**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**OTÁVIO THOMAS BERTUCINI**

# GARANTINDO A QUALIDADE DE DADOS NA FUSÃO DE DADOS SEMÂNTICOS: UM CASO DE USO DE SHACL EM DADOS DE MOBILIDADE E EDUCAÇÃO

**CURITIBA**

**2022**

**OTÁVIO THOMAS BERTUCINI**

# GARANTINDO A QUALIDADE DE DADOS NA FUSÃO DE DADOS SEMÂNTICOS: UM CASO DE USO DE SHACL EM DADOS DE MOBILIDADE E EDUCAÇÃO

## Ensuring data quality in semantic data fusion: A use case for SHACL in mobility and education data

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Dra. Rita Cristina Galarraga Berardi

**CURITIBA**

**2022**

**OTÁVIO THOMAS BERTUCINI**

# GARANTINDO A QUALIDADE DE DADOS NA FUSÃO DE DADOS SEMÂNTICOS: UM CASO DE USO DE SHACL EM DADOS DE MOBILIDADE E EDUCAÇÃO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 05/dezembro/2022

---

Rita Cristina Galarraga Berardi
Doutorado
Universidade Tecnológica Federal do Paraná

---

LUIZ CELSO GOMES JUNIOR
Doutorado
Universidade Tecnológica Federal do Paraná

---

HELOISE MANICA PARIS TEIXEIRA
Doutorado
Universidade Estadual de Maringá

**CURITIBA**
**2022**

## RESUMO

Para se obter análises completas e confiáveis através dos dados a quantidade de dados disponível é um fator muito importante é ideal que conjuntos de dados cresçam e se tornem mais úteis ao longo do tempo. No entanto, apenas aumentar a quantidade de dados disponíveis sem se preocupar a qualidade não torna os dados mais proveitosos para o usuário e pode até mesmo acabar deixando o conjunto inutilizável. Ao fundir dois ou mais conjuntos de dados, registros incorretos ou incoerentes podem ser fundidos, causando a perda de qualidade do conjunto como um todo. As dimensões de qualidade concisão, consistência e precisão são importantes para garantir a integridade e veracidade do conjunto de dados e o conceito de gerenciamento de dados baseado em ontologias (GDBO) pode ajudar para que estas características estejam presentes em conjuntos de dados que crescem ao longo do tempo. Este trabalho tem como objetivo criar um mecanismo de verificação da dimensões de qualidade de acurácia, consistência e concisão ao fundir conjuntos de dados ligados, permitindo que essas três dimensões de qualidade sejam aferidas e buscando oferecer mecanismos para pessoas que queiram aumentar o quantidade de dados possam entender se os conjuntos de dados utilizados têm qualidade. O mecanismo de verificação foi construído na linguagem SHACL(Shapes Constraint Language) e testado em um conjunto de dados semânticos do domínio de mobilidade urbana e educação. Também foi criado um *script* em Python que permite a execução deste mecanismo em conjunto de dados.



**Palavras-chave:** ontologia; qualidade de dados; web semântica; fusão de dados.

**ABSTRACT**

An important factor for the quality of knowledge gained from the data is the amount of data available for analysis and therefore it is ideal that data sets grow and become more useful over time. However, just increasing the number of available data without worrying about quality does not make the data more useful to the user and may even end up rendering the set unusable. When merging two or more semantic datasets, incorrect or incoherent data can be merged, causing the loss of quality of the set as a whole. The dimensions of quality, conciseness, consistency, and accuracy are important to ensure the integrity and veracity of the dataset, and the concept of ontology-based data management (OBDM) can help to ensure that these characteristics are present in datasets that grow over time. of time. This work aims to create a mechanism that allows these three dimensions of quality to be measured, seeking to provide resources for people who want to increase the number of data to understand whether the data sets used have quality. The mechanism was built in the SHACL language and tested on a set of semantic data from the urban mobility and education domain. A *script* in Python was also created that allows the execution of this mechanism in a dataset.

**Keywords:** ontology; data quality; semantic web; data fusion.

## LISTA DE ALGORITMOS

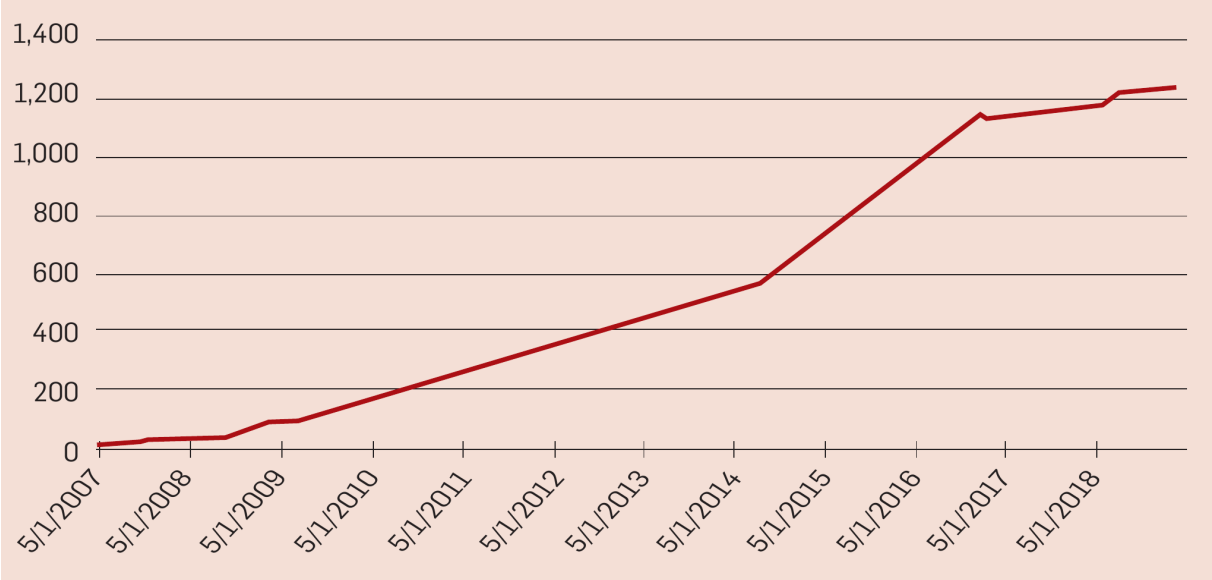
## LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A Web Semântica tem o potencial de revolucionar a maneira como descobrimos, acessamos, integramos e usamos dados, pois permite que dados de contexto diferentes se conectem (HEATH; BIZER, 2011) e dessa forma aumentar a flexibilidade, adaptabilidade e eficiência da gestão da informação no setor público e privado. A Web Semântica é feita de dados semânticos, que são dados que são estruturados de forma que o seu significado pode ser entendido por computadores. Para a comunidade geral, os dados ligados ajudam a obter e integrar as informações de contextos diferentes necessárias para a tomada de decisão de forma mais eficiente, consequentemente tornando a economia baseada em conhecimento e informação (NGOMO *et al.*, 2014). No contexto de cidades inteligentes, por exemplo, é necessário integrar dados de fontes diferentes, com características semânticas diferentes. A criação e aplicação de um modelo de informação unificado permite obter uma visão mais completa da atividade urbana, integrando todas as fontes de dados em um único lugar (NAPHADE *et al.*, 2011).

**Figura 1 – Número de grafos em RDF ao longo dos anos**
**(a)**

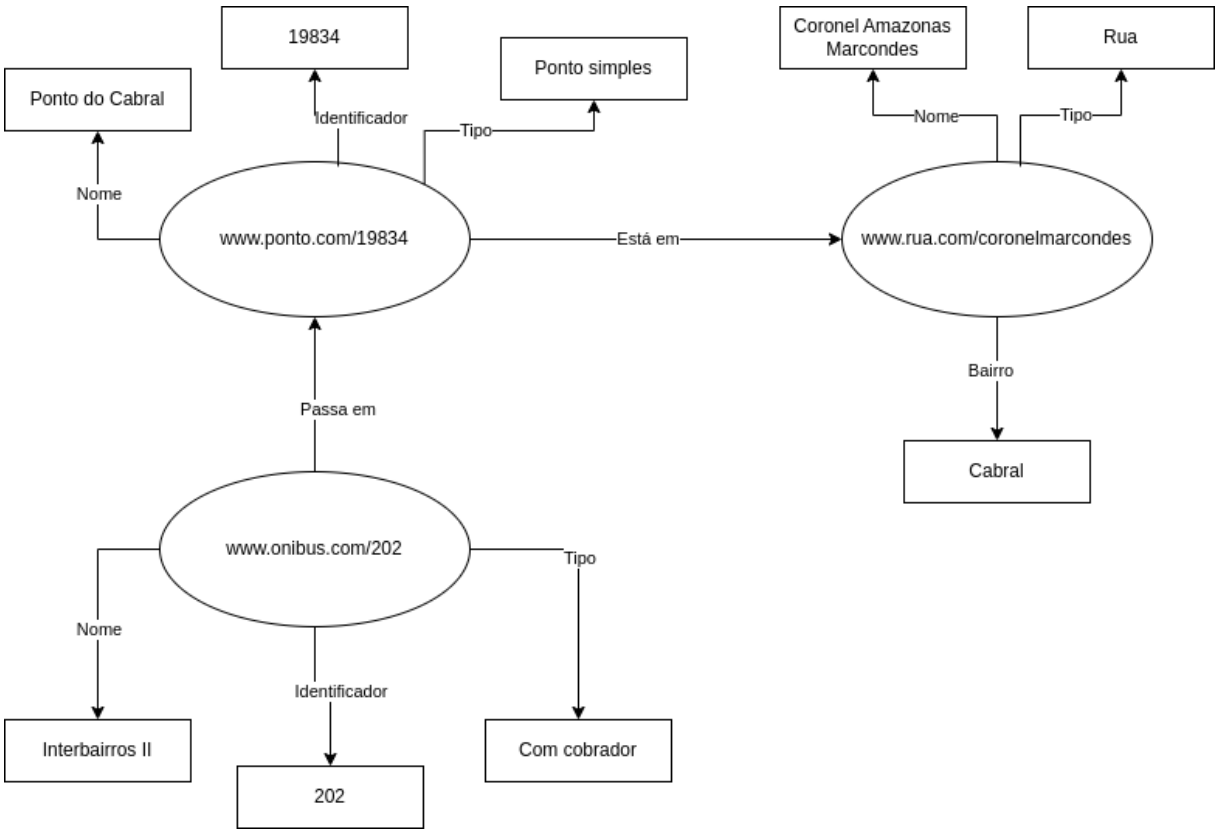



**Fonte: (HITZLER, 2021).**

Um fator importante para a qualidade das inferências feitas a partir de dados, sejam eles da Web Semântica ou de modelos tradicionais, é a quantidade de dados disponível para análise (HALEVY; NORVIG; PEREIRA, 2009). Dessa forma é desejável que modelos de dados cresçam cada vez mais e se tornem cada vez mais favoráveis para a tomada de decisões. Nos últimos anos, houve um grande crescimento no número de informação gerada e armazenada na Web, como demonstrado na Figura 1. Em 2015 foram identificadas mais de 37 bilhões de triplas provenientes de mais de 650 mil documentos com dados acessíveis ao público (RIETVELD; BEEK; SCHLOBACH, 2015). Além do mais, houve um aumento no interesse do setor privado pelo uso

**Figura 2 – Recorte do conjunto de dados inicial**

**(a)**



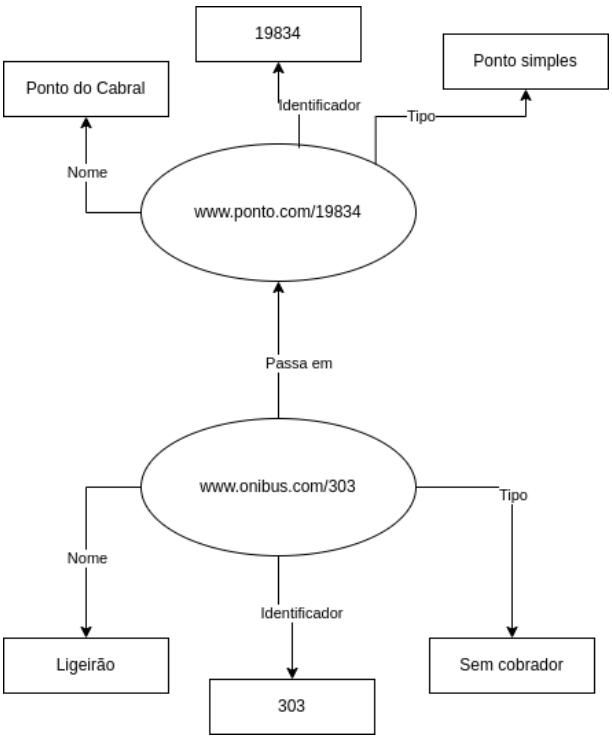

**Fonte: Autoria própria.**

das tecnologias e metodologias da Web Semântica por empresas como a CNN, o Facebook e o New York, demonstrando que existe uma tendência de mais dados serem publicados e utilizados na rede de dados. Porém, apenas crescer o volume de dados na Web não é suficiente para que conhecimento útil seja inferido a partir da Web Semântica (ZAVERI *et al.*, 2016).

Segundo Hyland e Wood (2011) devemos aumentar conjuntos de dados possibilitando que o usuário consiga usar esses dados para inferir conhecimento. Um exemplo que ilustra essa situação é o de (HOGAN *et al.*, 2010): "Alice pesquisa dados sobre ela mesma em uma rede de dados ligados e começa a explorar dados que estão integrados ao seus dados pessoais: país de origem, blogs, universidade, datas, lugares de interesse. Porém, Alice percebe que muitos desses dados conectados não contêm informações corretas, têm valores inválidos, se contradizem, contêm informações divergentes ou as conexões com outros objetos da Web não estão certas ou não existem. Portanto, mesmo Alice tendo uma base de dados abrangente e extensa, ela não consegue encontrar nenhum dado relevante por causa da baixa qualidade dos dados e que consequentemente se tornam inúteis.“ Dito isso, é importante aferir a qualidade dos dados que serão utilizados para a obtenção do conhecimento, principalmente após a fusão de dois ou mais conjuntos distintos de forma a reduzir o número de situações como a de Alice.

**Figura 3 – Recorte do conjunto de dados que será adicionado**

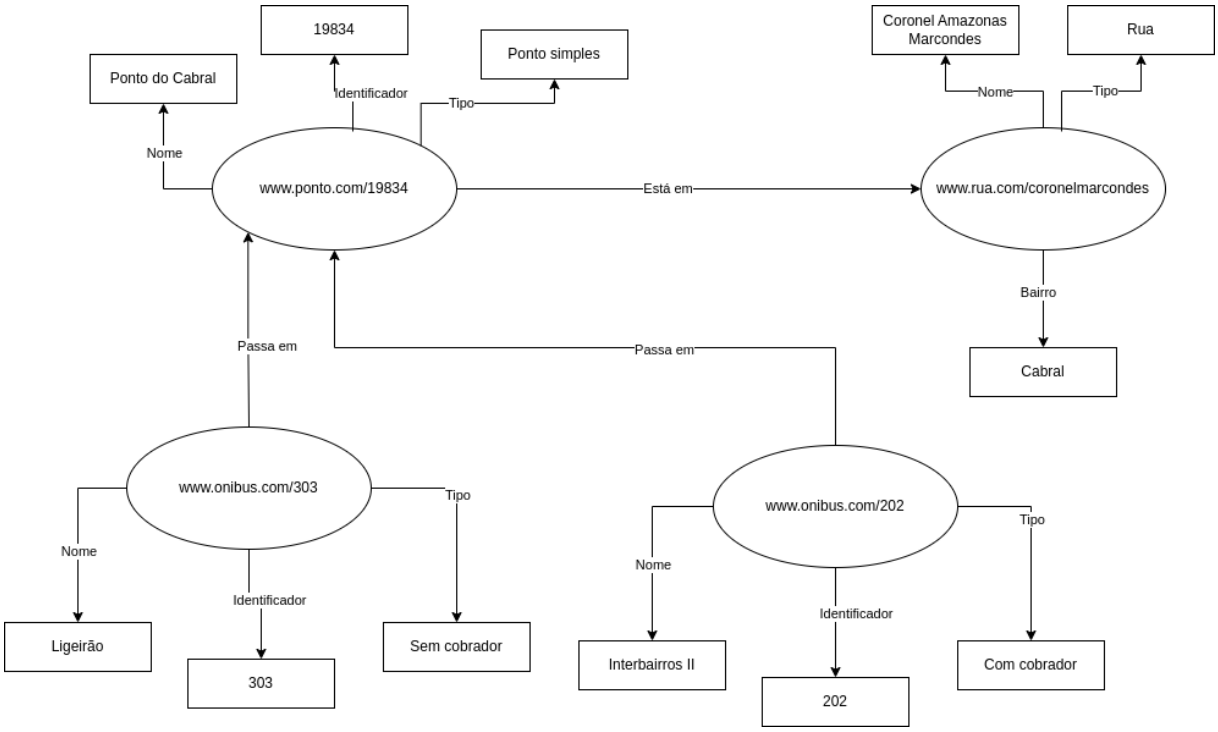
**(a)**




**Fonte: Autoria própria.**

**Figura 4 – Recorte dos dois conjuntos após a fusão**

**(a)**




**Fonte: Autoria própria.**

## 1.1 Problema

Para explanar melhor o problema tratado neste trabalho, vamos supor que determinado usuário esteja interagindo com um conjunto de dados semânticos no domínio de mobilidade urbana que possui três entidades: ruas, pontos de ônibus e linhas de ônibus. O conjunto também descreve a relação entre essas entidades que são em quais ruas os pontos estão e quais as linhas que param em cada ponto. O conjunto de dados pertence a uma cidade que identifica os pontos de ônibus em dois tipos, que são ponto simples e ponto tubo e existe uma regra de que apenas ônibus com cobradores podem passar por pontos do tipo simples. Um recorte do conjunto de dados pode ser visto na Figura 2.

Além de interagir com o conjunto de dados, o usuário deseja estendê-lo com um conjunto de dados semânticos que ele desenvolveu com dados novos ou atualizados (Figura 3) e, ao fazer a fusão dos seus dados com os dados originais, pode acabar inserindo dados que firam a consistência, acurácia ou concisão do conjunto de dados. Por exemplo, o usuário poderia adicionar no conjunto de dados linhas de ônibus sem cobradores que passam em linhas simples, o que fere a regra de que apenas ônibus com cobradores podem passar por pontos do tipo simples. O conjunto fundido hipotético pode ser visto na Figura 4.

Tendo em vista que a adição de novos dados pode ferir a consistência, precisão e concisão em um conjunto de dados, é fundamental verificar se a fusão dos novos dados com os dados originais não resultarão em um conjunto final com baixa qualidade e consequentemente inútil para a inferência de conhecimento. Logo, é importante que o usuário saiba antes de fundir os dados que a potencial fusão

## 1.2 Objetivo Geral

Criar um mecanismo de verificação da dimensões de qualidade de acurácia, consistência e concisão ao fundir conjuntos de dados ligados, buscando garantir a usabilidade dos dados fundidos para as entidades interessadas.

## 1.3 Objetivos específicos

- Fazer uma revisão bibliográfica das dimensões de qualidade a serem escolhidas
- Pesquisar por mecanismos de verificação de qualidade de dados baseado em ontologias
- Mapear os componentes do mecanismo de verificação para as dimensões de qualidade escolhidas
- Testar o mecanismo de verificação em um caso de uso

# 2 REVISÃO DE LITERATURA

## 2.1 Conceitos

### 2.1.1 Web Semântica

O formato que a Internet foi concebida inicialmente não permite que o computador entenda semanticamente os dados armazenados e apenas humanos conseguem interpretar e inferir conhecimento dos dados armazenados na Web. Por exemplo, um humano consegue interpretar o resultado de um arquivo HTML, pois sabe inferir significado do elementos textuais (parágrafos, cabeçalhos, títulos) e não-textuais (imagens, cores). No entanto, para um computador um arquivo HTML são apenas tags sem significado. Segundo Berners-Lee, Hendler e Lassila (2001), a Web apenas armazena dados que podem ser interpretados por humanos, não permitindo que computadores possam inferir conhecimento através deles. O conceito de Web Semântica surge para resolver esse problema, permitindo que homem e máquina cooperem para extrair significado deste amontoado de informações. A Web Semântica é a evolução da Web de Documentos, que apenas armazenava dados que não podiam ser interpretados pelo computador.

### 2.1.2 Ontologias

Para que as máquinas consigam entender os dados contidos na Web, devem existir estruturas de dados que possibilitem que essas informações sejam acessadas e deve existir uma lógica que defina como os componentes desta Web de dados interagem entre si. Na Web tradicional, cada sistema nomeia e interpreta os dados da sua própria maneira, fazendo que definições iguais sejam representadas de maneiras diferentes entre dois sistemas distintos. A Web Semântica, por outro lado, se baseia no conceito de universalidade, onde qualquer coisa pode se conectar com qualquer coisa (BERNERS-LEE; HENDLER; LASSILA, 2001). Dessa forma, dados de sistemas diferentes que representam o mesmo conceito podem ser unificados e descritos pelo mesmo conjunto de regras chamadas ontologias.

Ontologia é uma forma de generalizar conceitos do mundo real, definindo recursos e como estes recursos interagem entre si (GUARINO; OBERLE; STAAB, 2009). Essa generalização permite que entidades de diferentes contextos possam usar uma mesma linguagem para representar o mundo, buscando um acordo no lugar da separação de conceitos. Em uma ontologia são definidos os recursos, que descrevem as entidades do mundo e as relações, que explicam como os recursos se relacionam entre si. Ontologias são úteis em cenários onde um mesmo conceito é descrito de maneiras diferentes por duas entidades. Por exemplo: duas cidades desejam fundir seus dados de educação. Uma cidade tem uma coluna chamada “Escola“

e a outra tem uma coluna “Colégio“, onde ambas têm o mesmo significado. Uma ontologia unificaria esses dois conceitos através de um recurso “Entidade Educacional“ e antes de unir os conjuntos de dados seria informado ao sistema que as duas colunas representam o mesmo recurso “Entidade Educacional“.

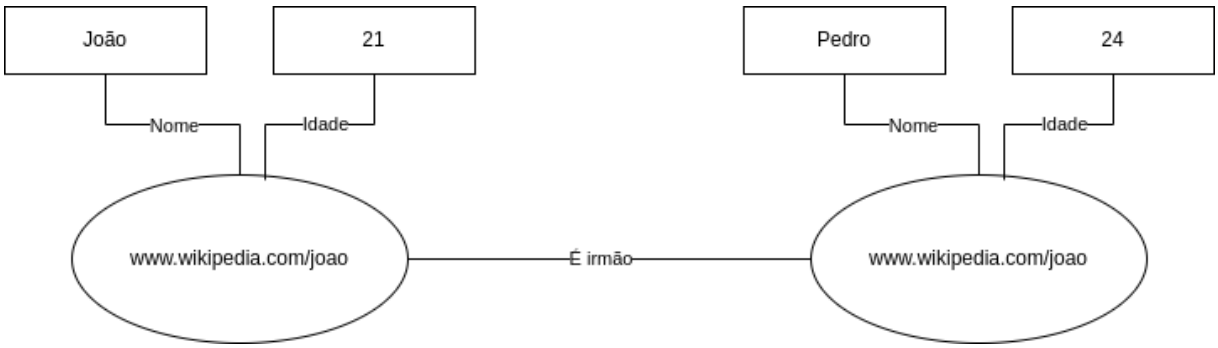
### 2.1.3 RDF e RDFS

O RDF (Resource Description Framework) é uma linguagem que permite que elementos do mundo sejam descritos e relacionados. O RDF é composto de representações dos 3 elementos da estrutura de dados semânticos: sujeito, predicado e objeto. O sujeito e o objeto representam dois recursos e o predicado indica qual a relação entre eles. Por exemplo, se João e Pedro são irmãos, um sujeito “Joã“ e um objeto “Pedr“ podem ser relacionados através do predicado “É irmão“, como exemplificado na Figura 5 . Um objeto também pode ser um valor literal. Por exemplo, se João tem 21 anos um sujeito “João“ pode ser ligado ao valor “21“ através do predicado “Idade“ (Figura 5).

Uma extensão da linguagem RDF que permite descrever os recursos com mais detalhes é o RDFS (Resource Description Framework Schema), que é uma extensão do RDF. O RDFS usa o conceito de classes que podem ser utilizadas para classificar os recursos e que estas classes tenham subclasses criando-se um conceito de hierarquia. Por exemplo, supondo que a classe “Cachorro“ é subclasse da classe “Animal“ e que o recurso “Toby“ é da classe cachorro, podemos inferir que Toby é um animal. Além disso é possível criar relações de generalização e especialização entre classes: supondo que a classe “Cachorro“ e a classe “Mamífero“ sejam ligadas pelo predicado “É um“, podemos inferir que Toby é um mamífero. Outro conceito do RDFS é o de domínio, podendo indicar quais classes fazem parte de uma relação. Por exemplo, é possível definir que o domínio da relação “É um“ é da classe “Animal“ e que o contradomínio e da classe “Categoria“.

**Figura 5 – Exemplo de grafos em RDF**

**(a)**




**Fonte: Autoria própria.**

### 2.1.4 OWL

A Ontology Web Language (OWL) é uma linguagem que é estendida do RDF e RDFS e permite que restrições sejam definidas nas triplas. Na OWL existem três entidades: instâncias, são a representação dos recursos, as classes, descrevem os grupos nos quais as instâncias pertencem, e propriedades que descrevem as ligações entre as instâncias, podendo elas ser entre duas instâncias ou entre uma instância e um objeto literal. Com a OWL é possível definir propriedades sobre os predicados como simetria, transitividade, inversão, assimetria, disjunção e cardinalidade. Exemplo: supondo que a relação “é colega“ seja transitiva, caso a instância “Pedro“ tenha a relação “é colega“ com “João“ e “João“ tenha a relação “é colega“ com “Maria“, podemos inferir que “Maria“ é colega de “Pedro“. Essas propriedade são úteis para inferir conhecimento e verificar a consistência dos conjuntos de dados (ZAVERI *et al.*, 2013). As triplas resultantes das linguagens descritas acima podem ser armazenadas em bancos de dados de triplas como o GraphDB, que é um banco de dados baseado em grafos, ou podem ser mapeadas em bancos de dados relacionais. A linguagem de consulta utilizada no contexto de Web Semântica é a SPARQL (SPARQL Protocol and Query Language), que está para o RDF assim como o SQL está para bancos de dados relacionais.

### 2.1.5 Qualidade de dados

O conceito de qualidade em um conjunto de dados varia de acordo com o contexto em que os dados estão sendo utilizados (WANG; STRONG, 1996). Dados que podem ser úteis para um determinado contexto podem não ter serventia para outro dependendo das características apresentadas por estes dados. Por exemplo, dados utilizados para a construção de uma aplicação voltada para a área da medicina devem ter características como consistência e acurácia, enquanto dados utilizados em uma aplicação de rede social devem ter alta interconexão e completude (ZAVERI *et al.*, 2013). Desta forma, é importante que o consumidor de determinado conjunto de dados tenha a capacidade de avaliar se este conjunto serve ou não para o contexto desejado por meio da escolha das características desejadas e da verificação das métricas das características escolhidas.

Dentro do conceito de qualidade de dados, essas características são chamadas de dimensões (ou critérios). Cada dimensão é medida por meio de uma métrica, que é um número gerado por uma função de pontuação que leva em conta indicadores específicos de cada métrica para fazer o cálculo (BIZER; CYGANIAK, 2009). Segundo Zaveri *et al.* (2016) essas dimensões podem estar enquadradas nos seguintes grupos:

- Acessibilidade: as dimensões desta categoria tratam das características de acesso e recuperação dos dados

- Dinamicidade: capturam o aspecto da atualidade e frequência de atualização dos dados do conjunto
- Contextualidade: são as dimensões que estão fortemente ligadas ao domínio dos dados, em contraste com a categoria “Intrínsecas“.
- Representatividade: aferem a qualidade da forma em que os dados estão representados. Por exemplo, se mede se todos as entidades do conjunto tem os utilizam os mesmo atributos.
- Intrínsecas: dimensões que independem do contexto do domínio dos dados. Visam dizer se o dado representa corretamente as informações do mundo real. Todas as dimensões deste trabalho, que são consistência, precisão e concisão, são desta categoria.

As dimensões também podem ser categorizadas como objetivas, que são aquelas que podem ser representadas por um valor concreto, como consistência; ou subjetivas, que dependem da percepção dos usuário em relação ao dado sendo analisado, como reputação (ZAVERI *et al.*, 2016). Para este trabalho, foram escolhidas as dimensões de qualidade de acurácia, concisão e consistência. Nos subcapítulos seguintes, uma descrição mais detalhada de cada uma é apresentada.

#### 2.1.5.1 Acurácia

A acurácia tem como objetivo indicar se os dados literais estão de acordo com os padrões definidos pela sua tipagem. Dados literais são representações em string de datas, inteiros e decimais, textos, tempo, etc que podem estar malformados. Os cinco tipos de dados mais comuns em dados ligados são *xsd:string*, *xsd:nonNegativeInteger*, *xsd:integer*, *xsd:dateTime* e *xsd:unsignedLong* (HOGAN *et al.*, 2010). Um exemplo de literal malformado é uma data sem caracteres separadores (como *1205/2000*) ou um inteiro que contém casas decimais. Além de estarem malformados, os dados podem estar fora do alcance do tipo, como uma data que contém um mês 13 (uma vez que o ano só tem doze meses) ou um literal para media a altura de um indivíduo com valor negativo. Nesses casos os dados estão bem formados (sintaxe) porém a informação é incorreta (semântica).

#### 2.1.5.2 Concisão

A dimensão de concisão pode ser dividida em dois tipos: de objetos e de atributos. A concisão de objetos, que também pode ser chamada de singularidade, se refere a quantidade de indivíduos repetidos em um determinado conjunto de dados. Por indivíduo repetido se entende duas ou mais representações de um mesmo objeto do domínio em um conjunto de dados

(ZAVERI *et al.*, 2013). Por exemplo, dois indivíduos de um conjunto de dados que representa a população do Brasil com o predicado *temCPF 03526411198* (que indica o número do Cadastro de Pessoas Físicas) seriam incoerentes com a realidade do domínio, uma vez que não podem haver duas pessoas com o mesmo número de CPF no Brasil. Assim, a concisão de objetos mede o número de indivíduos únicos em relação ao número total de indivíduos no conjunto de dados.

Já a concisão de atributos se refere ao número de predicados únicos de um determinado indivíduo, ou seja, verifica a existência de predicados redundantes nos indivíduos do conjunto de dados (MENDES; MÜHLEISEN; BIZER, 2012). Por exemplo, um indivíduo que representa um objeto tem dois predicados: *dataDeFabricacao* e *fabricadoEm*. Os predicados deste indivíduo estão sendo redundantes, uma vez que representam a mesma informação porem com nomes diferentes. O indivíduo estaria conciso se apenas um desses predicados estivesse ligado a ele. Dessa forma, a concisão de atributos mede o número de predicados únicos de um conjunto de dados em relação ao número total de predicados de uma determinada ontologia.

#### 2.1.5.3 Consistência

Por último, a dimensão de qualidade de consistência se refere ao quanto o conjunto de dados está livre de informações conflitantes. Pode se dizer que um conjunto de dados sem inconsistência é aquele que não contém contradições nos dados com respeito a determinada ontologia (ZAVERI *et al.*, 2013). Um exemplo de dados inconsistentes seria um time de futebol com apenas 5 jogadores (sendo que são necessários no mínimo 11 para uma partida iniciar) ou uma república federativa sem nenhuma federação (que viola o conceito de república federativa). Essa dimensão de qualidade se difere das outras duas pois é necessário um conhecimento de como as entidades do conjunto de dados se relacionam no mundo real por parte de quem está validando os dados, uma vez que a consistência é relativa a uma lógica específica para identificar contradições.

### 2.1.6 Fusão de dados

Fusão de dados é definida como o ato de juntar registros que representam o mesmo objeto do mundo real em uma representação única e consistente com a realidade (BLEIHOLDER; NAUMANN, 2009). A fusão de dados geralmente ocorre após o mapeamento de esquema (que busca integrar os esquemas das bases de dados) e da resolução de identidade, que tem como objetivo identificar os registros ou atributos que estão repetidos. Caso no processo de fusão sejam encontrados dois atributos iguais de esquemas diferentes, o sistema pode gerar um atributo único com os dois valores diferentes ou o sistema pode reduzir dois atributos de objeto em um, dependendo da estratégia. Por exemplo, suponto um conjunto com o predicado *dataDeFabricacao* e outro conjunto com o predicado *fabricadoEm*, que representam a mesma

informação, quando esses conjuntos forem fundidos esses dois atributos podem ser transformados em um só. A fusão de dados é importante pois, como dito em Halevy, Norvig e Pereira (2009), todo conjunto de dados deve crescer para agregar mais valor ao usuário final e por isso métodos corretos de como juntar dois ou mais modelos devem ser aplicados.

### 2.1.7 Gerenciamento de dados baseado em ontologias

Lenzerini (2011) formaliza o conceito de gerenciamento de dados baseado em ontologias (GDBO), onde a ontologia é uma descrição formal do domínio trabalhado e é considerada o ponto mais importante do sistema. O GDBO oferece aos consumidores de informação não apenas uma estrutura de dados que se encaixa com os dados nas fontes, mas uma descrição semântica dos conceitos do domínio, bem como as relações entre esses dados.

Em Console e Lenzerini (2014) é demostrado através de provas formais de lógica e ilustrações que é possível aferir as de dimensões de qualidade através da utilização de uma estrutura formal que representa um modelo conceitual do domínio estudado, que são as ontologias. Neste trabalho, apenas a dimensão de consistência foi tratada e nenhum exemplo prático de ontologia foi apresentado pelos autores.

### 2.1.8 SHACL

Com o aumento da utilização de dados em RDF, foi necessária a criação de uma linguagem padrão para a criação de restrições de qualidade e de mecanismos que pudessem interpretar essa linguagem e apontar as violações destas restrições. O SHACL (Shapes Constraint Language) surgiu com o objetivo de suprir essa demanda e se tornou uma recomendação do W3C em 2017. O SHACL funciona por meio da criação de *shapes*, que são grafos em RDF que definem as restrições para indivíduos específicos. Formalizando, um *shape* é uma tupla (s, t, d) definida por três componentes: o nome do *shape* s, que o identifica exclusivamente; os indivíduos a serem validados t, e o conjunto de restrições d.

**Figura 6 – Exemplo de *shape* do SHACL**

**(a)**

```
:PersonShape a sh:NodeShape ;
   sh:targetClass :Person ;
   sh:property [
       sh:message 'Nome inválido' ;
       sh:path :hasName ;
       sh:minCount 1 ;
       sh:maxCount 1 ;
       sh:datatype xsd:string ;
   ] ;
   sh:property [
       sh:message 'Idade inválida' ;
       sh:path :hasAge ;
       sh:minCount 1 ;
       sh:maxCount 1 ;
       sh:datatype xsd:integer ;
   ] ;
.
```

**Fonte: Autoria própria.**

O componente t define quais indivíduos serão validados pelo *shape*, que são definidos através do objeto do predicado *sh:targetClass*. Esse predicado define que os indivíduos que serão validados serão aqueles ligados pelo predicado *rdf:type* à classe definida pelo predicado *sha:targetClass*. Por exemplo, o *shape PersonShape* (componente s) da Figura 6 cria duas restrições para indivíduos da classe Person (componente t, através do predicado *sh:targetClass*): uma para o predicado *hasName* e outra para o predicado *hasAge*, validando se esses dois predicados apontam para objetos do tipo *string* e se cada um dele aparece exatamente uma vez para cada indivído da classe Person (componente d). Na Figura 6, o componente s foi marcado em amarelo, o componente t em vermelho e o componente d em azul.

# 3 TRABALHOS RELACIONADOS

Paulheim e Stuckenschmidt (2016) mostrou que é possível criar algoritmos de inferência que avaliam a consistência de um determinado conjunto de dados semânticos através do uso de aprendizado de máquina. Com esse método, foi atingido uma precisão de 95% nos resultados e 50x mais velocidade ao executar o algoritmos. Essa eficiência foi gerada pelo fato de que apenas algumas classes e propriedades definidas pela ontologia foram escolhidas para serem usadas nos algoritmos de aprendizado, uma vez que foi observado que um conjunto pequeno de classes e propriedades são responsáveis por 99% das inconsistências no domínio estudado. Porém o autor deixa em aberto como as ontologias que serão utilizadas pelos algoritmos devem ser criadas. Neste trabalho, o algoritmo de inferência utilizado foi implementado através da biblioteca *pyshacl* da linguagem *Python* e foi utilizado para realizar as validações com os shapes em SHACL.

Kovalenko *et al.* (2015) criou uma ontologia para permitir que dados fossem trocados entre máquinas em um contexto de PSE (Projetos de Sistema de Engenharia), com o objetivo de homogeneizar os dados produzidos pelas ferramentas e máquinas do domínio, realizar análise e fazer verificações de consistência nos dados coletados. A ontologia foi criada com base em modelos de dados pré-existentes e com a ajuda de especialistas na área e foi implementada utilizando a linguagem OWL. Os resultados obtidos foram satisfatórios, uma vez que a ontologia conseguiu capturar todas as regras semânticas no domínio, que posteriormente foram usadas para fazer a checagem de consistência. Neste trabalho, uma lógica parecida é utilizada, onde foram estudados os axiomas e relacionamentos entre as classes para gerar as restrições de qualidade.

Spahiu, Maurino e Palmonari (2018) propôs uma metodologia para melhorar a qualidade dos dados por meio de restrições SHACL geradas a partir dos análises feitas pelo ABSTAT, que é uma ferramenta de análise semântica que ajuda os consumidores de dados a entender melhor os dados, extraindo padrões de ontologia orientados por dados e estatísticas. Por meio do ABSTAT é possível inferir conhecimentos como "90% dos dados do conjunto de dados são ligados pelo predicado X" ou "50% dos objetos do predicado X são do tipo inteiro" e com esses conhecimentos obter um maior entendimento do conjunto de dados estudado e posteriormente transformá-los em *shapes* do SHACL.

Pandit, O'Sullivan e Lewis (2018) formalizou a criação de *shapes* em SHACL por meio da utilização de Padrões de Design do Ontologias (PDO), que são axiomas que capturam apenas os conceitos e relacionamentos necessários para definir determinado domínio. Axiomas em uma ontologia definem restrições sobre conceitos e relacionamentos que devem ser satisfeitos pelas instâncias que usam a ontologia. Por exemplo, em um ontologia do domínio de mobilidade urbana, um exemplo de axioma seria que todo ônibus deve passar em pelo menos um ponto. Essa abordagem incentiva a reutilização de PDOs além da fase de modelagem de dados. Este trabalho propõe que é possível traduzir os axiomas gerados pela PDO em *shapes* do SHACL,

sendo necessária apenas a criação de um mapeamento entre as restrições e os componentes do SHACL, que foi criada neste trabalho

Rabbani, Lissandrini e Hose (2022) buscou como os *shapes* estão sendo gerados em SHACL e como eles estão sendo utilizados. Primeiramente, foi realizada uma pesquisa na comunidade, tanto acadêmica quanto empresarial, para analisar as necessidades e comportamentos dos usuários ao gerar *shapes*. Em seguida, foram realizadas pesquisas acerca de ferramentas que permitem a criação de *shapes* de forma semi-automatizada. Essas pesquisas foram cruzadas com os resultados da primeira etapa. Por último, foram analisadas como as abordagens de extração automática e semi-automática de *shapes* existentes funcionam na prática em conjuntos reais de dados ligados. Os resultados mostraram que métodos automáticos de geração de *shapes* apenas são aplicáveis em pequenos conjuntos de dados. Os resultados também apontaram que os *shapes* gerados por essas ferramentas revelou que nenhuma das abordagens extrai todas as restrições necessárias. Por exemplo, não foi encontrada nenhuma restrição para objetos predicados não literais (por exemplo, para indicar que objetos para “hasBus” devem ser do tipo “Bus”). Além disso, alguns *shapes* foram extraídas devido à presença de alguns indivíduos com predicados errados, como uma cidade confundida com um membro de uma banda musical.

# 4 METODOLOGIA

Na primeira etapa foi buscado material bibliográfico que é usado como base para escolher quais dimensões de qualidade seriam tratadas no decorrer do trabalho, assim como referências para criação da base conceitual utilizada na pesquisa.

Com base nos trabalhos encontrados na primeira etapa, o passo seguinte foi escolher as dimensões que seriam tratadas no trabalho. A partir da análise das características de cada dimensão de qualidade foi decidido que este trabalho tratará apenas de dimensões objetivas, pois podem ser aferidas por meio de regras bem claras e não dependem do contexto do conjunto de dados ou da opinião do usuário.

De início foram escolhidas as dimensões de completude, consistência, precisão, concisão e interconexão. Porém, ao estudar as referências sobre GDBO, que foi a terceira etapa do trabalho, notou-se que as dimensões de completude e interconexão não fariam sentido dentro do contexto de aferição de qualidade por meio de ontologias, por isso ambas não serão tratadas no trabalho. Dessa forma as dimensões escolhidas foram acurácia, consistência e concisão.

A próxima fase do desenvolvimento do projeto foi a busca por ferramentas ou técnicas que permitissem com que um conjunto de dados ligados fosse validado por meio de ontologias e por meio da leitura da literatura relacionada foram encontradas diversas técnicas e ferramentas que permitiam a validação do conjunto. A que mais chamou atenção foi o SHACL, pois permitia criar validações diretamente no nível dos dados da ontologia através da verificação de relacionamentos e padrões dos dados.

Uma opção semelhante com o SHACL foi encontrada na literatura, chamada ShEx. Porém, esta linguagem não oferece alguns recursos importantes que seriam necessários para a implementação da proposta deste trabalho, como herança de classes (por exemplo, supondo que existe uma classe base Ponto de Acesso e suas derivadas Ponto e Terminal, as restrições da classe base não seriam herdadas pelas classes derivadas), comparação de valores entre predicados do mesmo objeto (por exemplo, verificar se o predicado *dataInicio* é menor que o predicado *dataFim*) e também pela falta de suporte para utilização de consultas SPARQL nas restrições. Além do mais, o SHACL é recomentado pelo World Wide Web Consortium (W3C) desde 2017, em contraste com o ShEx que não tem essa recomendação.

Uma vez definida qual a ferramenta a ser utilizada para fazer a validação dos dados, foram lidos artigos e documentações que permitiram que o conhecimento do funcionamento da ferramenta fosse aprofundado. Os materiais que mais serviram de auxílio foi a documentação oficial do SHACL [1] e o artigo A Review of SHACL: From Data Validation to Schema Reasoning for RDF Graphs (PARETI; KONSTANTINIDIS, 2021). Para testar na prática o funcionamento da linguagem, foi criado um conjunto de dados e restrições fictícias com base na ontologia escolhida, do domínio de conhecimento de mobilidade urbana e educação, no software Protegé com a adição do *plugin* SHACL4P, que habilita a utilização do SHACL dentro do Protegé. Esse teste

[1] https://www.w3.org/TR/shacl/

prático foi muito importante para entender características mais técnicas da linguagem, como sintaxe e o comportamento de algumas funcionalidades que não estavam bem documentadas nos materiais escritos.

Com um maior conhecimento da ferramenta, foi possível fazer o mapeamento entre os componentes do SHACL e as dimensões de qualidade escolhidas para o trabalho. Nesta fase, buscou-se entender quais componentes e funcionalidades do SHACL suportariam a criação das restrições que validariam se os dados cumprem as dimensões de consistência, acurácia e concisão. Para isso, foi identificado como cada dimensão de qualidade se comportava e como esse comportamento poderia ser traduzido para o SHACL através da utlização de seus componentes.

Uma vez feito o mapeamento dos componentes e as dimensões de qualidade, foram criadas as restrições que garantirão o cumprimento das dimensões de qualidade da ontologia, primeiramente em linguagem natural, através da análise das classes e do comportamento do domínio.

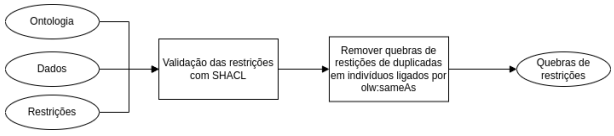
Com o mapeamento dos componentes e as dimensões de qualidade em linguagem natural feitos, o próximo passo foi criar os *shapes* em SHACL, que seria traduzir as restrições em linguagem natural para SHACL por meio do mapeamento de componentes. Além disso, foi criado também a query em SPARQL utilizada para a validação da dimensão de concisão de objetos, uma vez que o SHACL não suporta essa dimensão nativamente.

O próximo passo foi a criação de um *script* em Python que permite a ignorar quebras de restrição de concisão de objetos por indivíduos que estão ligados pelo predicado *owl:sameAs*. Foi utilizada a versão 3.6 da linguagem e além das funcionalidades básica da linguagem foram adicionadas as bibliotecas *pyshacl*, *rdflib*, *xml* e *json*. O fluxo deste *script* pode ser observado na Figura 7.

Por fim, foram realizados testes em conjuntos de dados fictícios baseados na ontologia escolhida, simulando a fusão de um conjunto de dados sem erros com um conjunto que adiciona dados que quebra das restrições escolhidas.

**Figura 7 – Fluxo de validação em SHACL em Python**

**(a)**




**Fonte: Autoria própria.**

# 5 DESENVOLVIMENTO

## 5.1 Mapeamento entre dimensões de qualidade e componentes do SHACL

**Tabela 1 – Mapeamento das dimensões de qualidade e dos componentes em SHACL**

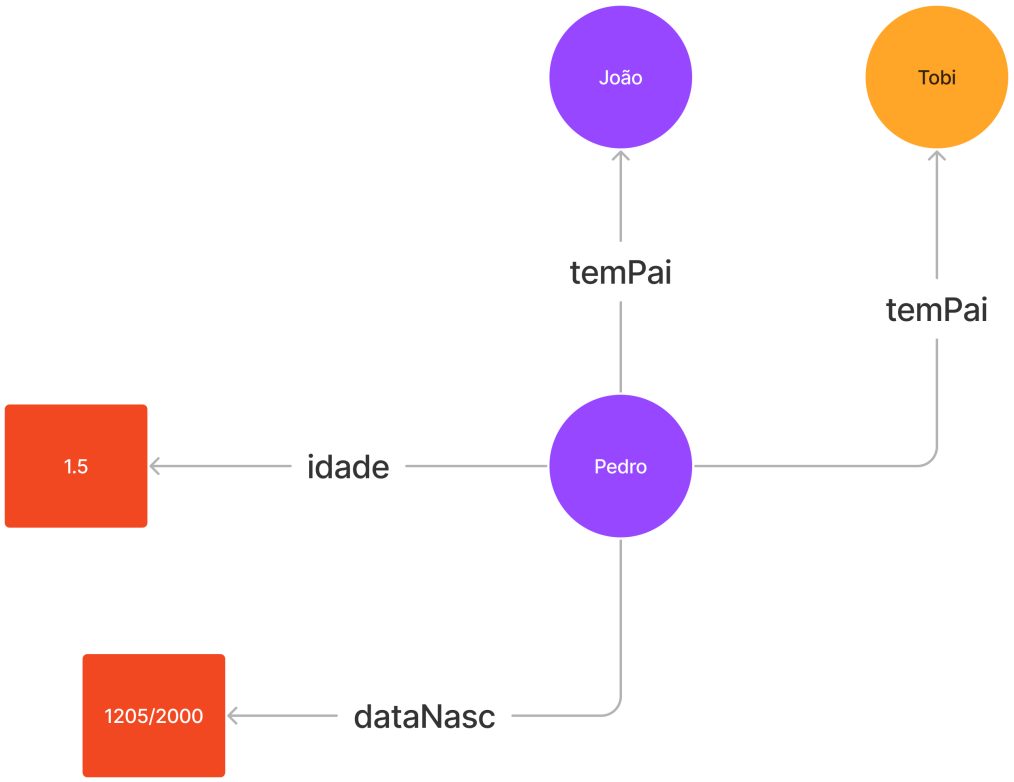
| Dimensão | Componentes | Justificativa |
|---|---|---|
| Acurácia | sh:property<br>sh:path<br>sh:dataType<br>sh:maxInclusive<br>sh:minInclusive<br>sh:pattern<br>sh:length | Componentes de acesso a predicados e de validação de literais |
| Consistência | sh:property<br>sh:path<br>sh:class<br>sh:minCount<br>sh:maxCount | Componentes de acesso a predicados e validação de relacionamento entre classes |
| Concisão | sh:closed<br>sh:ignoredProperties<br>sh:property | Componentes validação de existencia de predicados |

**Fonte: Autoria própria (2022).**

Nesta etapa, foram criados os mapeamentos entre os componentes do SHACL e as dimensões de qualidade escolhidas. Os componentes do SHACL são funcionalidades fornecidas pela linguagem, que juntas formam os *shapes* de validação. Como proposto em Pandit, O'Sullivan e Lewis (2018), o mapeamento entre os componentes da linguagem e as dimensões de qualidade deve ser feito para que, posteriormente, as restrições de qualidade criadas para a ontologia sejam implementadas em SHACL. O mapeamento, uma vez feito, pode ser reutilizado em diversos domínios do conhecimento, uma vez que as dimensões de qualidades podem ser aferidas em qualquer ontologia.

Para cada dimensão de qualidade, buscou-se entender qual o seu comportamento (como a dimensão é definida) e posteriormente investigar quais funcionalidades do SHACL que suportavam a verificação desta definição no conjunto de dados. A definição das dimensões foi feita com a ajuda de definições formais encontradas na literatura e complementado com observações do autor deste trabalho. O mapeamento final pode ser visto na Tabela 1. Para um melhor entendimento sobre o mapeamento das dimensões nos elementos SHACL, as dimensões serão explicadas utilizando como base o conjunto de dados fictício da Figura 8.

**Figura 8 – Exemplo de conjunto de dados**
**(a)**




**Fonte: Autoria própria.**

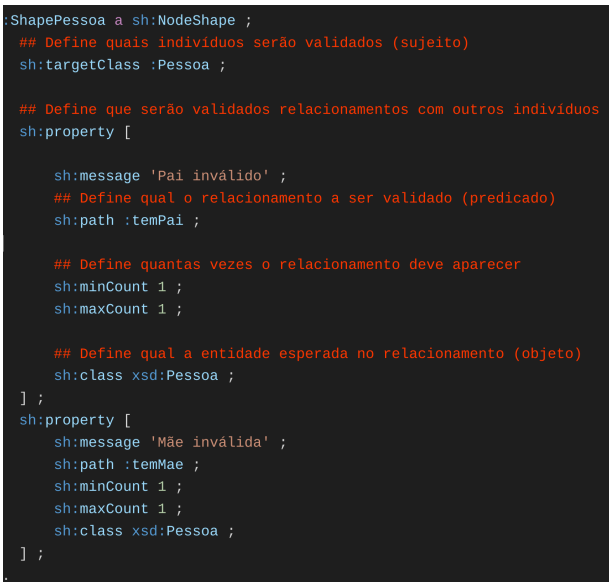
### 5.1.1 Consistência

Um conjunto de dados consistente é aquele que não contém contradições nos dados com respeito a determinada ontologia (ZAVERI *et al.*, 2013). Contradições, no contexto de dados ligados, são formadas quando um indivíduo está ligado a outros de forma errada, seja por meio de uma relação que não deveria existir, por meio de uma relação que aparece um número errado de vezes ou por meio de uma relação que não existe. Por exemplo, a Figura 8 representa um conjunto de dados onde as bolas roxas são indivíduos da classe *Pessoa* e as Amarelas da classe *Cachorro*. As inconsistências neste caso são nítidas, partindo do pressuposto de que uma pessoa só pode nascer tendo um pai e uma mãe: o indivíduo *Pedro* tem dois pais (cardinalidade errada), sendo que um deles é um cachorro (relacionamento errado) e não possui uma mãe (relacionamento inexistente).

Logo, a dimensão de consistência pode ser aferida através da observação de como o indivíduo analisado se relaciona com outros indivíduos. No SHACL, esse relacionamento entre

indivíduos pode ser validado através do componente *sh:property*, que verifica a existência de predicados entre o sujeito sendo validado e determinado objeto, do componente *sh:class*, que verifica se o objeto pertence a determinada classe e do componente *sh:path*, que define qual o predicado que deve ligar os dois indivíduos. Além disso, podem ser utilizados componentes de cardinalidade, como o *sh:minCount* e o *sh:maxCount*, que indicam quantas vezes esses relacionamentos devem aparecer.

**Figura 9 – Exemplo de shape do SHACL**
**(a)**



```
:ShapePessoa a sh:NodeShape ;
  ## Define quais indivíduos serão validados (sujeito)
  sh:targetClass :Pessoa ;

  ## Define que serão validados relacionamentos com outros indivíduos
  sh:property [

      sh:message 'Pai inválido' ;
      ## Define qual o relacionamento a ser validado (predicado)
      sh:path :temPai ;

      ## Define quantas vezes o relacionamento deve aparecer
      sh:minCount 1 ;
      sh:maxCount 1 ;

      ## Define qual a entidade esperada no relacionamento (objeto)
      sh:class xsd:Pessoa ;
  ] ;
  sh:property [
      sh:message 'Mãe inválida' ;
      sh:path :temMae ;
      sh:minCount 1 ;
      sh:maxCount 1 ;
      sh:class xsd:Pessoa ;
  ] ;
```

**Fonte: Autoria própria.**

Por exemplo, na Figura 9 é possível observar um shape que valida se os indivíduos da classe *Pessoa* (através do componente *sh:targetClass*) estão ligadas à um outro indivíduo de classe *Pessoa* (através dos componentes *sh:property* e *sh:class*) por meio do predicado *temPai* (através do componente *sh:path*), no mínimo uma vez e no máximo uma vez (através dos componentes *sh:minCount* e *sh:maxCount*).

### 5.1.2 Acurácia

Já a dimensão de acurácia pode ser descrita como a validação com o objetivo de verificar se os dados literais de determinado indivíduo estão dentro do padrão definido pelo tipo do dado e pelo domínio sendo validado. Novamente utilizando o exemplo da Figura 8, a falta de acurácia pode ser vista no predicado *idade*, onde o objeto é um decimal (sendo que deveria ser um inteiro) e no predicado *dataNasc*, onde o formato da data está inválido (deveria ser dd/mm/aaaa). Logo a dimensão de acurácia também trata dos relacionamentos de um indivíduo, porém agora em vez de serem relacionamentos com outros indivíduos (como na consistência) o relacionamento e com dados literais. Por isso a maneira como essas validações são feitas no

SHACL é parecida com a de consistência, uma vez que o componente *sh:property* também é utilizado. Porém, como a acurácia é validada nos dados literais de um indivíduo, o componente *sh:dataType* é utilizado no lugar do componente *sh:class*. O componente *sh:dataType* valida qual o tipo literal do objeto que está ligado ao indivíduo sendo analisado (string, número, data). A validação do formato dos dados pode ser feito no SHACL através dos componentes chamados de *Value Range* como *sh:maxInclusive* e *sh:minInclusive* ou os *String Patter*, como *sh:pattern* e *sh:length*.

**Figura 10 – Exemplo de shape do SHACL**

**(a)**

```
:ShapePessoaLiteral a sh:NodeShape ;
  sh:targetClass :Pessoa ;
  sh:property [
      sh:message 'Idade inválida' ;
      sh:path :idade ;
      sh:minCount 1 ;
      sh:maxCount 1 ;
      ## Define qual o tipo do literal
      sh:dataType xsd:integer ;
  ] ;
  sh:property [
      sh:message 'Data de nascimento inválida' ;
      sh:path :dataNasc ;
      sh:minCount 1 ;
      sh:maxCount 1 ;
      sh:dataType xsd:date ;
      ## Define qual o padrão do literal
      sh:pattern "^[0-9]{2}/[0-9]{2}/[0-9]{4}$" ;
  ] ;
  ## Define quais predicados são aceitos no indivíduo
  sh:closed true ;
  sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

Um exemplo de *shape* que valida esses o exemplo acima é o da Figura 10. Note que os mesmos componentes da Figura 9 foram utilizados, com a adição dos componentes de dados literais *sh:dataType* e *sh:pattern*, que validam o tipo do dados literal e o seu padrão, respectivamente.

### 5.1.3 Concisão

A dimensão de concisão de atributos se refere ao número de predicados únicos de determinado indivíduo. Logo, buscou-se no SHACL um componente que permitisse verificar quais os predicados ligados à determinado indivíduo e foi encontrado o componente *sh:closed*. Este componente indica que apenas os predicados que aparecem definidos no componente *sh:property* podem estar ligados ao indivíduo sendo validado. O componente *sh:ignoredProperties* serve

para declarar as exceções desta regra, permitindo com que alguns predicados não entrem na validação do *sh:closed*, geralmente utilizado para ignorar predicados do RDF e OWL.

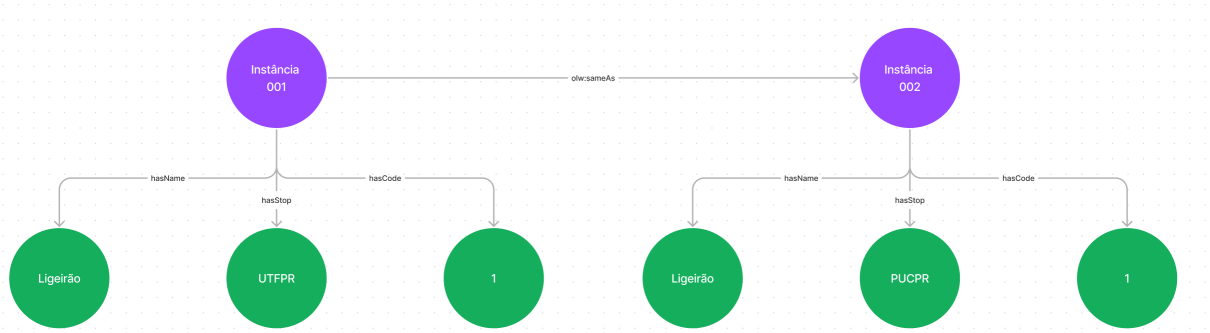
No caso do *shape* exemplo da Figura 10 o componente *sh:closed* está configurado com o valor *true*. Neste caso, o SHACL validará se os indivíduos sendo validados por esse *shape* estão ligados apenas aos predicados *dataNasc* e *idade*, que estão definidos no componente *sh:property*. Além disso, o componente *sh:ignoredProperties* tem como valor o predicado *rdf:type*, que indica que este predicado é uma exceção a regra do *sh:closed*, podendo estar ligado ao individuo.

Já a dimensão de concisão de objetos se refere à não duplicidade de indivíduos em um conjunto de dados e por isso é necessária a verificação de indivíduos repetidos dentro de um conjunto de dados por meio de uma iteração por todos os indivíduos. Após busca da documentação do SHACL, foi constatado que esta funcionalidade não é suportada nativamente pela linguagem, uma vez que não é possível comparar predicados de dois indivíduos diferentes, apenas os predicados de um mesmo indivíduo. Por exemplo, não é possível comparar, em SHACL, o valor do predicado *hasCPF* do indivíduo *Pedro* com o valor do mesmo predicado do indivíduo *Joao*. Logo, não haveria como conferir se dois predicados têm o mesmo valor. Por meio da leitura da documentação de recursos avançados do SHACL, foi verificado que a linguagem suportava a adição de consultas personalizadas por meio da linguagem SPARQL. Dessa forma, seria possível criar uma consulta que comparasse se dois ou mais indivíduos diferentes têm um predicado com o mesmo valor.

Porém ao identificar indivíduos que têm o mesmo valor para um predicado que deveria ser único, devemos verificar se os indivíduos estão ligados pelo predicado *old:sameAs*, que indica que dois indivíduos com a URI diferente na verdade se tratam de um mesmo indivíduo na vida real. Caso estejam ligados por esse predicado, não podem ser considerados como duplicados. Por exemplo, na Figura 11 é possível observar que a linha de ônibus Ligeirão é representada por dois indivíduos diferentes dentro do conjunto de dados, onde em ambos o predicado *hasCode* têm o valor 1. Por definição o predicado *hasCode* deveria ser único, porém neste caso ele está repetido, ferindo a dimensão de qualidade de concisão de objetos. No entanto, esses dois indivíduos estão ligados pelo predicado *olw:sameAs*, fazendo com que a quebra da restrição de concisão de objetos deva ser ignorada, uma vez que ambos invíduos representam a mesma coisa no mundo real. Já no caso da Figura 12, a quebra da restrição de concisão de objetos deve permanecer, uma vez que indivíduos que têm o mesmo valor para o predicado *hasCode* representam duas coisas diferentes no mundo real.

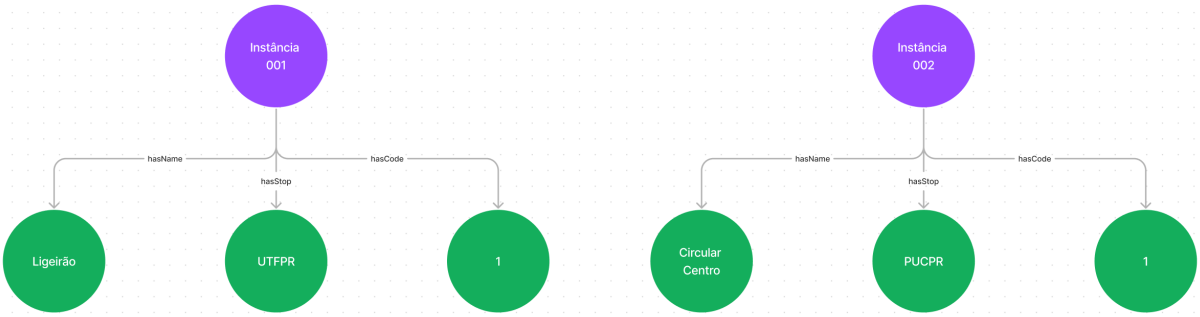
Porém, novamente, o SHACL não tem suporte para esse tipo de verificação, nem mesmo utilizando consultas com SPARQL. Por isso, foi necessária a criação de um script em Python para ignorar o aviso de duplicidade quando verificada a existência do predicado *sameAs* entre os indivíduos com valores únicos repetidos, que será detalhado com mais profundidade na sessão 5.4.

**Figura 11 – Exemplo concisão**
**(a)**




**Fonte: Autoria própria.**

**Figura 12 – Exemplo concisão**
**(a)**




**Fonte: Autoria própria.**

## 5.2 Caso de Uso - Dados de mobilidade e educação

Para a criação das restrições de qualidade, foi utilizado o método proposto em Pandit, O'Sullivan e Lewis (2018), que consiste em utilizar os Padrões de Design de Ontologia (PDO). Os PDOs são os conceitos e relacionamentos necessários para gerar a ontologia. Por exemplo, para criar uma ontologia que descreve o domínio do ser humano, podemos definir um axioma de que todo ser humano deve ser gerado por outros dois seres humanos, um pai e uma mãe. A partir deste axioma, podemos inferir que a ontologia deve ter uma classe *Pessoa* e os predicados *temPai* e *temMae*, que ligam indivíduos da classe *Pessoa*. Agora, podemos utilizar o mesmo axioma para criar restrições de qualidade para essa ontologia, como:

- Todo ser humano deve ter exatamente um pai, que também deve ser um ser humano
- Todo ser humano deve ter exatamente uma mãe, que também deve ser um ser humano

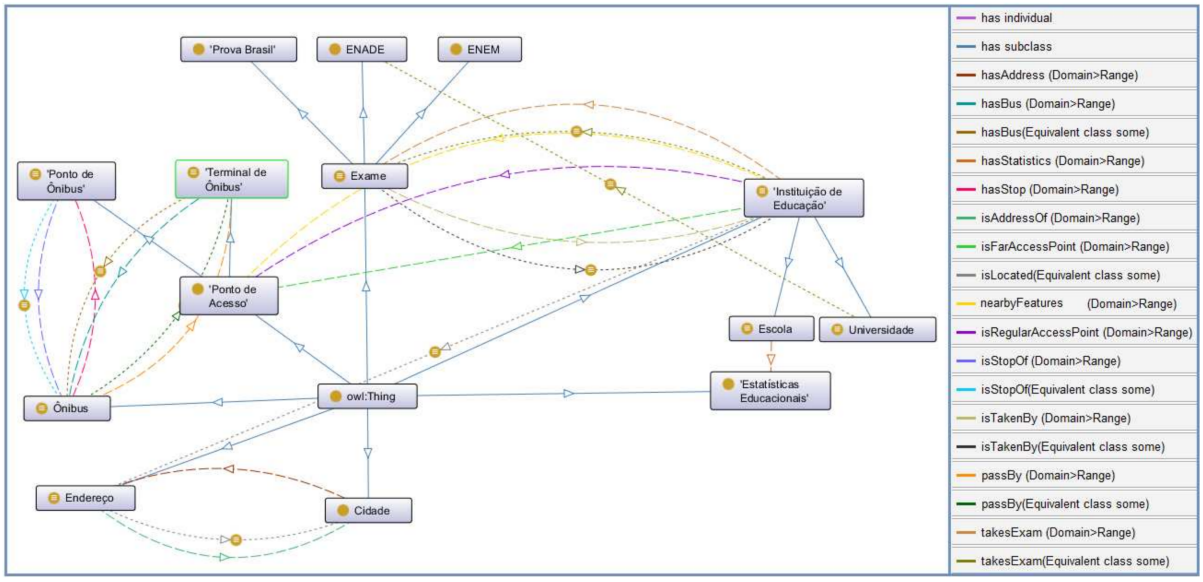
Uma lógica semelhante foi aplicada para a o domínio escolhido para este trabalho, que foi de mobilidade e educação, pois os axiomas que geraram a ontologia não estavam dispo-

níveis. No lugar, a partir do relacionamento entre as classes da ontologia e da observação do comportamento do domínio, foram geradas as restrições de qualidade em linguagem natural.

### 5.2.1 Descrição da Ontologia de Mobilidade e Educação

A ontologia utilizada para a criação das restrições tem como objetivo representar as principais entidades relacionadas ao transporte público, como pontos de ônibus, linhas de ônibus e terminais de ônibus. Além disso, também incluí entidades de educação, como ensino básico, fundamental e superior e por fim as estatísticas de avaliação de cada instituição de ensino, como ENEM, ENADE e Prova Brasil. A Figura 13 é uma representação gráfica de todas as classes da ontologia, descrevendo as relação entre elas através das setas coloridas.

**Figura 13 – Ontologia de Mobilidade Urbana e Educação**
**(a)**




**Fonte: (BELIZARIO, 2020).**

### 5.2.2 Definição das restrições das dimensões de qualidade

Para a dimensão de consistência, foi feita uma análise empírica de como as classes se relacionam entre si no domínio da ontologia. Por exemplo, para a criação das restrições da classe Bus, foi identificado que uma linha de ônibus só faz sentido se parar em pelo menos dois pontos de acesso (ponto de ônibus ou terminal). Um ônibus que para em apenas um lugar não faz sentido, pois os passageiros irão desembarcar no mesmo lugar que embarcaram, assim como um ônibus sem paradas não teria como embarcar passageiros. Ou seja, a partir da análise dos axiomas das classes, foi possível determinar como elas se comportam no domínio e consequentemente quais incoerências feririam a essência deste objeto.

Para a dimensão de acurácia, foram observados quais os valores aceitáveis para determinada variável no domínio. Por exemplo, no caso do ano de realização de um exame educacional não faz sentido esse valor ser menor que 1980 (uma vez que dados muito antigos não devem ser considerados) ou esse valor ser maior que o ano atual (uma vez que é impossível que um exame tenha sido realizado no futuro). Já para o valor do INSE, foi feita uma pesquisa de como esse dado é calculado e quais os possíveis valores que ele pode receber. Por se tratar de um dado do governo, essa informações foram encontradas no site do MEC.

Para a dimensão de concisão de objetos, foram analisadas quais características dos dados seriam únicas à cada indivíduo, ou seja, que não poderiam se repetir no domínio. No caso da ontologia utilizada no trabalho, foi utilizado um código de identificação para pontos de acesso, linhas de ônibus e instituições educacionais. Logo, foi escolhido o predicado *hasCode* para a verificação de duplicatas nos dados.

Já para a concisão de atributos, foram considerados os predicados que já estavam definidos na ontologia. Ou seja, se um predicado que não está na ontologia for utilizado, haverá a quebra da restrição.

### 5.2.3 Dimensões de qualidade para Mobilidade e Educação

**Ônibus (classe Bus)**

- Deve ter um nome que seja uma string de letras (acurácia)
- Deve ter um código que deve ser único (concisão)
- Deve parar em pelo menos dois pontos OU deve passar em pelo menos dois terminais OU deve passar em pelo menos um ponto e um terminal (consistência)

**Ponto de Acesso (classe Access_Point)**

- Deve ter um código que deve ser único (concisão)
- Deve estar localizada em algum endereço (consistência)

**Parada de Ônibus (classe *Bus_Stop*)**

Herdas as restrições da classe *Access_Point*, que é a classe pai de *Bus_Stop* e contém mais a seguinte restrição:

- Deve ser parada de pelo menos um ônibus (consistência)

**Terminal (classe *Bus_Station*)**

Herdas as restrições da classe *Access_Point*, que é a classe pai de *Bus_Station* e contém mais a seguinte restrição:

- Deve ter pelo menos um ônibus passando (consistência)

**Instituição de Ensino (classe Educational_Institution)**

- Deve ter um nome que seja uma string de letras (acurácia)
- Deve ter um tipo, que é uma string que pode ter os valores Particular, Público Municipal, Público Estadual, Público Federal (acurácia)
- Deve ter código INEP, que deve ter 8 números (acurácia)
- Pode conter pontos de acessos (consistência)
- Deve fazer pelo menos um exame (consistência)
- Deve estar localizada em algum endereço (consistência)

**Universidade (classe *University*)**

Herdas as restrições da classe *Educational_Institution*, que é a classe pai de *University* e contém mais as seguintes restrições:

- Herda as constraints de Instituição de Ensino
- Deve ter código IES, que dever ter de 3 a 5 dígitos (hasCodeIES)
- Deve ter iniciais que é uma string de letras e símbolos (hasInitials)

**Escola (classe *School*)**

Herdas as restrições da classe *Educational_Institution*, que é a classe pai de *School* e contém mais as seguintes restrições:

- Deve ter pelo menos uma estatística vinculada (consistência)

**Estatísticas Educacionais (classe Educational_Statistics)**

- Pode conter taxa de abandono, que deve ser um número float (acurácia)
- Pode conter taxa de aprovação, que deve ser um número float (acurácia)
- Pode conter taxa de reprovação, que deve ser um número float (acurácia)
- Pode conter taxa de permanência, que deve ser um número float (acurácia)
- Deve ter um ano, que é um inteiro entre 1980 e 2022 (acurácia)
- Deve ter um INSE, que dever ser um inteiro entre 1 e 9 (acurácia)

**Exame (classe Exam)**

- Deve ter um INSE, que dever ser um inteiro entre 1 e 9 (acurácia)

**Endereço (classe Address)**

- Deve ter um nome que seja uma string de letras (acurácia)
- Deve ter um bairro que seja uma string de letras (acurácia)

## 5.3 Ontologia em SHACL

Uma vez feito o mapeamento dos componentes do SHACL e com a criação das restrições para cada classe da ontologia, foi possível fazer a tradução das restrições de linguagem natural para SHACL. No total, foram criados dez shapes do SHACL, sendo nove específicos para cada classe da ontologia e mais um que é reutilizado nas classes *Educational_Institution*, *Bus* e *Access_Point*.

**Figura 14 – *Shape* da classe *Access_Point***
**(a)**

```
:AccessPointShape a sh:NodeShape ;
    sh:targetClass :Access_Point ;
    sh:property [
        sh:message '''{"message": "Access Point has invalid code.", "type": "property"}''' ;
        sh:path :hasCode ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:minInclusive 1 ;
        sh:maxInclusive 9999 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Access Point has invalid address.", "type": "property"}''' ;
        sh:path :isLocated ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class :Address ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

Por exemplo, na Figura 14 é possível observar um exemplo de um *shape* criado para as restrições descritas na sessão 5.2.3. A partir de qual dimensão da qualidade cada restrição resolvia, o mapeamento de componentes e dimensões foi consultado para saber qual o componente do SHACL deveria ser utilizado na restrição. Neste exemplo, uma restrição era de consistência, logo seriam utilizados os componentes *sh:property* e *sh:datatype* juntamente com os componentes de cardinalidade (*sh:minCount* e *sh:maxCount*) e de faixa de valor (*sh:minInclusive* e *sh:maxInclusive*). A outra restrição se refere à qualidade de consistência, portanto os componentes *sh:property* e *sh:class* foram utilizados, juntamente com os componentes

de cardinalidade *sh:minCount* e *sh:maxCount*. Essa mesma lógica foi aplicada para as outras restrições, gerando por fim os *shapes* em SHACL de toda a ontologia, que serão mostrados no Anexo A.

## 5.4 Validação em Python

**Algoritmo 1 – Algoritmo de validação de restrições**

```
1: VALIDOS ← ∅
2: DADOS, SHACL, ONTOLOGIA
3: QUEBRAS ← VALIDAR(DADOS, ONTOLOGIA, SHACL)
4: para todos QUEBRA ∈ QUEBRAS faça
5:     se EH_QUEBRA_CONCISAO(QUEBRA) então
6:         VALOR_OBJETO ← QUEBRA.VALOR_PREDICADO
7:         retorna individuos com mesmo valor que valor_objeto R
8:         para todos I ∈ R faça
9:             para todos J ∈ R faça
10:                se J ≠ I então
11:                    verifica se I e J estão ligados pelo predicado owl:sameas (ESTA_LIGADO)
12:                    se ESTA_LIGADO então
13:                        VALIDOS ← VALIDOS ∪ i, j
14:                    finaliza se
15:                finaliza se
16:            finaliza para
17:        finaliza para
18:    finaliza se
19: finaliza para
20: para todos QUEBRA ∈ QUEBRAS faça
21:    se QUEBRA é do tipo concisão então
22:        INDIVIDUO ← QUEBRA.INDIVIDUO
23:        se INDIVIDUO ∈ VALIDOS então
24:            continue para próxima iteração
25:        finaliza se
26:    finaliza se
27:    IMPRIMA(QUEBRA.MENSAGEM)
28:    IMPRIMA(QUEBRA.INDIVIDUO)
29: finaliza para
```

**Fonte: Autoria própria (2022).**

Além dos *shapes* em SHACL, foi necessária a criação de um *script* em Python [1] para permitir a execução do SHACL em cima de uma ontologia e de um conjunto de dados. Além disso, o *script* foi estendido para permitir a validação da dimensão de qualidade de concisão, pelos motivos descritos na sessão 5.1.3. O pseudocódigo do algoritmo está apresentado no Algoritmo 1.

Neste *script* são utilizadas as bibliotecas *pyshacl* (para a validação das restrições), *rdflib* (para a manipulação de dados em RDF), *xml* (para a manipulação de dados em XML) e *json*

[1] https://github.com/otaviobertucini/SHACL-validator/tree/master

(para a manipulação de dados em JSON). Os primeiros passos são carregar todos os dados dentro do *script*, que nesse caso são a ontologia, os *shapes* do SHACL e os dados em si. Esse carregamento é feito através da biblioteca *rdflib* e do método *parse*, que transforma um arquivo *.ttl* em uma instância da classe *Graph*, que permite que os dados em RDF sejam manipulados (linha 2 do Algoritimo 1).

Feito isso, essas intâncias são passadas para o método *validate* do *pyshacl*. Este método deve receber as três informações necessárias para a validação, que é a ontologia, os *shapes* do SHACL e os dados, além de qual o método de inferência utilizado nos dados (neste caso são os métodos *owlrl* e *rdfs*) (linha 3 do Algoritimo 1). O resultado desta chamada é uma instância da classe Graph do *rdflib*, que posteriormente é convertida em XML e salva em um arquivo de texto. Esse arquivo contém todas as restrições que foram quebradas, indicando em qual indivíduo dos dados ocorreu essa quebra. Além das restrições quebradas, algumas outras informações estão presentes neste XML, que são ignoradas através de um filtro que escolhe apenas tags do XML que têm a tag *resultSeverity* como filho, que indica que o elemento encontrado se trata de uma quebra de restrição dos *shapes* do SHACL. Cada restrição encontrada, que são instancias da classe *Element* da biblioteca *xml*, é adicionada a uma lista.

Com as restrições em mãos, o próximo passo é identificar as restrições que se referem à dimensão de qualidade de concisão de objetos (linha 4 do Algoritimo 1). Isso é feito passando um filtro na lista de restrições para considerar apenas os elementos que têm o filho *sourceConstraintComponent* com o valor *SPARQLConstraintComponent*. Em seguida, é feita uma busca em SPARQL por todos os objetos (valores) do predicado *hasCode*, que não pode ter valores duplicados (linha 7 do Algoritimo 1). Para cada objeto retornado pela busca anterior, é feita uma outra busca, também em SPARQL, para retornar os sujeitos que têm esse objeto ligado pelo predicado *hasCode*. Com os resultados desta busca, é feita uma verificação (em SPARQL, também) para identificar se os sujeitos encontrados estão ligados pelo predicado *olw:sameAs* (linha 11 do Algoritimo 1). Caso positivo, os sujeitos que estão ligados são adicionados a um conjunto de indivíduos que será utilizada posteriormente, chamada de *valids* (linha 13 do Algoritimo 1). Em caso negativo, nada é feito.

Feita a validação de indivíduos duplicados, o próximo passo é mostrar ao usuário, por meio do terminal, quais são as restrições que foram quebradas no conjunto de dados selecionado, tendo como base a ontologia e os *shapes* do SHACL informados. Para isso, é feita uma iteração na lista de restrições encontrada (linha 20 do Algoritimo 1) e para cada elemento é identificado em qual indivíduo foi identificada a quebra de restrição (através da tag XML *focusNode*) e qual a mensagem da quebra da restrição (através da tag XML *resultMessage*). Caso a mensagem esteja no formato JSON, será feita uma conversão de JSON para dicionários do Python através da biblioteca *json* e verificado se a propriedade *type* tem o valor *duplicate*. Esse valor indica que a restrição se trata de uma quebra de concisão de objetos e, caso o indivíduo desta restrição esteja na lista de sujeitos a serem ignorados (linha 23 do Algoritimo 1), esta quebra de restrição será desconsiderada pois o indivíduo não é uma duplicata real. Caso o

indivíduo não, o esteja nesta lista, a quebra de restrição é mostrada normalmente ao usuário (linha 27 e 28 do Algoritimo 1). Para as quebras de restrições que não se tratam da dimensão concisão de objetos não é feita nenhuma validação especial, sendo apenas mostradas para o usuário.

## 5.5 Validação no Cenário de Caso de Uso

Para a validação da verificação das 3 dimensões de qualidade implementadas em SHACL foram utilizados dois conjuntos de dados, um contendo dados sem erro (conjunto A), que seriam os dados originais, e outro contendo dados que quebram as restrições de qualidade escolhidas (conjunto B). O conjunto A contém pelo menos uma instância de cada classe da ontologia e é baseado no transporte público de Curitiba, por isso alguns nomes podem ser comuns para quem utiliza a rede de ônibus da cidade de Curitiba. Da mesma forma, o conjunto B contém dados adicionais com algumas irregularidades inseridas de propósito, como um ônibus que não passa em nenhum ponto, uma estação de ônibus que não tem nenhum linha, uma faculdade que contém alguns dados mal formatados e indivíduos duplicados.

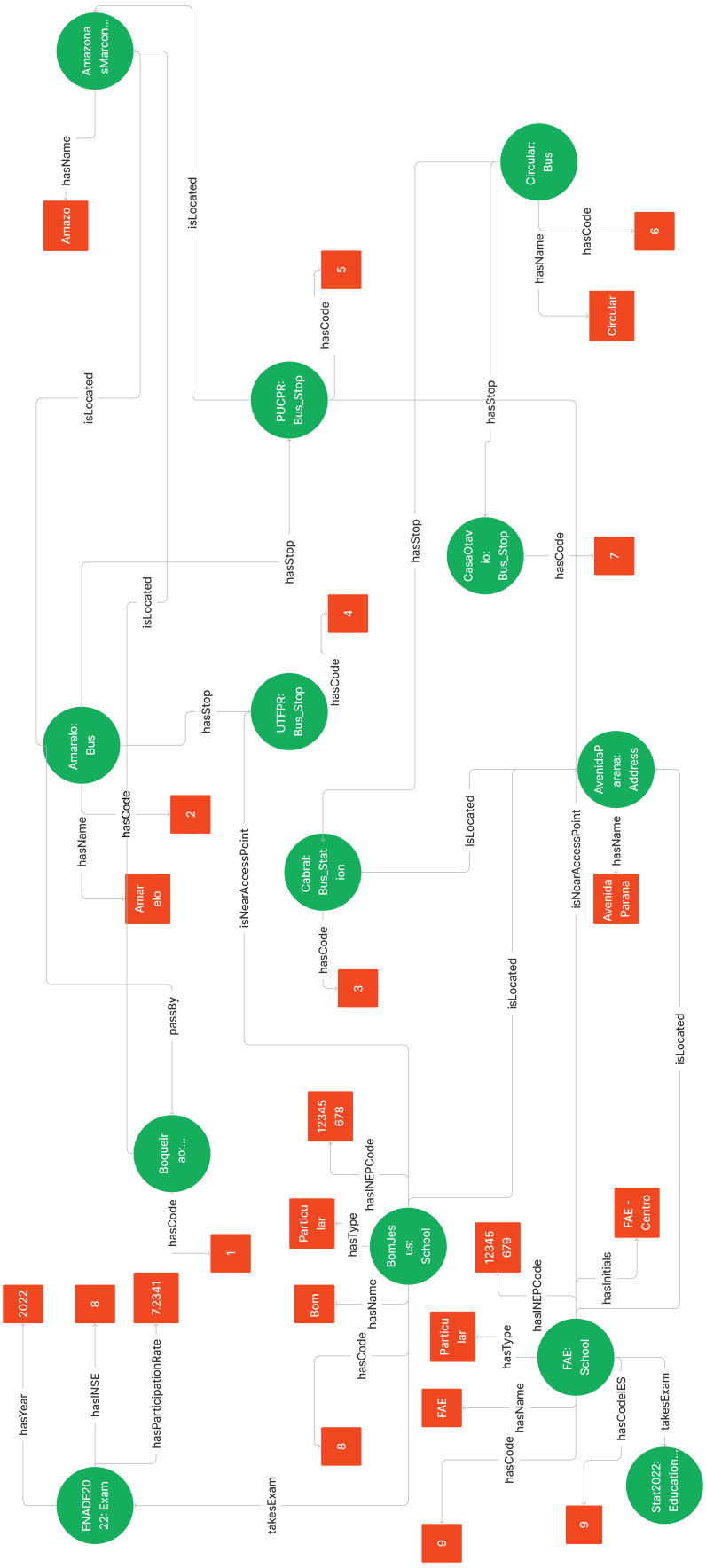
A Figura 15 é uma representação gráfica do conjunto de dados A, onde os círculos verdes representam os indivíduos, sendo indicado dentro do circulo a qual classe pertencem, e os quadrados vermelhos são dados literais, como *string*, inteiros e decimais. Os predicados que ligam os indivíduos com outros indivíduos ou com literais são representados por uma flecha, sendo o começo da flecha o sujeito e a fim da flecha o objeto. Por exemplo, o indivíduo PUCPR (sujeito) está ligado ao literal ”5” (objeto) através da flecha *hasCode* (predicado).

A Figura 16 é uma representação gráfica do conjunto B, utilizando a mesma notação do conjunto A com a adição dos círculos azuis, que indicam indivíduos que também estão presentes no conjunto A. No conjunto B foram criados dados que quebram as restrições de qualidade propostas. É importante notar que este conjunto é composto de três indivíduos disjuntos e por isso não têm conexão entre si. A primeira quebra de restrição é no indivíduo *Osorio*, que é um ônibus que passa apenas em um ponto. A segunda quebra ocorre no indivíduo *RuiBarbosa*, que é uma estação que não tem ônibus parando e que possui um predicado inválido (*hasName*). Por último, o indivíduo *Tuiuti* é uma universidade que tem o código INEP inválido (pois o código deve ter 8 dígitos) e que possui o predicado *hasCode* com um valor que, quando fundido com o conjunto A, vai ser duplicado. Com isso, as quebras de restrições deverão passar do conjunto B para o conjunto A ao fundir os dois conjuntos.

Primeiramente, o *script* de validação em Python foi executado utilizando o conjunto de dados A. Uma vez verificado que o conjunto de dados não contém nenhuma quebra de restrições que foram definidas em SHACL, o conjunto A e o conjunto B foram fundidos utilizando o VSCode e posteriormente o *script* de validação em Python foi executado utilizando o conjunto resultante da fusão. O objetivo desta etapa é verificar se as quebras de restrições que foram inseridas no conjunto de dados através da fusão foram identificadas pelo *script*. Feito isso, o

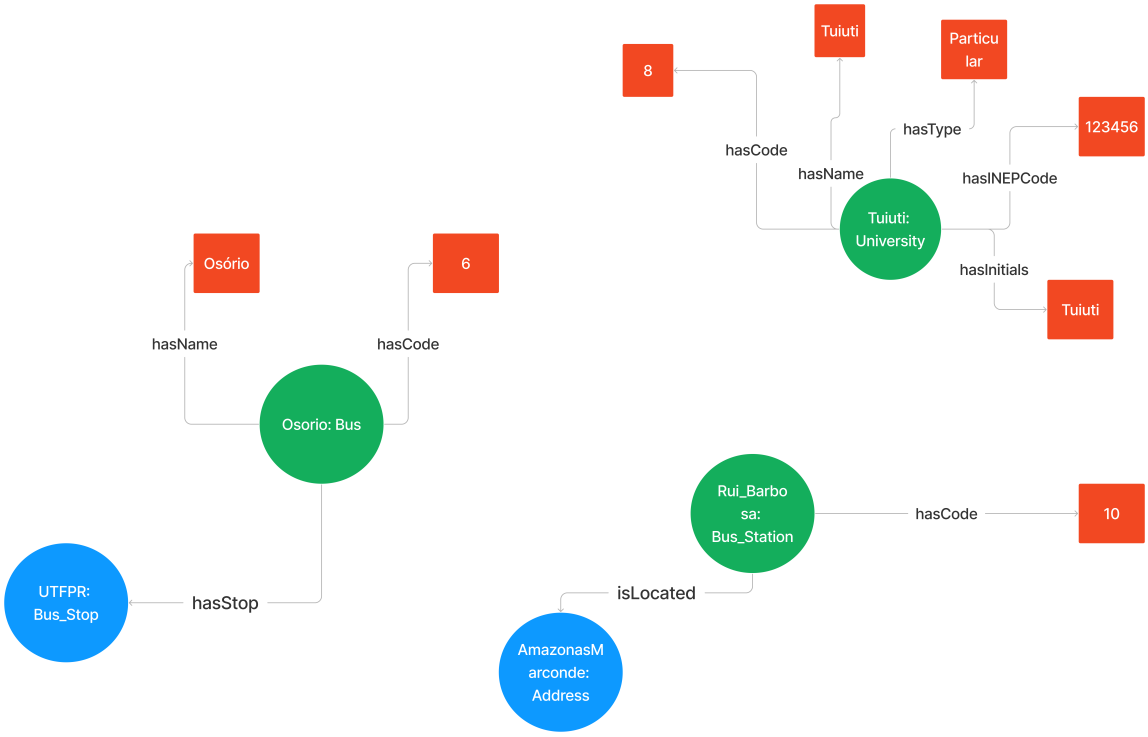
*script* foi executado novamente, porém agora com os indivíduos duplicados ligados através do predicado *olw:sameAs*, que indica que dois indivíduos representam o mesmo objeto do domínio. Após a execução, foi validado se os os casos de duplicação identificados pelas restrições de concisão de objetos foram ignorados nas situações onde os objetos continham o predicado *owl:sameAs* entre eles.

**Figura 15 – Conjunto de testes A**
**(a)**




**Fonte: Autoria própria.**

**Figura 16 – Conjunto de testes B**
**(a)**



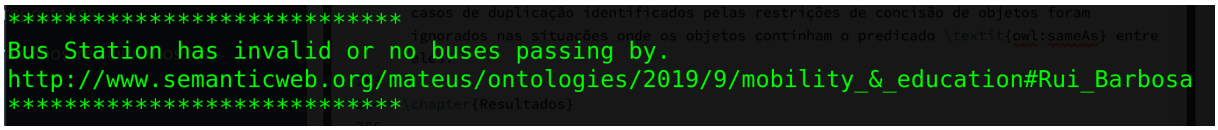

**Fonte: Autoria própria.**

# 6 RESULTADOS

Através dos testes realizados, mostrou-se que o *script* desenvolvido para a verificação das dimensões de qualidade de acurácia, concisão e consistência valida corretamente os dados da ontologia através dos *shapes* do SHACL e ignora os casos de duplicação de predicados únicos quando os indivíduos duplicados estão ligados pelo predicado *owl:sameAs*, uma vez que foram mostradas para o usuário todas as restrições definidas no SHACL que foram quebradas, com exceção daquelas que não se tratavam de duplicatas.

**Figura 17 – Exemplo de erro de consistência**

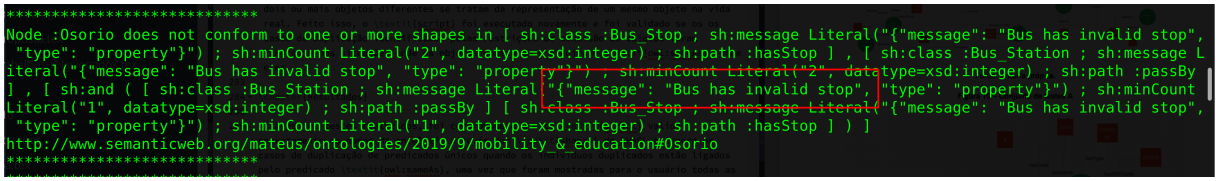
**(a)**




**Fonte: Autoria própria.**

**Figura 18 – Exemplo de erro de consistência**

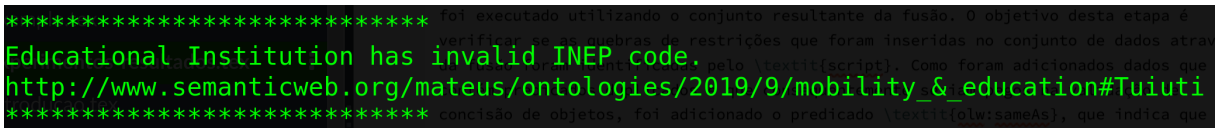
**(a)**




**Fonte: Autoria própria.**

**Figura 19 – Exemplo de erro de acurácia**

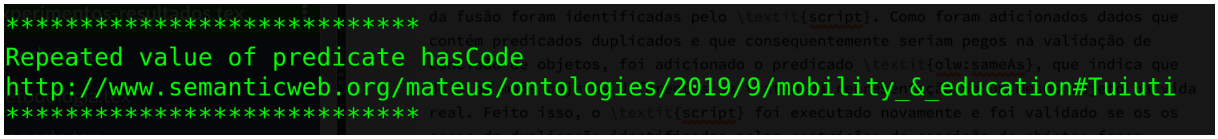
**(a)**




**Fonte: Autoria própria.**

**Figura 20 – Exemplo de erro de concisão de objetos**

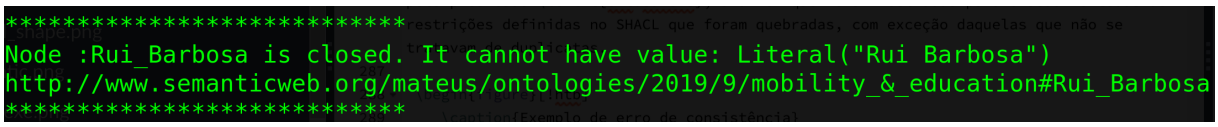
**(a)**




**Fonte: Autoria própria.**

**Figura 21 – Exemplo de erro de concisão de atributos**

**(a)**




**Fonte: Autoria própria.**

Após a execução do *script* de validação, a lista de quebras de restrições foi apresentada no terminal. Cada quebra de restrição possui duas linhas, onde a primeira linha do erro indica qual o tipo do erro e a segunda em qual indivíduo ocorreu a quebra da restrição. Já para a restrição de acurácia, presente no indivíduo *Tuiuti*, foi gerado o erro da Figura 19. Por exemplo, para a quebra de restrição de consistência, presente nos indivíduos *Osorio* e *RuiBarbosa*, foram gerados os erros da Figura 17 e 18. É importante notar que o erro da Figura 18, referente ao indivíduo *Osorio* tem um padrão diferente dos outros, que se deve ao fato do *shape* correspondente do SHACL utilizar o componente *sh:and*.

Por último, as quebras de restrição referentes à qualidade de concisão podem ser encontradas nos indivíduos *Tuiuti* e *RuiBarbosa*. Para a quebra de concisão de objetos, foi gerado o erro da Figura 20. Já para a quebra de concisão de atributos, o erro da Figura 21 foi gerado. Esses resultados mostram que o *script* valida com sucesso todas as dimensões de qualidade propostas no trabalho, indicando para o usuário qual foi a quebra de restrição e em qual indivíduo do conjunto de dados ela ocorreu.

Além disso, após adicionar o predicado *owl:sameAs* entre os indivíduos duplicados, as quebras de restrição de concisão de objetos referentes a esses indivíduos foram ignoradas e não apareceram no terminal após a execução. Isso comprova que o *script* desconsidera corretamente os indivíduos duplicados mas que representam a mesma entidade no mundo real.

# 7 CONCLUSÃO

Neste trabalho foram apresentados um dos potenciais problemas gerado pela fusão de dois ou mais conjuntos de dados ligados, que é quebra das dimensões de qualidade de consistência, concisão e acurácia. Esse trabalho propôs a resolução deste problema através da validação dos dados após a fusão, permitindo com que o usuário dos dados seja alertado dos possíveis erros inseridos após a fusão. Foram definidos os seguintes objetivos para sanar este problema:

Primeiro, buscou-se entender com mais profundidade as dimensões escolhidas e quais as suas implicações foi feita uma revisão bibliográfica das dimensões, buscando entender em quais contextos cada uma se aplica e como poderiam ser aferidas em determinado conjunto de dados.

O próximo objetivo foi criar as formas de aferição de cada uma das dimensões de qualidade. O SHACL foi a ferramente escolhida para fazer a aferição das dimensões, porém foi necessário que os componentes do SHACL fossem estudados e mapeados com as dimensões escolhidas. Para isso, estudos foram feitos na documentação do SHACL e o mapeamento entre as dimensões de qualidade e os componentes foi criado.

O objetivo seguinte foi criar a ontologia que seria utilizada pelo algoritmo de raciocínio para aferir a qualidade dos dados. Para que esse objetivo fosse completado, primeiro fez-se necessária a criação, em linguagem natural, de todas as restrições de qualidade da ontologia em que os dados se basearam. Para isso foi utilizado o conhecimento obtido no primeiro objetivo deste trabalho.

Com as restrições de qualidade criadas e o mapeamento obtido pelo segundo objetivo, foi possível criar a ontologia de validação com a linguagem SHACL. Esse objetivo foi concluído utilizando cada restrição de qualidade criada e sua respectiva dimensão de qualidade e aplicando o mapeamento de dimensões de qualidades e componentes feitos no objetivo dois.

Por fim, o último objetivo consistia em realizar testes fundindo dois ou mais conjuntos de dados e validar, através da ontologia gerada pelo terceiro objetivo, se quebras das dimensões de qualidade seriam identificadas. Para isso, utilizando a ontologia base, foram criados dois conjuntos de dados, um sem quebra de restrições, que era o conjunto original, e outro com quebra de restrições, que era o conjunto a ser fundido, que foram fundidos e submetidos ao teste.

Os resultados mostram que, por meio da ontologia de validação em SHACL, foi possível identificar as quebras nas restrições de qualidade no conjunto de dados fundido. Além disso, esse trabalho formaliza a validação das três dimensões escolhidas através da linguagem SHACL, permitindo com que outros trabalhos utilizem o mapeamento feito para criarem suas próprias ontologias de validação em SHACL.

## 7.1 Trabalhos futuros

Como proposta de trabalhos futuros, este trabalho pode ser estendido para que usuários que desejem criar *shapes* no SHACL para as dimensões tratadas neste trabalho não precisem utilizar a linguagem SHACL diretamente. No lugar, pode ser criada uma interface gráfica onde o usuário pode montar as restrições que posteriormente seriam mapeadas através dos mapeamentos feitos no capítulo 5.3. Também poderiam ser criados algoritmos e técnicas para fazer o mapeamento automático de restrições a partir das restrições em texto, sem a necessidade da interação com a ontologia ou com o SHACL.

Além disso, poderia ser criada uma interface gráfica para a execução da validação e apresentação das quebras de restrições para o usuário, onde o usuário pode importar a ontologia, os *shapes* e os dados sem a necessidade de ter o conhecimento de *Python* para alterar os inputs do *script* e também visualizar os erros fora do terminal. Além disso, uma biblioteca em Python poderia ser criada com base no *script* desenvolvido neste trabalho e publicada para utilização futura.

Os mapeamentos de componentes do SHACL também podem ser estendidos para outras dimensões de qualidade, além das tratadas deste trabalho. Dessa maneira, uma cobertura maior de dimensões de qualidade seria formalizada pelo SHACL dessa forma permitindo com que *shapes* mais completos possam ser criados para a validação de dados.

Na área de otimização, poderiam ser criados algoritmos para melhorar a performance do SHACL em *datasets* muito grandes ou em ontologias com muitas classes. Além disso, testes de desempenho poderiam ser feitos com o SHACL para entender as limitações e restições do SHACL.

# REFERÊNCIAS


BELIZARIO, M. G. Shacl and shex in the wild: A community survey on validating shapes generation and adoption. 2020.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific american**, JSTOR, v. 284, n. 5, p. 34–43, 2001.

BIZER, C.; CYGANIAK, R. Quality-driven information filtering using the wiqa policy framework. **Journal of Web Semantics**, Elsevier, v. 7, n. 1, p. 1–10, 2009.

BLEIHOLDER, J.; NAUMANN, F. Data fusion. **ACM computing surveys (CSUR)**, ACM New York, NY, USA, v. 41, n. 1, p. 1–41, 2009.

CONSOLE, M.; LENZERINI, M. Data quality in ontology-based data access: The case of consistency. *In*: **Proceedings of the AAAI Conference on Artificial Intelligence**. [*S.l.*: *s.n.*], 2014. v. 28, n. 1.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an ontology? *In*: **Handbook on ontologies**. [*S.l.*]: Springer, 2009. p. 1–17.

HALEVY, A.; NORVIG, P.; PEREIRA, F. The unreasonable effectiveness of data. **IEEE Intelligent Systems**, v. 24, n. 2, p. 8–12, 2009.

HEATH, T.; BIZER, C. Linked data: Evolving the web into a global data space. **Synthesis lectures on the semantic web: theory and technology**, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–136, 2011.

HITZLER, P. A review of the semantic web field. **Communications of the ACM**, ACM New York, NY, USA, v. 64, n. 2, p. 76–83, 2021.

HOGAN, A. *et al.* Weaving the pedantic web. *In*: **LDOW**. [*S.l.*: *s.n.*], 2010.

HYLAND, B.; WOOD, D. The joy of data-a cookbook for publishing linked government data on the web. *In*: **Linking government data**. [*S.l.*]: Springer, 2011. p. 3–26.

KOVALENKO, O. *et al.* Modeling automationml: Semantic web technologies vs. model-driven engineering. *In*: IEEE. **2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)**. [*S.l.*], 2015. p. 1–4.

LENZERINI, M. Ontology-based data management. *In*: **Proceedings of the 20th ACM international conference on Information and knowledge management**. [*S.l.*: *s.n.*], 2011. p. 5–6.

MENDES, P. N.; MÜHLEISEN, H.; BIZER, C. Sieve: linked data quality assessment and fusion. *In*: **Proceedings of the 2012 joint EDBT/ICDT workshops**. [*S.l.*: *s.n.*], 2012. p. 116–123.

NAPHADE, M. *et al.* Smarter cities and their innovation challenges. **Computer**, IEEE, v. 44, n. 6, p. 32–39, 2011.

NGOMO, A.-C. N. *et al.* Introduction to linked data and its lifecycle on the web. *In*: SPRINGER. **Reasoning Web International Summer School**. [*S.l.*], 2014. p. 1–99.

PANDIT, H. J.; O'SULLIVAN, D.; LEWIS, D. Using ontology design patterns to define shacl shapes. *In*: **WOP@ ISWC**. [*S.l.*: *s.n.*], 2018. p. 67–71.

PARETI, P.; KONSTANTINIDIS, G. A review of shacl: From data validation to schema reasoning for rdf graphs. **Reasoning Web International Summer School**, Springer, p. 115–144, 2021.

PAULHEIM, H.; STUCKENSCHMIDT, H. Fast approximate a-box consistency checking using machine learning. *In*: SPRINGER. **European Semantic Web Conference**. [*S.l.*], 2016. p. 135–150.

RABBANI, K.; LISSANDRINI, M.; HOSE, K. Shacl and shex in the wild: A community survey on validating shapes generation and adoption. 2022.

RIETVELD, L.; BEEK, W.; SCHLOBACH, S. Lod lab: Experiments at lod scale. *In*: SPRINGER. **International semantic web conference**. [*S.l.*], 2015. p. 339–355.

SPAHIU, B.; MAURINO, A.; PALMONARI, M. Towards improving the quality of knowledge graphs with data-driven ontology patterns and shacl. *In*: **ISWC (Best Workshop Papers)**. [*S.l.*: *s.n.*], 2018. p. 103–117.

WANG, R. Y.; STRONG, D. M. Beyond accuracy: What data quality means to data consumers. **Journal of management information systems**, Taylor & Francis, v. 12, n. 4, p. 5–33, 1996.

ZAVERI, A. *et al.* User-driven quality evaluation of dbpedia. *In*: **Proceedings of the 9th International Conference on Semantic Systems**. [*S.l.*: *s.n.*], 2013. p. 97–104.

ZAVERI, A. *et al.* Quality assessment for linked data: A survey. **Semantic Web**, IOS Press, v. 7, n. 1, p. 63–93, 2016.

# ANEXO A – Shapes em SHACL

**Figura 22 – *Shape* da classe *Bus***

**(a)**

```
:BusShape a sh:NodeShape ;
    sh:targetClass :Bus ;
    sh:or (
        [
            sh:message '''{"message": "Bus has invalid stop", "type": "property"}''' ;
            sh:path :hasStop ;
            sh:minCount 2 ;
            sh:class :Bus_Stop ;
        ]
        [
            sh:message '''{"message": "Bus has invalid stop", "type": "property"}''' ;
            sh:path :passBy ;
            sh:minCount 2 ;
            sh:class :Bus_Station ;
        ]
        [
            sh:and (
                [
                    sh:message '''{"message": "Bus has invalid stop", "type": "property"}''' ;
                    sh:path :passBy ;
                    sh:minCount 1 ;
                    sh:class :Bus_Station ;
                ]
                [
                    sh:message '''{"message": "Bus has invalid stop", "type": "property"}''' ;
                    sh:path :hasStop ;
                    sh:minCount 1 ;
                    sh:class :Bus_Stop ;
                ]
            )
        ]
    ) ;
    sh:property [
        sh:message '''{"message": "Bus has invalid code.", "type": "property"}''' ;
        sh:path :hasCode ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Bus has invalid name.", "type": "property"}''' ;
        sh:path :hasName ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
.
```

**Fonte: Autoria própria.**

**Figura 23 – *Shape* da classe *Access_Point***

**(a)**

```
:AccessPointShape a sh:NodeShape ;
    sh:targetClass :Access_Point ;
    sh:property [
        sh:message '''{"message": "Access Point has invalid code.", "type": "property"}''' ;
        sh:path :hasCode ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:minInclusive 1 ;
        sh:maxInclusive 9999 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Access Point has invalid address.", "type": "property"}''' ;
        sh:path :isLocated ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class :Address ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

**Figura 24 – *Shape* da classe *Bus_Stop***

**(a)**

```
:BusStopShape a sh:NodeShape ;
    sh:targetClass :Bus_Stop ;
    sh:property [
        sh:message '''{"message": "Bus Stop has invalid or no buses passing by.", "type": "property"}''' ;
        sh:path :isStopOf ;
        sh:minCount 1 ;
        sh:class :Bus ;
    ] ;
.
```

**Fonte: Autoria própria.**

**Figura 25 – *Shape* da classe *Bus_Station***

**(a)**

```
:BusStationShape a sh:NodeShape ;
    sh:targetClass :Bus_Station ;
    sh:property [
        sh:message '''{"message": "Bus Station has invalid or no buses passing by.", "type": "property"}''' ;
        sh:path :hasBus ;
        sh:minCount 1 ;
        sh:class :Bus ;
    ] ;
.
```

**Fonte: Autoria própria.**

**Figura 26 – *Shape* da classe *Educational_Intitution***

**(a)**

```
:EducationalInstitutionShape a sh:NodeShape ;
    sh:targetClass :Educational_Institution ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid code.", "type": "property"}''' ;
        sh:path :hasCode ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid name.", "type": "property"}''' ;
        sh:path :hasName ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid address.", "type": "property"}''' ;
        sh:path :isLocated ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class :Address ;
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid type.", "type": "property"}''' ;
        sh:path :hasType ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^Particular$|^Público Municipal$|^Público Estadual$|^Público Federal$"
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid INEP code.", "type": "property"}''' ;
        sh:path :hasINEPCode ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^[0-9]{8}$"
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid near Access Point", "type": "property"}''' ;
        sh:path :isNearAccessPoint ;
        sh:minCount 0 ;
        sh:class :Access_Point
    ] ;
    sh:property [
        sh:message '''{"message": "Educational Institution has invalid exam", "type": "property"}''' ;
        sh:path :takesExam ;
        sh:minCount 1 ;
        sh:class :Exam
    ] ;
    sh:closed true ;      You, 1 second ago • Uncommitted changes
    sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

**Figura 27 – *Shape* da classe *University***

**(a)**

```
:UniversityShape a sh:NodeShape ;
    sh:targetClass :University ;
    sh:property [
        sh:message '''{"message": "University has invalid IES.", "type": "property"}''' ;
        sh:path :hasCodeIES ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
        sh:minInclusive 100 ;
        sh:maxInclusive 99999 ;
    ] ;
    sh:property [
        sh:message '''{"message": "University has invalid initials.", "type": "property"}''' ;
        sh:path :hasInitials ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

**Figura 28 – *Shape* da classe *Educational_Statistics***

**(a)**

```
:EducationalStatisticsShape a sh:NodeShape ;
    sh:targetClass :Educational_Statistics ;
    sh:property [
        sh:message '''{"message": "Statistc has invalid Abandonment Rate.", "type": "property"}''' ;
        sh:path :hasAbandonmentRate ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
        sh:datatype xsd:double ;
    ] ;
    sh:property [
        sh:message '''{"message": "Statistc has invalid Approval Rate.", "type": "property"}''' ;
        sh:path :hasApprovalRate ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
        sh:datatype xsd:double ;
    ] ;
    sh:property [
        sh:message '''{"message": "Statistc has invalid Disapproval Rate.", "type": "property"}''' ;
        sh:path :hasDisapprovalRate ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
        sh:datatype xsd:double ;
    ] ;
    sh:property [
        sh:message '''{"message": "Statistc has invalid Permanence Rate.", "type": "property"}''' ;
        sh:path :hasPermanenceRate ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
        sh:datatype xsd:double ;
    ] ;
    sh:property [
        sh:message '''{"message": "Exam has invalid INSE.", "type": "property"}''' ;
        sh:path :hasINSE ;
        sh:minCount 0 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties (rdf:type) ;
.
```

**Fonte: Autoria própria.**

**Figura 29 – *Shape* da classe *Exam***

**(a)**

```
:ExamShape a sh:NodeShape ;
    sh:targetClass :Exam ;
    sh:property [
        sh:message '''{"message": "Exam has invalid year.", "type": "property"}''' ;
        sh:path :hasYear ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Exam has invalid INSE.", "type": "property"}''' ;
        sh:path :hasINSE ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:integer ;
    ] ;
    sh:property [
        sh:message '''{"message": "Exam has invalid participation rate.", "type": "property"}''' ;
        sh:path :hasParticipationRate ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:double ;
    ] ;
.
```

**Fonte: Autoria própria.**

**Figura 30 – *Shape* da classe *Address***

**(a)**

```
:AddressShape a sh:NodeShape ;
    sh:targetClass :Address ;
    sh:property [
        sh:message '''{"message": "Address has invalid name.", "type": "property"}''' ;
        sh:path :hasName ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:message '''{"message": "Address has invalid neighborhood.", "type": "property"}''' ;
        sh:path :hasNeighborhood ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
.
```

**Fonte: Autoria própria.**