République Algérienne Démocratique et Populaire
Ministère de l'enseignement Supérieur et de la Recherche Scientifique
Université de Mohamed El Bachir El Ibrahimi de Bordj Bou Arréridj
Faculté des Mathématiques et d'Informatique
Département d'informatique

MEMOIRE

Présenté en vue de l'obtention du diplôme

Master en informatique

Spécialité : Réseaux & Multimédia

THEME

Étude de cas avec le langage de requêtes ShACL

Présenté par :

Boukhelifa Abdeslam

Mansouri Mounir

Soutenu publiquement le :21 /06/2023

Devant le jury composé de:

Président : Zitouni Sihem

Examinateur : Khelifi Hakima

Encadreur : Sabri Lyazid

2022/2023

Dédicace

A mes très chers parents, je dédie ce travail Aucun mot n'est assez fort pour leur exprimer la reconnaissance sincère que je leur porte pour la richesse de leurs enseignements. A ma très chère grande mère A mon très cher frère Et à toute ma famille de Mansouri et Mebarek

Dédicace

A mes très chers parents, je dédie ce travail Aucun mot n'est assez fort pour leur exprimer la reconnaissance sincère que je leur porte pour la richesse de leurs enseignements. A ma très chère grande mère A mon très cher frère et soeur Et à toute ma famille de Boukhelifa

Remerciement

« Louange à Allah qui nous a guidés à ceci. Nous n'aurions pas été guidés, si Allah ne nous avait pas guidés »

[Sourate 7. Al Araf verset 43]

Mes remerciements les plus sincères à toutes les personnes qui ont contribué de près ou de loin à l'élaboration de cette mémoire ainsi qu'à la réussite de cette formidable année universitaire. Je remercie piètrement Allah le tout puissant de m'avoir donné le courage et la volonté de mener à terme ce présent travail. J'adresse mes vifs remerciements : A mon encadrant Mr. Sabri Lyazid pour son encadrement, son soutien sans failles et sa disponibilité. Ses conseils, ses suggestions de lecture, ses commentaires, se s corrections et ses qualités scientifiques ont été très précieux pour mener à bien ce travail. Je tiens également à remercier et exprimer mon profond respect aux membres de jury d'avoir accepté de juger ce travail

Résumé

Les observations environnementales sont essentielles pour mesurer les paramètres physiques d'une personne et/ou la détection de données environnementales. L'avènement de RDF et XML a rendu donc possible d'encoder ou d'annoter un contenu Web d'une manière plus compréhensible par la machine. Ce qui permet une rapidité d'évolution vers une riche base de connaissances. Par conséquent, fournira un moyen à la création de nombreuses nouvelles applications intéressantes, dont l'une des plus importantes sera la recherche et la récupération précises de contenu. La question qui se pose est comment extraire des données tout en garantissant la qualité et la cohérence des données RDF. Ainsi, valider un graphe RDF est considéré comme un filtre indispensable pour vérifier si les instances (données) sont conformes à une définition de schéma. Ce qui permet à un concepteur de se focaliser sur la modélisation que sur le modèle d'ontologie dédié.

Shapes Constraint Language (ShACL) comme standard du W3C pour la validation des graphes RDF décrit deux graphes, le graphe de données et le graphe de formes. Le graphe RDF à valider est le graphe de données, tandis que l'ensemble des contraintes, exprimées également en RDF et décrivant la forme d'un graphe, est connu sous le nom de graphe de formes. Ce graphe de formes décrit formellement l'ensemble des conditions que le graphe de données doit satisfaire.

Mot clefs : RDF, ShACL, Graphe, Web Sémantique, Validation de données.

Abstract

Environmental observations are essential for measuring a person's physical parameters and/or detecting environmental data. Therefore, the advent of RDF and XML has made it possible to encode or annotate web content in a more machine-readable way. Thus, it allows a rapid evolution towards a rich knowledge base. Therefore, it will provide a means to create many exciting new applications, one of the most important of which will be precise content search and retrieval. The question that arises is how to extract data while ensuring the quality and consistency of RDF data. Thus, the validation of an RDF graph is an indispensable filter to check if the instances (data) conform to a schema definition and allows a designer to focus on modeling only the dedicated ontology model.

Shapes Constraint Language (SHACL) is a W3C standard for validating RDF graphs and describes two graphs, data and shape graphs. The RDF graph to be validated is the data graph, while the set of constraints, also expressed in RDF and describing the shape of a graph, is called shape graph. This shape graph formally describes the set of conditions that the data graph must satisfy.

Keywords: RDF, ShACL, Graph, Semantic Web, Data Validation.

ملخص

تعتبر الملاحظات البيئية ضرورية لقياس المعلمات الفيزيائية للشخص و / أو الكشف عن البيانات البيئية.لذلك ، أتاح ظهور RDF و XML إمكانية تشفير محتوى الويب أو التعليق عليه بطريقة أكثر قابلية للقراءة آليًا.وبالتالي ، فإنه يسمح بالتطور السريع نحو قاعدة معرفية غنية.لذلك، سيوفر وسيلة لإنشاء العديد من التطبيقات الجديدة والمثيرة، ومن أهمها البحث الدقيق واسترجاع المحتوى.

السؤال الذي يطرح نفسه هو كيفية استخراج البيانات مع ضمان جودة واتساق بيانات RDF .كما سنرى في دراستنا ، فإن التحقق من صحة البيانات واعد للغاية ، وهو ما يفسر تفاني العديد من الباحثين

إن لغة قيد الأشكال (ShACL) هي معيار W3C للتحقق من صحة الرسوم البيانية RDF وتصف رسمين بيانيين ، الرسوم البيانية للبيانات والشكل.الرسم البياني RDF المراد التحقق من صحته هو الرسم البياني للبيانات ، بينما تُعرف مجموعة القيود ، التي يتم التعبير عنها أيضًا في RDF وتصف شكل الرسم البياني ، بشكل جماعي باسم الرسم البياني للشكل.يصف الرسم البياني للشكل هذا بشكل رسمي مجموعة الشروط التي يجب أن يستوفيها الرسم البياني للبيانات.

الكلمات الأساسية : RDF. ShACL، الرسم البياني ، الويب الدلالي ، التحقق من صحة البيانات.

Table des matières

Chapitre 1: Sommaire

Liste des abréviations

CWA	Closed World Assumption
FOAF	Friend Of A Friend
IOT	Internet Of Things
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
OWL	Ontology Web Language
OWA	Open World Assumption
RDF	Resource Description Framework
RDF(S)	Resource Description Framework Schema
SPARQL	Simple Protocol And RDF Query Language
SQL	Structured Query Language
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

Liste des figures

Liste des tableaux

Chapitre 1: Introduction Générale

Ce travail que nous présentons dans ce mémoire est une suite du travail d'une étude menée lors de l'année 2022-2023 par nos collègues Bahloul Ouahchia et Saoudi Nour El Houda. Les deux étudiantes ont étudiées le langage de requêtes SPARQL (SPARQL Protocol and RDF Query Language) et ses extensions existantes.

1.1. Contexte

La collecte, le traitement et l'analyse des flux de données émanant des capteurs, des informations issues de textes, des réseaux sociaux, etc. est devenu une tâche courante dans de nombreuse solutions industriels ou académiques. Dans le contexte de l'Internet des Objets (IoD) et les dossiers médicaux, les données sont hétérogènes, c'est à dire provenant de domaines et de format de données différents. Cela impose aux applications de l'IoD de gérer efficacement l'intégration de données à partir de ressources diverses. Le traitement des flux en particulier et les modèles ontologiques en général est dès lors devenu un domaine de recherche important. Ainsi, l'apparition du terme Big Data n'est pas un hasard, puisqu'il décrit des données volumineuses, la vélocité par laquelle les données sont générées et la variété de données supprimant en conséquence la simplicité de traitement de données traditionnelles. Bien que le volume et la vitesse soient des caractéristiques clés, la diversification est un enjeu majeur empêchant l'intégration des données et génère de nombreux problèmes. Le modèle (Resource Description Framework (RDF), comme un modèle de données basé sur des graphes est devenu une partie de la vision du Web sémantique, a été proposé pour palier au problème d'hétérogénéité de données.

Lors de la première étude, les étudiantes ont fait un constat que de nombreuses fonctionnalités importantes demeurent sans réponses dans de nombreux cas, en particulier lorsqu'il s'agit d'extraire la notion spatiale ou temporelle. Toutefois, force de constater lors de notre étude que l'adoption de RDF au sein de la communauté de chercheurs, exige un effort supplémentaire pour mieux introduire RDF dans le développement et de publication Web. Par exemple, les consommateurs de données, humains ou machines, sont appelés à connaitre les

structure de données, les propriétés, les valeurs associées aux nœuds RDF. Ainsi, la validation des schémas RDF est plus que nécessaire avant même le traitement des données pour des raisons diverses telles que la sécurité, la véracité, les performances, etc.

1.2. Objectifs

Il s'agit de décrire les technologies de description et de validation qui permet aux humains/machines de décrire le schéma souhaité des données et de vérifier si certaines données existantes sont conformes à ce schéma. De nombreux outils les plus utilisés pour définir et valider les données s'appuient sur la notion de grammaire. Cette dernière telle que XML Schéma est considérée comme convention formelle précisant des groupes de propriété, des relations existantes entre les structures, représenter des types de données et des cardinalités. Il est clair donc que la complexité d'une grammaire croît avec la complexité du langage. Dans le cas de RDF qui est considéré comme un modèle de graphe simple pour décrire des données sous formes de triplets, quant à SPARQL, nécessite une grammaire pour expliciter des contraintes sur des données RDF. Cette grammaire serait utile pour s'interfacer avec les utilisateurs, en particulier pour la validation des données lors de leurs extractions. Ce travail sera un support pour garantir la sécurité, la validation de données dans des applications sensibles. Les futurs étudiants pourront facilement appréhender via des exemples comment ces deux langages permettent de valider les données collecter à partir de capteurs, dossiers médicaux, etc.Il s'agit de vérifier si certaines données existantes sont conformes à attente des application qui vont les utiliser.

1.3. Méthodologie et résultats

Deux langages de haut niveau pour la validation RDF. Il s'agit de ShEx[1] (Shape Expressions) et ShACL[2](Shapes Constraint Language). ShEx a été au départ utilisé pour qu'il

/http://shex.io[1]
/https://www.w3.org/TR/shacl[2]

soit lisible par un humain, ensuite il s'est développé pour des exigences plus complexes tel que le domaine médical. En 2017 ShEx 2.0 est retenu en tant que brouillon du groupe communautaire W3C (World Wide Web Consortium) et en 2019, a été adopté par Wikidata. Quant à ShACL est accepté comme recommandation du W3C. Il permet de valider les graphes RDF (nommées dans ce cas graphes de données) par rapport à un ensemble de conditions. Notre travail est une couche supérieure à la première étude réalisée l'année dernière, et qui va sans aucun doute permettre une bonne prise en main pour la réalisation d'applications d'envergures.

1.4. Structure du rapport

Nous allons présenter sans s'attarder dans le second chapitre le modèle RDF et les technologies qui lui sont associées. Ensuite, dans le chapitre trois, nous allons décrire de manière assez claire en quoi consiste le problème de validation de données. Pour ce faire, nous allons nous appuyer principalement sur le langage ShACL puisqu'il est le langage recommandé par le W3C. Ce langage sera décrit dans le chapitre quatre. Enfin, le dernier chapitre est dédié à l'ontologie et l'étude développée.

Chapitre 2: **Schémas de Validation de données**

2.1 Introduction

Tout programme en intelligence artificiel nécessite des données, généralement structurées. Cette structure doit être facilement partageable entre humain ou machines et interprétable explicitement pour être communiquée via les différentes technologies. S'appuyer sur un langage naturel peut s'avérer efficace, mais vue les développements actuels, un traitement du langage naturel demeure encore ambigu et est difficile à traiter par les machines. En effet, l'humain peut facilement comprendre la sémantique des données. De plus, peut facilement détecter les redondances d'informations dans documents. Bien que le RDF est largement utilisé comme un langage d'ontologie pour la représentation de données, cependant d'autres langages existent tels que XML, JSON, CSV, UML, etc. par exemple, UML contient quatorze types de diagrammes, qui sont classés en trois catégories : structure, comportement et interaction. Le diagramme le plus répondu est le diagramme de classes UML, qui définit la structure logique d'un système en termes de classes et de relations ces classes. UML permet également de spécifier les cardinalités entre les classes. Ce qui permet de vérifier et de valider les relations entre classes en cas d'erreur. Certainement l'accès le plus connu aux données relationnelles est Structured Query Language (SQL). SQL est conçu pour enrichir la base de données et en extraire des données tabulaires. Souvent, des contraintes d'intégrité référentielle sont utilisées pour une liaison cohérente entre les tables. Un schéma de base de données est introduit pour décrire les enregistrements, les champs et les ensembles du modèle de données utilisateur incluant des valeurs numériques telles que des entiers, des caractères et des dates.

Dans ce qui suit, nous allons présenter de manière non exhaustive les structures de données utilisées pour la représentation des informations. Nous allons beaucoup plus montrer comment et par quel moyen la validation des données sont réalisées. Il s'agit des décrire les technologies de description et de validation qui permet aux humains/machines de décrire le schéma souhaité des données et de vérifier si certaines données existantes sont conformes à ce schéma.

2.2 XML (Extensible Markup Language)

XML[3] a été proposé par le W3C comme langage de balisage extensible pour le Web vers 1996. XML est un métalangage qui fournit une syntaxe commune pour les systèmes de balisage textuel. Le modèle XML se compose d'une structure arborescente, où chaque nœud de l'arbre est défini comme étant un élément d'information d'un type particulier. Un élément a un ensemble d'éléments d'attribut et une liste d'éléments enfants ou de nœuds de texte. Les éléments d'attribut peuvent contenir des éléments de caractère ou ils peuvent contenir des données typées telles que des identificateurs et des références. Les identifiants et références d'éléments peuvent être utilisés pour connecter des nœuds transformant l'arbre sous-jacent en un graphe. Le

Tableau 2.1 Document XML représentant une liste d'étudiants sous forme de structure.

présente un exemple de document XML. Quant à la Figure 2.1 présente l'arborescence du même document.

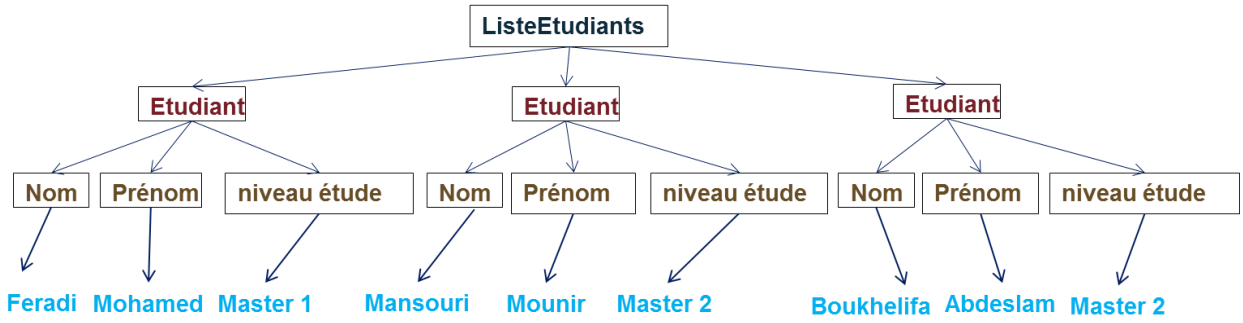


Figure 2.1 Représentation sous forme d'arbre des données du Tableau 2.1

Le langage XML est un méta-langage de balisage et universel. Il fournit un formalisme et des outils pour l'échange de données à travers le web. Toutefois, XML ne fournit aucun moyen pour ajouter de la sémantique aux données. En effet, le modèle XML permet de définir d'un côté une grammaire (syntaxe) pour structurer des éléments d'un document, et d'un autre côté n'impose aucune condition sur le choix des termes (vocabulaire) pour décrire les données d'un domaine.

/https://www.w3.org/XML[3]

```
<?xml version="1.0" encoding="UTF-8"?>
<listeEtudiants>
<!--  La liste des étudiants -->
<etudiant>
<nom>Feradi</nom>
<prenom>Mohamed</prenom>
<niveau>Master 1</niveau>
</etudiant>
<etudiant>
<nom>Mansouri</nom>
<prenom>Mounir </prenom>
<niveau>Master 2 Recherche</niveau>
</etudiant>
<etudiant>
<nom>Boukhelifa</nom>
<prenom>Abdeslam</prenom>
      niveau>Master 2 Recherche</niveau>
</etudiant>
</listeEtudiants>
```

Tableau 2.1 Document XML représentant une liste d'étudiants sous forme de structure.

2.3 Validation des données d'un document XML

Les documents bien formés XML sont des documents XML avec une syntaxe correcte tandis que les documents valides sont des documents qui, en plus d'être bien formés, sont conformes à une définition de schéma (dite de grammaire). Cette grammaire peut être définit selon divers possibilité telles que Définition du type de Document (DTD) , RelaxNG[4] ou Schéma XML[5]. Ces dernières permettent de définir une famille de document XML. La grammaire XML Schéma étant plus puissante que les DTD sont de fait les plus utilisées. Quant à RELAX NG est une évolution et une généralisation des DTD XML. Il partage le même paradigme basé sur la grammaire. Parmi les avantages majeurs de RELAX NG est que : Les DTD XML peuvent être

/https://relaxng.org[4]
/https://www.w3.org/TR/xmlschema11-1[5]

automatiquement converties en RELAX NG. Un objectif majeur de RELAX NG est qu'il soit facile à apprendre et facile à utiliser et permet de suivre la structure du document.

Un validateur de schéma XML définit chaque structure du document XML avec des informations supplémentaires appelées ensemble d'informations de validation post-schéma. Cette structure contient des informations sur le processus de validation qui peuvent ensuite être utilisées par d'autres outils XML. Pour comprendre la nécessité de validation de données véhiculées via le réseau, considérons les deux exemples suivants :

1. Exemple 1 : Contrôle de maison intelligente avec le paradigme Internet des Objets.

La Figure 2.2 présente une grammaire pour la création d'une structure de données XML. Les éléments *homeAutomation* et *location* par exemple doivent être respectivement des sous éléments de *environment* et *components*. Par conséquent, toute structure qui ne respecte pas cette arborescence sera rejetée. Quant à la Figure 2.3 illustre clairement l'intérêt de validation des données. Ainsi, l'outil alerte l'utilisateur que le document en cours de création n'est pas valide, tandis que le second document est conforme à la structure imposé par la grammaire XML Schema.

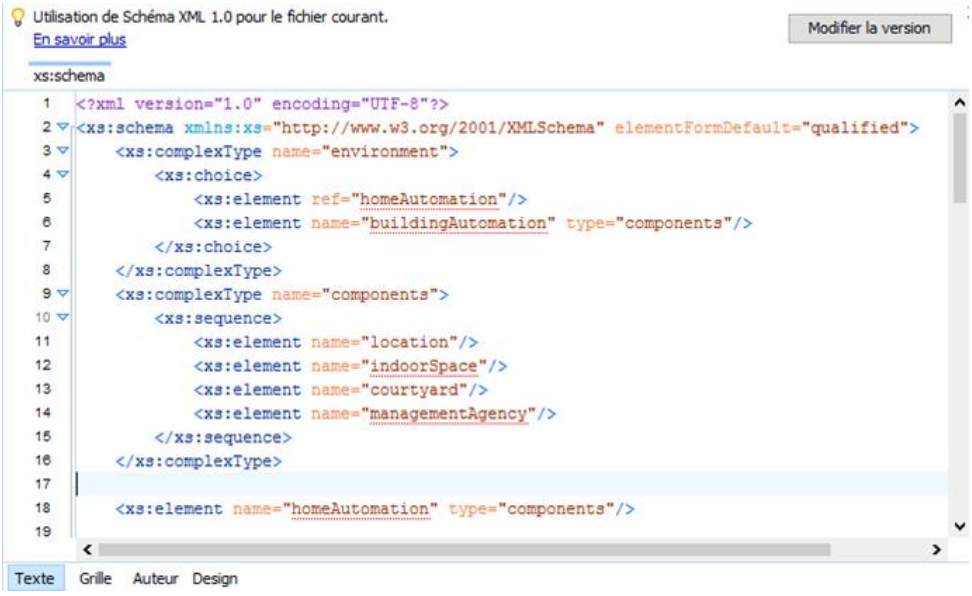


Figure 2.2 Document XML Schema pour la création d'espace de localisation créé avec l'outil Oxgen https://www.oxygenxml.com/

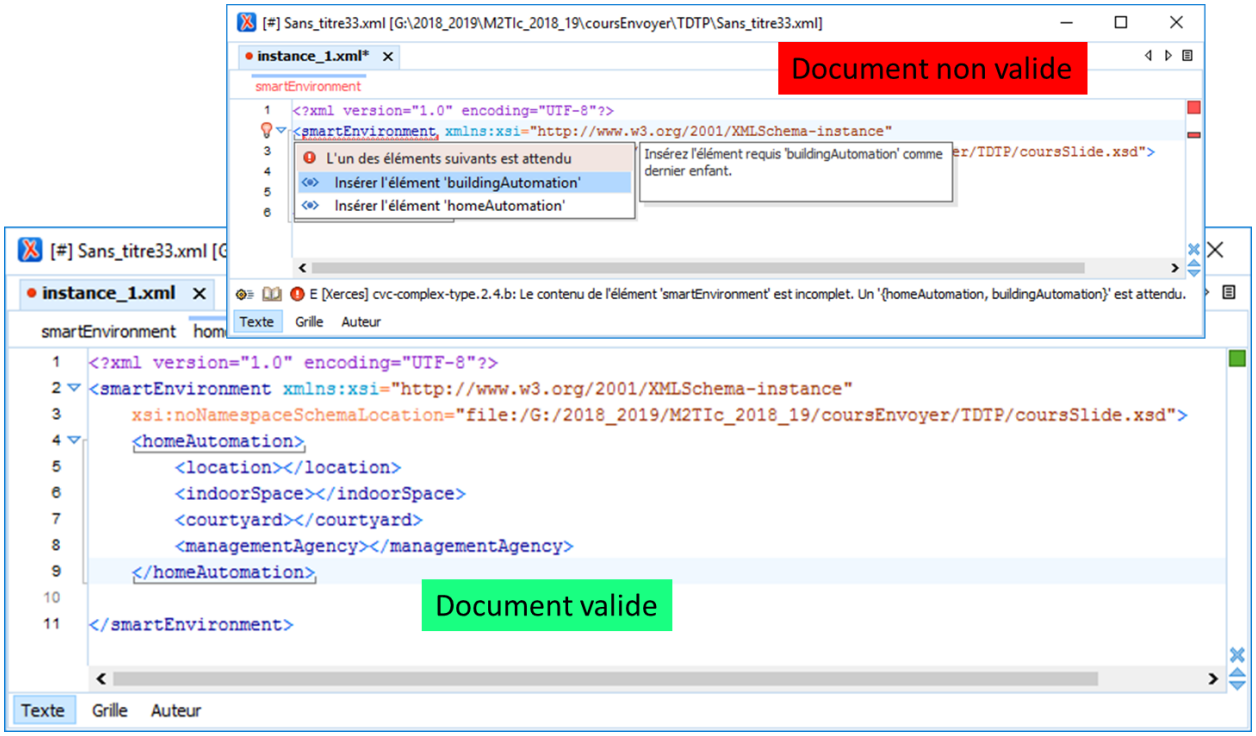


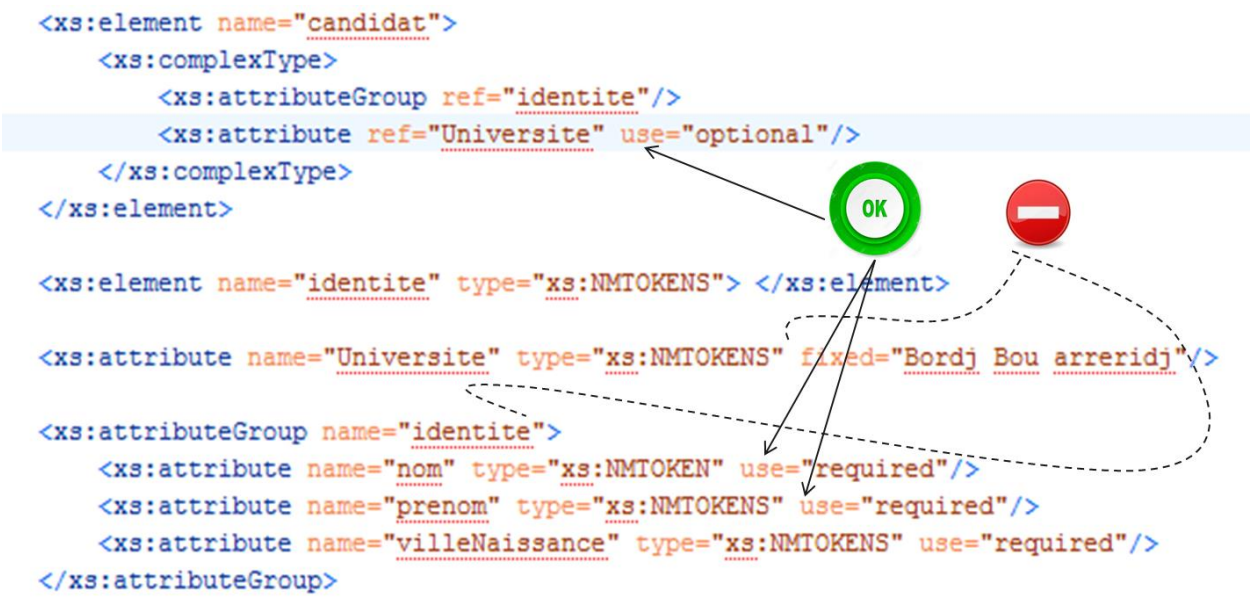
Figure 2.3 Validation des données au moment de la création. Les deux documents doivent être conforment au document XML Schema Figure 2.2

1. Exemple 2 : Validation d'un dossier de candidat

Considérons les données véhiculées par un document XML :

<candidat **prenom**="n" **villeNaissance**="fak" **nom**="ML" >< /candidat>

Bien que les éléments prénom, villeNaissance aient un sens pour un être humain, toutefois, la valeur des contenus n'a aucun sens. La Figure 2.4 présente une définition de schéma XML permettant de vérifier la conformité d'un candidat. Remarquons ici, que cette grammaire exige que le candidat soit de l'université de Bordj Bou Arreridj. De plus, contrôle uniquement le type des valeurs mais ne contrôle pas les valeurs des champs.




```
<xs:element name="candidat">
    <xs:complexType>
        <xs:attributeGroup ref="identite"/>
        <xs:attribute ref="Universite" use="optional"/>
    </xs:complexType>
</xs:element>

<xs:element name="identite" type="xs:NMTOKENS"> </xs:element>

<xs:attribute name="Universite" type="xs:NMTOKENS" fixed="Bordj Bou arreridj"/>

<xs:attributeGroup name="identite">
    <xs:attribute name="nom" type="xs:NMTOKEN" use="required"/>
    <xs:attribute name="prenom" type="xs:NMTOKENS" use="required"/>
    <xs:attribute name="villeNaissance" type="xs:NMTOKENS" use="required"/>
</xs:attributeGroup>
```

Figure 2.4 Définition de Shéma pour l'exemple 2.

2.3.1 Validation des données avec Schematron et JSON Schema

Schematron est utilisé dans des secteurs tels que la finance, la santé, l'aérospatiale, la sécurité intérieure, la fiscalité, le calcul scientifique et l'édition. Il a pour objectif la validation des règles métier, les rapports de données, la validation générale, le contrôle qualité, l'assurance qualité, etc. Il est basé sur des règles basées sur des modèles, des règles et des assertions. Schematron est considéré plus expressif que les autres langages puisqu'il peut exprimer des contraintes complexes qui sont difficile voire impossibles avec les autres langages de validation. En fait, il est souvent utilisé pour définir des règles métier. Néanmoins, les règles Schematron peuvent devenir complexes à définir et à déboguer. Une approche populaire consiste à combiner les deux approches, en définissant la structure du document XML avec un langage de schéma traditionnel et en le complétant avec des règles de schématron. Ainsi, schematron diffère dans son concept de base des autres langages de schéma en ce sens qu'il n'est pas basé sur des grammaires mais sur la recherche de modèles d'arborescence dans le document analysé.

1. Exemple de validation des données avec schématron

Supposons les données dans un document XML représentant des capteurs disséminés dans l'environnement. Nous pouvons définir le fichier schématron pour valider des données concernant un capteur de lumière, Tableau 2.2. Il s'agit ici de renseigner obligatoirement les informations d'identification et de localisation de chaque capteur. L'élément idtype permet de contrôler la valeur de l'id de chaque capteur.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema   xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron">
<xs:element name="sensorLight">
<xs:annotation>
<xs:appinfo>
<sch:pattern>
<sch:rule context="sensorLight">
<sch:assert test="(@id and @localisation)">
        Un capteur de lumière doit forcément avoir un identifiant et l'espace où est installer.
</sch:assert>
</sch:rule>
</sch:pattern>
</xs:appinfo>
</xs:annotation>
<xs:complexType>
<xs:attribute name="id" type="xs:idtype" use="optional"/>
<xs:attribute name="localisation"  type="xs:string" use="optional"/>
</xs:complexType>
        <xs:simpleType name="idtype">
        <xs:restriction base="xs:positiveInteger">
                <xs:pattern value="0[1-9]{1}[0-9]{8}"/>
                </xs:restriction>
</xs:simpleType>
</xs:element>
</xs:schema>
```

Tableau 2.2 Validation des données via schématron.

JSON Schema[6] est un langage déclaratif qui permet d'annoter et de valider des documents JSON (JavaScript Object Notation). Il a évolué en tant que format d'échange de données indépendant avec sa propre spécification. Décrit les formats de données existants, fournit une documentation claire lisible par l'homme et la machine, valide les données utiles pour des tests automatisés et en particulier assurer la qualité des données soumises par les clients.

JSON Schema a été proposé comme un langage Schema (i.e. ; grammaire) pour JSON avec un rôle similaire à XML Schema pour XML. Il est lui-même écrit en utilisant la syntaxe JSON et est indépendant du langage de programmation. Il contient les types de données prédéfinis suivants : null, booléen, objet, tableau, nombre et chaîne, et permet de définir des contraintes sur chacun d'eux. Dans JSON Schema, il est possible d'avoir des définitions réutilisables qui peuvent ensuite être référencées. L'exemple décrit dans

Tableau 2.3 contient un schéma JSON qui peut être utilisé pour valider notre exemple de capteurs.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "SensorLight",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
            "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
```

https://json-schema.org/understanding-json-schema/index.html[6]

```
                    "minimum": 0,
    },
    "localisation": {
      "description": "space where the sensor is installed",
      "type": "string"
    },      },
  "required": ["id", "localisation"]
}
```

Tableau 2.3 Validation de données de capteur de lumière avec JSON Schema

2.4 Validation de document RDF : Comment ?

Nous venons de voir les différentes technologies permettant de garantir une qualité de données en termes de vérification de conformité aux schémas. Dans cette section, nous décrivons les concepts particuliers de RDF qui doivent être pris en compte pour sa validation.

Dans le travail d'étude présenté l'année 2021/2022, nos collègues ont longuement décrit les structures d'un document RDF et les requêtes SPARQL. Pour plus de détail, nous invitons le lecteur à lire cette étude. Toutefois, dans ce qui suit, nous allons faire un petit rappel et allons présenter des concepts clefs non abordés par nos collègues.

L'avènement de RDF et XML a rendu donc possible d'encoder ou d'annoter un contenu Web d'une manière plus compréhensible par la machine. Ce qui permet une rapidité d'évolution vers une riche base de connaissances. Par conséquent, fournira un moyen à la création de nombreuses nouvelles applications intéressantes, dont l'une des plus importantes sera la recherche et la récupération précises de contenu.

Le traitement des flux RDF (Resource Description Framework) en particulier et les modèles ontologiques en général est dès lors devenu un domaine de recherche important. Toutefois, il faut noter qu'il existe une réelle différence entre concevoir une ontologie et définir un modèle RDF

2.4.1 Ontologie vs modèle RDF

Une ontologie est la « spécification explicite d'une conceptualisation partagée pour un domaine de connaissance ». Cette définition a été l'une des plus reprises dans la littérature. Elle permet de préciser l'objet d'étude sans entrer dans des considérations d'appartenance à un courant du domaine. Aimé[7] précise deux points qui émanent de cette définition : « la conceptualisation d'un domaine » se réfère à « un choix quant à la manière de décrire un domaine » ; et « la spécification de cette conceptualisation » est « sa description formelle ». Une ontologie est vue comme une bibliothèque de concepts bien définie. Cette bibliothèque décrit la structure de l'information pour un domaine particulier.

Un domaine particulier de l'utilisation des ontologies est le partage commun des connaissances et d'un vocabulaire standardisé entre humains, humain-machine et machine-machine. De ce fait, une ontologie a pour but de capturer l'essence des termes, le signifié, le concept et de leurs associer leur réalisation dans un langage courant, leur référent.

L'objectif premier du W3C est de trouver un mécanisme pour que les machines puissent manipuler des données du Web. Ces données sont des ressources étant donné qu'elles sont identifiées sur le Web. Par conséquent, l'utilisation des métadonnées pour annoter des informations qui circulent sur le Web facilitera l'automatisation du traitement de ces quantités énormes de données. Donc, RDF est devenu de facto un modèle pour décrire ces ressources et modéliser les relations pouvant exister entre elles.

RDF est basé sur des primitives de modélisation : classes, propriétés et instances permettant de fournir un mécanisme pour décrire les métas-données du web sous forme de triplets. RDF est considéré comme un simple modèle de données permettant de décrire les ressources web. Il utilise un vocabulaire extensible et basé sur le mécanisme des URIs (Uniform Resource

---

[7] Aimé, X. (2011). Prototypicality gradients, similarity and proximity measures : a contribution to the Ontology Engineering. Theses, Université de Nantes. https://tel. archives-ouvertes.fr/tel-00660916.

Identifier), voir un exemple de données RDF, Tableau 2.4. Par ailleurs, Il s'appuie sur une syntaxe XML et prend en charge les types de données de XML Schema.

Les informations à partir du Web sont représentées par un triplet. Chaque triplet est représenté par : SUJET (Subject), PREDICAT (Predicate) et OBJET (Object). L'ensemble des triplets forment un graphe RDF. Dans un graphe, le Sujet et l'Objet sont représentés par un nœud, et le prédicat est représenté par un arc orienté. Ce dernier relie un Sujet à un Objet. Le prédicat RDF (pouvant être une autre ressource) permet de représenter des prédicats binaires. Il a pour rôle de décrire la relation entre le sujet et l'objet représentant respectivement le domaine et le range.

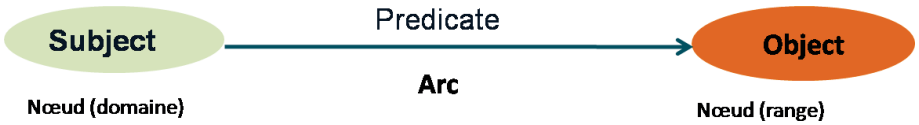


Figure 2.5 Représentation graphique d'un Statement RDF

Ainsi, valider un graphe RDF est considéré comme un filtre indispensable pour vérifier si les instances (données) sont conformes à une définition de schéma. Ce qui permet à un concepteur de se focaliser sur la modélisation que sur le modèle d'ontologie dédié. Notons ici, que ces modèles diffèrent des validateurs XML, puisque les modèles RDF ou d'ontologie peuvent s'appuyer également sur le mécanisme d'inférence. Par conséquent, le processus de validation d'un document RDF utilisera deux branches, la première validation avant inférence, quant à la seconde, elle consiste en la validation des données après inférence. Un autre type de validation s'appuie sur le langage de requête SPARQL.

```
N-Triples
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ml: <http://www.mysite.fr/mlOnt#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
ml:Robml:assistsml:mo ;
ml:hasName "Kompai" .
ml:mofoaf:name "Mounir" .


 Turtle
<http://www.mysite.fr/mlOnt#Rob><http://www.mysite.fr/mlOnt#hasName> "Kompai" .

<http://www.mysite.fr/mlOnt#Rob><http://www.mysite.fr/mlOnt#assists><http://www.mysite.fr/mlOnt#mo>
.

<http://www.mysite.fr/mlOnt#mo><http://xmlns.com/foaf/0.1/name> "Mounir" .
```

Tableau 2.4 Document RDF décrivant qu'un Robot nommé Kompai assiste la personne nommée John

2.4.2 Langage de requêtes SPARQL : Rappel

SPARQL est une recommandation de consortium World Wide Web Consortium (W3C) pour l'extraction des données à partir du Web. Devenu une norme du W3C le 15 janvier 2008. Le nom SPARQL (signifie scintillement en Anglais), est un acronyme récursif, Protocol et RDF Query Language. Ici protocole fait référence aux conventions établies entre client/serveur pour échanger des données, il s'agit ici de requête/réponses. En outre, le W3C de SPARQL défini trois briques de base de recommandations ; la première concerne le langage de requête, la seconde est le format de réponse retenu par W3C qu'est le format XML (SPARQL Query Results XML Format (Second Edition) W3C Recommandation 21 Mars 2013), enfin la dernière est SPARQL qui utilise WSDL pour définir les protocoles http et SOAP pour interroger une base de données RDF.

SPARQL permet aux applications d'obtenir des résultats à partir de triplets décrits en RDF. Il permet également d'interroger un serveur distant pour collecter des réponses en continue et de manière concise.

Le partage de données entre plusieurs applications est possible avec le serveur Fuseki (https://jena.apache.org/documentation/fuseki2/). Ce dernier fournit un serveur SPARQL

utiliser pour le stockage persistant et fournit les protocoles pour extraire des triplets via des mécanismes de requêtes SPARQL. Ce dernier, offre quatre types de requêtes : SELECT, ASK, DESCRIBE et CONSTRUCT. Ces requêtes sont basées sur deux types de concepts de SPARQL : triplet et les graphes. Une déclaration RDF est un triplet : Sujet, Prédicat et Objet. Ainsi, le pattern triplet SPARQL tire profit de cette syntaxe en ajoutant un point à la fin de chaque déclaration. Contrairement au triplet RDF, une requête SPARQL peut inclure des variables. Une variable SPARQL commence par le symbole ? ou $. Une requête SPARQL permet de faire correspondre le motif de la requête (triplet de recherche) avec l'ensemble des triplets du graphe RDF en cours d'exploitation. La forme générale d'une requête SPARQL est composée de plusieurs parties :

- La clause FROM indique quel graphe interroger. FROM peut être optionnelle si l'on connait la base à interroger ;
- WHERE contient les motifs graphiques qui spécifient les résultats souhaités (le mot Where n'est pas obligatoire) ;
- La liste de tous les préfixes à utiliser dans une requête ;

Via la requête SPARQL suivante, nous pouvons savoir qui assiste qui :

```
@prefix ml: <http://www.mysite.fr/mlOnt#>.
SELECT ?artifact ?who
FROM NAMED <http://www.mysite.fr/mygraphe>
WHERE {
   GRAPH ?g {
          ?artifact ml:assists ?who .      }}
```

Notons au passage que SPARQL est comme SQL, mais il existe des divergences dans l'interrogation de BD relationnelles comme illustré dans le tableau suivant :

SQL	SPARQL
• Basé sur le calcul relationnel • Les requêtes sont exprimées dans un langage logique de description du résultat : les champs sélectionnés, les jointures, les filtrages, les groupes et les opérations de groupe.	• Basé sur la notion de graphes de triplets • Les requêtes sont décrites par des motifs (patterns) et des variables.

Tableau 2.5 SPARQL vs SQL

2.4.3 Schéma RDF

Le langage RDF-S permet de décrire sémantiquement les informations échangées. Dans cette perspective, la relation d'héritage entre classes ou entre propriétés n'est autre qu'une taxonomie des termes organisés de manière hiérarchique. Le langage RDF-S[8] permet également de définir des restrictions sur les valeurs de propriétés offrant ainsi un moyen d'effectuer de simples inférences comme l'appartenance à une classe ou à une sous-classe. La vérification des valeurs de propriétés et des relations de sous-propriétés est réalisée à l'aide du principe de généralisation/spécialisation entre propriétés.

RDF Schema (RDF(S), RDF-S) est totalement différent de ce que représente XML Schema pour une instance XML. Au fait, quand on parle de RDF-S, on fait référence à un vocabulaire. En effet, RDF Schema ou plutôt RDF Vocabulary offre des mécanismes pour décrire des classes de ressources et les liens susceptibles existaient entre ces ressources. Un lien consiste en la description du domaine d'une ressource et la valeur (range) de ses propriétés. Le W3C a proposé une norme RDFS quioffre plus de souplesse et une description indépendante entre les propriétés

[9]https://www.w3.org/TR/rdf-schema/

et les classes. Parexemple, en RDFS, la propriété assiste est considérée toujours comme propriété indépendante,tandis que Robot est un domaine de la propriété assiste et Person est le range (valeur de lapropriété de Person) de cette même propriété.

2.4.4 Défis RDFS et validation de données

Les spécifications présentées par le W3C n'a pas vocation d'énumérer toutes les formes possibles de représentation et de sens des classes et des propriétés RDF. Toutefois, RDF Schema souligne l'existence de nombreuses techniques par lesquelles la signification des classes et des propriétés peut être décrite. Ainsi, un vocabulaire RDF pourrait décrire des limitations sur les types de valeurs qui sont appropriées pour certaines propriétés, ou sur les classes auxquelles il est logique d'attribuer de telles propriétés.

Les vocabulaires RDF peuvent décrire les relations entre les éléments de vocabulaire de plusieurs vocabulaires développés indépendamment. Étant donné que les IRI sont utilisés pour identifier les classes et les propriétés sur le Web, il est possible de créer de nouvelles propriétés qui ont un domaine ou une plage dont la valeur est une classe définie dans un autre espace de noms.

RDF Schema fournit quant à lui un mécanisme permettant de décrire des données, mais n'impose pas de crainte en termes de comment une application doit les utiliser. A titre d'exemple, un vocabulaire RDF peut affirmer qu'une propriété assiste est utilisée pour indiquer des ressources qui sont des instances de la classe Robot ou Humain, mais ne précise pas les types auxquels est appliqués (nous parlons de rdfs :range). Différentes applications utiliseront ces informations de différentes manières. Dans ce contexte, des outils de vérification de données peuvent l'utiliser pour souligner des erreurs probables, quant à un moteur d'inférence, peut l'utiliser pour en séduire d'autres valeurs à partir de données d'instances.

2.5 Validation pendant le traitement des requêtes SPARQL

Les processus de validation de données basées sur les requêtes SPARQL consiste à exprimer des contraintes sur les plages de valeurs, c'est-à-dire sur le range. Considérons dans la suite de cette section l'exemple présenté dans le Tableau 2.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY f "http://xmlns.com/foaf/0.1/">
<!ENTITY mln "http://www.lyazidsabri-aures.fr/mlOnt#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ml="http://www.lyazidsabri-aures.fr/mlOnt#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Property rdf:about="http://www.lyazidsabri-aures.fr/mlOnt#assists">
<rdfs:domain rdf:resource="&mln;Agent"/>
<rdfs:domain rdf:resource="&mln;Artifact"/>
<rdfs:range rdf:resource="&f;Person"/>
</rdf:Property>
<rdfs:Clas
s rdf:about="&f;Person">
<rdfs:subClassOf rdf:resource="http://www.lyazidsabri-aures.fr/mlOnt#Agent"/>
</rdfs:Class>
</rdf:RDF>
```

Tableau 2.6 Exemple de Graphe RDF. Ce graphe sert d'exemple pour la requête

L'étude menée par nos collègues en 2021/2022, présente le langage SPARQL et expose une comparaison entre les moteurs de langage et leurs fonctionnalités. De plus, l'étude souligne que le langage SPARQL est très verbeux. En conséquence, les requêtes SPARQL sont très difficiles à écrire, en particulier par les non-experts. L'exercice devient encore plus compliqué puisqu'il

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix ml:<http://www.lyazidsabri-aures.fr/mlOnt#>
prefix foaf: <http://xmlns.com/foaf/0.1/>


SELECT distinct  ?who
 WHERE { ?subject  ml:assists   ?what.
           ?subject  rdf:type     foaf:Person.
           ?subject  foaf:name   ?who
 }
```

Figure 2.6 Cette requête Sparql permet d'imposer que la propriété ml :etudie ait un valeur de type foaf :Person. Ainsi, le graphe RDF exprimé dans le Tableau 2.6 peut fournir une réponse à cette requête.

n'existe pas de standard pour imposer des contraintes. Ainsi, il n'est pas évident de rédiger toutes les possibilités qu'une valeur puisse posséder ou en exclure les autres. Des outils ont été introduits pour pallier aux inconvénients cités ci-dessus. Parmi ces outils, SPIN[9], ShEx[10] et ShACL[11]. La notation d'inférence SPARQL (SPIN) combine des concepts de langages orientés objet, de langages de requête et de systèmes basés sur des règles pour décrire le comportement des objets sur le Web de données. L'une des idées de base de SPIN est de lier les définitions de classe aux requêtes SPARQL pour capturer les contraintes et les règles qui formalisent le comportement attendu de ces classes. SPIN est un langage de règles et de contraintes basé sur SPARQL pour le Web sémantique. Il est également un mécanisme permettant de représenter des requêtes SPARQL réutilisables sous forme de modèles et de définir de nouvelles fonctions SPARQL avec une syntaxe conviviale pour le Web. SPIN peut être utilisé pour : Calculer la valeur d'une propriété en fonction d'autres propriétés, l'âge d'une personne en tant que différence entre la date en cours et l'anniversaire de la personne. Quant à ShEx, il partage des similarités

https://spinrdf.org[10]
https://shex.io[11]
[12]https://w3c.github.io/shacl/shacl-af/#SPARQLRule

avec ShACL. Ils ont comme objectif la description et la validation des données RDF. Par conséquent, les deux considèrent que la notion de forme consiste à exprimer des contraintes sur les nœuds RDF (Subject, Predicate, Object). Dans le chapitre suivant, nous allons passer en revue le langage ShACL.

2.6 Contraintes à respecter par les outils de validation

Contrairement aux autres graphes, RDF ne possède pas un nœud racine. Chaque nœud est identifié via une IRI. Ainsi, il faut respecter des conditions pour garantir une qualité de données en termes de conformité à une grammaire. Dans ce qui suit nous avons énuméré une liste non exhaustive :

1. Cibler les nœuds de graphe auxquels à appliquer les filtres (contraintes).
2. Les propriétés qui sont représentées par des arcs orientés sont également des IRIs. Ces propriétés peuvent être réutilisées par un nœud pouvant avoir des structures différentes. Cette contrainte rend la validation plus compliquée.
3. La propriété inverse ne peut pas être définie dans un graphe RDF, toutefois ShACL propose d'utiliser la propriété sh :inversepPath pour indiquer l'autre sens de l'arc du prédicat (c'est-à-dire inversePah permet de vérifier la propriété à partir de la ressource Object vers la ressource Subject).
4. Attention aux nœuds vide, en particulier dans le cadre de définition de conteneur RDF (Bag, List, etc.)
5. Etre en mesure de décrire les types de données que des nœuds peuvent exprimer.
6. Notion de sous classes qui peut s'avérer difficile à gérer si un graphe importe un autre graphe.
7. Possibilité de gérer la négation sur la valeur d'un nœud.

Parmi les contraintes qui serait pratiquement très difficile à réaliser et entraver le processus de validation est le principe du monde ouvert/fermé sur lequel repose les langages du Web.

Le monde fermé (en Anglais, Closed World Assumption (CWA)) est généralement appliqué dans les systèmes qui disposent d'informations complètes tandis que l'OWA est plus naturel pour

les systèmes d'information incomplets comme le Web. L'hypothèse du monde ouvert, sur laquelle s'appuie le raisonnement à base d'ontologies, qu'on appelle interprétation logique, considère que l'absence d'un fait dans la base de connaissances présuppose que ce fait est inconnu et non qu'il soit faux. L'hypothèse du monde ouvert ait été adoptée compte tenu du caractère distribué des connaissances du Web sémantique. Par conséquent, cette hypothèse a un impact sur la façon de modéliser et l'interprétation des informations ce qui rend la validation des données pratiquement impossible vue l'absence de l'information. En effet, l'hypothèse du monde ouvert affirme que la véracité d'une déclaration (un fait) est indépendante de savoir si elle est connue. En d'autres termes, ne pas savoir si une déclaration est explicitement vraie n'implique pas que la déclaration est fausse. Inversement, l'hypothèse du monde fermé atteste que toute déclaration dont la véracité n'est pas connue est considérée comme fausse. Dans l'hypothèse d'un monde ouvert, l'ajout de nouvelles informations ne révoquent pas des informations déjà existantes même si elles sont contradictoires.

Cette tâche devient encore plus compliquée, étant donné qu'au niveau des ontologies, on distingue trois niveaux de raisonnement :

- Le niveau classe : Il s'agit principalement de vérifier l'héritage des classes (subsomption) et de construire la taxonomie des classes de l'ontologie.
- Le niveau propriétés : Il s'agit de vérifier les relations entre individus de classes différentes à travers différents types de propriétés (transitive, fonctionnelle, etc.). Par exemple, la propriété transitive permet d'inférer l'existence d'une relation entre les individus de deux classes qui n'ont pas de relation explicite dans l'ontologie.
- Niveau individu : Dans ce niveau, il s'agit d'une part, de vérifier la consistance des connaissances (Consistency Checking) vis-à-vis du modèle d'ontologie, c'est-à-dire de vérifier si une classe possède un individu (instance) dans la base de connaissances, et d'autre part, de rechercher dans la base de connaissances les classes qui correspondent à une description partielle ou complète d'un individu donné.

Chapitre 3: **Shapes Constraint Language (ShACL)**

3.1 Introduction

ShApes Constraint Language (ShACL) a été développé par le W3C RDF Data Shapes Working Group, qui a été créé en 2014 dans le but de « produire un langage permettant de définir des contraintes structurelles sur les graphes RDF». Les vocabulaires RDF définissent les termes et les ressources pour un domaine d'intérêt. Ces vocabulaires sont souvent spécifiés de manière ouverte, sans fournir d'informations telles que les assertions de domaine, les types des assertions, les propriétés, les cardinalités, etc. Cela permet de garder le vocabulaire applicable à un large éventail d'utilisations et de favoriser l'intégration avec d'autres vocabulaires. Toutefois, dans de nombreuses applications, il est utile de définir le vocabulaire dans un but précis avec des contraintes spécifiques pour faciliter son utilisation. Dans ce contexte, ShACL a été influencé par SPARQL Inferencing Notation (SPIN) et certaines parties des formes de ressources Open Services for Lifecycle Collaboration (OSLC). ShACL est divisé en deux parties. La première partie, appelée ShACL Core, décrit un vocabulaire RDF de base pour définir des formes et des contraintes communes tandis que la deuxième partie décrit un mécanisme d'extension en termes de SPARQL et a été appelée : ShACL-SPARQL.

3.2 Quelques Fondements de SHACL

La validation standard W3C ShACL est un outil précieux pour une vérification efficace de la cohérence des informations. Il est utile dans les efforts d'intégration des données, ainsi que pour l'examen de la conformité des données, par exemple, vérifié que chaque ressource est commencée par une URI. Chaque URI doit correspondre à un espace de nom. Le langage ShACL valide les graphes RDF par rapport à un ensemble de conditions. Ces conditions sont fournies sous forme de formes et d'autres constructions exprimées sous la forme d'un graphe RDF. Dans ShACL, les graphes RDF utilisés de cette manière sont appelés graphes de formes, et

les graphes RDF qui sont validés par rapport à un graphe de formes sont appelés graphes de données.

3.2.1 Formes ShACL

Deux types de forme en ShACL: nœud et propriétés. Les nœuds spécifient des contraintes sur un nœud tandis que les formes de propriété spécifient des contraintes sur les valeurs. Le diagramme suivant Figure 3.1 présent un aperçu de certaines des classes clés du vocabulaire ShACL. Chaque boîte représente une classe. Le contenu des cases sous le nom de classe répertorie certaines des propriétés que les instances de ces classes peuvent avoir, ainsi que leurs types de valeur. Les flèches orientées indiquent la hiérarchie de classe RDFS.

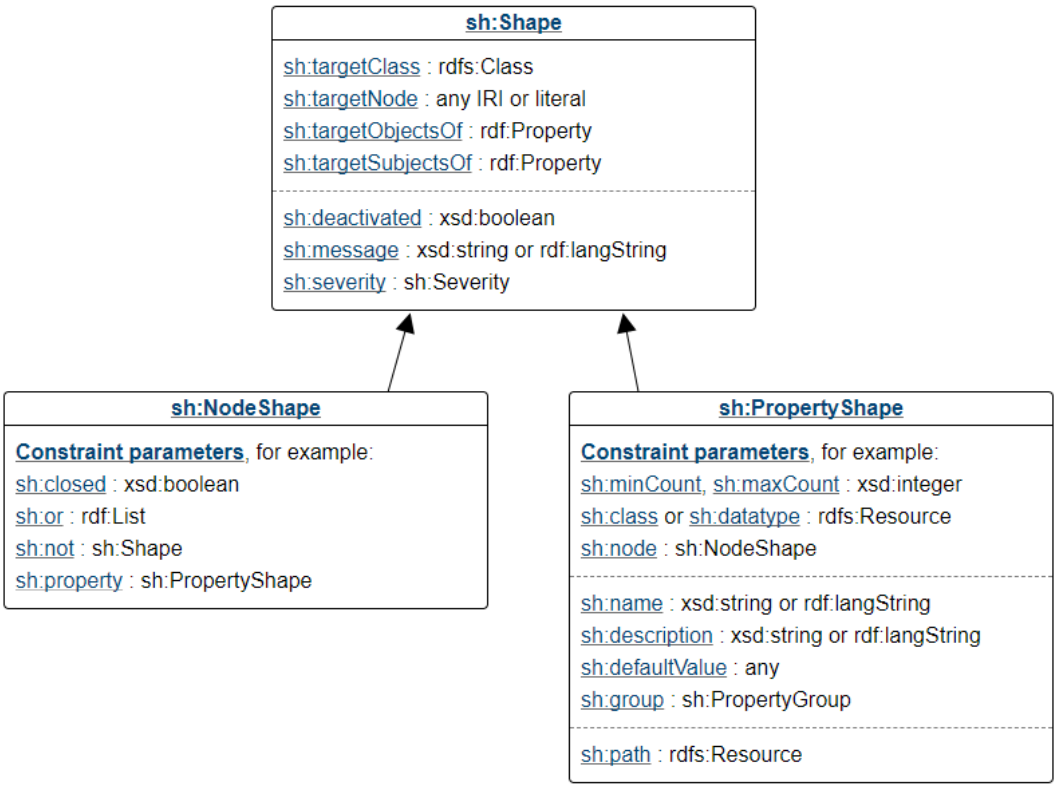


Figure 3.1 Diagramme de forme ShACL.

Une forme ShACL soit est une IRI ou un nœud vide noté v telle que ce nœud vide qui devrait remplir une des conditions suivantes :

1. Une instance ShACL de sh:NodeShape ou sh:PropertyShape.
2. Le sujet d'un triplet RDF a sh:targetClass, sh:targetNode, sh:targetObjectsOf ou sh:targetSubjectsOf comme prédicat.
3. Sujet d'un triplet possède un paramètre comme prédicat
4. Une valeur d'un paramètre attendant une forme et ne prenant pas de liste tel que sh:node, ou un membre d'une liste ShACL qui est une valeur d'un paramètre attendant une forme et prenant une liste.

ShACL englobe des informations et des contraintes applicables aux nœuds de données dans certaines constructions appelées *formes*. Les formes ShACL peuvent véhiculer également des informations sur les nœuds cibles. Notons ici que la relation entre le ShACL et RDFS ShACL est que ShACL utilise les vocabulaires RDF et RDFS, mais l'inférence RDFS complète n'est pas requise. Cependant, les processeurs ShACL peuvent fonctionner sur des graphiques RDF qui incluent des déductions de types de classes avant d'être soumis à un processeur ShACL, soit effectué à la volée dans le cadre du traitement ShACL (sans modifier le graphique de données ou les formes graphiques). Les valeurs de la propriété sh:implication sont des IRIs. Les implémentations de ShACL peuvent, mais ne sont pas tenues d'implémenter le processus d'implications de classes.

3.2.2 Les chemins ShACL

Une « Property shape » contient obligatoirement un « sh:path ». ShACL comprend des termes RDF pour représenter le sous-ensemble suivant des chemins de propriété SPARQL : PredicatePath, InversePath, SequencePath, AlternativePath, ZeroOrMorePath, OneOrMorePath et ZeroOrOnePath. Ces chemins utilisent un codage RDF basé sur les règles suivantes :

1. Les prédicats directs utilisent un seul IRI.
2. Les chemins inverses sont déclarés par un nœud vide avec la propriété sh:inversePath.
3. Les chemins de séquence sont codés par des listes RDF dont les valeurs sont les chemins ShACL eux-mêmes.
4. Les chemins alternatifs sont déclarés par un nœud vide avec la propriété sh:alternativePath dont la valeur est une liste RDF avec les différentes alternatives.

5. Les modificateurs de chemin ?, * et + sont codés par un nœud vide avec les propriétés correspondantes sh:zeroOrOnePath, sh:zeroOrMorePath ou oneOrMorePath. Un chemin à un ou plus est un nœud vide qui fait l'objet d'un triple exactement dans G. Ce triple a SH: sh:oneOrMorePath comme prédicat, et l'objet V est un chemin de propriété ShaCl bien formé

3.3 Processus de validation des données RDF via ShACL

Pour mieux illustrer le rôle de ShACL, présentons l'exemple ShACL dans le Tableau 3.2 illustre un exemple de contraintes et de validation d'un document RDF via le modèle ShACL. Il s'agit ici de vérifier que l'id d'un capteur est de type xsd :NCNAME d'XML schema, ensuite de vérifier qu'un capteur de type DoorSensor est installé sur une entité de type Door accessible via un IRI. Ainsi, grâce à ShACL qui définit les formes comme une conjonction de contraintes que les nœuds doivent satisfaire. Un processeur ShACL vérifie chacune des contraintes et renvoie des erreurs de validation pour chaque contrainte non satisfaite. Si le processus de validation ne renvoie aucune erreur, le graphe RDF correspondant est considéré comme valide.

Les exemples Tableau 3.1 et Tableau 3.3illustrent respectivement un graphe RDF valide et un autre non valide. La Figure 3.2 présente le processus de validation via un code Python. Quant à la Figure 3.3 présente un rapport négative de validation.

```
@prefix ex: <http://example.org/ns#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Sensor1
  rdf:type ex:DoorSensor ;
ex:id 5125 .
```

Tableau 3.1 Exemple Valide. Fichier nommé instDoor.ttl

```
@prefix ex: <http://example.com/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#>  .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>  .
ex:SensorShape
   a sh:NodeShape  ;
   sh:targetClass ex:DoorSensor;
   sh:property [
   sh:path ex:id;
   sh:mincount  1 ;
   sh:maxCount  1 ;
   sh:datatype xsd: NCName;
] ;
sh:property [
      sh:path   ex:installedOn ;
      sh:minCount  1 ;
      sh:maxCount  1 ;
      sh:nodeKind sh:IRI ;
      sh:class ex:Door ;  # le capteur doit être installé sur un objet de class Door
    ] .
ex:DoorShape
  a sh:NodeShape ;
  sh:targetClass ex:Door ;
  sh:property [
      sh:path ex:locatedAt;
      sh:datatype xsd:string ;
] .
```

Tableau 3.2 Document ShACL nommé exsh1.tt. Notons ici que xsd: NCName est un type de nom XML qui ne contient pas de signe deux-points, et l'extension .ttl correspond à un fichier Turtle

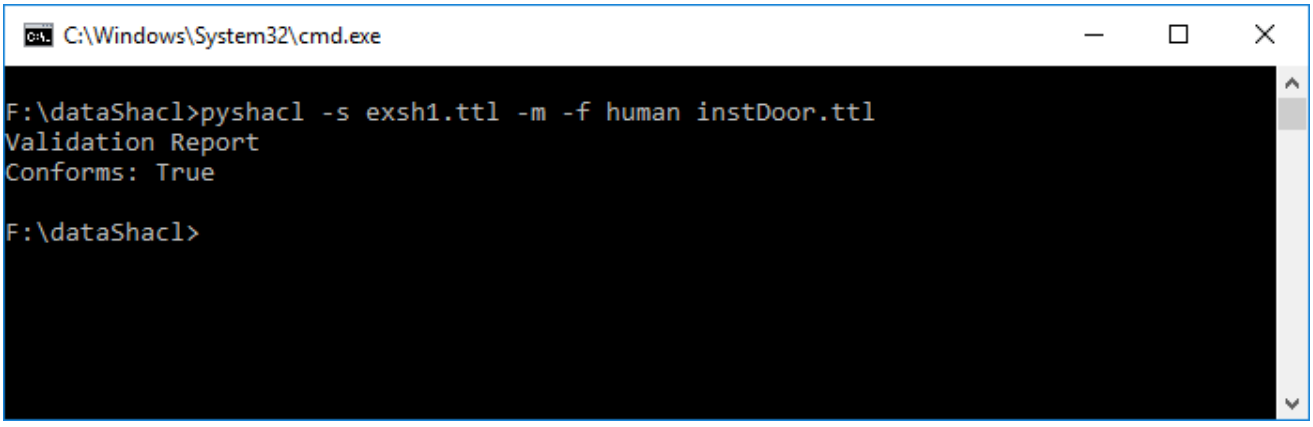


Figure 3.2 Résultat de validation du document instDoor.ttl

```
@prefix ex: <http://example.org/ns#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:Sensor1
  rdf:type ex:DoorSensor ;
  ex:id 5125 .
```

Tableau 3.3 Exemple de graphe RDF non valide par rapport à SCHACL

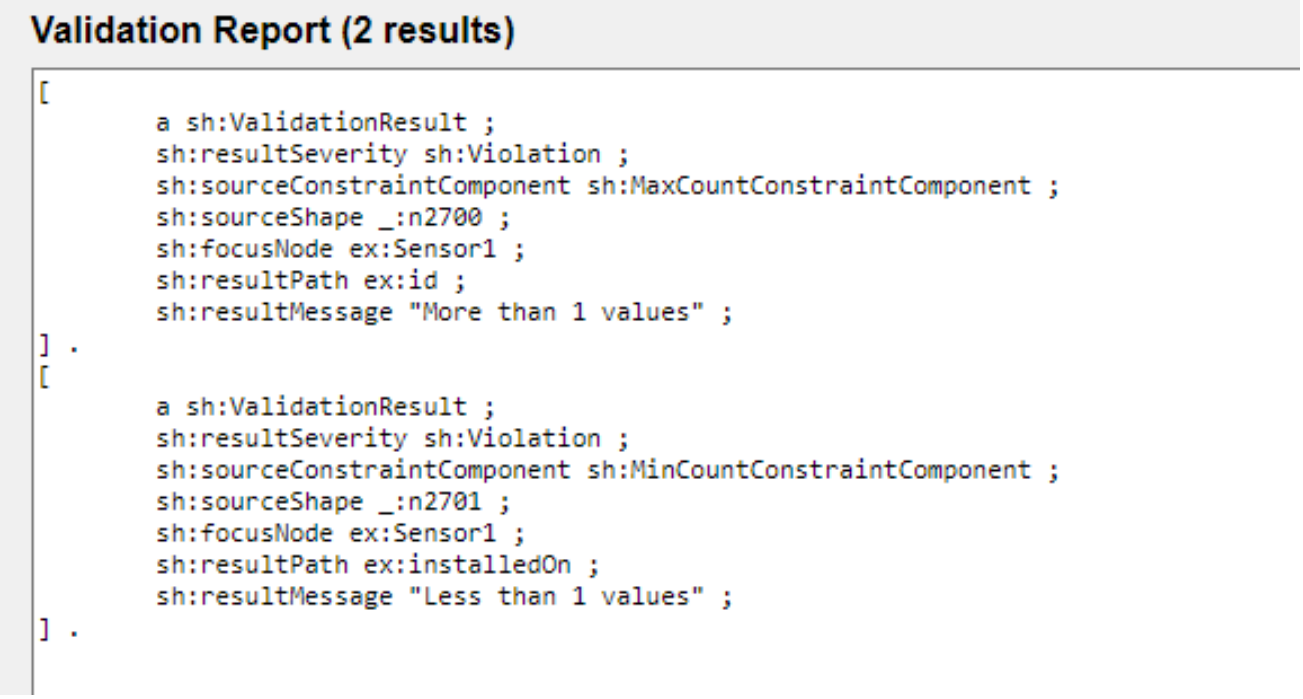


Figure 3.3 Echec de validation présenté par l'outil playground https://shacl.org/playground/

Le processus de validation signale deux erreurs, le premier concerne l'id du capteur, en effet, le capteur possède deux ids, Figure 3.3. Quant au second problème concerne le fait que la propriété installedOn n'est pas définie. En effet, ShACL déclare des contraintes sur les valeurs et celles appliquées sur les classes. Le Tableau 3. 4 résume ces contraintes.

Contraintes de cardinalités	
Contraintes	**Description**
sh:minCount	Nombre mininal de triplets qui incluent le noeud et le prédicat ciblé. Valeur par défaut: 0
sh:maxCount	Idem à sh:minCount, mais selon le nombre maximale Par défaut, aucune limite
Contraintes principales	
Type de contraintes	**Contraintes**
Interval de valeurs	sh:minInclusive, sh:maxInclusive sh:minExclusive, sh:maxExclusive
Contriantes entre deux propriétés	sh:equals, sh:disjoint, sh:lessThan, sh:lessThanOrEquals
Basé sur les chaînes de caractères	sh:minLength, sh:maxLength, sh:uniqueLang
sh:datatype	Restriction du type de valeur possible.

Tableau 3. 4Contraintes sur les valeurs.

sh:and	Conjonction d'une liste de property shape. Valeur par défaut
sh:or	Disjonction d'une liste de property shape.
sh:not	Négation d'un property shape
sh:xone	Au moins 1 property shape doit être validé

Tableau 3.5 Contraintes sur les valeurs de classes

Notons également que l'on peut définir une contrainte sur la classe des valeurs pour imposer que les valeurs doivent être une instance d'une classe donnée. La notion d'instance est définie par: rdf:type et /rdfs:subClassOf*.

Quant aux opérateurs logiques, ils sont décrits par le Rappelons ici que tous les littéraux du modèle de données RDF ont un type de données associé. Les littéraux de chaîne simples ont le type de données xsd:string par défaut. Dans ce contexte, ShACL contient une liste de types de données intégrés basés sur les types de données XML Schema (qui sont les mêmes que dans SPARQL 1.1).

3.4 Processus de validation

La validation considère en entrée un graphe de données et un graphe de forme, et produit un rapport de validation contenant les résultats de la validation. La vérification de la conformité est une version simplifiée de la validation, produisant un résultat booléen. Un système capable d'effectuer une validation est appelé processeur. ShACL définit un vocabulaire de rapport de validation RDF qui peut être utilisé par les processeurs qui produisent des rapports de validation comme graphiques de résultats RDF. Cette spécification utilise le vocabulaire des résultats ShACL pour les définitions normatives des validateurs associés aux composants de contrainte. Seules les implémentations de ShACL qui peuvent produire toutes les propriétés obligatoires du vocabulaire du rapport de validation sont conformes aux normes.

Bien qu'un processeur de validation ShACL repose sur deux type d'entrées : un graphe de données qui contient les données RDF à valider et un graphe de formes qui contient les formes de validation. Toutefois, il est possible de fusionner les deux graphes (données, conditions de validation) dans un seul et même fichier. Il demeure conseiller d'utiliser des fichiers séparés. Il existe deux principaux types de formes : les formes de nœud et les formes de propriété. Les formes de nœud déclarent des contraintes directement sur un nœud. Les formes de propriété déclarent des contraintes sur les valeurs associées à un nœud via un chemin. Les formes de propriété (prédicat dans le jargon RDF) ont une propriété sh:path qui déclare le chemin (arc) sortant du nœud Subject vers le nœud Object qui spécifie la valeur. Le champ propriété sh:property permet de déclarer un ensemble de propriétés.

Dans l'exemple Tableau 3.6 , la forme SensorShape déclare deux propriété, la première définie quatre déclarations, quant à la seconde définie 5 déclarations.

```
ex:SensorShape
   a sh:NodeShape  ;
   sh:targetClass ex:DoorSensor;
   sh:property [
   sh:path ex:id;
   sh:mincount  1 ;
   sh:maxCount  1 ;
   sh:datatype xsd:integer ;
] ;
sh:property [
      sh:path   ex:installedOn ;
      sh:minCount  1 ;
      sh:maxCount  1 ;
      sh:nodeKind sh:IRI ;
      sh:class ex:Door ;  # le capteur doit être installé sur un objet de class Door
   ] .
```

Tableau 3.6 Forme ShACL, SensorShape, fichier nommé e2.ttl

Remarquons que dans la forme SensorShape, il est possible de définir le minimum et le maximum d'une valeur donnée. Le point important est la possibilité de définir le type de nœud cible (c'est-à-dire l'arc sortant vers le nœud Object). L'utilisation d'IRI est conseillée pour pouvoir référencer les ressources. Signalons ici qu'il est possible d'importer des graphes. Cela signifie que des graphes nommés peuvent être réutilisé comme composant externe dans le processus de validation. Cette possibilité offre une grande flexibilité et de puissance au langage ShACL. Pour ce faire, ce langage offre le prédicat owl :imports permettant en conséquence une pré-validation utile pour fusionner le graphe ou les graphes référencés. Toutefois, il est possible de désactiver des formes lors de la réutilisation des conditions ShACL. Un cas d'utilisation typique pour les formes désactivées est lorsque l'on importe des formes d'un autre graphique par un tiers et que l'on souhaite désactiver certaines des formes du graphique de formes locales qui ne s'appliquent pas dans le contexte actuel. Notez que si l'auteur d'une bibliothèque de formes prévoit qu'une forme devra peut-être être désactivée ou modifiée par d'autres, il peut être préférable d'utiliser des IRIs au lieu de nœuds vides, afin qu'ils puissent être référencés ultérieurement.

De plus, ShACL permet de définir plusieurs déclarations cibles en spécifiant un ensemble de nœuds qui seront validés par rapport à une forme. Le Tableau 3.7 contient les différentes déclarations cible définies dans le noyau ShACL.

Valeur	Description
sh:targetNode	Pointer directement vers un nœud
sh:targetClass	Tous les nœuds qui sont des instances d'une certaine classe
sh:targetSubjectsOf	Tous les nœuds qui sont sujets d'un prédicat
sh:targetObjectsOf	Tous les nœuds qui sont des objets d'un prédicat

Tableau 3.7 Déclarations des cibles ShACL. Par exemple, sh:targetNode déclare un nœud cible.

sh:focusNode	Le nœud de focus qui était en cours de validation lorsque l'erreur s'est produite
sh:resultPath	Le chemin depuis le nœud de focus. Cette propriété facultative correspond généralement à la déclaration sh:path des formes de propriété
sh:valeur	La valeur qui a violé la contrainte, lorsqu'elle est disponible
sh:sourceShape	La forme par rapport à laquelle le nœud de focus a été validé lorsque la contrainte a été violée.
sh:sourceConstraintComponent	IRI identifiant le composant à l'origine de la violation.
sh:détail	Peut indiquer plus de détails sur la cause de l'erreur. Cette propriété peut être utilisée pour signaler des erreurs dans des formes imbriquées.
sh:resultMessage	Détails textuels sur l'erreur. Ce message peut être affecté par la propriété sh:message
sh:resultSeverity	Valeur égale à la valeur sh:severity de la forme qui a provoqué l'erreur de violation. Si la forme n'a pas de déclaration sh:severity, la valeur par défaut sera sh:Violation.

Tableau 3.8 Propriétés des résultats de validation ShACL

Quant au nœud de type nœud cible (en Anglais, focusnode), est un terme RDF qui est validé par rapport à une forme en utilisant les triplets d'un graphe de données. L'ensemble des nœuds nommés focusnode pour une forme peut être identifié comme suit:

- Spécifié sous une forme en utilisant des déclarations cibles.
- Spécifié dans toute contrainte qui fait référence à une forme dans les paramètres des contraintes de forme.
- Spécifié comme entrée explicite au processeur ShACL pour valider un terme RDF spécifique contre une forme.

Toutes les propriétés décrites dans le Tableau 3.8 peuvent être spécifiées dans un sh:ValidationResult.Les propriétés sh:focusNode, sh:resultSeverity et sh:sourceConstraintComponent sont les seules propriétés obligatoires pour tous les résultats de validation.Les figures Figure 3.4 et Figure 3.5illustrent des résultats de validation de l'exemple ci-dessus, la première utilise python en s'appuyant sur la librairie *pyshacle* et la seconde utilise le service web : https://shacl.org/playground.

```
F:\dataShacl>pyshacl -s e2.ttl -m -f human in1.ttl
Validation Report
Conforms: False
Results (1):
Constraint Violation in MaxCountConstraintComponent (http://www.w3.org/ns/shacl#MaxCountConstraintCo
mponent):
        Severity: sh:Violation
        Source Shape: [ sh:datatype xsd:integer ; sh:maxCount Literal("1", datatype=xsd:integer) ; s
h:mincount Literal("1", datatype=xsd:integer) ; sh:path ex:id ]
        Focus Node: ex:Sensor1
        Result Path: ex:id
        Message: More than 1 values on ex:Sensor1->ex:id

F:\dataShacl>
```

Figure 3.4 Résultat de validation avec la librairie pyshacl

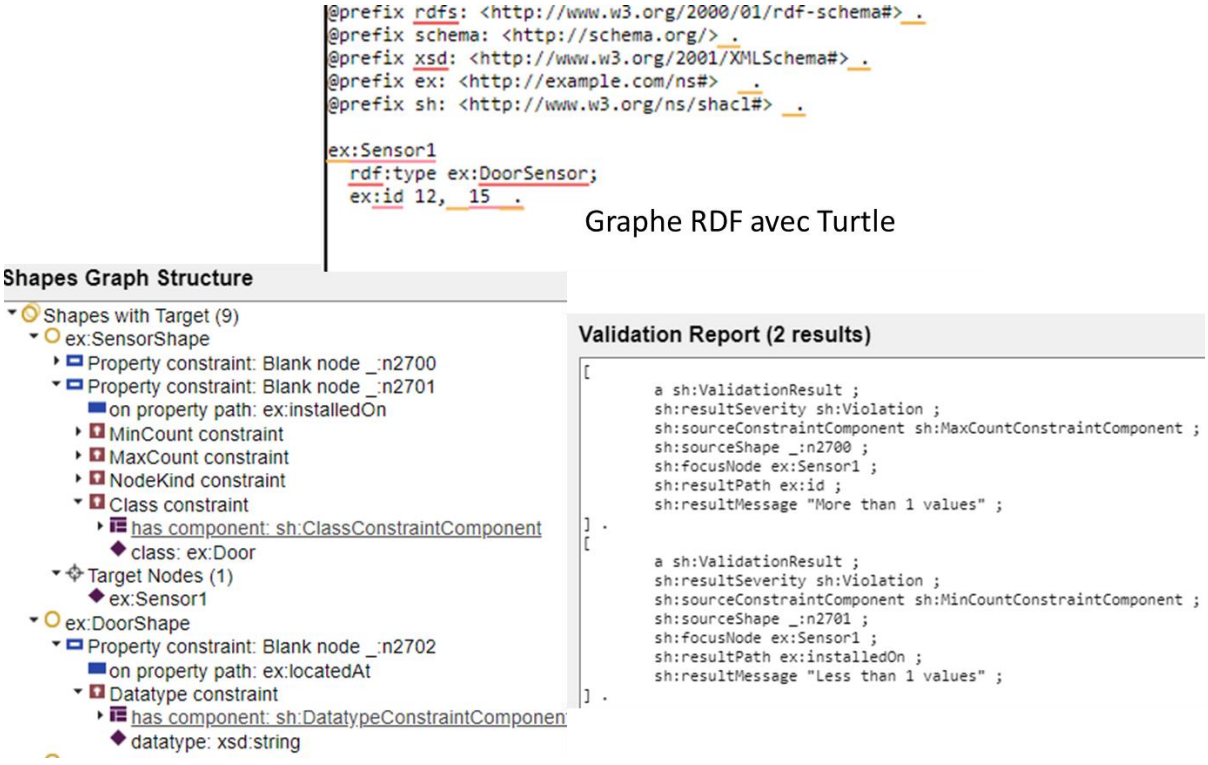


Figure 3.5 Résultat de validation du graphe RDF selon la forme Shapes Graph Structure correspondant au Tableau 3.6.

3.5 Composants de contrainte

SHACL définit le concept de composants de contrainte en s'appuyant sur le composant *constraint*. Ils sont associés à des formes pour déclarer des contraintes. Chaque nœud ou forme de propriété peut être associé à plusieurs composants de contrainte. Les composants de contrainte sont identifiés par un IRI et ont deux types de paramètres : obligatoires et facultatifs. L'association entre une forme et un composant de contrainte se fait en déclarant des valeurs pour les paramètres. Supposons qu'un composant de contrainte C avec des paramètres obligatoires p1, ... pn, une forme s dans un graphe de formes SG (Shapes Graph) déclare une contrainte de type C avec des valeurs de paramètres obligatoires <p1,v1>, ... <pn,vn> dans SG. Pour les composants de contrainte avec des paramètres facultatifs, la déclaration de contrainte se compose des valeurs que la forme a pour tous les paramètres obligatoires et facultatifs de ce composant.

Nous avons déjà utilisé ces contraintes dans l'exemple ci-dessus où la validation du graphe RDF exige qu'un capteur ne peut avoir qu'un seul ID. Nous allons voir dans le chapitre quatre que cette manière est dangereuse dans des applications sensibles, telles que santé, véhicule intelligent, etc.

```
@prefix ex: <http://example.com/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:SensorShape
   a sh:NodeShape ;
   sh:targetClass ex:DoorSensor;
   sh:property [
   sh:path ex:id;
   sh:mincount 1 ;
   sh:maxCount 1 ;
   sh:datatype xsd:integer ;
] ;
```

Tableau 3.9 Forme ShACL déclarant des contraintes sur des valeurs

Modifions l'exemple précédent Tableau 3.9, en imposant que la description de l'objet sur lequel un capteur est installé ne puisse dépasser 7 caractères, Tableau 3.10 ci-dessous, ensuite définissons un graphe de données Tableau 3.11 . Lors du processus de validation, la donnée du capteur Door2 soulève une erreur comme illustré par laFigure 3.6.

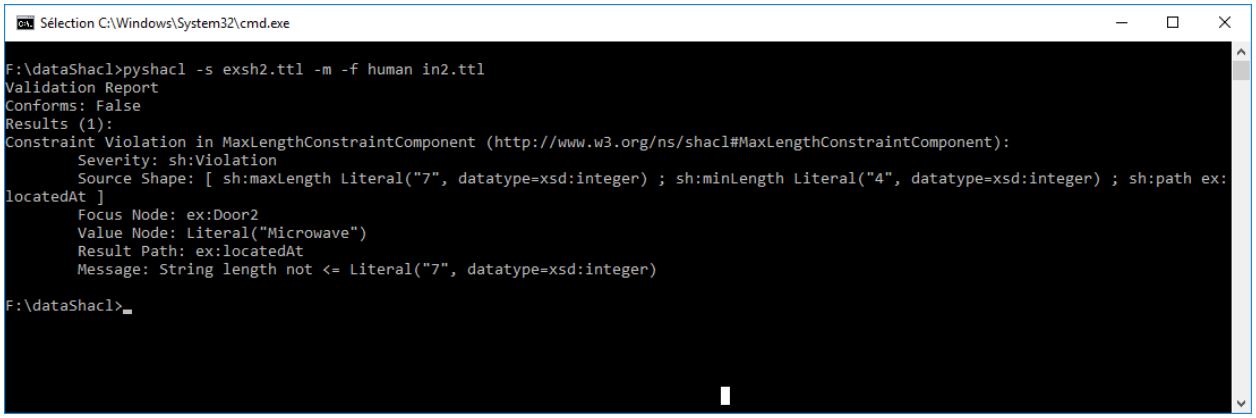


Figure 3.6 Echec de validation avec la librairie pyshacl

La Figure 3.6 présente l'échec de validation des données représentées par le Tableau 3.11.

```
@prefix ex: <http://example.com/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#>  .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>  .
ex:SensorShape
   a sh:NodeShape  ;
            sh:targetClass ex:DoorSensor;
            sh:property [
            sh:path ex:id;
            sh:mincount  1 ;
            sh:maxCount  1 ;
            sh:datatype xsd:integer ;] ;
sh:property [
      sh:path   ex:installedOn ;
      sh:minCount  1 ;
      sh:maxCount  1 ;
      sh:nodeKind sh:IRI ;
      sh:class ex:Door ;  # le capteur doit être installé sur un objet de class Door
   ] .
ex:DoorShape
  a sh:NodeShape ;
  sh:targetClass ex:Door ;
  sh:property [
      sh:path ex:locatedAt;
      sh:minLength 4 ;
      sh:maxLength 7 ;    ] .
```

Tableau 3.10 Un capteur doit être installé sur une entité de type Door, la localisation est typé par un string n'excédant pas sept caractères.

Nous attirons l'attention du lecteur que si nous avions déclaré que la propriété locatedAt est de type string, alors la **sh:maxLength et sh:minLength** seront omises et les contraintes ne seront pas prises en compte.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.com/ns#>  .
@prefix sh: <http://www.w3.org/ns/shacl#>  .
ex:Sensor1
  rdf:type ex:DoorSensor;
  ex:id 12 ;
 ex:installedOn  ex:Door1  .
ex:Door1
   rdf:type ex:Door;
   ex:locatedAt "kitchen" .
ex:Sensor2
  rdf:type ex:DoorSensor;
  ex:id 25 ;
 ex:installedOn  ex:Door2  .
ex:Door2
   rdf:type ex:Door;
   ex:locatedAt  "Microwave" .
```

Tableau 3.11 Graphe RDF

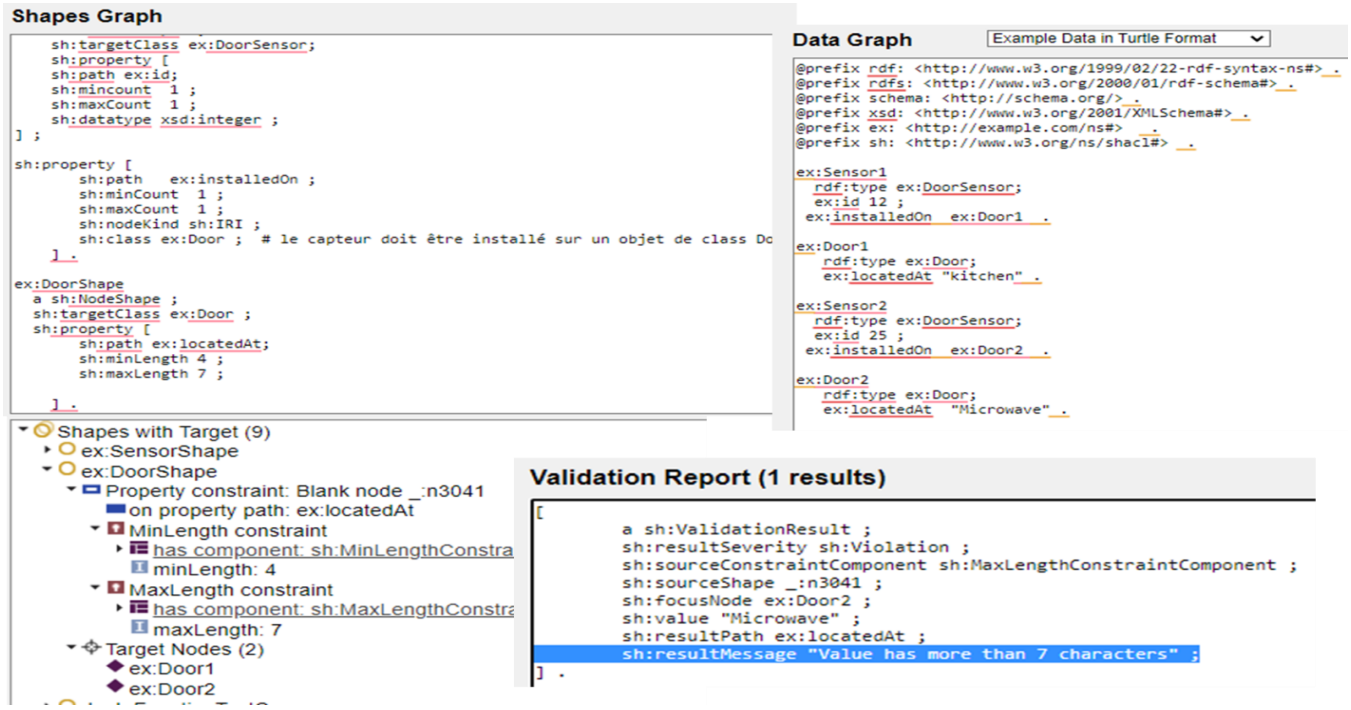


Figure 3.7 Echec de validation signalée par ShACL playground

La Figure 3.8 illustre l'échec de validation du document définit dans le Tableau 3.12.

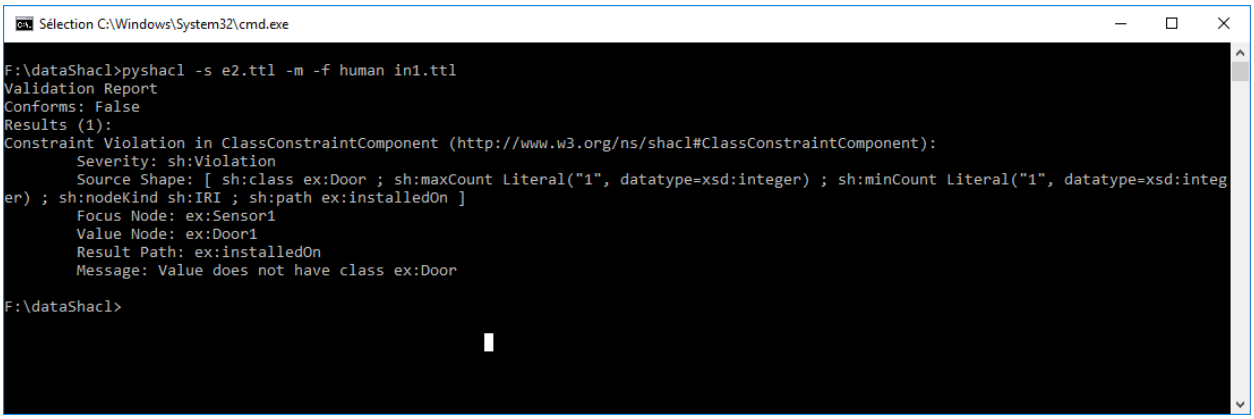


Figure 3.8 Echec de validation d'un graphe RDF déclarant un type Fridge au lieu de type Door.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.com/ns#>  .
@prefix sh: <http://www.w3.org/ns/shacl#>  .

ex:Sensor1
  rdf:type ex:DoorSensor;
  ex:id 12 ;
  ex:installedOn  ex:Door1  .

ex:Door1
   rdf:type ex:Fridge;
   ex:locatedAt "kitchen" .

ex:Door1
   rdf:type ex:Door;
  ex:locatedAt "kit" .
```

Tableau 3.12 Exemple de données illustrant que l'instance Door1 est de type Fridge

3.6 de négation de graphe.

La négation est considérée dans le domaine des ontologies une contrainte difficile à gérer. Pour cette raison, nous avons préféré montré comment la négation est utilisée dans le cadre du modèle ShACL. Supposons que nous souhaitons être sûr ne pas pouvoir utilisé l'ontologie foaf

pour définir des connaissances, étant donné que cette ontologie est dédié à la définition d'un vocabulaire entre amis. Ainsi, le code suivant va contrôler que toute information ne peut jamais être préfixée par foaf.

<table>
<tr><td colspan="2">@prefix ex: <http://example.com/ns#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf:<http://xmlns.com/foaf/0.1/> .</td></tr>
<tr><td>Shape Graph</td><td>Graphe RDF</td></tr>
<tr><td>ex:DoorShape
a sh:NodeShape ;
sh:targetClass ex:Door ;
sh:property [
sh:path ex:locatedAt;
sh:minLength 4 ;
sh:maxLength 7 ;

] ;

sh:not [
sh:property [
sh:path foaf:name;
sh:minCount 1 ;
] ;
] .</td><td>ex:Sensor2
rdf:type ex:DoorSensor;
ex:id 25 ;
ex:installedOn ex:Door2 .

ex:Door2
rdf:type ex:Door;
foaf:name "message" .</td></tr>
</table>

Tableau 3.13 Exemple de validation par le mécanisme de négation.

La validation d'un nœud cible par rapport à une contrainte est définie par les validateurs du composant de contrainte. Ces validateurs prennent généralement en entrée le focus node, les valeurs spécifiques des paramètres de la contrainte dans le graphe de formes, et les nœuds de valeur de la forme. Lors de la validation, le graphe de données et le graphe de formes doivent rester immuables, c'est-à-dire que les deux graphes en fin de validation doivent être identiques au graphe en début de validation. Les processeurs ShACL ne doivent pas modifier les graphes qu'ils utilisent pour construire le graphe de formes ou le graphe de données, même si ces graphes font partie d'une base RDF.

La validation est donc un appariement entre des entrées et les résultats de la validation. Ainsi, un graphe de données et un graphe de formes sont représentés par l'union des résultats de la validation du graphe de données par rapport à toutes les formes du graphe de formes. Quant à la validation d'un graphe de données par rapport à une forme est représentée par des résultats de la validation présentée par l'union des résultats de la validation de tous les nœuds de focus qui se trouvent dans la cible de la forme dans le graphe de données.

Le résultat de la validation d'un nœud de focus par rapport à une forme est l'union des résultats de la validation du nœud de focus par rapport à toutes les contraintes déclarées par la forme, sauf si la forme a été désactivée, auquel cas les résultats de la validation sont vides.

Chapitre 4: **Etude De Cas.**

4.1 Introduction

Dans ce chapitre nous allons étudier deux cas de la vie courante. La première concerne les environnements ambiants et les technologies associés au paradigme de l'internet des objets. Quant à la seconde, nous voulons introduire une ontologie RDF de la gestion d'un environnement quoi que simple mais complexe à gérer, il s'agit de la gestion de données d'un environnement hôtelier. Ces deux cas nécessitent la création des ontologies pour la représentation des connaissances. Nous avons également besoin d'utiliser trois outils que nous avions trouvé très utile pour la mise en œuvre. Nous avons utilisé le langage de programmation Python, la librairie pyshacl, Figure 4.1 et enfin un outil très pratique en ligne, ShACL Playground.

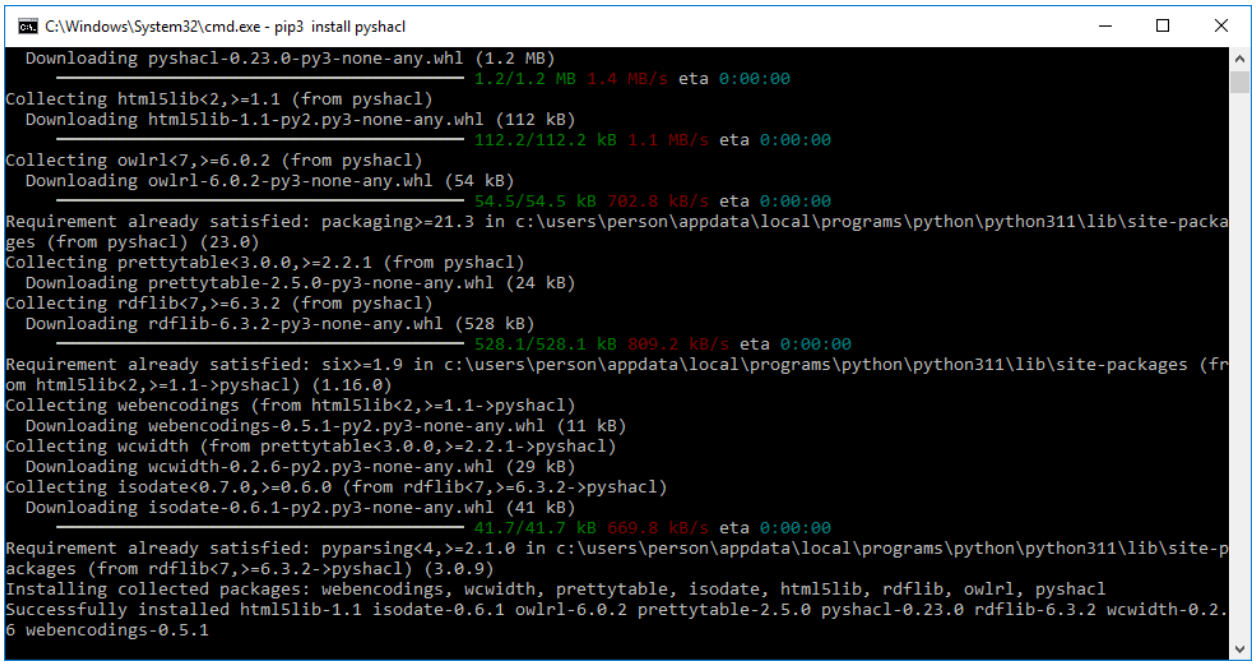


Figure 4.1 Processus d'installation de la librairie pyschal via la commande pip3 install pyshacl.

4.2 Modèle d'ontologies

Les Figure 4.2 et Figure 4.4et présentent respectivement quelques concepts tels que Camera, Lumière, Pression, et les triplets RDF (Sujet, Prédicat et Objet) existants entre les différents concepts et instances de notre ontologie pour la gestion d'un environnement intelligent. Les principaux triplets de cette ontologie sont décrits par la :

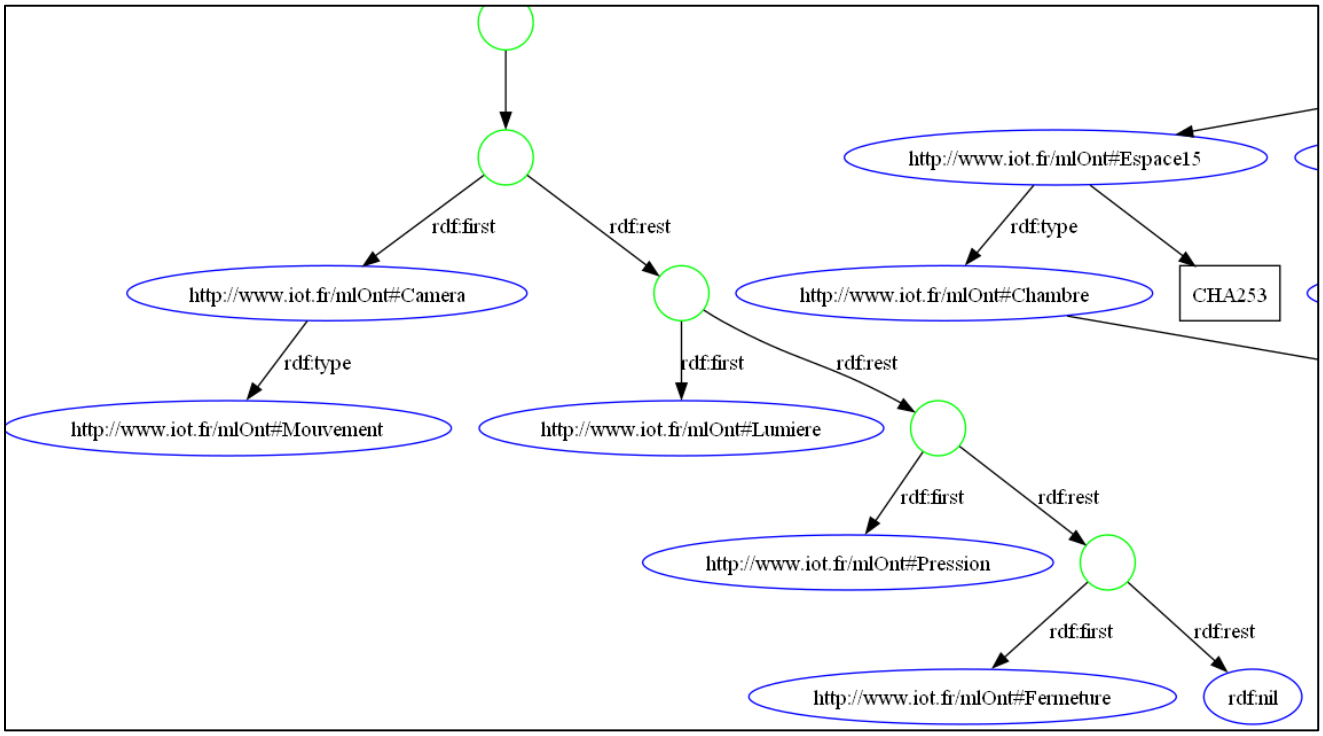


Figure 4.2 Fragment des concepts définis dans l'ontologie

La suite de ce chapitre continue à enrichir avec des exemples simples développés tout au long des chapitres précédents. Il ne s'agit pas ici de faire tout un inventaire des possibilités offertes par le langage ShACL, mais plutôt de présenter sa puissance en termes de contrôle de la qualité de données et de par conséquent de validation.

Un des fondements clefs à tester concerne la supposition de non unique (en Anglais, Unique Name Assumption). Du fait que les ressources sont distribuées, il est tout à fait admis que des personnes utilisent le même IRI pour identifier des ressources différentes. D'autres utilisent différents IRIs pour décrire une même ressource. Par exemple, une même ville peut être décrite

par plusieurs personnes utilisant chacun son propre IRI. Faire l'hypothèse du non unique permet d'attester que des ressources identifiées par des IRIs ne soient pas différentes sauf indications explicite.

Your RDF document validated successfully.

Triples of the Data Model

Number	Subject	Predicate	Object
1	genid:UlisteCapteurs	http://www.iot.fr/mlOnt#listetudier	genid:A14740
2	genid:A14740	http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.iot.fr/mlOnt#Camera
3	http://www.iot.fr/mlOnt#Camera	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Mouvement
4	genid:A14740	http://www.w3.org/1999/02/22-rdf-syntax-ns#rest	genid:A14741
5	genid:A14741	http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.iot.fr/mlOnt#Lumiere
6	genid:A14741	http://www.w3.org/1999/02/22-rdf-syntax-ns#rest	genid:A14742
7	genid:A14742	http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.iot.fr/mlOnt#Pression
8	genid:A14742	http://www.w3.org/1999/02/22-rdf-syntax-ns#rest	genid:A14743
9	genid:A14743	http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.iot.fr/mlOnt#Fermeture
10	genid:A14743	http://www.w3.org/1999/02/22-rdf-syntax-ns#rest	http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
11	http://www.iot.fr/mlOnt#Espace15	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Chambre
12	http://www.iot.fr/mlOnt#Espace15	http://www.iot.fr/mlOnt#id	"CHA253"
13	http://www.iot.fr/mlOnt#Espace23	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Couloir
14	http://www.iot.fr/mlOnt#Espace23	http://www.iot.fr/mlOnt#id	"COU0387"
15	http://www.iot.fr/mlOnt#Espace4	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Parking
16	http://www.iot.fr/mlOnt#Espace4	http://www.iot.fr/mlOnt#id	"CHAI598"
17	http://www.iot.fr/mlOnt#Espace66	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Cours
18	http://www.iot.fr/mlOnt#Espace66	http://www.iot.fr/mlOnt#id	"PT230"
19	http://www.iot.fr/mlOnt#Mouvement23	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Capteur
20	http://www.iot.fr/mlOnt#Mouvement23	http://www.iot.fr/mlOnt#id	"P1253"
21	http://www.iot.fr/mlOnt#Mouvement23	http://www.iot.fr/mlOnt#installer	http://www.iot.fr/mlOnt#Espace15
22	http://www.iot.fr/mlOnt#Lumiere5	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Capteur
23	http://www.iot.fr/mlOnt#Lumiere5	http://www.iot.fr/mlOnt#id	"L3625"
24	http://www.iot.fr/mlOnt#Lumiere5	http://www.iot.fr/mlOnt#installer	http://www.iot.fr/mlOnt#Espace01
25	http://www.iot.fr/mlOnt#Pression1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Capteur
26	http://www.iot.fr/mlOnt#Pression1	http://www.iot.fr/mlOnt#id	"PR563"
27	http://www.iot.fr/mlOnt#Pression1	http://www.iot.fr/mlOnt#installer	http://www.iot.fr/mlOnt#Espace4
28	http://www.iot.fr/mlOnt#Fermeture3	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Capteur
29	http://www.iot.fr/mlOnt#Fermeture3	http://www.iot.fr/mlOnt#id	"F569"
30	http://www.iot.fr/mlOnt#Fermeture3	http://www.iot.fr/mlOnt#installer	http://www.iot.fr/mlOnt#Espace66
31	http://www.iot.fr/mlOnt#Chambre	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Localisation
32	http://www.iot.fr/mlOnt#Couloir	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Localisation
33	http://www.iot.fr/mlOnt#Parking	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Localisation
34	http://www.iot.fr/mlOnt#Cours	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.iot.fr/mlOnt#Localisation

Figure 4.3 Principaux Triplets RDF de l'ontologie

Ce principe est le fil conducteur du fondement RDF et OWL. Similaire au monde ouvert, le non support du non unique à un impact sur la capacité d'inférence à cause des informations redondantes. Le fondement de OWL sur l'hypothèse du monde ouvert et le "Non Unique Name Assumption" était un choix murement réfléchit par le consortium W3C. Les fondateurs d'OWL reconnaissent que pour de nombreuses applications, ces hypothèses ne sont pas admises et leur utilisation serait un échec, mais Ils avaient justifiés leur choix comme suit :

- Le faite qu'OWL était destiné pour le web, alors l'utilisateur devrait être capable de spécifier que tous les espaces de noms véhiculés par les documents doivent référencer différents objets.
- Due aux grands nombre d'échange de pages web, le monde fermé n'est pas approprié. En effet, L'absence d'une information est considérée par OWL qu'elle n'est pas disponible pour le moment. Contrairement à des nombreuses applications devant avoir une réponse concrète oui/non.

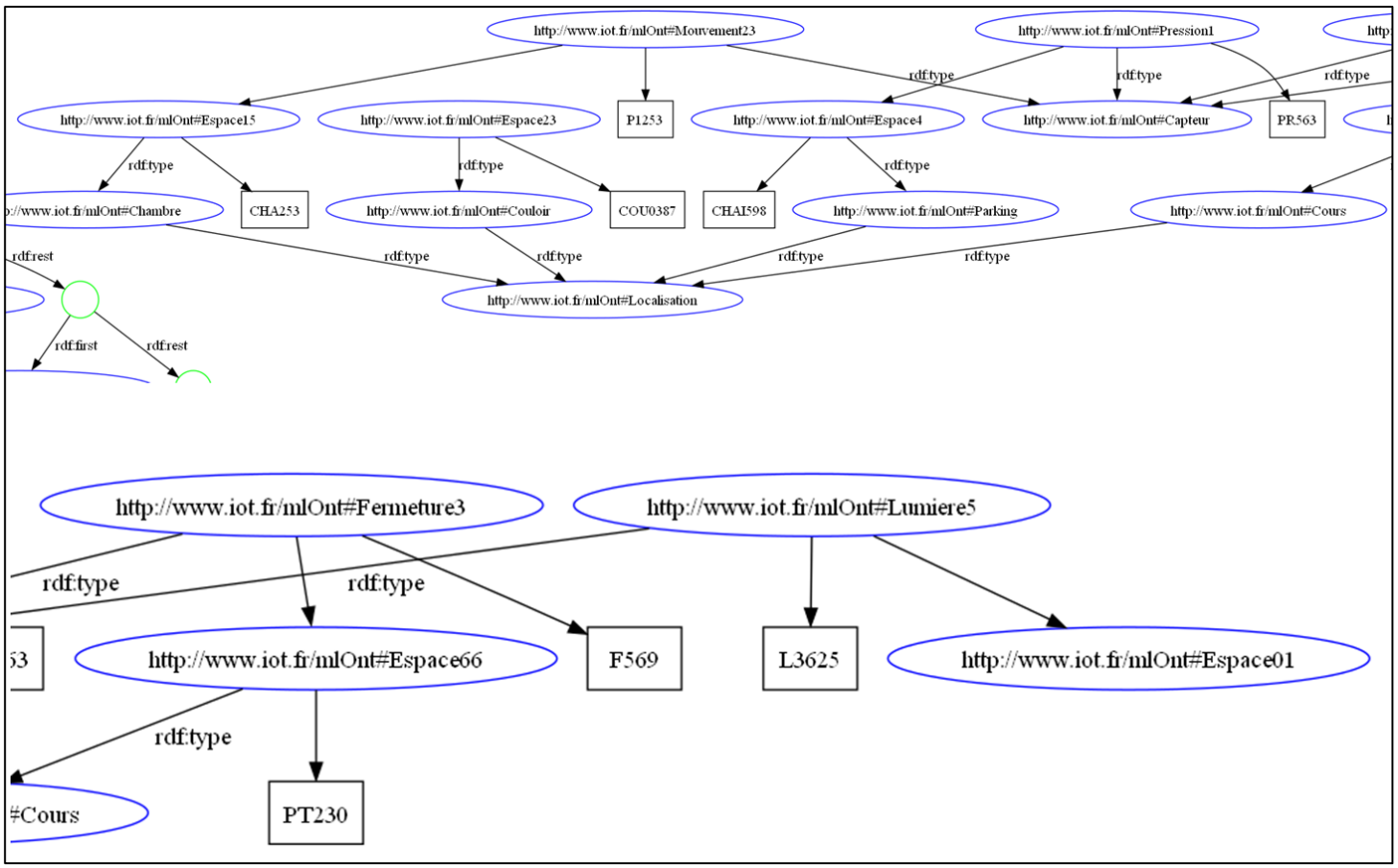


Figure 4.4 Fragment de l'otologie dédiée à la gestion d'un environnement ambiant

Pour ce faire, nous avons créé la forme ShACL présentée par le Tableau 4.1 et la Figure 4.5. Il s'agit ici de vérifier qu'une propriété possède exactement un seul nœud. Dans notre étude de cas, nous allons montrer que le paramètre XONE permet définir un id sans que l'on se souci du

type de la valeur pourvu que l'espace ou le capteur possède un seul et unique ID. En effet, le contenu de l'id doit être géré par d'autres attributs.

```
@prefix  : <http://www.iot.fr/mlOnt#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:ObjectShape a sh:NodeShape;
 sh:targetClass  :Parking;
 sh:xone (
  [
     sh:path      :id ;
    sh:minCount 1;
    sh:maxCount 1

] ) .
```

Tableau 4.1 Forme ShACL pour contrôler l'unicité d'un prédicat

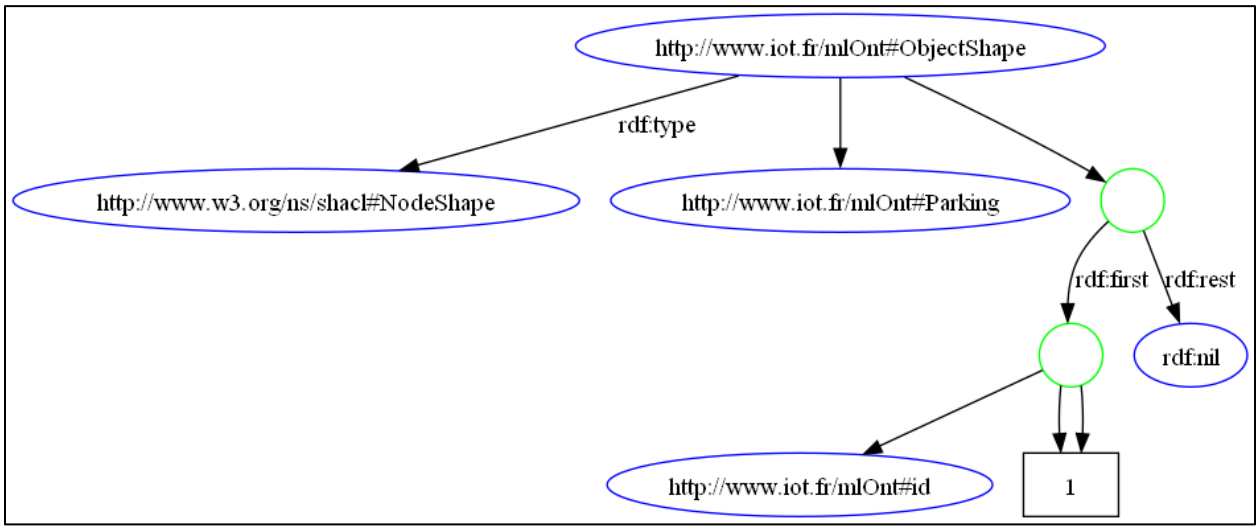


Figure 4.5 Représentation graphique de la forme ShACL pour garantir l'unicité d'une valeur.

Il est très important de souligner ici que la règle ShACL représentée par la Figure 4.5illustre que le format général de la règle est un conteneur, de plus, cette règle indique bien qu'une unique valeur id doit être appariée avec une donnée dans un graphe RDF. Le résultat d'exécution de cette règle est présenté par les figures respectives et.

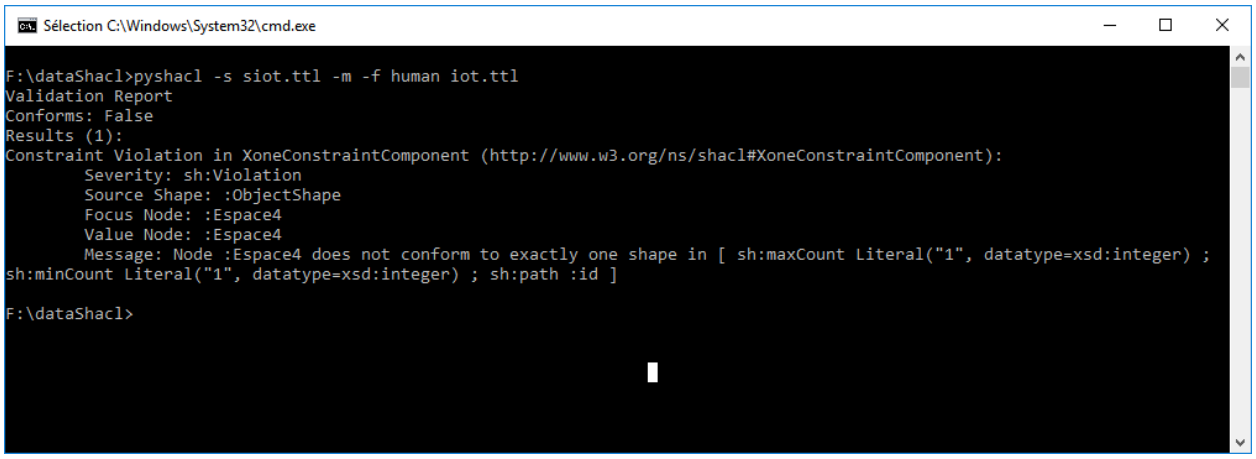


Figure 4.6 Résultat montrant l'échec de validation lors du processus de validation de la règle ShACL pour le contrôle de l'unicité d'un prédicat des données Tableau 4.2. (la librairie pyshacl)

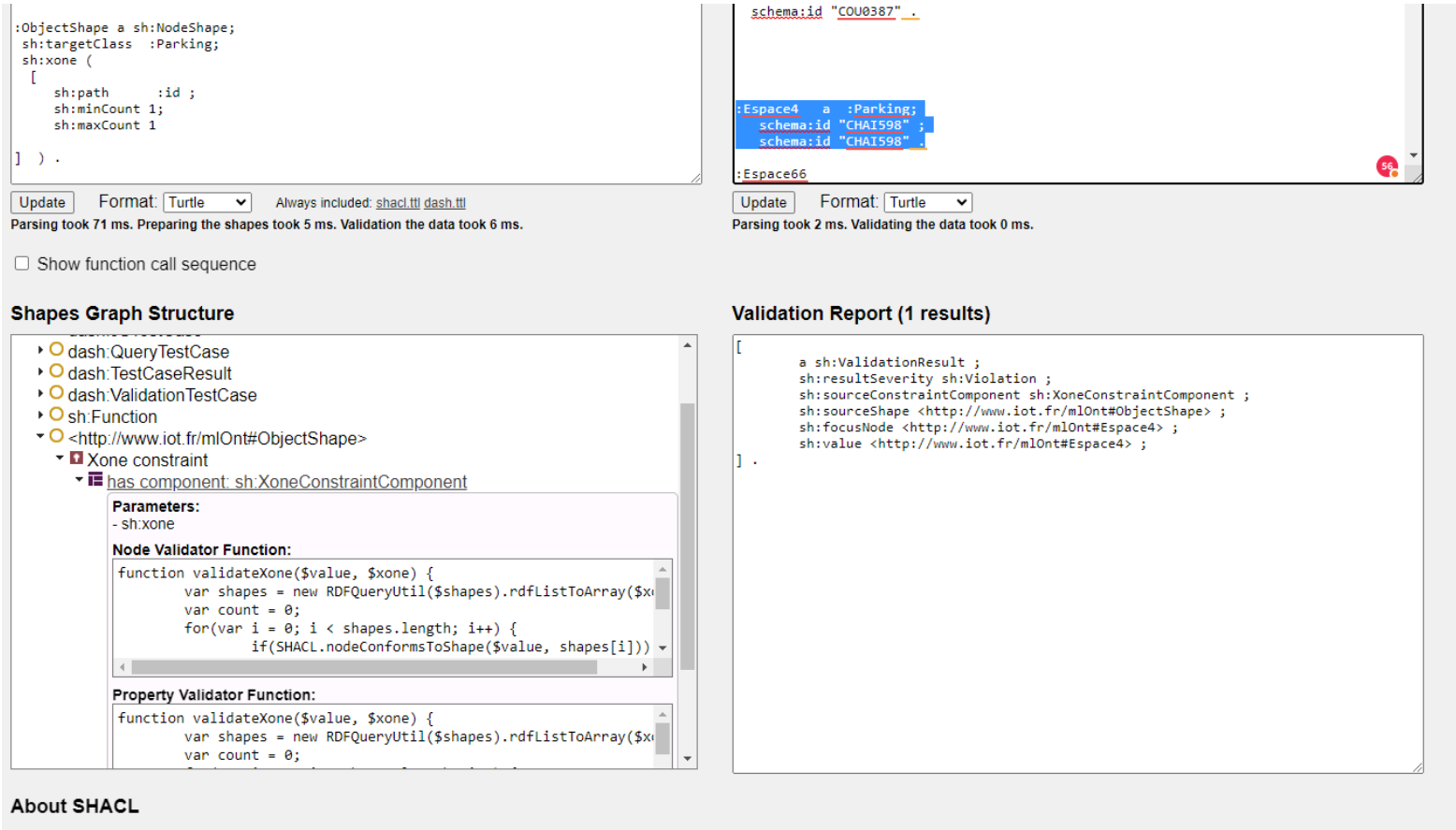


Figure 4.7 Résultat d'exécution de la même règle ShACL pour le contrôle de l'unicité d'une valeur avec l'outil ShACL Playground.

```
@prefix  : <http://www.iot.fr/mlOnt#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://www.iot.fr/mlOnt#Camera> a <http://www.iot.fr/mlOnt#Mouvement> .
<http://www.iot.fr/mlOnt#Espace15>
  a <http://www.iot.fr/mlOnt#Chambre>, :Localisation  ;
   :id "CHA253" .
:Chambre a :Localisation .
:Couloir a :Localisation .
:Parcking a :Localisation .
:Cours a :Localisation .
:Espace23
  a  :Couloir  , :Localisation ;
  schema:id "COU0387" .
:Espace4   a  :Parking,  :Localisation ;
schema:id "CHAI598" ;
   schema:id "CHAI598" .
:Espace66
  a  :Cours,  :Localisation ;
   schema:id "PT230" .
:Mouvement23
  a  :Capteur ;
   schema:id "P1253" ;
   schema:installer  :Espace15 .
:Lumiere5
  a  :Capteur ;
   schema:id "L3625" ;
   schema:installer  :Espace01 .
:Pression1
  a  :Capteur ;
   schema:id "PR563" ;
   schema:installer  :Espace4 .
:Fermeture3
  a  :Capteur ;
   schema:id "F569" ;
   schema:installer  :Espace66 .
[]  :listetudier (
    :Camera
    :Lumiere
    :Pression
:Fermeture  ) .
```

Tableau 4.2 Quelques données décrites en format Turtle

Il est très important de noter ici que même si le champ des valeurs les deux prédicats ids sont identiques, le langage ShACL signale tout de même une erreur de validation. En effet, la règle vérifie qu'un seul prédicat nommé id doit être utilisé par un nœud (ressource rdf). Lors de l'exécution de la règle avec le document RDF, ShACL signale l'impossibilité qu'un espace de type Parking lui soit attribué plus qu'un seul id. Cette possibilité de verrier et valider la valeur d'une information est indispensable dans les systèmes intelligents fondés sur le Web sémantique telles que la santé, le transport, etc.

L'ontologie suivante un peu plus grande que la première est représentée par la. Dans cette ontologie nous cherchons à concevoir un modèle RDF pour la gestion sémantique d'un service hôtelier. Dans ce teste, nous avions voulu vérifier une autre manière de gérer la possibilité d'interdire qu'un prédicat apparaisse plus qu'une fois. La différence avec XONE testé ci-dessus, est que XONE autorise un et un seul attribut, et donc préserve la notion de Unique Name Assomption (UNA) , mais il faut éviter d'utiliser minCount pour le principe UNA , même si cela fonctionne.

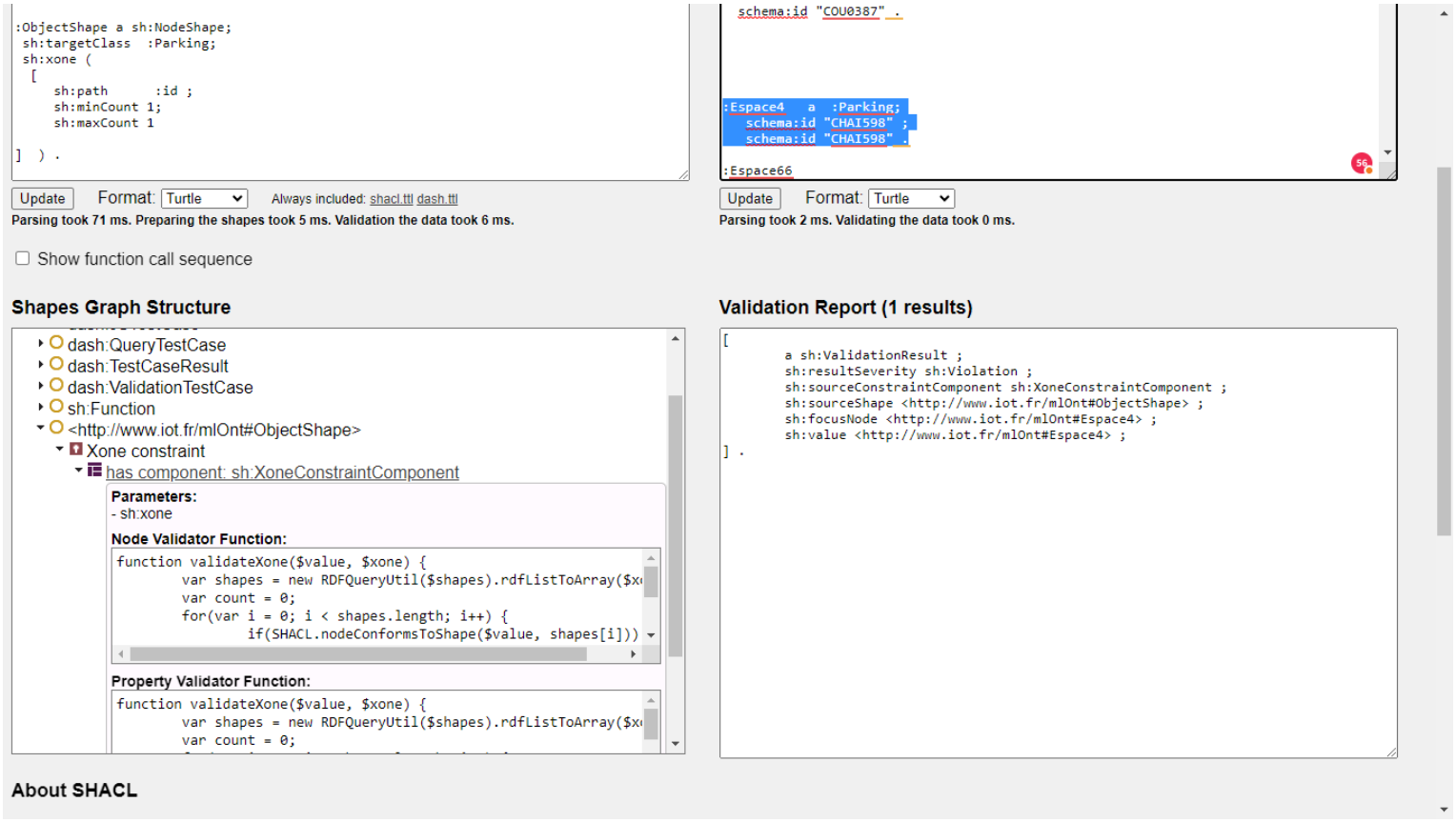


Figure 4.8 Résultat d'exécution de la même règle ShACL pour le contrôle de l'unicité d'une valeur avec l'outil SHACL Playground

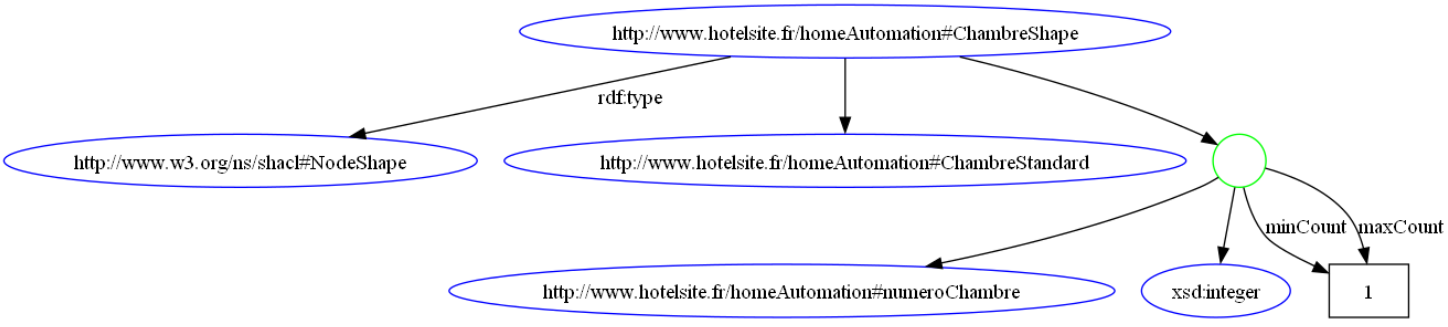


Figure 4.9 Représentation sous forme graph RDF de la règle ShACL Tableau 4.3.

Le Tableau 4.1 montre la forme ShACL pour appliquer le UNA. Il devient très claire que les valeurs minCount et max Count sont soumises à des erreurs de saisies, ou d'oubli, tandis que XONE agit comme un opérateur logique.

```
    @prefix      :<http://www.hotelsite.fr/homeAutomation#> .
    @prefix sh: <http://www.w3.org/ns/shacl#>  .
    @prefix xsd: <http://www.w3.org/2001/XMLSchema#>  .
    @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
    @prefix schema: <http://schema.org/> .
    @prefix sh: <http://www.w3.org/ns/shacl#> .

    :ChambreShape a sh:NodeShape;
     sh:targetClass ns0:ChambreStandard;

      sh:property  [
        sh:path     ns0:numeroChambre ;
        sh:datatype xsd:integer;
        sh:minCount 1;
        sh:maxCount 1
 ]  .
```

Tableau 4.3 Garantir l'unicité d'un prédicat via minCount et maxCount (à éviter)

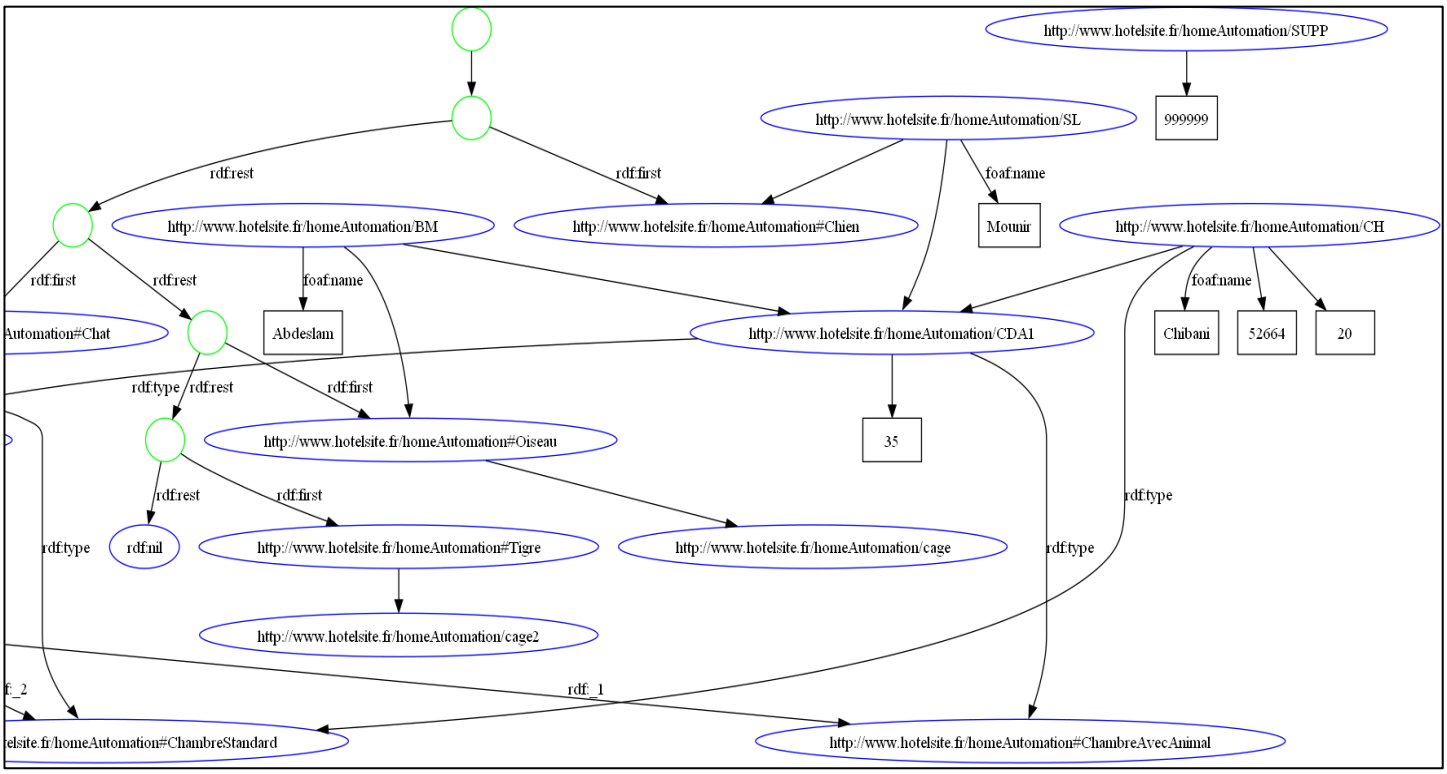


Figure 4.10 Fragment de l'ontologie RDF pour la gestion hôtelier.

```
   @prefix foaf:<http://xmlns.com/foaf/0.1/> .
   @prefix ns0:<http://www.hotelsite.fr/homeAutomation#> .
   @prefix xsd:<http://www.w3.org/2001/XMLSchema#> .
   @prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

   http://www.hotelsite.fr/homeAutomation/SL>
     foaf:name "Mounir";
     ns0:possedeAnimal ns0:Chien;
     ns0:occupe <http://www.hotelsite.fr/homeAutomation/CDA1> .
   <http://www.hotelsite.fr/homeAutomation/CDA1>
     a ns0:ChambreAvecAnimal, ns0:ChambreDouble;
     ns0:numeroChambre "35"^^xsd:Integer.
   <http://www.hotelsite.fr/homeAutomation/BM>
     foaf:name "Abdeslam";
     ns0:possedeAnimal ns0:Oiseau;
     ns0:occupe <http://www.hotelsite.fr/homeAutomation/CDA1> .
   <http://www.hotelsite.fr/homeAutomation/RS>
     foaf:name "Manel";
     ns0:possedeAnimal ns0:Chat;
     ns0:occupe <http://www.hotelsite.fr/homeAutomation/CDS2> .
   <http://www.hotelsite.fr/homeAutomation/CH>
     foaf:name "Chibani";
     ns0:pieceIDentite "52664"^^xsd:Integer;
     ns0:occupe <http://www.hotelsite.fr/homeAutomation/CDA1> ;
     a ns0:ChambreStandard;
     ns0:numeroChambre "20"^^xsd:Integer.

   <http://www.hotelsite.fr/homeAutomation/CDS2>
     a ns0:ChambreStandard, ns0:ChambreTriple;
     ns0:numeroChambre "21"^^xsd:Integer.
   <http://www.hotelsite.fr/homeAutomation/EspacesAOccupes> ns0:capaciteChambres [
       a rdf:Bag;
       rdf:_1 ns0:ChambreDouble;
       rdf:_2 ns0:ChambreTriple
   ].
   ns0:ChambreDouble ns0:typeChambres ns0:Chambres.
   ns0:ChambreTriple ns0:typeChambres ns0:Chambres.
   ns0:Chambres ns0:typeChambres [
       a rdf:Alt;
       rdf:_1 ns0:ChambreAvecAnimal;
       rdf:_2 ns0:ChambreStandard
   ].
   ns0:Oiseau ns0:seTrouve <http://www.hotelsite.fr/homeAutomation/cage> .
   ns0:Tigre ns0:seTrouve <http://www.hotelsite.fr/homeAutomation/cage2> .
   <http://www.hotelsite.fr/homeAutomation/SUPP>                    ns0:numeroChambre
"999999"^^xsd:Integer .
   [] ns0:listeanimaux (
      ns0:Chien
      ns0:Chat
      ns0:Oiseau
      ns0:Tigre
).
```

Tableau 4. 4Information associées à un hôtel représentées par un graphe RDF.

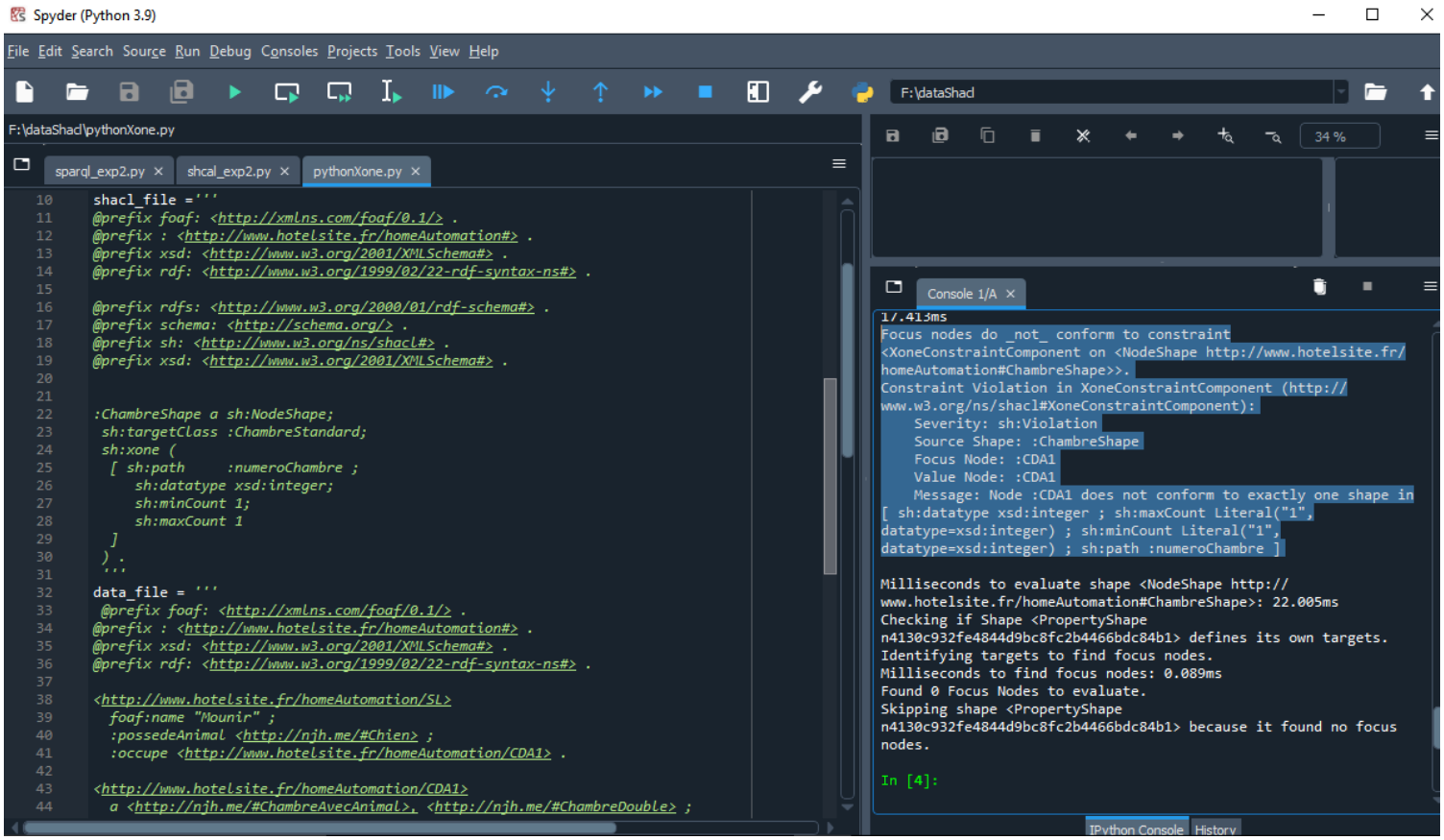


Figure 4.11 Résultat du processus de validation de la forme ShACL Tableau 4.3.

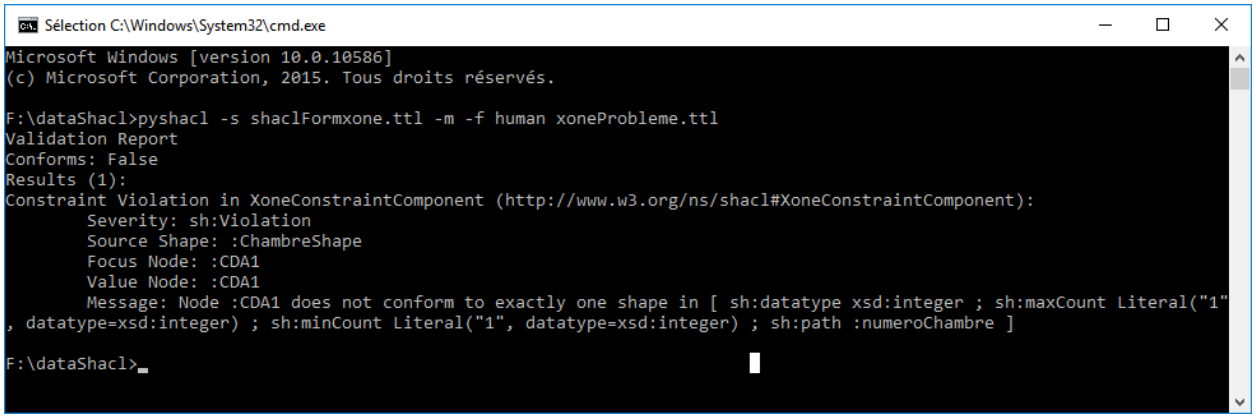


Figure 4.12 Même résultat de violation de la règle obtenu via la librairie pyshacl.

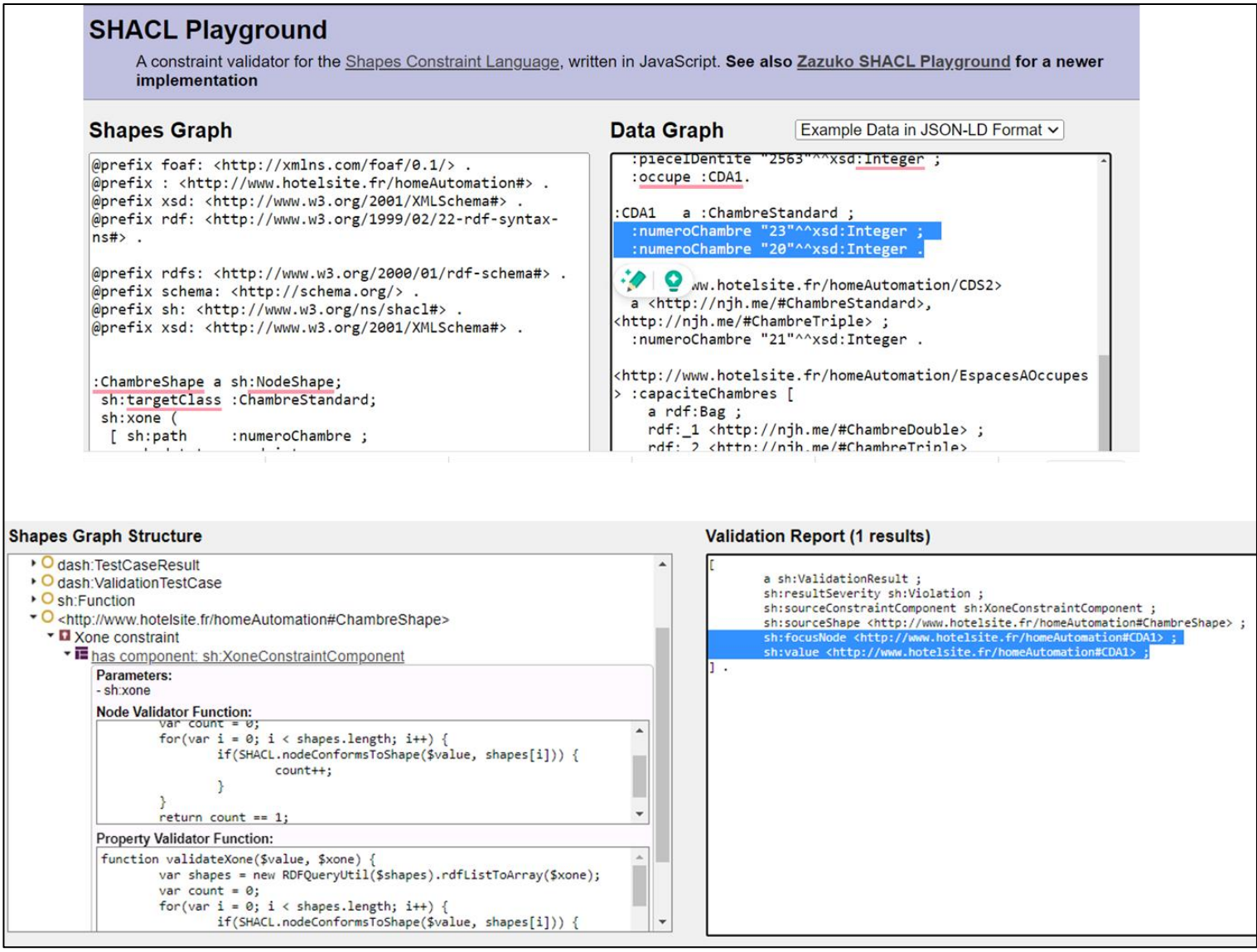


Figure 4.13 Même résultat de violation de la règle obtenu avec ShACL Playground

4.2.1 Tests portés sur les motifs et la notion de class

Dans cette section, nous essayons de revenir sur la validation de la valeur d'un nœud de type littéral. Ici nous utilisons les expressions régulières. Pour ce faire, l'attribut sh:pattern de ShACL, est utilisée. Cette instruction est similaire à celle définie dans la spécification SPARQL. Le Tableau 4. 5 présente deux propriétés, la première stipule qu'une valeur doit correspondre à une expression régulière commençant par P et doit avoir au moins 5 chiffres. Quant à la seconde propriété, elle vérifier que si un capteur doit être installé, il ne peut l'être que dans un espace

(c'est à dire Localisation qui est le concept le plus abstrait dans notre ontologie). Le paramètre flag est optionnel, son rôle est de préciser que la valeur du champ id est sensitive à la case. Dans notre cas, si on retire ce flag, une information n'étant pas majuscule ne serait pas validé. Par exemple, l''information suivante ne sera pas validée :

```
:Mouvement78
    a  :Capteur ;
    schema:id "p1021" ;
```

```
@prefix  : <http://www.iot.fr/mlOnt#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
:CapteurShape a sh:NodeShape ;
 sh:targetClass :Capteur ;
 sh:property [
   sh:path    schema:id ;
   sh:pattern "^TC\\d{5,10}" ;
   sh:flags   "i" ;
  ] ;

 sh:property [      # propriété 2 (notion de classes)
   sh:path    schema:installer ;
   sh:class :Localisation
] .
```

Tableau 4. 5Forme ShACL présentant deux propriétés pour valider un graphe RDF contenant des informations sur un capteur.

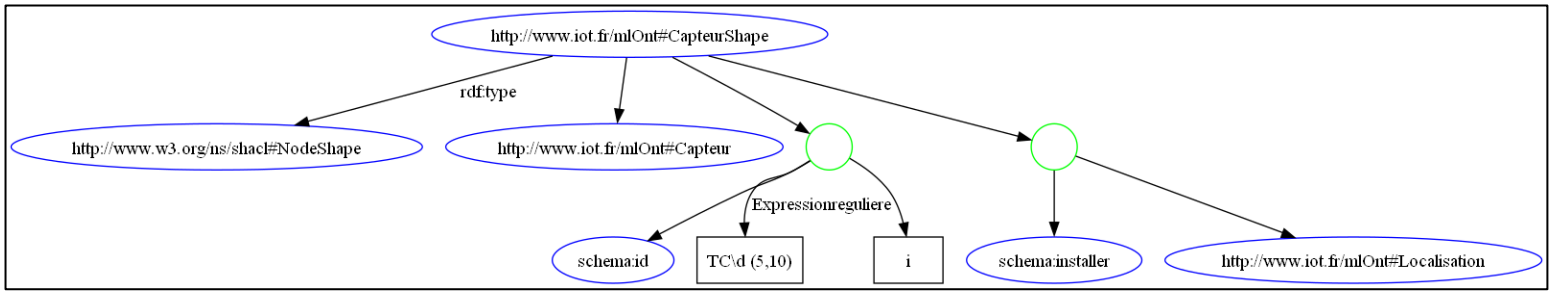


Figure 4.14 Graph de la règle ShACL Tableau 4. 5.

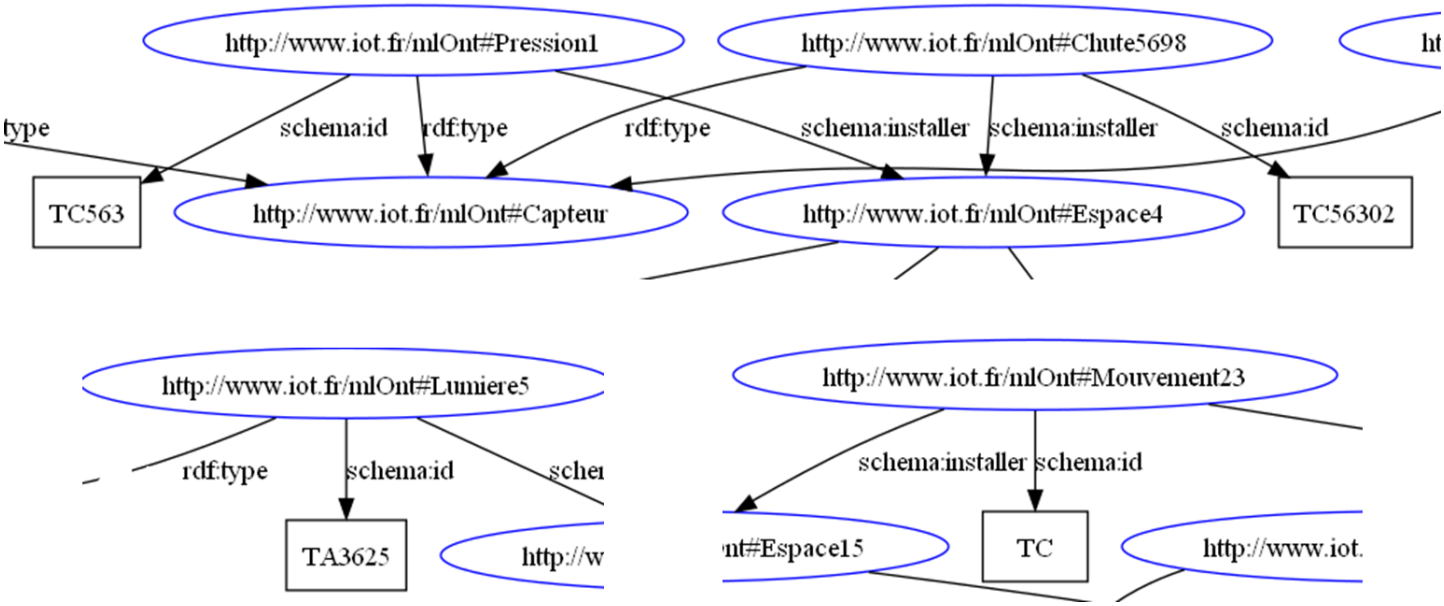


Figure 4.15 Graphe de données à valider par la règle Figure 4.14.

A partir des deux graphes, on observe bien que la seule donnée qui devrait être validée est celle associée au détecteur de chute 5698. L'exécution de la règle via la librairie pyshacl valide ce résultat comme le montre la Figure 4. 16.

```
Sélection C:\Windows\System32\cmd.exe
F:\dataShacl>pyshacl -s shonto.ttl -m -f human onto2.ttl
Validation Report
Conforms: False
Results (3):
Constraint Violation in PatternConstraintComponent (http://www.w3.org/ns/shacl#PatternConstraintComponent):
        Severity: sh:Violation
        Source Shape: [ sh:flags Literal("i") ; sh:path schema1:id ; sh:pattern Literal("^TC\d{5,10}") ]
        Focus Node: :Mouvement23
        Value Node: Literal("TC")
        Result Path: schema1:id
        Message: Value does not match pattern '^TC\d{5,10}'
Constraint Violation in PatternConstraintComponent (http://www.w3.org/ns/shacl#PatternConstraintComponent):
        Severity: sh:Violation
        Source Shape: [ sh:flags Literal("i") ; sh:path schema1:id ; sh:pattern Literal("^TC\d{5,10}") ]
        Focus Node: :Pression1
        Value Node: Literal("TC563")
        Result Path: schema1:id
        Message: Value does not match pattern '^TC\d{5,10}'
Constraint Violation in PatternConstraintComponent (http://www.w3.org/ns/shacl#PatternConstraintComponent):
        Severity: sh:Violation
        Source Shape: [ sh:flags Literal("i") ; sh:path schema1:id ; sh:pattern Literal("^TC\d{5,10}") ]
        Focus Node: :Lumiere5
        Value Node: Literal("TA3625")
        Result Path: schema1:id
        Message: Value does not match pattern '^TC\d{5,10}'

F:\dataShacl>
```

Figure 4. 16 Résultat d'exécution du processus de validation de la valeur d'un Objet.

Le dernier point à montrer dans nos tests est la notion de classe RDF. Pour ce faire, nous avons ajouté à notre ontologie, un triplet comme illustré par la

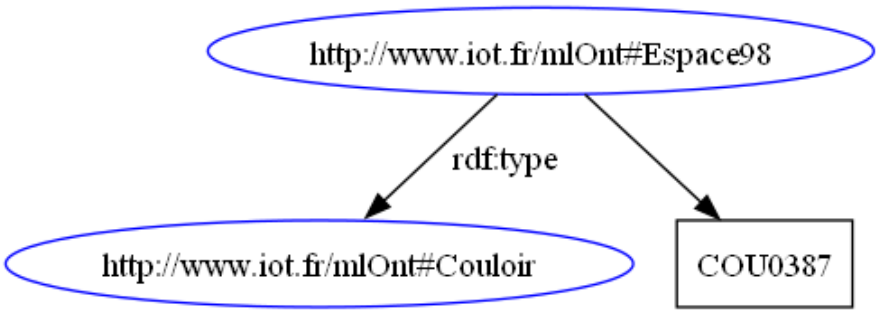


Figure 4.17 Donnée à valider par la règle ShACL propriété 2 dans le Tableau 4. 5.

Remarquons ici que l''instance Espace98 est un Couloir, et rien nous nous dit que Espace98 est une Localisation. La validation de cette nouvelle donnée a échoué comme le montre la Figure 4. . Le résultat détaillé présenté par ShACL est impressionnant. En effet, on peut facilement remonter l'erreur à savoir que le capteur Lumiere5 installé au niveau d'un espace représenté par Espace98. Cette connaissance est obtenue grâce au prédicat installer qui associe les deux ressources Lumiere5 et Espace98. Le message d'erreur indique clairement que le path du prédicat ne trouve pas que Espace58 est de type Localisation comme l'exige la règle ShACL Tableau 4. 5.

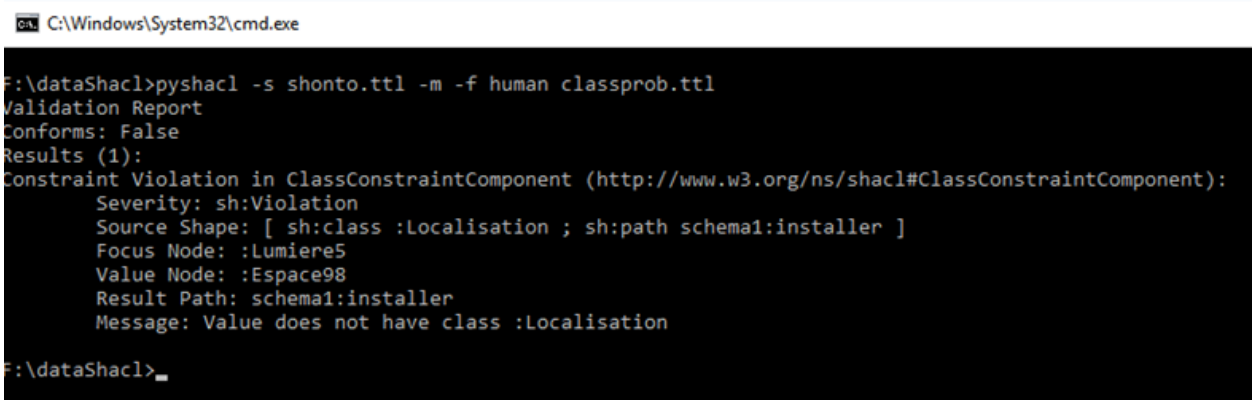


Figure 4.18 Echec de validation de l'information.

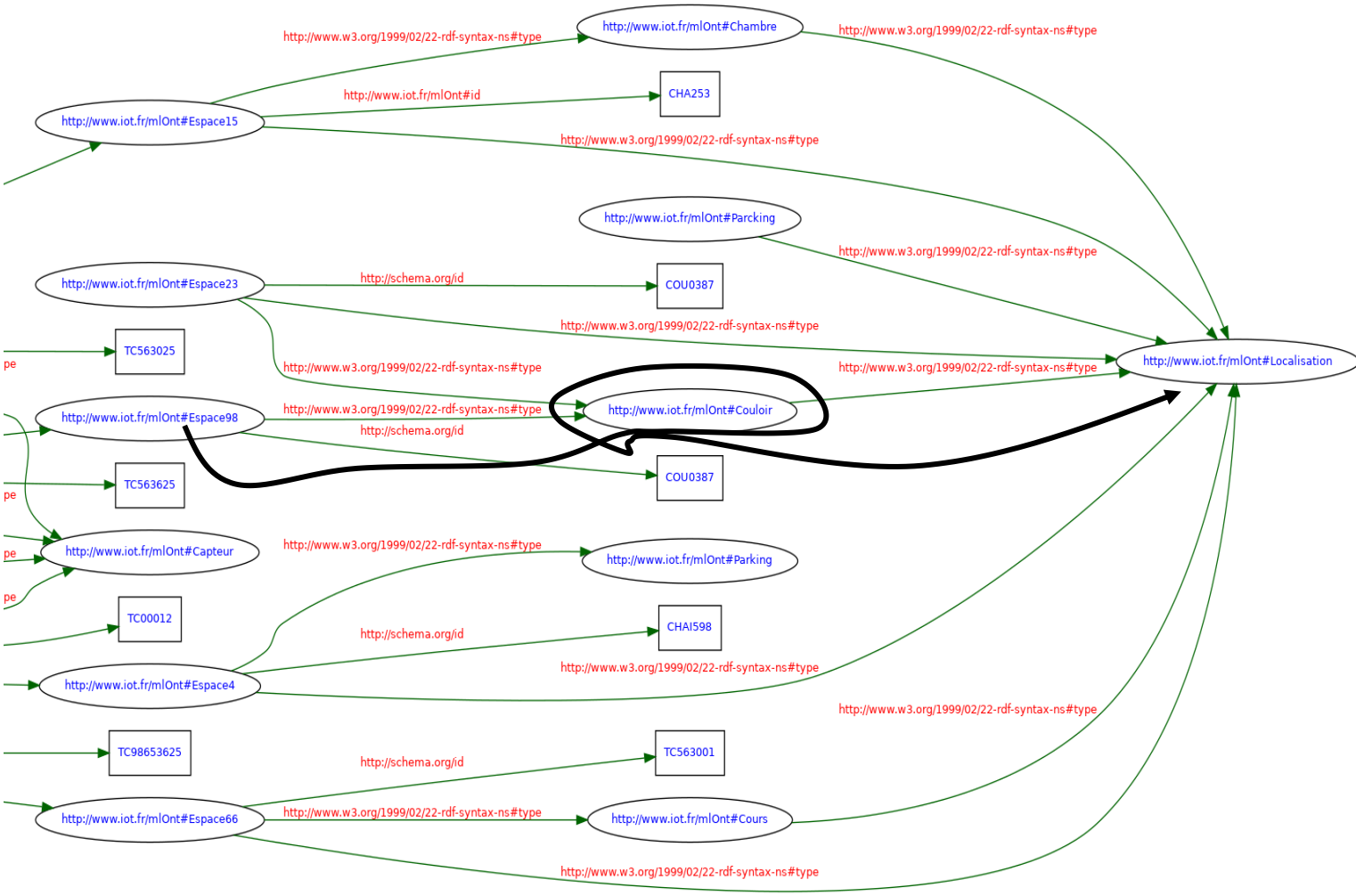


Figure 4.19 Graphe RDF montrant bien que la ressource Espace98 est de type Couloir qui lui-même de type Localisation, mais cette connaissance n'a permis de valider la donnée

Ce qui attire notre attention est que la connaissance Espace98 est de type Localisation existe, mais implicitement dans le graphe RDF de données, Figure 4.. Pour remédier à cette situation, il fallait décrire explicitement que le concept Couloir est une sous classe du concept Localisation. La montre que cette manière la règle ShACL qui a échoué, valide cette fois-ci la donnée.

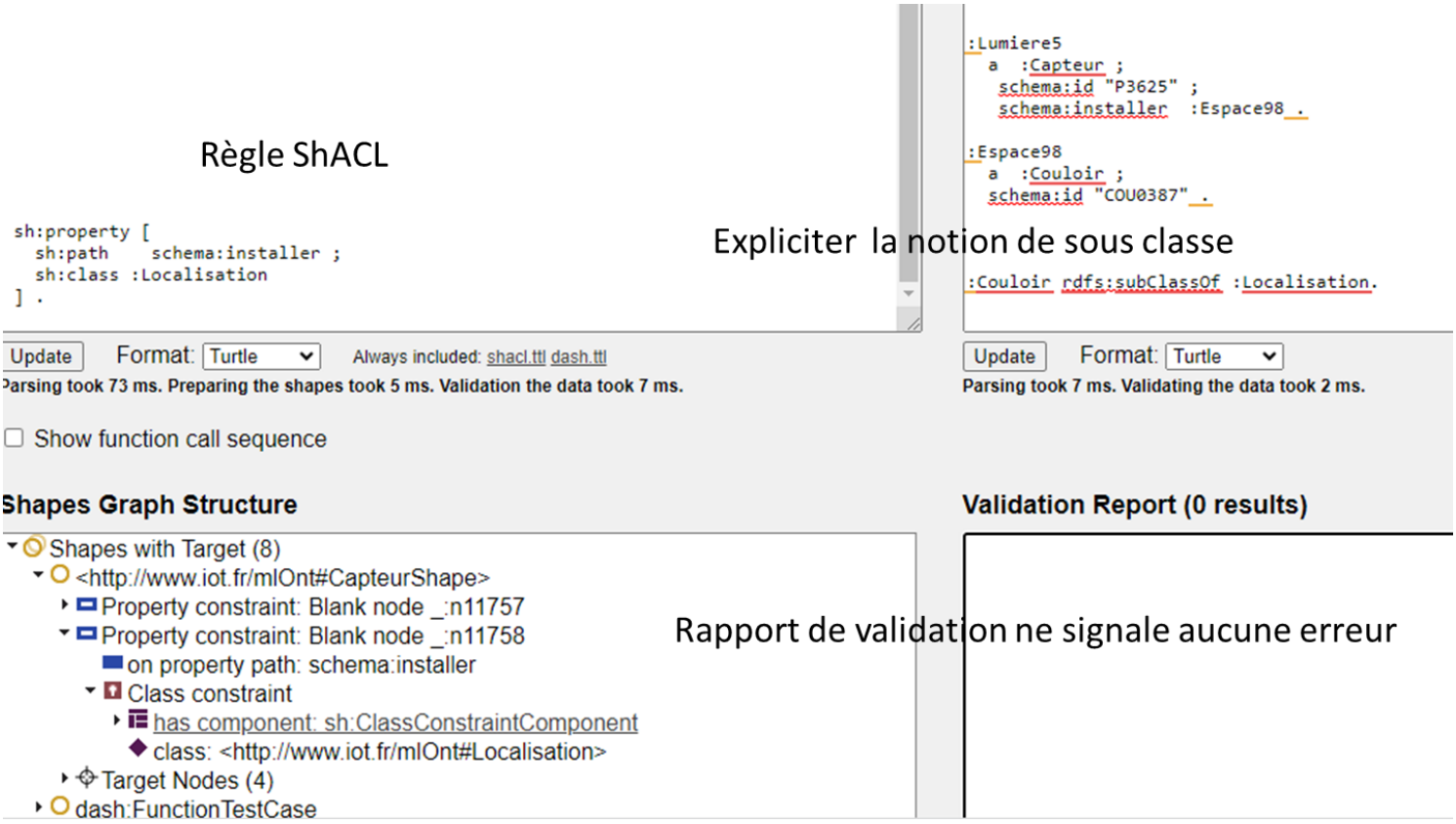


Figure 4.20 Résultat de validation après explicitement déclaré que Couloir est sous classe de Localisation

4.3 Discussion

Pourconclure, Shapes Constraint Language (ShACL) est un standard du W3C pour la validation des graphes RDF; la norme décrit donc deux graphes, le graphe de données et le graphe de formes. Le graphe RDF à valider est le graphe de données, tandis que l'ensemble des contraintes, exprimées également en RDF connu sous le nom de graphe de formes. Ce dernier véhicule des conditions que le graphe de données doit satisfaire. ShACL fournit un ensemble de contraintes de base intégrées qui sont souvent utilisées dans la validation des graphiques, telles que sh:class pour limiter le nœud de focus à une classe spécifique, sh:minCount pour restreindre le nombre de nœuds de valeur pour un nœud de focus/ propriété, et sh:xone pour restreindre la possibilité de définir un prédicat à une seule fois. Lors de notre étude de cas, nous avons constaté que les nœuds de données à atteindre sont définis par sh :NodeShape. Il faut toujours retenir qu'un nœud doit être un IRI et les cardinalités peuvent être non liées (pas de limite) ou imposer une valeur. Dans le cas où aucune cardinalité n'est spécifiée, ShACL suppose la valeur 0 ou plusieurs. De plus ShACL utilise directement la syntaxe RDF. Ce qui donne plus de souplesse à utilisateur qui connait déjà RDF.

ShACL s'appuie également sur SPARQL donc ne s'inspire pas de XML Schema. De ce faite, il peut être comparé à schématron. En effet, shématron utilise des expressions XPATH pour exprimer la présence ou l'absence de motif dans le graphe RDF. Ce qui permet de faire un lien entre des éléments et des attributs éloignés dans un graphe. Cette manière permet de décrire avec précision les erreurs soulevées lors du processus de validation. Toutefois cela pose le problème quand un nœud est validé. En effet, il se peut qu'un nœud ne soit pas valide, mais non atteint par le processus de validation car il n'était pas dans la cible. Ainsi, dans ce précis, il serait plus utile d'utiliser un mécanisme similaire à une grammaire XML Schéma.

La modularité de ShACL permet d'importer des graphique de formes en utilisant la propriété owl:imports. Cette technique permet à ShACL de supporter la notion d'extension. Il s'agit ici d'étendre un schéma, par exemple en tirant profit de la notions de sous-classe, mais

cette possibilité peut devenir très restreinte si le nœud n'a pas retenu la déclaration rdf :type appropriée et la notion de rds :subClassOf, comme nous l'avons constater dans notre dernier test. Enfin, la sémantique SHACL Core est définie en langage naturel avec certains modèles SPARQL. Sa complexité dépend de la complexité de SPARQL, qui peut également être assez coûteux, notamment dans l'utilisation des chemins de propriété.

Chapitre 5: **Conclusion générale**

5.1 Contributions

Comme a été souligné par les auteurs dans [5], la capacité de collecter et de stocker les observations de divers capteurs a explosé. Plusieurs façons potentielles existent pour intégrer ces observations, par exemple, dans l'espace et dans le temps, ainsi que les caractéristiques d'intérêt observées. Toutefois, l'hétérogénéité des modèles de représentation des données des capteurs rend entrave généralement le processus d'intégration.

Dans la vie réelle, les données sont imparfaites, incomplètes et contiennent des erreurs de diverses natures. Par conséquent, le contrôle de la qualité des données RDF en vue d'en extraire ou intégrer les données garantissant la cohérence des données RDF sont des cas que l'on peut aborder dans ShACL. Nous avons que cette approche de validation de données est très prometteuse, comme a été souligné par les auteurs [1]. ShACL est le langage recommandé par le W3C pour exprimer des modèles que les données RDF doivent respecter afin d'assurer la cohérence de l'ensemble de données. Du fait que ShACL a émerger récemment, beaucoup d chercheurs s'y investissent et proposent des travaux comme dans le cas de [2,3,4] et la validation des données RDF avec ShACL est une question de recherche d'actualité largement abordée dans la littérature. En particulier dans le cadre des inférences. La difficulté augmente quand lors qu'une approche tente de considéré le principe du monde ouvert sur lequel est fondé RDF et SPARQL. ShACL est riche en termes d'expressivité puisqu'il épuise se syntaxe de RDF, RDFS ET OWL.

5.2 Critique du travail

Vu la panoplie de règles que nous pouvons exprimer avec ShACL, il nous est impossible donc de tous présenter dans ce document toutes les possibilités de contrôle d'informations. Les règles d'inférences, les langages d'extension de SPARQL rendent l'exercice difficile à appréhender. Le monde des ontologies est complexe, nécessite une compréhension parfaite du principe de représentation des connaissances pour pouvoir avancer

rapidement et aller plus loin de tout étude ayant un lien étroit au Web sémantique. Par conséquent, nous estimons qu'il reste encore un travail à faire qui sera dans le cadre de ce travail de master.

5.3 Travaux futurs et perspectives

Comme nous l'avons souligné dans la section précédente, ShACL n'a pas été étudié dans ce document avec la possibilité d'utiliser les inférences avant de lancer le processus de validation. Cette tâche très utile mais difficile puisqu'elle reste encore une piste de recherche au moment de la rédaction de ce document. Un dernier point concerne la possibilité de comparer d'autres langage de validation en particulier ceux qui sont similaires à XML Schéma.

Les références

[1] R. Felin, C. Faron, et A. G. B. Tettamanzi, « A Framework to Include and Exploit Probabilistic Information in SHACL Validation Reports », présenté à ESWC 2023 - 20th International European Semantic Web Conference, mai 2023. Consulté le: 8 juin 2023. [En ligne]. Disponible sur: https://inria.hal.science/hal-04031744

[2] P. Ghiasnezhad Omran, K. Taylor, S. Rodríguez Méndez, et A. Haller, « Learning ShACL shapes from knowledge graphs », Semantic Web, vol. 14, no 1, p. 101-121, janv. 2023, doi: 10.3233/SW-223063.

[3] G. Kober, L. Robaldo, et A. Paschke, « Modeling Medical Guidelines by Prova and ShACL Accessing FHIR/RDF. Use Case: The Medical ABCDE Approach », Stud Health Technol Inform, vol. 293, p. 59-66, mai 2022, doi: 10.3233/SHTI220348.

[4] M. Verhaeg, L. W. Rutledge, et B. J. Heeren, « ShACL-based Ontology Design Patterns for Evidence-based Decision-making: 11th Workshop on Ontology Design and Patterns », Proceedings of the 11th Workshop on Ontology Design and Patterns, vol. 51, p. 292-298, 2021, doi: 10.3233/SSW210020.

[5] R. Zhu et al., « SOSA-ShACL: Shapes Constraint for the Sensor, Observation, Sample, and Actuator Ontology », in Proceedings of the 10th International Joint Conference on Knowledge Graphs, Virtual Event Thailand: ACM, déc. 2021, p. 99-107. doi: 10.1145/3502223.3502235.