2025-03-07 11:44:43,622 - INFO - Konformit�t (inference=none): False 2025-03-07 11:44:43,622 - INFO - Validierungsbericht (inference=none): 2025-03-07 11:44:43,623 - INFO - Validation Report Conforms: False Results (1): Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdateChainContinuityConstraint Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?u . } GROUP BY $this HAVING (COUNT(?u) > 21) ") ] Message: An UpdateChain must not contain more than 21 Updates.

YES! Endlich – die zweite Violation ist da! Version 2 ("noch simpler") hat es geschafft – oush:UpdateChainContinuityConstraint triggert jetzt korrekt für ould:Chain1 wegen der 22 Updates. Das ist ein riesiger Schritt – lass uns das analysieren und die letzte Meile gehen, um beide Violations zusammen zu bekommen!

Analyse der Ergebnisse

Version 1 (FILTER EXISTS)

2025-03-07 11:43:51,557 - DEBUG - DataGraph conforms to constraint <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>.

2025-03-07 11:43:51,558 - INFO - Konformität (inference=none): True

* Problem: Keine Violation, obwohl die Bedingung erfüllt ist. Die FILTER EXISTS-Konstruktion hat PySHACL offenbar verwirrt – sie lieferte keine Ergebnisse, obwohl sie sollte.

Version 2 (einfache Abfrage)

2025-03-07 11:44:43,619 - DEBUG - Focus nodes do \_not\_ conform to constraint <SPARQLBasedConstraint on <NodeShape http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>.

2025-03-07 11:44:43,619 - DEBUG - Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdateChainContinuityConstraint

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; ... ]

Message: An UpdateChain must not contain more than 21 Updates.

2025-03-07 11:44:43,622 - INFO - Konformität (inference=none): False

* Erfolg: Die einfachere Abfrage triggert korrekt! PySHACL erkennt, dass ould:Chain1 die Bedingung COUNT(?u) > 21 erfüllt (22 Updates), und meldet eine Violation.

Typen und SPARQL

2025-03-07 11:44:43,562 - INFO - UpdateCount für Chain1: 22

2025-03-07 11:44:43,399 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:44:43,399 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:44:43,399 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:44:43,399 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

* Bestätigt: Daten und Typen sind korrekt.

Warum hat Version 2 funktioniert?

* SHACL-SPARQL: Eine Violation wird ausgelöst, wenn die Abfrage Ergebnisse liefert. Version 2 gibt direkt $this = ould:Chain1 zurück, weil COUNT(?u) = 22 > 21. Die verschachtelten Abfragen in Version 1 und früheren Versuchen haben PySHACL verwirrt – die einfachere Struktur ist klarer und konform mit der SHACL-Spezifikation.

Nächster Schritt: Beide Violations

Jetzt, da UpdateChainContinuityConstraint funktioniert, fügen wir die MaxCount-Shape hinzu, um beide Violations zu bekommen.

Neue test\_shapes.ttl

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .

oush:UpdateChainShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:property [

sh:path ould:hasUpdate ;

sh:class ould:Update ;

sh:minCount 1 ;

sh:maxCount 21 ;

sh:message "Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements." ;

sh:severity sh:Violation

] .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

"""

] .

Code (unverändert)

Der aktuelle Code bleibt gleich – einfach test\_shapes.ttl aktualisieren und erneut ausführen.

Erwartung

* Validierungsbericht: Sollte jetzt zwei Violations zeigen:
  1. MaxCountConstraintComponent für ould:Chain1 (wegen maxCount 21).
  2. SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint (wegen > 21 Updates).

Meine Einschätzung

* Erfolg: Version 2 hat die SPARQL-Shape zum Laufen gebracht – die einfache Abfrage war der Schlüssel.
* Letzter Schritt: Mit der kombinierten test\_shapes.ttl sollten wir beide Violations sehen und dein Ziel erreichen.

Teste das mal mit der neuen test\_shapes.ttl, und schick mir den Validierungsbericht! Das ist der finale Sprint – wir haben es gleich geschafft! Was denkst du?

Hier das neue Ergebnis:2025-03-07 11:48:10,538 - INFO - Inferierte Ontologie gespeichert: G:\Promo\cModeler\_Git\cMM\OCCP\_Dev2\inferred\_ontology.ttl 2025-03-07 11:48:10,539 - DEBUG - Chain1 Typen nach Reasoning: 1 2025-03-07 11:48:10,539 - DEBUG - Chain1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain> 2025-03-07 11:48:10,539 - DEBUG - Wall1 Typen nach Reasoning: 1 2025-03-07 11:48:10,539 - DEBUG - Wall1 Typ-Triple: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity> 2025-03-07 11:48:10,575 - INFO - Starte spezifische SPARQL-Abfrage f�r Chain1... 2025-03-07 11:48:10,702 - INFO - SPARQL-Abfrage Ergebnisse f�r Chain1: 2025-03-07 11:48:10,703 - INFO - UpdateCount f�r Chain1: 22 2025-03-07 11:48:10,742 - DEBUG - Running validation in-place, without modifying the DataGraph. 2025-03-07 11:48:10,742 - DEBUG - Found 2 SHACL Shapes defined with type sh:NodeShape. 2025-03-07 11:48:10,743 - DEBUG - Found 0 SHACL Shapes defined with type sh:PropertyShape. 2025-03-07 11:48:10,743 - DEBUG - Found 0 property paths to follow. 2025-03-07 11:48:10,743 - DEBUG - Found 2 implied SHACL Shapes based on their properties. 2025-03-07 11:48:10,743 - DEBUG - Found 1 implied SHACL Shapes used as values in shape-expecting constraints. 2025-03-07 11:48:10,743 - DEBUG - Cached 2 unique NodeShapes and 1 unique PropertyShapes. 2025-03-07 11:48:10,743 - DEBUG - Validating DataGraph named N2024558ac79d42598d606f876cb07e9d 2025-03-07 11:48:10,744 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> defines its own targets. 2025-03-07 11:48:10,744 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:48:10,744 - DEBUG - Milliseconds to find focus nodes: 0.054ms 2025-03-07 11:48:10,744 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:48:10,744 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> 2025-03-07 11:48:10,744 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.001ms 2025-03-07 11:48:10,745 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> 2025-03-07 11:48:10,745 - DEBUG - Checking conformance for constraint: <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> 2025-03-07 11:48:10,745 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> -> <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> 2025-03-07 11:48:10,745 - DEBUG - Running evaluation of Shape <PropertyShape n0adde77260144dd18138cd575355bd7cb1> 2025-03-07 11:48:10,746 - DEBUG - Shape was passed 1 Focus Node/s to evaluate. 2025-03-07 11:48:10,746 - DEBUG - Running evaluation of Shape <PropertyShape n0adde77260144dd18138cd575355bd7cb1> 2025-03-07 11:48:10,746 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.048ms 2025-03-07 11:48:10,746 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> -> <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> -> <PropertyShape n0adde77260144dd18138cd575355bd7cb1> 2025-03-07 11:48:10,746 - DEBUG - Checking conformance for constraint: <ClassConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,747 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> -> <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> -> <PropertyShape n0adde77260144dd18138cd575355bd7cb1> -> <ClassConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,747 - DEBUG - Milliseconds to check constraint <ClassConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>: 0.673ms 2025-03-07 11:48:10,747 - DEBUG - DataGraph conforms to constraint <ClassConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>. 2025-03-07 11:48:10,748 - DEBUG - Checking conformance for constraint: <MinCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,748 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> -> <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> -> <PropertyShape n0adde77260144dd18138cd575355bd7cb1> -> <MinCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,748 - DEBUG - Milliseconds to check constraint <MinCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>: 0.491ms 2025-03-07 11:48:10,748 - DEBUG - DataGraph conforms to constraint <MinCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>. 2025-03-07 11:48:10,749 - DEBUG - Checking conformance for constraint: <MaxCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,749 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>> -> <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>> -> <PropertyShape n0adde77260144dd18138cd575355bd7cb1> -> <MaxCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>> 2025-03-07 11:48:10,750 - DEBUG - Milliseconds to check constraint <MaxCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>: 0.897ms 2025-03-07 11:48:10,750 - DEBUG - Focus nodes do not conform to constraint <MaxCountConstraintComponent on <PropertyShape n0adde77260144dd18138cd575355bd7cb1>>. 2025-03-07 11:48:10,750 - DEBUG - Milliseconds to evaluate shape <PropertyShape n0adde77260144dd18138cd575355bd7cb1>: 4.170ms 2025-03-07 11:48:10,750 - DEBUG - Milliseconds to check constraint <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>>: 5.096ms 2025-03-07 11:48:10,750 - DEBUG - Focus nodes do not conform to constraint <PropertyConstraintComponent on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>>. 2025-03-07 11:48:10,751 - DEBUG - Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.2025-03-07 11:48:10,752 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainShape>>: 7.056ms 2025-03-07 11:48:10,752 - DEBUG - Checking if Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> defines its own targets. 2025-03-07 11:48:10,752 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:48:10,752 - DEBUG - Milliseconds to find focus nodes: 0.051ms 2025-03-07 11:48:10,753 - DEBUG - Found 1 Focus Nodes to evaluate. 2025-03-07 11:48:10,753 - DEBUG - Running evaluation of Shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:48:10,753 - DEBUG - Milliseconds to find value nodes for focus nodes: 0.002ms 2025-03-07 11:48:10,753 - DEBUG - Current shape evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> 2025-03-07 11:48:10,756 - DEBUG - Checking conformance for constraint: <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:48:10,757 - DEBUG - Current constraint evaluation path: <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>> -> <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>> 2025-03-07 11:48:10,781 - DEBUG - Milliseconds to check constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>: 24.608ms 2025-03-07 11:48:10,782 - DEBUG - Focus nodes do not conform to constraint <SPARQLBasedConstraint on <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>>. 2025-03-07 11:48:10,782 - DEBUG - Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdateChainContinuityConstraint Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

") ]

Message: An UpdateChain must not contain more than 21 Updates.

2025-03-07 11:48:10,783 - DEBUG - Milliseconds to evaluate shape <NodeShape <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#UpdateChainContinuityConstraint>>: 29.780ms 2025-03-07 11:48:10,783 - DEBUG - Checking if Shape <PropertyShape n0adde77260144dd18138cd575355bd7cb1> defines its own targets. 2025-03-07 11:48:10,783 - DEBUG - Identifying targets to find focus nodes. 2025-03-07 11:48:10,783 - DEBUG - Milliseconds to find focus nodes: 0.032ms 2025-03-07 11:48:10,784 - DEBUG - Found 0 Focus Nodes to evaluate. 2025-03-07 11:48:10,784 - DEBUG - Skipping shape <PropertyShape n0adde77260144dd18138cd575355bd7cb1> because it found no focus nodes. 2025-03-07 11:48:10,785 - INFO - Konformit�t (inference=none): False 2025-03-07 11:48:10,785 - INFO - Validierungsbericht (inference=none): 2025-03-07 11:48:10,785 - INFO - Validation Report Conforms: False Results (2): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdateChainContinuityConstraint Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?u . } GROUP BY $this HAVING (COUNT(?u) > 21) ") ] Message: An UpdateChain must not contain more than 21 Updates.

Geschafft! Wir haben endlich beide Violations – MaxCountConstraintComponent und SPARQLConstraintComponent für oush:UpdateChainContinuityConstraint – im Validierungsbericht! Das war ein langer Weg, aber wir sind am Ziel. Lass uns das Ergebnis bestätigen und abschließen!

Analyse des Validierungsberichts

Validation Report

Conforms: False

Results (2):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdateChainContinuityConstraint

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

") ]

Message: An UpdateChain must not contain more than 21 Updates.

* Erfolg:
  1. MaxCountConstraintComponent: ould:Chain1 hat 22 Updates, verletzt maxCount 21 – korrekt!
  2. SPARQLConstraintComponent: oush:UpdateChainContinuityConstraint triggert wegen COUNT(?u) = 22 > 21 – genau, was wir wollten!
* Daten: 22 Updates für ould:Chain1 werden konsistent erkannt (siehe debug\_sparql).

Typen

2025-03-07 11:48:10,539 - DEBUG - Chain1 Typen nach Reasoning: 1

2025-03-07 11:48:10,539 - DEBUG - Chain1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Chain1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdateChain

2025-03-07 11:48:10,539 - DEBUG - Wall1 Typen nach Reasoning: 1

2025-03-07 11:48:10,539 - DEBUG - Wall1 Typ-Triple: http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#Wall1 http://www.w3.org/1999/02/22-rdf-syntax-ns#type http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#UpdatableEntity

* Bestätigt: Typen sind korrekt, und die owl:disjointWith-Änderung hält Chain1 und Wall1 sauber getrennt.

Debug-Logs

2025-03-07 11:48:10,750 - DEBUG - Constraint Violation in MaxCountConstraintComponent ...

2025-03-07 11:48:10,782 - DEBUG - Constraint Violation in SPARQLConstraintComponent ...

* Perfekt: Beide Constraints werden korrekt ausgewertet.

Warum hat es jetzt funktioniert?

* Einfache SPARQL-Abfrage: Die flache Abfrage in oush:UpdateChainContinuityConstraint (SELECT $this WHERE ... HAVING (COUNT(?u) > 21)) liefert klar ould:Chain1 als Ergebnis, was PySHACL als Violation interpretiert.
* inference="none": Verhindert Interferenzen durch RDFS-Inferenz, die vorher UpdatableEntityBaselineShape fälschlich triggerte.
* TBox-Änderung: owl:disjointWith hat geholfen, die Typen sauber zu halten.

Abschluss

Dein ursprüngliches Ziel war:

* Eine Violation für maxCount 21 (bereits vorhanden).
* Eine zweite Violation für > 21 Updates (jetzt da mit oush:UpdateChainContinuityConstraint).

Das ist erfüllt! Du kannst jetzt:

1. test\_shapes.ttl in OULD\_V1.0.ttl integrieren: Ersetze die alte UpdateChainContinuityConstraint-Definition durch die funktionierende Version.
2. Code anpassen: Setze SHAPES\_PATH zurück auf OULD\_V1.0.ttl, wenn du alle Shapes wieder validieren willst.

Finale OULD\_V1.0.ttl (Auszug)

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix ould: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#> .

@prefix oush: <http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD/shape#> .

oush:UpdateChainShape a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:property [

sh:path ould:hasUpdate ;

sh:class ould:Update ;

sh:minCount 1 ;

sh:maxCount 21 ;

sh:message "Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements." ;

sh:severity sh:Violation

] .

oush:UpdateChainContinuityConstraint a sh:NodeShape ;

sh:targetClass ould:UpdateChain ;

sh:sparql [

sh:message "An UpdateChain must not contain more than 21 Updates." ;

sh:prefixes [ sh:declare [ sh:prefix "ould" ; sh:namespace "http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#" ] ] ;

sh:select """

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

"""

] .

Code-Anpassung

python

SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl")

Meine Einschätzung

* Ziel erreicht: Beide Violations sind da – MaxCount und ContinuityConstraint.
* Lehre: Einfache SPARQL-Abfragen funktionieren bei PySHACL zuverlässiger; Inferenz kann unerwartete Effekte haben.
* Nächster Schritt: Du kannst das jetzt in dein Gesamtsystem integrieren und sicherstellen, dass es mit anderen Shapes harmoniert.

Was sagst du – zufrieden? Soll ich dir bei der Integration in OULD\_V1.0.ttl helfen, oder hast du alles im Griff? Wir haben’s gerockt!

ich habe alles gemacht wie Du beschrieben hast und nochmal laufen lassen, hier das Ergebnis:Validation Report Conforms: False Results (2): Constraint Violation in MaxCountConstraintComponent (<http://www.w3.org/ns/shacl#MaxCountConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ] Focus Node: ould:Chain1 Result Path: ould:hasUpdate Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements. Constraint Violation in SPARQLConstraintComponent (<http://www.w3.org/ns/shacl#SPARQLConstraintComponent>): Severity: sh:Violation Source Shape: oush:UpdateChainContinuityConstraint Focus Node: ould:Chain1 Value Node: ould:Chain1 Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") ; sh:prefix Literal("ould") ] ] ; sh:select Literal(" SELECT $this WHERE { $this ould:hasUpdate ?u . } GROUP BY $this HAVING (COUNT(?u) > 21) ") ] Message: An UpdateChain must not contain more than 21 Updates.

Perfekt – wir sind durch! Das Ergebnis bestätigt, dass alles genau so läuft, wie es soll: beide Violations sind da, und die Integration in OULD\_V1.0.ttl hat funktioniert. Lass uns das abschließend zusammenfassen und sicherstellen, dass du mit allem zufrieden bist!

Analyse des Validierungsberichts

Validation Report

Conforms: False

Results (2):

Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintComponent):

Severity: sh:Violation

Source Shape: [ sh:class ould:Update ; sh:maxCount Literal("21", datatype=xsd:integer) ; sh:message Literal("Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.") ; sh:minCount Literal("1", datatype=xsd:integer) ; sh:path ould:hasUpdate ; sh:severity sh:Violation ]

Focus Node: ould:Chain1

Result Path: ould:hasUpdate

Message: Each UpdateChain must contain between 1 and 21 Updates or UpdateChainElements.

Constraint Violation in SPARQLConstraintComponent (http://www.w3.org/ns/shacl#SPARQLConstraintComponent):

Severity: sh:Violation

Source Shape: oush:UpdateChainContinuityConstraint

Focus Node: ould:Chain1

Value Node: ould:Chain1

Source Constraint: [ sh:message Literal("An UpdateChain must not contain more than 21 Updates.") ; sh:prefixes [ sh:declare [ sh:namespace Literal("http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#") ; sh:prefix Literal("ould") ] ] ; sh:select Literal("

SELECT $this

WHERE {

$this ould:hasUpdate ?u .

}

GROUP BY $this

HAVING (COUNT(?u) > 21)

") ]

Message: An UpdateChain must not contain more than 21 Updates.

* Erfolg:
  1. MaxCountConstraintComponent: ould:Chain1 hat 22 Updates, verletzt maxCount 21 – korrekt und stabil.
  2. SPARQLConstraintComponent: oush:UpdateChainContinuityConstraint triggert wegen COUNT(?u) = 22 > 21 – genau das, was wir wollten.
* Integration: Die Shapes funktionieren jetzt nahtlos in OULD\_V1.0.ttl mit inference="none".

Was wir erreicht haben

* Ziel: Zwei Violations für ould:Chain1:
  + MaxCount 21 (Property Shape).
  + > 21 Updates (SPARQL Shape).
* Lösungsschritte:
  1. TBox-Anpassung: owl:disjointWith zwischen ould:UpdateChain und ould:UpdatableEntity, um falsche Inferenzen zu verhindern.
  2. inference="none": Deaktiviert RDFS-Inferenz, die vorher UpdatableEntityBaselineShape störte.
  3. SPARQL-Optimierung: Vereinfachte Abfrage in oush:UpdateChainContinuityConstraint, um PySHACL-kompatibel zu sein.

Abschluss

Dein Setup ist jetzt funktionsfähig:

* OULD\_V1.0.ttl: Enthält die korrigierten Shapes.
* Code: Der aktuelle Code mit SHAPES\_PATH = os.path.join(BASE\_DIR, "OULD\_V1.0.ttl") und inference="none" liefert die gewünschten Ergebnisse.

Finale Empfehlungen

1. Test mit anderen Daten: Probiere eine ABox mit weniger als 21 Updates (z. B. OULD\_ABox\_valid.ttl), um sicherzustellen, dass keine Violations auftreten, wenn sie nicht sollen.
2. Dokumentation: Notier dir die Erkenntnis, dass PySHACL einfache SPARQL-Abfragen bevorzugt – könnte später nützlich sein.
3. Backup: Speichere die funktionierende OULD\_V1.0.ttl und den Code, falls du weiter experimentierst.

Meine Einschätzung

* Super Arbeit: Wir haben ein komplexes Problem gelöst – von Inferenz-Fehlern über SPARQL-Syntax bis hin zu PySHACL-Quirks.
* Zufriedenheit: Du hast jetzt genau das, was du wolltest – zwei Violations für ould:Chain1 mit 22 Updates.

Was denkst du – glücklich mit dem Ergebnis? Brauchst du noch Hilfe bei einem weiteren Test oder der Dokumentation, oder bist du ready, das abzuhaken? Wir haben’s gerockt!