INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Reescrita de operações de limpeza de dados

FÁBIO ROBERTO BARROS DOS SANTOS
Outubro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Reescrita de operações de limpeza de dados

Fábio Roberto Barros dos Santos

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Paulo Alexandre Fangueiro Maio

Co-orientador: Paulo Jorge Machado Oliveira

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, outubro 2016

Dedicatória

Quero dedicar este trabalho em primeiro lugar à minha avó, Angelica Pedrinha que era das pessoas que mais orgulho tinha em ver-me a estudar e que enquanto esteve presente, sempre me deu força para lutar pelos meus objetivos.

Como é óbvio quero também dedicar este trabalho também aos meus pais, José Santos e Anabela Barros pelo apoio incondicional que sempre me deram durante toda a minha vida. Apesar de serem pessoas de origem humilde, sempre investiram no melhor para a minha educação, mostrando os seus princípios e educação que pretendem transmitir.

Por último, mas não menos importante, queria dedicar este trabalho à minha namorada, Jéssica Silva pelo apoio incondicional que me deu para a conclusão desta etapa na minha vida. Ela mais que ninguém sentiu de perto todo o esforço que tive de fazer para terminar o curso como aluno e trabalhador ao mesmo tempo, estando sempre do meu lado nas situações mais difíceis.

Resumo

Este trabalho surge num contexto em que as tecnologias estão presentes em grande escala no quotidiano das organizações/pessoas, produzindo uma elevada quantidade de dados. Nesses dados, por sua vez residem problemas de qualidade que precisam de ser tratados de uma forma simples e eficaz. O problema abordado neste trabalho é a reescrita de operações de limpeza de dados e a sua aplicabilidade entre diferentes repositórios/ferramentas de limpeza de dados. Essa necessidade surge pelo facto de no momento em que se define uma operação, a mesma se encontra dependente do repositório de dados/linguagem que utiliza. O objetivo proposto neste trabalho consiste em especificar as operações a um elevado nível conceptual (permitindo o seu acesso a outros utilizadores sem ser unicamente os especialistas) e permitir a automatização da sua reescrita.

A abordagem adotada consiste na elaboração de uma solução que implementa um processo de reescrita genérico, tendo por base conteúdo de configuração definido de acordo com o contexto em que vai ser utilizada. O conteúdo de configuração utilizado consiste na captura sobre um formato de ontologias do domínio (descrição da estrutura) do repositório de dados utilizado para definir as operações como também do domínio do repositório escolhido para a reescrita, do vocabulário utilizado para definir as operações como também do vocabulário que descreve a linguagem utilizada na ferramenta de limpeza de dados (para a qual a operação vai ser reescrita), alinhamentos entre as ontologias definidas anteriormente com o objetivo de obter as correspondências entre elementos das ontologias e por fim uma gramática que permite a definir a estrutura das operações. A partir da gramática e do conteúdo criado pelo processo de reescrita a um nível conceptual, é possível construir a operação final devidamente formatada para ser utilizada na ferramenta de dados.

Os resultados alcançados permitem a reescrita de operações simples utilizadas na maioria dos casos para reutilização em outros contextos de uma forma automática/semiautomática entre diferentes repositórios de dados/ferramentas de limpeza de dados. As operações que se encontram a um nível semiautomático de reescrita deve-se ao facto de necessitarem de parâmetros definidos explicitamente pelo utilizador. Apesar ser possível reescrever operações simples, existem limitações de reescrita quer ao nível da complexidade das operações, como também do suporte prestado pelo vocabulário base utilizado relativamente às funcionalidades das ferramentas de limpeza de dados.

Palavras-chave: qualidade dos dados, DQM, operações de limpeza de dados, ferramentas de limpeza de dados, reescrita de operações, ontologias

Abstract

This work arises in a context where technologies are present in large scale in the daily life of organizations/individuals producing a high amount of data. In the data, there are quality problems that need to be addressed in a simple and effective manner. The problem addressed in this paper is the rewriting of data cleaning and its applicability across different repositories/data cleansing tools. This need arises because when the user defines an operation, it is dependent on the data repository/language it uses. The goal proposed in this paper is to specify the operations at a high conceptual level (allowing access to other users not experts only) and enable automation of its rewrite.

The approach consists in developing a solution that implements a generic rewriting process, with the defined configuration content according to the context in which it will be used. The configuration content used is the capture of a domain through an ontology (description of structure) of data repository used to define the operations as well as the repository of the domain chosen for the rewrite, the vocabulary used to define the operations as well as the vocabulary that describes the language used in data cleaning tool (for which the operation will be rewritten), alignments between the previously defined ontologies in order to obtain correspondence between elements of ontologies and finally a grammar that allows to define operations' structure. From the grammar and content created by the rewrite process at a conceptual level, it is possible to construct the final operation with the properly formats for use in data tool.

The results achieved allow the rewriting of simple operations used in most cases for reuse in other contexts in an automatic/semiautomatic manner between different data repositories/data cleansing tools. The operations which are in a semiautomatic level of rewriting due to the fact that require explicit user-defined parameters. Although it is possible to rewrite simple operations, there are some rewriting limitations at operations' complexity level, as well as in the support provided by the base vocabulary used comparing to the features of data cleansing tools.

Keywords: Data Quality, DQM, Data Cleaning Operations, Query rewriting, ontologies

Agradecimentos

Quero agradecer em primeiro lugar aos meus colegas de curso pelo apoio e companheirismo que prestaram durante o meu percurso académico. Esse apoio é extremamente importante para a conclusão desta etapa. Se existe um curso que dificilmente pode ser concluído sem um espirito de mutuo auxilio e companheirismo, com certeza que é o de Engenharia Informática.

Quero agradecer aos meus orientadores por me terem proposto a elaboração deste trabalho, isto só demonstra a confiança que depositam nas minhas capacidades, como também no trabalho que desempenharam enquanto professores.

Em especial queria expressar o meu enorme agradecimento ao professor Paulo Maio pela dedicação que teve na elaboração deste trabalho, estando sempre disponível para o seu acompanhamento, dando um contributo importante para a sua elaboração. Quero também agradecer todo o conhecimento que me transmitiu enquanto professor, esse conhecimento permitiu tornar-me uma pessoa mais competente no âmbito das minhas funções enquanto engenheiro informático.

Quero agradecer também ao Sr. Engenheiro Pedro Bailão pela compreensão e flexibilidade que demonstrou no contexto laboral quando trabalhava na Glintt HS. Sem isso, a minha vida de trabalhador estudante seria muito mais complicada.

Por último, mas não menos importante, quero agradecer também ao Sr. Engenheiro Marco *Tschan* Carvalho pelo apoio e flexibilidade que demonstrou como entidade patronal, mas também como amigo para a conclusão deste trabalho. Sem esse apoio não era possível concluir este trabalho.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

API	*Application programming interface*
BAV	*Both As View*
BNF	*Backus-Naur Form*
DB	*Database*
DCO	*Data Cleaning Operation*
DQM	*Data Quality Management*
FURPS	Functionality Usability Reliability Performance Supportability
GAV	*Global As View*
GLAV	*Global Local As View*
IRI	*Internationalized Resource Identifier*
LAV	*Local As View*
OWL	Web Ontology Language
P2P	*Peer to Peer*
RDF	*Resource Description Framework*
SPARQL	*SPARQL Protocol and RDF Query Language*
SQL	*Structured Query Language*
URI	*Uniform Resource Identifier*

Lista de Símbolos

α	alfa
β	beta

1 Introdução

Neste capítulo introdutório é contextualizado o trabalho realizado, mostrando de que forma o mesmo se encontra enquadrado técnico-cientificamente como também no contexto socioeconómico, quais as motivações que levaram à sua elaboração, quais são os objetivos que se pretendem atingir, qual a abordagem aplicada ao trabalho e de que forma o documento se encontra estruturado.

1.1 Contexto e motivação

Atualmente os sistemas de informação têm vindo a assumir gradualmente um papel importante tanto para as organizações, como para as pessoas. Tal facto deve-se por um lado à acentuada e rápida evolução tecnológica que se tem vindo a sentir na última década, como também à alteração socioeconómica que tem vindo a acontecer, o que leva a uma maior abertura para o aumento da utilização/confiança nos sistemas de informação devido à sua massificação /introdução no quotidiano da vida das pessoas tanto a nível profissional como pessoal.

Esta utilização em massa leva a que seja importante garantir o correto funcionamento dos sistemas, de forma a manter a confiança na sua utilização. O correto funcionamento destes sistemas está associado a um conjunto de fatores, um deles a qualidade de dados.

Cada vez mais a qualidade dos dados começa a ser um requisito fundamental em diversas áreas, uma delas a elaboração de sistemas de informação onde assume maior importância. Para isso acontecer têm contribuído vários fatores, um deles a drástica evolução que se tem sentido em áreas como manipulação e/ou transformação dos dados para posterior análise. Conceitos associados à análise de grandes volumes de dados como *business intelligence* (Ranjan, 2009) e *big data* (Hansmann and Niemeyer, 2014) ganharam atualmente um enorme relevo na implementação de soluções informáticas, mostrando a importância que as organizações e os utilizadores atribuem aos dados.

Apesar de atualmente ser possível processar um elevado volume de dados comparativamente ao passado, continuam a existir alguns problemas que prejudicam os resultados obtidos, um deles com maior relevância é a qualidade dos dados. Um processo que garanta a qualidade dos dados (normalmente designado por processo de limpeza de dados) pode conter um conjunto de fases que incluem a deteção dos problemas, elaboração da operação de correção e por último a sua aplicação. Para a execução deste tipo operações existem um conjunto de ferramentas que permitem a sua definição, no entanto é difícil a sua reutilização em outros repositórios de dados sobre o mesmo domínio de aplicação. Com vista a potenciar a reutilização destas operações entre diferentes contextos torna-se necessária a reescrita das mesmas. Esta última necessidade identificada serve como justificação para o trabalho efetuado.

A motivação para a abordagem deste tema está relacionada com o facto de o mesmo ser em primeiro lugar um desafio devido ao seu grau de complexidade, como também a

utilidade/importância que este tipo de soluções pode assumir no auxilio da resolução deste tipo de problemas. Através de uma análise mais aprofundada do problema apresentado, é possível concluir que atualmente existe um elevado acoplamento entre as operações de limpeza de dados e o repositório onde as mesmas vão ser executadas, isto significa que uma determinada operação não pode ser reutilizada em outro repositório/ferramenta de limpeza de dados. Com um exemplo prático é possível demonstrar que na definição de uma determinada operação de limpeza estão explícitas certas características do modelo onde a mesma vai ser executada, como também certos elementos na sua sintaxe que servem de suporte à execução através da ferramenta de limpeza de dados. Para além disso muito do conhecimento para a elaboração destas operações estão unicamente acessíveis a especialistas na matéria, portanto, torna-se um contributo importante a possibilidade da criação deste tipo de operações por outro tipo de utilizadores.

1.2 Objetivos

Ao iniciar este trabalho foi necessário definir quais são os objetivos que se pretendem atingir, devido ao facto de o problema em questão poder ser estendido em vários níveis que por sua vez estão relacionados com diferentes graus de complexidade. Um exemplo disso são operações de limpeza de dados simples que se encaixam perfeitamente no âmbito de reescrita devido à propensão para a sua reutilização e do outro lado operações complexas que são definidas unicamente para resolver um problema de qualidade de dados daquele em especifico. Perante este tipo de questões, torna-se necessário refletir se estamos perante um caso em que se justifique a reescrita da operação ou não.

Tendo por base o problema abordado e os diferentes níveis que o mesmo pode ter, foram identificados os seguintes objetivos:

- Capacidade de reescrita da solução para operações que podem ser reutilizadas em outros contextos.
- Construir uma solução que consegue suportar a reescrita de operações para diferentes ferramentas de limpeza de dados.
- Fácil interpretação dos dados gerados pela ferramenta de forma clara e objetiva descrever a sua execução.

1.3 Abordagem

Para a elaboração deste trabalho, adotou-se uma abordagem baseada na metodologia designada por "*Design Science*" (Hevner et al., 2004).

Esta abordagem permite um conhecimento e análise do domínio do problema e a sua solução é alcançada através do desenho e da construção de artefacto(s) concebido(s) para o propósito.

Esta metodologia assenta em sete normas:

1. Desenhar a solução como um artefacto: esta metodologia deve produzir um artefacto viável, tendo em conta a sua construção, modelação e metodologias usadas.
2. Relevância do problema: o objetivo é desenvolver uma solução tecnológica para importantes problemas de negócio.
3. Avaliação do desenho: a utilidade, qualidade e eficácia do artefacto concebido deve ser demonstrada de forma rigorosa através da realização de um conjunto de avaliações na solução concebida.
4. Contribuições do estudo: deve fornecer uma clara contribuição para outras áreas de conhecimento.
5. Rigor do estudo: a pesquisa deve basear-se na aplicação de métodos rigorosos tanto na construção, como na avaliação do desenho do artefacto.
6. Desenhar o processo de pesquisa: durante a pesquisa, é necessário utilizar os meios disponíveis para alcançar os fins desejados, com o objetivo de satisfazer os requisitos do problema.
7. Divulgação de resultados: é necessário que a passagem de conhecimento seja feita para que, tanto as pessoas ligadas à área da tecnologia, como outro tipo de pessoas percebam quais foram os resultados alcançados.

De acordo com as normas da metodologia descritas anteriormente, foi abordado em primeiro lugar a relevância do problema através de um estudo aprofundado da abordagem que serve de base a este trabalho através da sua descrição, como do contexto tecnológico em que o mesmo se insere.

O passo seguinte consiste na sistematização do conhecimento dos conceitos/áreas relacionadas com o problema através do processo de pesquisa e recolha de informação.

Depois de detetada a carência de ferramentas que lidassem com modelos/linguagens diferentes para a execução de operações de limpeza, foi desenhada uma solução para a construção de um artefacto que satisfizesse os requisitos levantados pelo problema.

1.4 Estrutura do documento

O documento é iniciado com uma contextualização do problema e da necessidade deste tipo de soluções no panorama atual, quais as motivações que estão na base da elaboração do trabalho e qual foi a abordagem seguida durante a sua composição.

No capítulo 2 é iniciada uma fase de estudo que levou à apresentação de diversos conceitos relacionados com o problema abordado neste trabalho e identificação das carências das abordagens/ferramentas de limpeza de dados face ao problema em questão.

No capítulo 3 é feita uma análise de valor de negócio sobre a solução em causa através da abordagem de conceitos ligados à área do empreendedorismo como a proposição de valor, segmentos de clientes e respetiva abordagem, o modelo de canvas e o processo de criação de valor.

No capítulo 4 é apresentada a abordagem que serve de suporte ao processo de conceção da solução através de uma perspetiva geral e exemplos práticos da sua aplicabilidade, concluindo com a informação relevante que deve ser assimilada para compreensão do trabalho.

No capítulo 5 são identificados os requisitos funcionais e não funcionais que a solução deve obedecer, realizando a sua classificação num modelo especifico. Este capítulo é importante, porque através dele é possível compreender o que é abordado nos restantes capítulos.

No capítulo 6 é analisada a solução consoante os requisitos definidos no capítulo anterior através da definição formal do processo de reescrita a ser implementado e da captura dos conceitos relacionados através de um modelo de domínio. Depois de analisada a solução, é feito o seu *design* através de duas perspetivas que são a arquitetural e a detalhada. Na perspetiva arquitetural é feito o *design* da arquitetura da solução onde é possível ter a perceção da sua divisão em diferentes módulos. Na perspetiva detalhada, o foco está na operação de reescrita e no seu fluxo de execução.

No capítulo 7 é descrita a implementação da solução, tendo como base a análise e o *design* efetuado no capítulo anterior. Essa descrição é feita através das etapas que dividiram este processo, seguido de um exemplo prático que permite mostrar de que forma a mesma foi configurada para a sua primeira utilização. Este capítulo termina com a sua validação ao nível dos seus componentes.

No capítulo 8 é estruturado o processo de experimentação e avaliação da solução concebida através da definição dos objetivos a avaliar, formalização do processo, ambiente de testes criado, apresentação dos resultados obtidos e conclusões obtidas na elaboração das experiências.

No capítulo 9 são apresentadas as conclusões obtidas através de uma reflexão sobre os objetivos alcançados, os objetivos que não foram atingidos, um balanço sobre que contributo este trabalho dá tendo em conta o que foi possível concretizar comparativamente aos pontos que ficaram por concretizar, terminando com indicações de trabalho futuro que pode ser realizado tendo por base este trabalho.

2 Estado da arte

Neste capítulo é apresentado o estado da arte onde são expostos todos os conceitos relacionados com o tema deste trabalho.

Este capítulo inicia-se com uma secção de apresentação da terminologia utilizada para a descrição do trabalho, onde são descritos os termos utilizados ao longo do documento, com o objetivo de evitar diferentes interpretações e tornar clara a sua leitura.

O primeiro tema a ser abordado são as diferentes taxonomias existentes para problemas de qualidade de dados, como forma de apresentar o conceito e que formalizações foram adotadas para a sua descrição.

Tendo como base o conhecimento adquirido sobre problemas de qualidade de dados, são apresentadas as ferramentas/abordagens existentes. O objetivo nesta fase é compreender em que estado se encontra a resolução de problemas de qualidade de dados, identificando que limitações existem e quais delas levam a que seja necessária a elaboração deste trabalho.

Como o âmbito deste trabalho é a reescrita de operações de limpeza de dados, é necessário um estudo sobre o trabalho efetuado no âmbito da reescrita de consultas. Para isso, são introduzidos alguns conceitos sobre gramáticas para perceber de que forma a linguagem se encontra estruturada e quais as ferramentas existentes para a sua criação. Ao nível da reescrita de consulta são estudadas diferentes arquiteturas que permitem ter a noção da reescrita entre diferentes modelos/esquemas e a reescrita entre linguagens utilizadas em repositórios de dados que permitem ter a noção da reescrita entre diferentes linguagens.

2.1 Terminologia

Antes de abordar os conceitos que são estudados na elaboração deste trabalho, é necessário apresentar os termos utilizado com o objetivo de tornar clara a sua leitura.

Podemos agrupar os conceitos mencionados em três grupos que são os conceitos relacionados com repositórios de dados, ferramentas de limpeza de dados e operações de limpeza de dados. Para cada conceito é descrito para que finalidade é usado, que sinónimos são utilizados para o referenciar e quais os conceitos que se encontram associados.

Os conceitos associados a repositórios/fonte de dados são sinónimos utilizados para referenciar o suporte de armazenamento dos dados alvo das operações de limpeza.

Quando é necessário referenciar um repositório de dados são utilizados sinónimos como modelo e esquema de dados. Quando se pretende referir à descrição da estrutura de um determinado repositório de dados, é utilizado o conceito de domínio.

No que toca aos conceitos abordados no âmbito das ferramentas de limpeza de dados, são utilizados os conceitos de vocabulário como referência à sintaxe das instruções suportadas por uma determinada ferramenta de dados e o conceito de formato que é um sinónimo da sintaxe suportada pelas instruções da ferramenta.

Relativamente ao conceito de operação de limpeza de dados, o mesmo refere-se a uma única ação que tem como objetivo resolver um determinado problema de qualidade de dados. A utilização do acrónimo DCO serve como sinónimo para referenciar uma determinada operação de limpeza.

O conceito de vocabulário referido anteriormente também é utilizado para falar sobre a sintaxe de uma determinada operação de limpeza de dados. Por vezes, ao falar sobre operações de limpeza de dados, é utilizado o conceito de processo de limpeza de dados. O que se pretende descrever é um conjunto de operações de limpeza de dados que compõe um processo.

Outro conceito associado a uma operação de limpeza de dados é a sua serialização. Quando se fala em serialização de operações de limpeza de dados, o que se pretende dizer é que dada uma operação reescrita é construída a respetiva instrução consoante a sintaxe suportada pela ferramenta de limpeza de dados escolhida para a serialização.

Depois de apresentada a terminologia utilizada neste trabalho, as seguintes secções consistem na apresentação do estudo efetuado e respetivas conclusões obtidas.

2.2 Taxonomias para problemas de limpeza de dados

Um dos conceitos abordados neste trabalho são os problemas de limpeza de dados. Devido ao facto de existirem diversos tipos de problemas de qualidade de dados, torna-se necessária a elaboração da sua classificação. Nesta secção são descritas algumas taxonomias criadas com o objetivo de classificar os diferentes problemas de qualidade de dados.

2.2.1 SmartClean

O SmartClean (Oliveira, 2009, p. 110) é um trabalho que incide na deteção e correção de problemas de qualidade de dados em bases de dados relacionais. Esta secção consiste na descrição da taxonomia desenvolvida pelo autor para categorizar os diferentes problemas de qualidade de dados que residem a diferentes níveis.

Cada problema de qualidade de dados encontra-se agrupado em cinco níveis distintos de acordo com a sua área de incidência. Existem problemas de qualidade de dados que podem ser processados ao nível mais elementar como é o caso do atributo, podendo passar para níveis superiores como a coluna onde são abordados os diferentes valores de uma coluna, ao nível do tuplo (linhas da tabela) onde são utilizadas diferentes colunas de um determinado registo, ao nível da relação onde são utilizadas chaves estrangeiras para outras tabelas terminando na múltipla relação onde são utilizadas diferentes fontes de dados. A Figura 1 mostra os tipos de

grupos que permitem classificar os problemas de qualidade de dados suportados por esta ferramenta de dados.

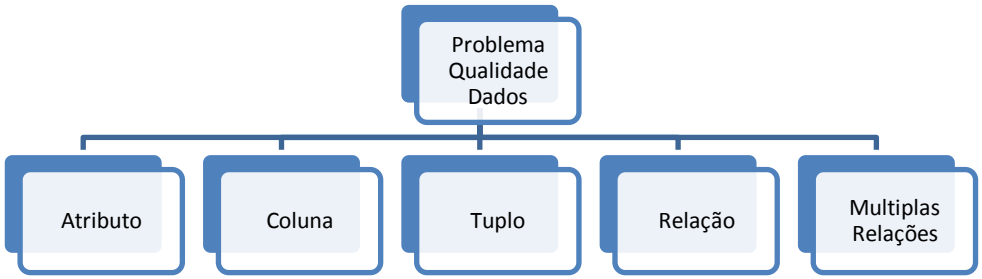


Figura 1 – Tipos de operações SmartClean

Depois de definido o seu tipo, uma operação pode ser definida em quatro níveis de acordo com o nível de granularidade que se pretenda aplicar. Os primeiros dois níveis encontram-se relacionados unicamente com a tabela onde se pretende executar a operação, ao contrário dos restantes que utilizam outras tabelas para a execução da operação. O nível de operação o primeiro nível é a coluna onde o foco é no valor do atributo, passando para a linha onde são utilizados os atributos contidos na mesma para a execução da operação. Os restantes níveis numa primeira fase relacionam outras tabelas com a tabela que se pretende utilizar na operação, ao contrário do último nível em que se pretende relacionar diferentes repositórios de dados.

A Figura 2 mostra os diferentes níveis de operação descritos anteriormente.

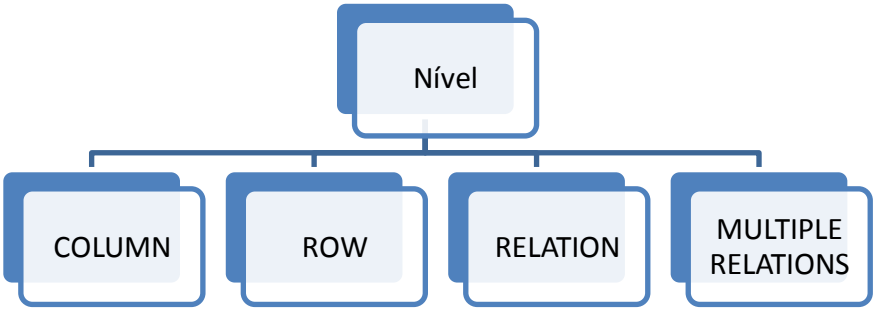


Figura 2 – Níveis de operações SmartClean

A partir da análise desta taxonomia é possível perceber que tipos de problemas de dados existem, permitindo assim a sua categorização, bem como o seu enquadramento em diferentes níveis. Isso permite ter uma noção de como um determinado problema pode afetar a qualidade de dados.

2.2.2 DQM

Esta seção tem como objetivo descrever o modelo conceptual que permite a representação de regras de limpeza de dados através de uma ontologia definida em RDF/OWL (Heflin and others, 2007). Esta ontologia providencia um vocabulário para representação de regras de limpeza e classificação de qualidade dos dados designada por DQM (Fürber and Hepp, 2011). Através da sua utilização é possível monitorizar possíveis problemas de qualidade, automatizar operações de limpeza das mesmas como também a sua classificação. O facto de as regras de limpeza de dados se encontrarem definidas neste formato permite a sua partilha e reutilização.

Este modelo conceptual foi desenvolvido com o objetivo de capturar um conjunto de conceitos relacionados com qualidade/classificação de limpeza de dados, permitindo assim a elaboração de regras/critérios de avaliação para os mesmos.

A Figura 3 representa parcialmente em UML o domínio das áreas referidas anteriormente que foram capturadas na ontologia. Atualmente esta ontologia é composta por 45 classes e 56 propriedades.

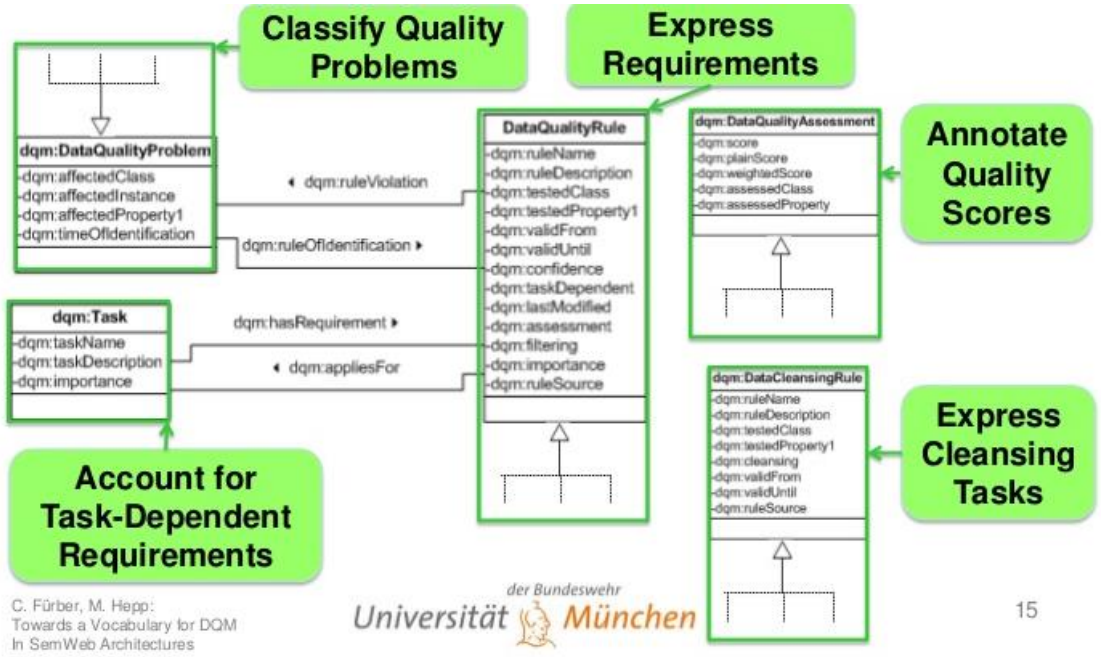


Figura 3 – Representação parcial do modelo UML da ontologia DQM ("Towards a Vocabulary for Data Quality Management in Semantic Web Arch…," 12:58:16 UTC, p. 15)

Os conceitos e respetivas ações relacionadas com qualidade/classificação de limpeza de dados e regras/critérios de avaliação presentes na imagem são os seguintes:

- Regras de qualidade de dados (*Express Requirements*).
- Classificação da qualidade dos dados (*Classify Quality Problems*).
- Avaliação da qualidade dos dados (*Annotate Quality Scores*).
- Regras de limpeza de dados (*Express Cleaning Tasks*).

Para compreender melhor cada conceito apresentado anteriormente, é feita uma descrição sobre cada uma através das classes que o compõem e caso seja necessário é feita uma descrição sobre os seus atributos.

As regras de limpeza foram identificadas através da análise dos tipos de problemas de qualidade de dados a nível das instâncias e encontram-se divididas em classes e propriedades.

As classes que compõem as regras de qualidade de dados são:

- Regra da qualidade dos dados: permite determinar se as instâncias de dados e os valores das propriedades servem para determinar a qualidade de dados.
- Regra da duplicação de instância: permite determinar se uma instância é única no que toca à combinação de determinados valores nas suas propriedades que a permitem identificar como única. Se esses valores se encontrarem em mais entidades, então podemos dizer que as mesmas são duplicadas.
- Regra do valor permitido: permite determinar quais os valores permitidos para uma determinada propriedade da instância.
- Regra da dependência funcional: permite determinar se a combinação de valores permitidos de duas ou mais propriedades são aceitáveis na mesma instância.
- Regra do intervalo de valores permitidos: permite determinar se o valor de uma propriedade numérica está compreendido ente os valores definidos no intervalo.
- Regra do intervalo de valores não permitidos: ao contrário da regra anterior, é definido o intervalo de valores que a propriedade da instância não pode assumir.
- Regra do valor não permitido: permite determinar se uma propriedade numa instância tem um valor que não é permitido.
- Regra da instância desatualizada: permite determinar se uma instância não está na sua ultima versão.
- Regra de caducidade: é uma regra derivada da anterior em que existe uma data que permite determinar a validade da instância.
- Regra de atualização: permite especificar qual é a data limite para a atualização da instância para que a mesma continue válida.
- Regra da propriedade completa: permite verificar se uma instância se encontra completa de acordo com os valores das suas propriedades.
- Regra da propriedade em falta: permite determinar se uma instância se encontra com falta de uma propriedade obrigatória.
- Regra do literal em falta: permite determinar se uma propriedade tem o valor em falta no seu literal em instâncias de determinadas classes.
- Regra da propriedade condicional: serve para atribuir uma propriedade a determinadas instâncias de uma classe de acordo com a condição definida.
- Regra do literal condicional: segue a mesma diretriz da regra anterior, só que aplicada a literais de uma determinada propriedade.
- Regra do valor único: permite especificar se o valor de uma determinada propriedade deve ser único entre as instâncias de uma determinada classe.
- Regra de sintaxe: permite determinar se o valor de uma propriedade segue o padrão definido para a mesma.

As propriedades das classes anteriormente descritas podem ser separadas em propriedades gerais que são aplicáveis a todas as classes e propriedades específicas que são só aplicáveis a um subconjunto de classes.

As propriedades gerais são as seguintes:

- Criador da regra
- Fonte da regra
- Classe testada
- Propriedade testada
- Validade
- Confiança
- Última alteração
- Relevância para avaliação
- Relevância para filtrar informação

Relativamente às propriedades específicas de cada classe, as mesmas existem para auxiliar a aplicação de determinada regra, como por exemplo a propriedade de expressão regular encontra-se associada à classe da regra de sintaxe, para validar o valor de determinada propriedade utilizando uma expressão regular.

Relativamente à classificação da qualidade dos dados, a mesma é feita recorrendo a um conjunto de classes que foram desenhadas a partir das regras de qualidade de dados definidas anteriormente. Existe para cada regra de problema de qualidade de dados, uma classificação de problema de dados.

As classes existentes para classificação da qualidade dos dados são as seguintes:

- Problema na qualidade dos dados: ocorre quando o valor de uma propriedade ou de uma instância não cumpre os requisitos. Pode ser aplicada no caso em que não seja clara a classificação do problema de dados
- Violação de sintaxe
- Violação de dependência funcional
- Valor não permitido
- Elemento em falta
- Propriedade em falta
- Valor em falta
- Instância desatualizada
- Valor fora do intervalo
- Instâncias duplicadas
- Violação da unicidade de valores das propriedades em diferentes instâncias

A avaliação da qualidade dos dados é feita através da definição de cinco dimensões de qualidade dos dados que servem de medição. Em cada dimensão são atribuídos valores numéricos ou categóricos e que servem da base à classificação. A classificação é realizada utilizando as regras de qualidade de dados definidas anteriormente.

A avaliação de qualidade de dados é representada de acordo com as seguintes classes:

- Avaliação da qualidade: é uma classe abstrata que serve para indicar a avaliação da qualidade de dados nas classes ou propriedades do modelo.
- Plenitude: serve para classificar nos caos em que as instâncias possuem um grau de informação que cumpre os requisitos para o âmbito em que as mesmas serão utilizadas.
- Plenitude de uma propriedade: serve para classificar entidades quando os valores de uma determinada propriedade ou a própria entidade não se encontram em falta.
- Plenitude da população: serve para classificar instâncias quando todas as instâncias de um determinado objeto são representadas só numa classe em específico.
- Precisão: serve para classificar uma propriedade quando o seu valor representa exatamente a sintaxe e o significado esperado.
- Precisão sintática: serve para classificar uma propriedade e consiste nos casos descritos no ponto anterior mesmo se o valor da propriedade não obedecer à semântica especificada.
- Precisão semântica: serve para classificar uma propriedade e consiste numa relação correta entre o nome da propriedade e o seu valor a nível semântico.
- Unicidade: serve para classificar instâncias e consiste em não existir classes com propriedades com valores iguais.
- Unicidade ao nível da instância: serve para classificar uma instância e consiste em não existir mais que uma classe para representar uma instância.
- Unicidade da propriedade: serve para classificar uma propriedade dentro de uma determinada classe em que o seu valor tem de ser único numa determinada classe.
- Válidas temporalmente: serve para avaliar se as instâncias cumprem o requisito de serem atualizadas dentro da data estipulada.

O último ponto a ser abordado consiste na definição de regras de limpeza de dados. Essas regras podem ser definidas através das seguintes classes:

- Regra de limpeza
- Regra de combinação estrita de valores
- Regra de remoção de espaços em branco
- Regra da limpeza de propriedade
- Valor atual/novo valor

A Figura 4 permite ter uma noção de algumas das classes que compõe uma regra de limpeza de dados, bem como os seus atributos.

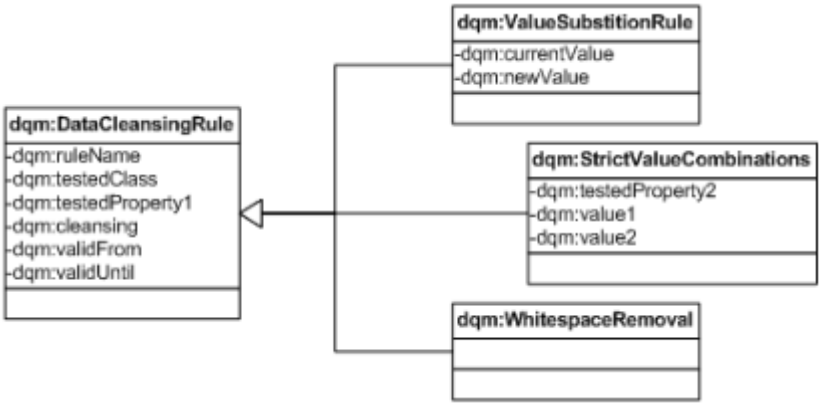


Figura 4 – Representação parcial do modelo UML da regra de limpeza de dados (Fürber and Hepp, 2011, p. 5)

De acordo com o que foi apresentado, podemos inferir que para a utilização do modelo conceptual é necessário definir em primeiro lugar as regras a aplicar no que toca à qualidade dos dados que se pretende obter. O passo seguinte consiste em classificar a regra consoante as classes existentes, seguido da avaliação do seu impacto no repositório de dados e por fim a aplicação da respetiva tarefa de limpeza de dados.

A Figura 5 mostra a relação existente entre a regra de qualidade de dados, o problema de qualidade de dados e tarefas de aplicação das regras. Um determinado problema de qualidade de dados pode ser definido através de uma regra de qualidade de dados que permite providenciar um maior detalhe sobre o mesmo. Uma regra de limpeza de dados depende diretamente de uma tarefa que permite a sua execução, atribuindo um nome e respetivo grau de importância nesse contexto.

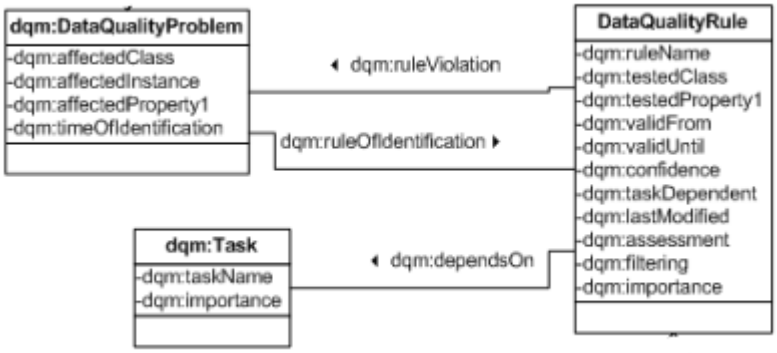


Figura 5 – Representação em UML do modelo de dependência entre regras de qualidade de dados, problemas de qualidade de dados e tarefas de avaliação dos problemas de dados (Fürber and Hepp, 2011, p. 5)

A vantagem do uso deste modelo conceptual consiste na fácil tradução de uma regra de limpeza para uma instrução SPARQL. Isto leva a que as consultas dependam unicamente do vocabulário de limpeza de dados existente na ontologia.

Após a análise deste modelo conceptual podemos concluir que o mesmo é bastante extenso para a classificação/expressão de problemas de qualidade de dados, isto deve-se a uma tentativa de disponibilizar um modelo o mais completo e abrangente possível para lidar com os problemas de qualidade de dados.

2.2.3 Sumário

Após a análise das taxonomias para problemas de qualidade de dados existentes, é possível ter uma noção das diferentes formas em que um problema de qualidade de dados pode ser classificado/definido. Uma clara noção sobre o conceito de limpeza de dados e as abordagens existentes permite uma maior compreensão de parte do problema alvo de estudo, porque só é possível reescrever uma operação de limpeza de dados se existir um conhecimento sobre que tipos existem e em que contextos são aplicados.

2.3 Ferramentas de limpeza de dados

Nesta secção são abordadas ferramentas/abordagens existentes para a realização de operações de limpeza de dados. Para cada item é feita uma descrição sobre a sua arquitetura e em alguns casos também sobre o seu funcionamento.

2.3.1 Data Wrangler

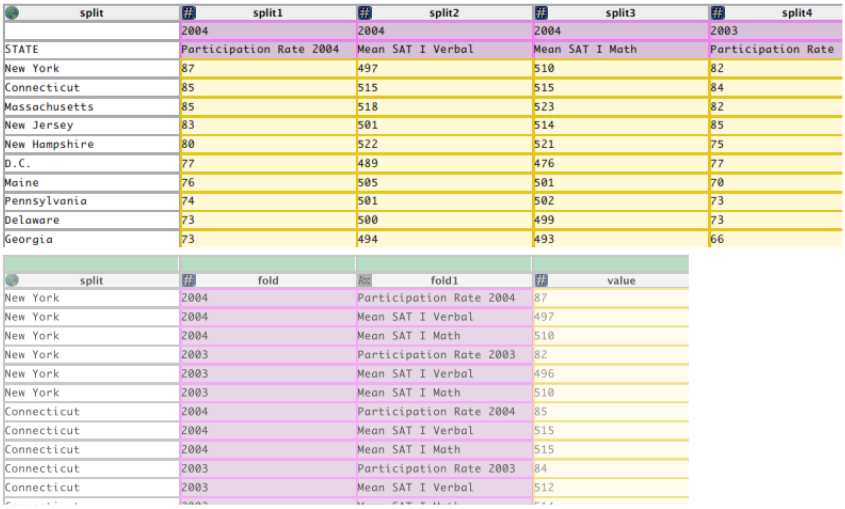
A ferramenta de limpeza de dados Data Wrangler (Kandel et al., 2011) nasceu de um projeto de investigação da universidade *Stanford* (que culminou numa solução comercial Trifacta Wrangler). Esta ferramenta surge com o intuito de simplificar a especificação de operações de limpeza/manipulação dos dados utilizando um mecanismo de inferência a partir de dados importados a partir de ficheiros de texto de vários formatos como por exemplo JSON, CSV, TSV e ficheiros Excel (nos exemplos estudados, foi sempre utilizado o formato CSV). Esta ferramenta possuí uma característica interessante no âmbito da automatização das operações de limpeza de dados, devido ao facto de a mesma analisar o estado atual dos dados e sugerir as operações a executar de acordo com o contexto em que o utilizador se encontra na utilização da aplicação.

A linguagem utilizada para especificar as operações de limpeza de dados é declarativa e tem como base a linguagem de transformação descrita em (Raman and Hellerstein, 2001). As operações seguem uma determinada sintaxe em que conjugam operadores e colunas existentes nos dados utilizados para a aplicação das operações. O exemplo seguinte mostra uma operação de limpeza construída a partir da ferramenta (Kandel et al., 2011, p. 4). Esta operação tem como objetivo substituir os valores em falta para a coluna "*State*", utilizando o valor preenchido na linha que se posiciona imediatamente abaixo através da sua cópia.

```
fill('State').method(COPY).direction(DOWN)
```

Os tipos de transformações existentes na ferramenta que suportam as operações de limpeza são as seguintes:

- Map: permite especificar operações a serem realizadas nas linhas da fonte de dados. Ao selecionar uma linha, é possível elimina-la reduzindo assim o conteúdo da fonte de dados, realizar operações sobre as suas colunas ou expandir as operações por diferentes linhas.
- Lookups e join: permite especificar operações para incorporar dados externos à nossa fonte de dados base. É utilizado num caso prático para determinar o distrito de um determinado código postal existente numa coluna.
- Positional: permite especificar operações para preenchimento de conteúdo em células através das operações *fill* e *lag*. A operação *fill* consiste na obtenção do valor a preencher na célula através dos valores existentes na proximidade da mesma (linha ou coluna). Operador *lag* permite preencher valores das células através da deslocação de valores existentes em outras colunas.
- Reshape: permite especificar operações para manipular a estrutura da fonte de dados utilizando dois operadores designados de *fold* e *unfold*. O operador *fold* serve para condensar várias colunas num número fixo de colunas permitindo classificar os dados de acordo com hierarquias. O operador *unfold* serve para realizar o processo oposto criando colunas de acordo com os valores existentes numa determinada coluna. A Figura 6 mostra o resultado da operação *fold*.



split	split1	split2	split3	split4
	2004	2004	2004	2003
STATE	Participation Rate 2004	Mean SAT I Verbal	Mean SAT I Math	Participation Rate
New York	87	497	510	82
Connecticut	85	515	515	84
Massachusetts	85	518	523	82
New Jersey	83	501	514	85
New Hampshire	80	522	521	75
D.C.	77	489	476	77
Maine	76	505	501	70
Pennsylvania	74	501	502	73
Delaware	73	500	499	73
Georgia	73	494	493	66

split	fold	fold1	value
New York	2004	Participation Rate 2004	87
New York	2004	Mean SAT I Verbal	497
New York	2004	Mean SAT I Math	510
New York	2003	Participation Rate 2003	82
New York	2003	Mean SAT I Verbal	496
New York	2003	Mean SAT I Math	510
Connecticut	2004	Participation Rate 2004	85
Connecticut	2004	Mean SAT I Verbal	515
Connecticut	2004	Mean SAT I Math	515
Connecticut	2003	Participation Rate 2003	84
Connecticut	2003	Mean SAT I Verbal	512

Figura 6 – Data Wrangler operação fold (Kandel et al., 2011, p. 7)

A interface utilizada para a aplicação das operações referidas anteriormente combina a possibilidade de especificação manual, utilização de uma operação das operações sugeridas automaticamente pela ferramenta (com a possibilidade de editar os seus parâmetros) e a utilização de um menu para a escolha de uma operação de acordo com as possíveis operações que podem ser realizadas.

A Figura 7 mostra um exemplo de uma lista de sugestões que está a ser editada manualmente.

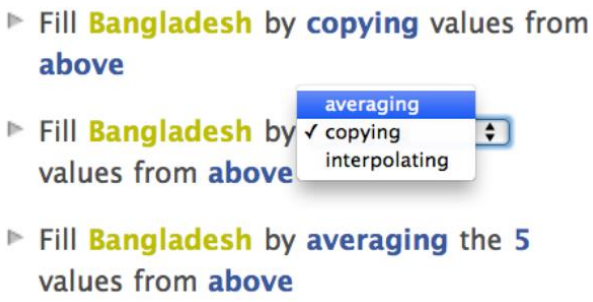
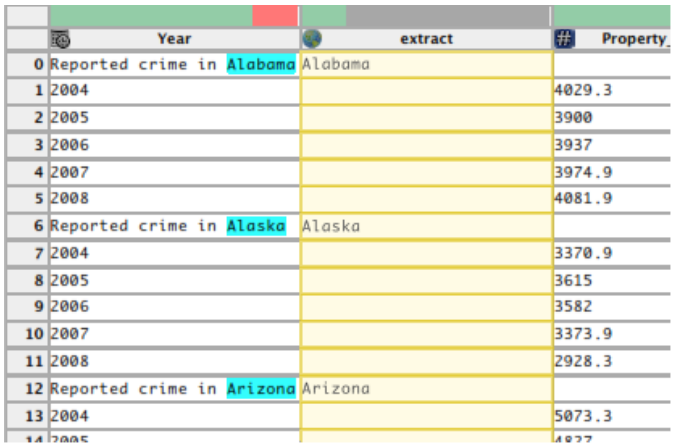


Figura 7 – Data Wrangler lista de operações de limpeza sugeridas

Outra particularidade na interface desta ferramenta consiste na indicação da qualidade de dados numa determinada coluna através de cores e sugestões para eliminar os problemas identificados. A Figura 8 mostra como a as colunas são sinalizadas e como são sugeridas as respetivas correções.



	Year	extract	Property_
0	Reported crime in Alabama	Alabama	
1	2004		4029.3
2	2005		3900
3	2006		3937
4	2007		3974.9
5	2008		4081.9
6	Reported crime in Alaska	Alaska	
7	2004		3370.9
8	2005		3615
9	2006		3582
10	2007		3373.9
11	2008		2928.3
12	Reported crime in Arizona	Arizona	
13	2004		5073.3

Figura 8 – Data Wrangler indicador de qualidade de dados

Cada coluna é sinalizada através de um conjunto de cores tendo cada uma um respetivo significado:

- Verde: percentagem de valores sem problemas de qualidades de dados;
- Amarelo: registos que não obedeçam ao tipo de dados detetado para a coluna em questão;
- Vermelho: registos com formato errado, por exemplo números por extenso onde existe valores numéricos;
- Cinza: registos com valores em falta, por exemplo: nulos ou em branco.

Para além das funcionalidades referidas anteriormente, a ferramenta utiliza um mecanismo de inferência para dar sugestões, sendo esse mecanismo baseado num conjunto de critérios para a construção e ordenação das mesmas. Os critérios utilizados para inferir as sugestões são divididos em três fases:

- Utilização de operações realizadas previamente pelos utilizadores: a ferramenta agrupa as transformações realizadas em três tipos (linha, coluna e texto) e para cada tipo

enumera os possíveis parâmetros que podem ser utilizados (fixos ou baseados em conteúdo selecionado pelo utilizador no preciso momento da utilização).

- Utilização dos parâmetros inferidos: de acordo com o conteúdo selecionado pelo utilizador ou dos possíveis valores existentes na coluna em questão, a ferramenta sugere um conjunto de operações.
- Ordenação dos resultados: existem cinco critérios de ordenação que começam pelo tipo de transformação, o segundo critério é aplicado de acordo com o grau de dificuldade de aplicação da operação (dificuldade em especificar os seus parâmetros), o terceiro é determinado pela frequência da utilização das operações por parte do utilizador, depois de ordenado pela frequência de utilização o quarto critério consiste na utilização da complexidade de uma operação que é obtida pelo número de cláusulas (exemplo: a=5 e b=6 equivale a complexidade é 2) e/ou pelo número de carateres numa expressão regular, por último é feito um rácio entre estes atributos para efetuar a sua classificação, permitindo assim obter as operações de limpeza a sugerir, sendo que uma operação sugerida não pode estar presente em mais de um terço dos resultados da lista de sugestões.

Para além de sugerir possíveis operações de limpeza, a ferramenta possibilita uma pré-visualização do resultado das mesmas, mostrando o valor anterior e o novo valor das células afetadas. Outra funcionalidade consiste na possibilidade de aplicação de operações anteriormente especificadas, devido ao facto de ser gerado um ficheiro com as operações selecionadas para posterior execução. Mesmo assim, estas operações só podem ser aplicadas num repositório de dados que tenha exatamente o mesmo modelo/esquema que serviu para a definição das mesmas, sendo uma das limitações referidas sobre as ferramentas de limpeza de dados que motivou a elaboração deste trabalho.

2.3.2 Sistema de limpeza de dados baseado em regras

A abordagem descrita em (Hao and Xing-Chun, 2008) consiste na conceptualização de uma solução de limpeza de dados baseada em regras para a especificação de operações de limpeza de dados, sendo as mesmas executadas através da aplicação de uma estratégia. Para descrever esta abordagem é usado como ponto de partida a sua arquitetura, permitindo ter uma ideia global do seu funcionamento, passando depois para uma descrição detalhada de cada componente.

A Figura 9 mostra de que forma a solução foi desenhada. Podemos observar que existe um módulo para gestão das regras que se encontram armazenadas e um processo de limpeza de dados. A partir das regras armazenadas é feita a escolha de uma regra para aplicar. Essa regra por sua vez está enquadrada num processo de limpeza de dados. Esta arquitetura utiliza uma interface para aceder aos repositórios de dados, permitindo assim o acesso a vários.

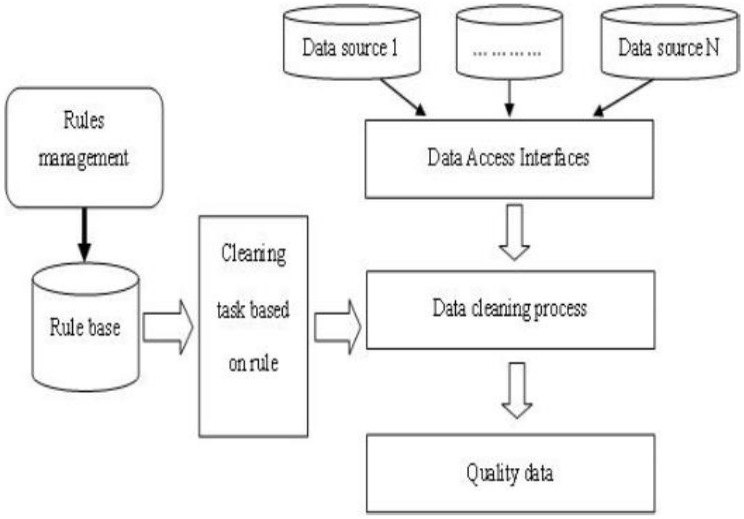


Figura 9 – Arquitetura do sistema de limpeza de dados baseado em regras (Hao and Xing-Chun, 2008, p. 2)

Uma regra de limpeza de dados é baseada em dois tipos de informação que são os requisitos e as respetivas operações de limpeza de dados (regras). Os requisitos estão relacionados com informação necessária para executar a operação e um processo para ser executado precisa de determinar que operações lhe estão associadas. Apesar de existir uma relação entre requisitos e as operações de limpeza de dados (um requisito pode abranger várias operações de limpeza e uma operação possuir os seus próprios requisitos para ser executada), os mesmos precisam de ser definidos de forma independente. Por exemplo, para executar um determinado operação e seu conjunto de operações num determinado tipo de repositório de dados, pode ser necessário que sejam cumpridos um conjunto de requisitos para a sua execução, no entanto para outro tipo de repositórios, para executar o mesmo processo e suas operações poderá ser necessário associar outro tipo de requisitos. Em termos práticos, se o objetivo é aplicar uma operação para corrigir valores em falta no atributo nome completo dos clientes de clientes no repositório de dados A, é preciso que exista a tabela cliente e a coluna nome completo, no repositório de dados B é preciso que exista a tabela utilizadores e que existam as colunas primeiro e último nome.

Como foi referido anteriormente, a regra (ou operação) é composta na sua base por um conjunto de requisitos e algoritmos que se encontram associados. Para a realização dessa associação foi desenhada uma funcionalidade baseada numa estratégia que consiste em armazenar os requisitos, os algoritmos base e a respetiva relação entre ambos. A Figura 10 mostra a interação entre o componente de gestão de regras e o componente que serve de base à construção das regras de limpeza.

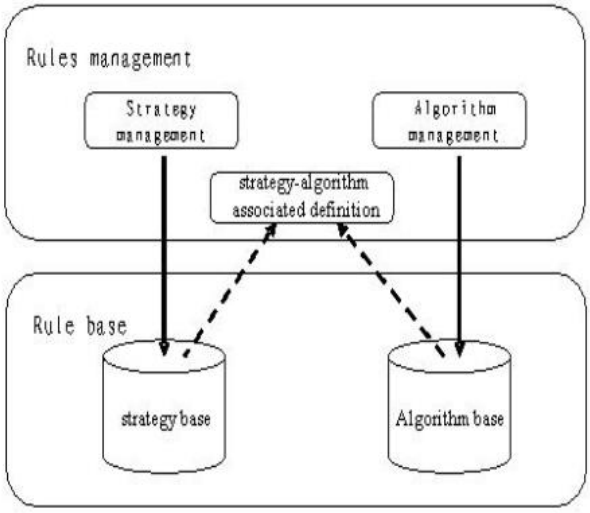


Figura 10 – Definição da base da regra de limpeza (Hao and Xing-Chun, 2008, p. 2)

Relativamente às estratégias que se encontram armazenadas, elas possuem o seguinte formato (Hao and Xing-Chun, 2008, p. 2) :

- Identificador da estratégia: identificador único da estratégia.
- Executor de eventos de limpeza: através da construção de condições que permitem determinar se é necessário ou não lançar operações de limpeza.
- Tipo de repositório de dados fonte: qual é o repositório de dados onde vão decorrer as operações de limpeza.
- Nome da tabela: qual a tabela onde vão ocorrer as operações de limpeza.
- Nome do campo: nome do campo da tabela referida anteriormente que se encontra relacionado com a operação de limpeza.
- Descrição da estratégia: serve para descrever a estratégia de limpeza, permitindo assim uma fácil leitura na escolha da mesma no momento da pesquisa.

Depois de apresentadas as estratégias que compõem as regras, é necessário descrever os algoritmos. Eles são definidos através de um nome que serve como identificador único e o corpo algoritmo onde são definidas as instruções a executar. A declaração do algoritmo e o seu corpo do algoritmo seguem o seguinte formato:

```
Function NomeDoAlgoritmo (Tipo Base de dados, Nome da tabela, Nome do
campo,Eventos de limpeza){
( Instruções)
 Return (sucesso/insucesso da operação)
}
```

O tipo de repositório de dados serve para escolher quais os operadores específicos a aplicar de acordo com a tecnologia do repositório de dados alvo, o nome da tabela e respetivo campo serve para indicar onde vai ser aplicada. As instruções são criadas através de uma sintaxe condicional (Se Condição Então Executar operação).

Depois de definidas as regras de limpeza, as estratégias e os algoritmos, o passo final consiste na implementação do processo de limpeza.

A implementação do processo de limpeza de dados segue as seguintes fases que são a construção do ficheiro para a sua execução, verificação do resultado da operação anterior, compilação do ficheiro de execução das operações e por fim a sua execução.

As principais conclusões do autor, apontam como mais-valias uma arquitetura simples com fronteiras bem definidas entre componentes e uma facilidade em interagir e reutilizar o conteúdo já existente com regras e algoritmos.

As principais conclusões obtidas no estudo desta abordagem permitem perceber que existe um esforço para reutilizar as operações de limpeza entre modelos/esquemas, no entanto não é possível utiliza-las entre diferentes tipos de repositórios de dados, visto que só são suportados os do tipo relacional. Esta limitação vem confirmar o que foi dito na contextualização do problema que serve de base para a elaboração deste trabalho.

2.3.3 SmartClean

No estudo desta ferramenta é realizada uma breve abordagem dos seus conceitos, como também da sintaxe utilizada. A partir da apresentação da sintaxe são descritos de uma forma breve os parâmetros contidos na mesma, sendo que em alguns casos essa descrição é acompanhada com um exemplo prático da sua aplicação, permitindo assim perceber de que forma podem ser aplicados.

As instruções suportadas pela ferramenta são baseadas numa linguagem declarativa, utilizando a gramática BNF estendida (cf. secção 2.4) para a especificação de operações. Algumas destas operações para além de possuir uma sintaxe própria, possuem algumas particulares em comum com a estrutura da linguagem SQL.

A sintaxe para a definição de operações de deteção de problemas de qualidade de dados é iniciada sempre pela palavra “DETECT” seguida de um conjunto de parâmetros que definem o objetivo da operação a executar. O exemplo seguinte mostra de uma forma completa a sintaxe que pode ser utilizada para elaboração de uma operação de deteção de limpeza:

```
OperaçãoDetecção → DETECT ProblemaQualidadeDados NívelOperação
                   NasColunas DasTabelas DasBasesDados
                   UsandoColunas UsandoChavesOrdenação
                   UsandoLimiar UsandoOperações
                   DependentesDe
                   DasTabelasReferência DasBasesDadosReferência
                   UsandoDomínio UsandoCondição
                   AgupandoPor
                   UsandoDicionário UsandoMétrica
```

Como forma de entender a sintaxe das operações de limpeza no SmartClean é dado um exemplo prático utilizando uma das operações mais simples de definir na ferramenta. Esta operação consiste na deteção de valores em falta numa determinada coluna, sendo composta pelo tipo de operação “MISSING-VALUE”, o nome da coluna onde vai ser executada que é designada por

"CodigoConclusao", a tabela a que a coluna pertence "Conclusoes" que se encontra na base de dados "BDD".

```
DETECT MISSING-VALUE ON CodigoConclusao FROM Conclusoes OF BDD
```

Depois de apresentada a sintaxe das operações de deteção e da sua contextualização através de um exemplo prático, é necessário descrever cada um dos parâmetros que compõe a mesma com o objetivo de perceber quais as suas potencialidades na deteção de problemas de qualidade de dados.

O primeiro parâmetro a ser descrito é designado por "ProblemaQualidadeDados" que tem como objetivo especificar qual o tipo de operação de deteção que vai ser executada.

Cada problema de qualidade de dados pode ser instanciado através de um conjunto níveis (de acordo com a sua taxonomia) através do parâmetro "NívelOperação". Este parâmetro serve para especificar de que forma a mesma vai ser aplicada relativamente ao(s) esquema(s) definidos na operação.

Depois de especificar os dois parâmetros inicias, o passo seguinte consiste em identificar as fontes dos dados que vão suportar a execução da operação.

Em primeiro lugar são definidas as colunas alvo da operação começando com a palavra "ON", seguido do identificador da tabela onde está a coluna (AliasTabelaDaColuna) e o nome definido pelo utilizador, separando cada coluna por vírgulas.

Depois de definidas as colunas, o passo seguinte consiste em definir de que tabelas as mesmas são originárias, começando com a palavra "FROM" seguido do identificador da tabela e o nome definido pelo utilizador (AliasTabela), separando cada tabela por vírgulas.

Por fim, o último passo para definir a fonte de dados consiste em especificar quais são as bases de dados. Para especificar as bases de dados é necessário começar com a palavra "OF" seguido dos identificadores das bases de dados, separando cada tabela por vírgulas.

Como exemplo prático é dada a seguinte operação de violação de integridade ao nível das múltiplas relações que utiliza duas tabelas designadas por "Dadores" e "Analises" de uma determinada base de dados e algumas colunas das respetivas tabelas (a restante parte da operação é omitida por não ser relevante para este exemplo, sendo abordada mais à frente).

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION

AT LEVEL OF MULTIPLE RELATIONS

ON T1.Data, T1.Dador, T1.NumColh, T2.Data, T2.Dador, T2.NumColh //Colunas

FROM Dadores T1, Analises T2 //Tabelas

OF BDD //Base de dados

//restante parte da operação…
```

De acordo com a sintaxe apresentada que serve para descrever uma operação na integra, os parâmetros que se seguem à especificação da fonte de dados estão relacionados com o tipo de

operação que se pretende executar. Existem parâmetros que são fáceis de especificar devido à simplicidade da instrução a que pertencem, como também existem parâmetros complexos que necessitam de um conhecimento técnico para a sua especificação.

Para especificar operações que utilizem as colunas auxiliares como parâmetro é necessário adicionar à operação de deteção a palavra "USING", seguido dos identificadores das colunas. Este tipo de parâmetros é utilizado em operações de deteção de duplicados. A seguinte operação mostra como é feita a sua utilização para a deteção de códigos postais duplicados utilizando as colunas "CodPostal" e "LocPostal".

```
DETECT DUPLICATE-TUPLES FROM CodigosPostais T1, CodigosPostais T2 OF BDD
USING CodPostal, LocPostal WHERE T1.CodPostal = T2.CodPostal AND
T1.LocPostal = T2.LocPostal
```

Existem outras formas de realizar a deteção de duplicados usando a ferramenta, uma delas é usando o parâmetro "UsandoChavesOrdenação" em que é utilizado um critério de ordenação específico dos registos. A instrução inicia com a especificação das chaves de ordenação que começa com a palavra "SORT KEY", seguido do número do índice que serve como critério de ordenação, o texto ou parte do mesmo a ser incluído como critério de ordenação e o identificador da coluna onde o mesmo se encontra. Este tipo de técnica permite obter uma chave composta capaz de ser utilizada em algoritmos de verificação de semelhanças na vizinhança como por exemplo o *sorted neighborhood* (Kejriwal and Miranker, 2015).

Devido a questões de performance na execução deste tipo de operações, é necessário adicionar um parâmetro designado por "WINDOW SIZE". Este parâmetro define o número de registos a ser utilizado no processo de comparação.

Para operações que validem um conjunto de valores válidos para um atributo, é necessário especificar o parâmetro de definição de domínio através das palavras "USING DOMAIN", seguido do identificador do domínio. Como exemplo é dada a seguinte operação que permite da deteção de violação de domínio para códigos postais, utilizando um conjunto de valores válidos designado por "Codigos_Postais".

```
DETECT DOMAIN-VIOLATION ON CodPostal FROM CodigosPostais OF BDD USING
DOMAIN Codigos_Postais
```

Caso a operação necessite de uma condição de baixo nível de complexidade para a sua execução, a mesma pode ser feita utilizando uma sintaxe baseada em SQL. O seguinte excerto do parâmetro condicional da operação de violação de integridade (apresentada num exemplo anterior) permite perceber qual o formato utlizado. A definição da condição é iniciada pela palavra "*WHERE*", seguida de um conjunto de expressões intercaladas por operadores que permitem fazer a conjunção de todos os elementos da operação.

```
WHERE T1.Data=T2.Data AND T1.Dador=T2.Dador AND T1.NumColh<>T2.NumColh
```

Em alguns casos é necessário agrupar os dados para garantir por exemplo o mesmo nível de granularidade entre os registos de diferentes fontes. Este parâmetro pode ser definido utilizando a sintaxe SQL para agrupar as consultas. Como parâmetro auxiliar, é possível aplicar uma condição à forma como os dados são agregados. Isso pode ser feito utilizando a sintaxe da instrução "HAVING" do SQL.

Para operações de deteção de erros ortográficos é necessário a inclusão de um dicionário para consulta de palavras, isso leva à inclusão de um parâmetro na operação em que a instrução começa por "USING DICTIONARY" e o respetivo identificador do dicionário.

O último parâmetro a ser descrito consiste na inclusão de uma métrica que serve basicamente para avaliar semelhanças entre valores através de funções de distância. Como exemplo da aplicação destes dois últimos parâmetros descritos anteriormente, é mostrada uma operação de deteção de erros de ortografia na coluna "DesignacaoConclusao", da tabela "Conclusoes", tendo como parâmetro o dicionário "DicPortugues" e utilizando a métrica de similaridade entre palavras "Jaro-Winkler".

```
DETECT MISSPELLING-ERRORS ON DesignacaoConclusao FROM Conclusoes OF BDD
USING DICTIONARY DicPortugues USING METRIC Jaro-Winkler
```

A partir da análise efetuada a esta ferramenta, podemos concluir que a mesma suporta a especificação de operações entre diferentes modelos de dados, possui uma linguagem de fácil compreensão para a grande parte dos utilizadores, o que possibilita o seu uso por pessoas que não são peritas na matéria. No entanto as operações encontram-se acopladas à sintaxe definida pela ferramenta/modelo de dados para o qual foram definidas, o que impossibilita a sua reutilização.

2.3.4 Sumário

Tendo em conta a inexistência de automatização e adaptação dos processos de limpeza de dados, existem três conclusões que podem ser retiradas:

1. As ferramentas de limpeza de dados necessitam de uma especificação manual das operações de deteção e limpeza por parte do especialista de domínio.
2. A maioria das ferramentas existentes para limpeza de dados são específicas para um determinado modelo, não podendo ser aplicadas a repositórios de dados com modelos diferentes, devido ao facto de não existirem correspondências entre modelos apesar de os mesmos poderem pertencer ao mesmo domínio.
3. Existem operações de limpeza de dados que são transversais ao domínio da aplicação, podendo ser aplicadas a qualquer repositório de dados independentemente do seu modelo.

Devido às limitações referidas anteriormente, surge a necessidade da existência de uma solução que permita a especificação de operações de deteção e limpeza de dados que seja compatível com diferentes modelos e independente da fonte de dados onde a mesma seria aplicada e que seja possível a definição de operações sem que as mesmas requeiram o conhecimento de um especialista.

2.4 Gramáticas

Nesta secção é abordado o conceito de gramáticas com o objetivo de perceber como são definidas as regras de sintaxe para uma determinada linguagem e de que forma as mesmas podem ser utilizadas num processo de reescrita.

Em primeiro lugar é definido o conceito de gramática através da descrição das duas dimensões que a definem, de que forma pode ser estruturada e quais os formalismos que existem para a sua definição.

2.4.1 Definição

Nesta secção é definido o conceito de gramática na área das ciências da computação. Em primeiro lugar são apresentados os dois conceitos chave que estão na base da sua definição, passando para a descrição formal da sua estrutura, de que forma o seu conteúdo é processado dando um exemplo prático e por último um breve resumo de todo o processo que envolve a sua utilização.

Uma gramática é definida em duas vertentes:

- Sintática: define a estrutura que as regras devem obedecer de acordo com gramática e à ordem dos seus símbolos. Um exemplo desse tipo de regras pode ser encontrado por exemplo na definição de uma operação de atribuição de um valor a uma variável. A primeira parte da instrução tem de ser o identificador da variável, seguido do caracter "=" e o respetivo valor que se pretende atribuir (ID = VAL).
- Semântica: define o significado que cada símbolo tem na gramática, permitindo assim uma correta contextualização do mesmo na construção de expressões válidas. Recorrendo ao exemplo anterior, o símbolo ID corresponde a um nome de uma variável, logo só pode ser aplicada em contextos em que se pretenda referir uma variável e não outra coisa qualquer como uma função por exemplo.

De acordo com ("Programming Languages: Syntax Description and Parsing - 15-syntax-parsing.pdf," n.d., p. 8), uma gramática formalmente é definida pela seguinte expressão:

$$G = (S, R, N, T)$$

- Símbolo inicial (S): para construir uma gramática é necessário definir qual é o seu símbolo inicial. A partir dele é definido quais as regras a serem processadas para determinar o seu valor.
- Símbolos terminais (T): são aqueles em que seu valor pode ser determinado diretamente sem a invocação de mais regras. Um exemplo disso pode ser apresentado numa gramática em que existe um símbolo designado por dígito em que o seu valor é uma expressão regular que define um algarismo compreendido entre 0 e 9.
- Símbolos não terminais (N): são aqueles em que o seu valor não pode ser determinado diretamente, necessitando do processamento do conjunto de regras que o definem para que seja possível determinar o seu valor. Um exemplo disso é uma regra que permite especificar uma expressão matemática que é constituída por um número e um conjunto de uma ou mais operações em que cada operação é constituída por um operador e/ou número ou por um operador e uma operação. Esta situação leva a que sejam determinadas todas as operações de forma recursiva até que o seu valor seja determinado através de símbolos terminais. Para este tipo de símbolos é dada a designação de *token*.

- Regras (R): servem para determinar um valor de parte da expressão utilizando símbolos terminais e não terminais.

Na maioria dos casos, para definir uma linguagem é utilizado o formalismo de Backus-Naur (BNF) (Garshol, 2003, p. 1) que é nada mais do que uma metalinguagem usada para descrever outras linguagens, sendo baseado em gramáticas de contexto livre (Gallier and Hicks, 2014). Este formalismo consiste em definir uma gramática como um conjunto de regras que são constituídas por:

```
lado esquerdo da regra  ::= lado direito da regra
```

O lado esquerdo da regra consiste na exposição do símbolo inicial e dos símbolos não terminais da gramática. No lado direito da regra é definido o conjunto de símbolos (terminais e/ou não terminais) que permitem atribuir o valor final ao símbolo que se encontra do lado esquerdo da regra.

Para mostrar na prática como este formalismo pode ser aplicado foi escolhido como exemplo uma gramática que permite processar declaração de variáveis na linguagem Pascal.

```
<vardecl>     ::= var <vardecllist> ;
<vardecllist> ::= <varandtype> { ; <varandtype> }
<varandtype>  ::= <ident> { , <ident> } : <typespec>
<ident>       ::= <letter> { <idchar> }
<idchar>      ::= <letter>  |  <digit>  |  _
<typespec>    ::= integer | float | double | char
```

Como exemplo prático para descrever o funcionamento desta gramática é apresentada a seguinte expressão:

```
var i : integer;
```

A Figura 11 mostra o resultado da análise da expressão num formato de árvore, onde é possível observar a derivação dos em símbolos terminais e não terminais ou caso seja necessário em outras regras.

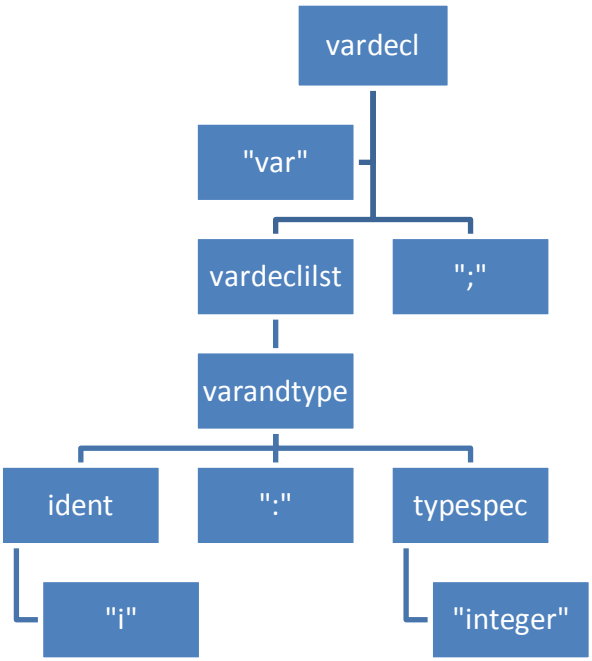


Figura 11 – Exemplo do resultado da análise de expressão no formato de árvore

Para além da notação descrita anteriormente, existem outras notações para gramáticas de contexto livre que derivam do formalismo BNF, como por exemplo o *Extended* BNF (Garshol, 2003, p. 4).

Na secção seguinte são apresentadas as soluções existentes ao nível das gramáticas através de uma breve descrição, seguido de um comparativo entre as soluções abordadas.

2.4.2 Soluções existentes

Nesta secção são abordadas algumas das soluções existentes ao nível da implementação de gramáticas. Cada solução possui um conjunto de particularidades, por isso é necessário escolher um conjunto de atributos em comum para que seja possível ter um ponto de comparação entre elas. Neste contexto, as soluções vão ser avaliadas de acordo com os seguintes atributos (“Comparison of Parser Generators - Deterministic Context-free Languages,” n.d.):

- Algoritmo de análise: define qual é o método utilizado para processar as regras da gramática.
- Notação: define o formato em que as regras devem ser definidas na ferramenta. A maioria das ferramentas utiliza como notação base o *Extended* BNF.
- Linguagem de saída: permite definir em que linguagem vai ser gerado o código que serve de implementação à gramática definida.
- Lexer: se o mesmo é gerado, carregado de fontes externas ou até mesmo se não existe suporte para o mesmo.
- Plataforma de desenvolvimento: permite determinar em que ambiente é possível utilizar a ferramenta.

Depois de apresentar o conjunto de atributos que são tidos como base para a análise das ferramentas a apresentar, as secções seguintes consistem na respetiva apresentação de duas ferramentas bastante utilizadas neste contexto. Estas ferramentas foram escolhidas com o objetivo de por um lado mostrar abordagens mais recentes (ANTLR), como por um lado mostrar o seu ponto de partida (Bison).

2.4.2.1 ANTLR

ANother Tool For Language Recognition (ANTLR) consiste numa ferramenta que permite a tradução de texto a partir de árvores de análise de expressões. A sua utilização em larga escala está associada à construção de linguagens de programação.

Esta ferramenta permite o suporte para gerar código para as múltiplas linguagens como Java, C#, C, Phyton, entre outras.

Desde a sua criação, esta ferramenta tem evoluído de uma forma constante e a prova disso são as alterações entre a versão 3 e a versão 4, sendo uma as principais alterações o algoritmo de análise que passou do tipo LL(*) para ALL(*). Esta alteração no algoritmo de análise consiste numa melhoria a nível de eficiência, como também ao nível do processamento das múltiplas opções que podem ocorrer no processamento de uma árvore de análise de expressões.

Para descrever um exemplo prático de utilização da ferramenta é utilizada uma gramática base, sendo a linguagem de saída para compilação o Java.

O primeiro passo consiste em definir a gramática num ficheiro com a extensão “g4”. O ficheiro para além das regras que definem a gramática possui um conjunto de parâmetros inicias:

```
gramar NomeGramática;
options{
//lista de opções da ferramenta
}

tokens{
 //definição de símbolos que não se encontram definidos diretamente na
gramática
}

@header{
 //secção normalmente utilizada para importar classes para o código
gerado na gramática
}



@rulecatch{
 //secção utilizada para tratamento de erros na gramática
}
```

```
@menbers{
 //secção utilizada para declarar variáveis e métodos a ser utilizados
nas regras.
}

/*regras…*/
```

Depois de definir o cabeçalho do ficheiro da gramática, o restante conteúdo do ficheiro consiste nas regras da gramática que são definidas no formato EBNF como é possível observar pelo seguinte excerto de código.

```
init : '{' value (',' value)* '}' ;
value : init | INT;
```

Cada regra pode ser composta por referências a outras regras ou a símbolos terminais da gramática. Um símbolo terminal pode ser definido em letra maiúscula como é possível observar neste exemplo.

```
INT : [0-9]+ ;
```

Depois de definida a gramática, a mesma é utilizada para gerar código de forma automática que permita a sua instanciação através de invocação das classes geradas.

A Figura 12 mostra um exemplo das classes geradas a partir de uma gramática designada por "ArrayInit"("Antlr.pdf," n.d., p. 23):

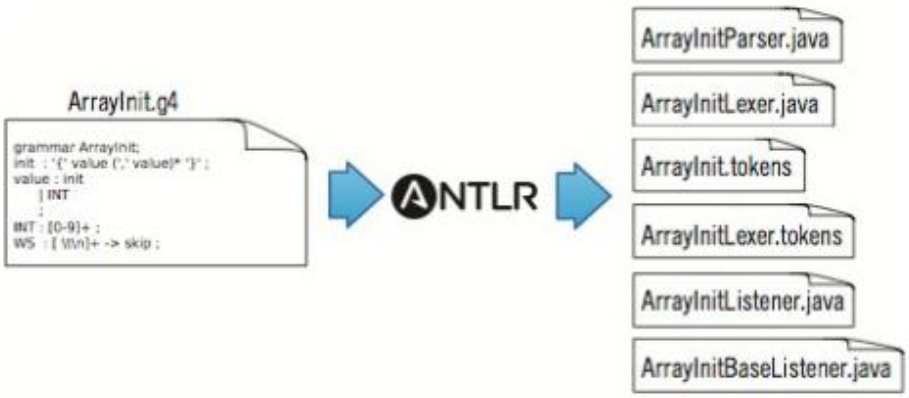


Figura 12 – Resultado da compilação da gramática ANTLR ("Antlr.pdf," n.d., p. 23)

ArrayInitParser.java: contem a definição da classe responsável pela análise das expressões ao nível das regras existentes na gramática.

ArrayInitLexer.java: contem a definição da classe responsável pela análise das expressões ao nível dos símbolos existentes na gramática.

ArrayInit.tokens: contem os *tokens* declarados na gramática e respetivos métodos para sua deteção e processamento.

ArrayInitListener.java: uma interface que permite implementar determinados eventos a serem executados durante a análise de uma expressão.

ArrayInitBaseListener.java: classe que possui a assinatura dos eventos referidos anteriormente sem nenhuma implementação.

Esta flexibilidade de interação com o processo de análise da gramática através da captura de eventos ou de inclusão de código a ser executado no decorrer do processamento de uma determinada regra, leva a que seja possível a implementação de outro tipo de funcionalidades como por exemplo processar o resultado de expressões matemáticas ou gerar instruções compatíveis com a gramática definida.

As principais vantagens no uso desta ferramenta prendem-se com o facto de a mesma permitir a integração com várias ferramentas de desenvolvimento, automatização da compilação da gramática, visualização da árvore de análise gerada e a possibilidade de integrar em projetos implementados em diferentes linguagens.

2.4.2.2 Bison

Esta ferramenta é uma das mais clássicas de análise de expressões que assenta numa estratégia de análise ascendente. Como suporte de linguagens de saída, esta ferramenta suporta unicamente as linguagens C e C++.

Um dos pontos fortes desta ferramenta é o suporte a vários tipos de algoritmos de análise, tais como:

- LR
- LALR
- GLR
- IELR

Para utilizar esta ferramenta, o primeiro passo consiste em definir a gramática num ficheiro com a extensão "y". A nível de sintaxe em que a gramática é definida, a mesma não segue nenhuma convenção em concreto (possui uma sintaxe própria). O ficheiro da gramática está estruturado da seguinte forma:

```
%{
 //declarações de bibliotecas e assinaturas de funções a utilizar na
gramática
%}



//permite definir o tipo de dados dos tokens
%union {
int val;
char* str;
}
```

```
//Declarações para a ferramenta Bison:
//token da gramática (NUM) e respetivo tipo (val), ex: %token <val> NUM
//precedência de operadores (%left ou %right), seguido do tipo que é
opcional e por fim símbolo, ex: %left "<="

%%
Regras da gramática
%%

//Código C adicional, como por exemplo tratamento de erros da gramática
void yyerror (char *const s) {
fprintf(stderr, "ERROR on line %d\n", lineno);
}
```

Relativamente ao formato das regras da gramática, as mesmas possuem um nome do lado esquerdo em letras minúsculas, separado por ":" e a respetiva definição da regra do lado direito. Cada regra pode conter referências a outras regras, o operador "ou" (|), *tokens* e código C entre chavetas.

No exemplo seguinte é definida a regra "exp" que tem como objetivo o processamento de operações aritméticas. As mesmas são executadas através do código existente entre chavetas e o retorno do respetivo resultado é realizado após o processamento da regra.

```
exp: NUM { $$ = $1; }
| exp '-' exp { $$ = $1 - $3; }
| exp '*' exp { $$ = $1 * $3; }
| '-' exp %prec NEG{ $$ = -$2 }
```

Depois de definidas todas as regras dando origem à gramática, o passo seguinte consiste na compilação da mesma para gerar o código C ou C++ para sua utilização. Este processo consiste na ferramenta ler o ficheiro da gramática e gerar um ficheiro em C, que por sua vez o compilador de C lê esse ficheiro e transforma-o num executável para ser utilizado no processamento de expressões. Quando é utilizado esse executável para processar uma determinada expressão é construída a respetiva árvore de análise.

A Figura 13 mostra a sequência do processo de compilação da gramática utilizando o Bison.

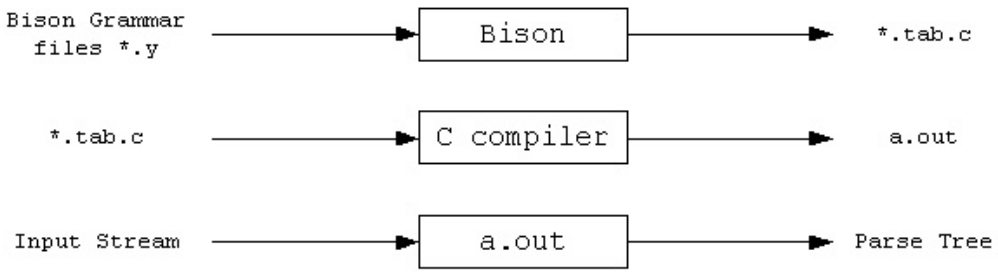


Figura 13 – Resultado da compilação da gramática utilizando o BISON

Relativamente ao nível de integração com ferramentas de desenvolvimento, esta ferramenta não possui nenhum suporte. O facto de suportar diferentes algoritmos de análise e de não precisar de nenhum ambiente de execução em particular, tornam esta solução de difícil utilização.

2.4.2.3 Comparação

Relativamente às duas soluções descritas anteriormente, existem um conjunto de pontos fortes e fracos que devem ser tidos em conta no momento de optar pela sua utilização num projeto.

O ANTLR é uma ferramenta que possui um suporte para integração com tecnologias mais recentes, pelo facto de permitir a criação automática de código em linguagens como Java ou C#. Para além disso, existe uma aposta em algoritmos específicos de análise de expressões, sendo este um fator diferenciador das restantes ferramentas. Outra mais valia desta ferramenta está na integração com as ferramentas de desenvolvimento normalmente usadas nas linguagens que suporta.

O Bison tem como mais valia o suporte de vários algoritmos de análise e uma simplicidade na sua sintaxe. Relativamente ao suporte a diferentes linguagens, esta ferramenta tem neste campo um ponto fraco que faz com que só seja compatível com projetos que utilizem como linguagem o C ou o C++. Outro fator desfavorável está relacionado com a falta de integração com uma ferramenta de desenvolvimento, sendo deste modo vista como uma unidade externa ao projeto onde está inserida.

2.4.3 Sumário

A partir da apresentação do conceito de gramáticas associado à área das ciências da computação e da análise destas soluções, é possível perceber como é definida a sintaxe de uma linguagem, como é o seu funcionamento das ferramentas existentes para a implementação de uma gramática, em que situações podem ser aplicadas e que funcionalidades oferecem num cenário de integração com a solução que pretendemos desenvolver.

2.5 Reescrita de consultas

O conceito de reescrita de consultas enquadra-se no âmbito dos repositórios de dados e consiste basicamente em processar uma dada consulta com o objetivo de a reformular numa nova consulta, como por exemplo reescrever uma consulta que tinha sido escrita para um determinado modelo de dados para outro modelo diferente.

A sua utilização encontra-se normalmente associada a otimização de consultas, armazéns de dados e integração de informação dispersa em diferentes repositórios. Quando é referida a situação de diferentes repositórios, a mesma pode significar diferentes linguagens de consulta ou diferentes modelos de dados. A Figura 14 mostra os quatro cenários possíveis tendo em conta as variáveis referidas anteriormente (linguagem e modelo).

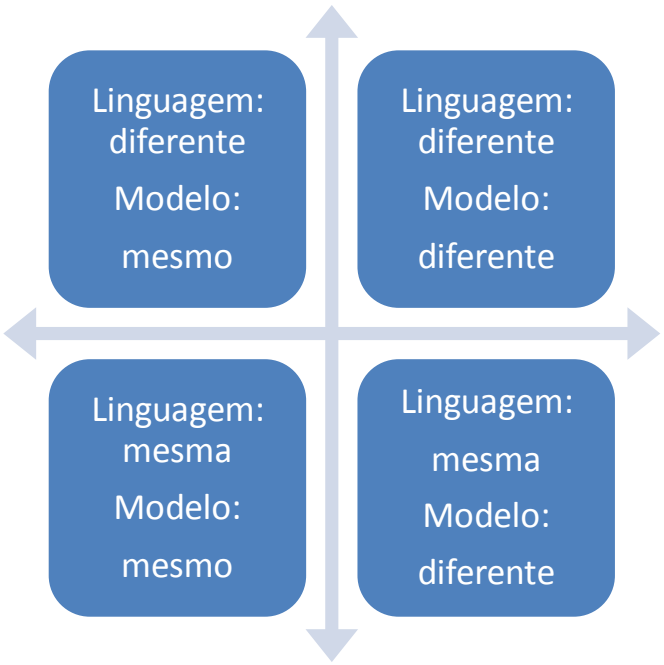


Figura 14 – Possíveis cenários de reescrita

(Num pequeno a parte, o caso abordado neste documento vai de encontro às situações em que temos a necessidade de reescrita de operações entre diferentes linguagens e/ou modelos).

Para a analisar este conceito são descritas as abordagens existentes que consistem na especificação de arquiteturas (mediadores, P2P) e na reescrita direta entre linguagens. Na arquitetura baseada em mediadores é realizada a descrição das várias abordagens existentes utilizando vistas (GAV, LAV, GLAV, BAV). Na reescrita entre linguagens é abordado o caso especifico de reescrita entre o SPARQL e o SQL.

2.5.1 Arquiteturas de reescrita de consultas

Nesta secção são apresentadas as abordagens existentes para o conceito de reescrita de consultas. De acordo com a Figura 14 é abordado o cenário de reescrita entre linguagens iguais, mas modelos diferentes.

A primeira abordagem é designada de arquitetura de mediadores (Wiederhold, 1992) que consiste numa camada lógica capaz de lidar com as heterogeneidades dos modelos existentes em diferentes repositórios de dados.

A segunda abordagem é designada de arquiteturas P2P e consiste na aplicação do paradigma P2P (Calvanese et al., 2003) para a implementação de sistemas de reescrita de consultas em que o processo é distribuído pelos vários *peers* pertencentes à rede.

2.5.1.1 Arquitetura de mediadores

A sua introdução foi realizada por (Wiederhold, 1992) e é baseada em mediadores que consiste numa espécie de módulo de *software* cujo objetivo é lidar com as heterogeneidades dos modelos em repositórios de dados distribuídos.

O seu objetivo é fornecer uma interface para acesso a dados, capaz de auxiliar no processo de integração de dados de diferentes modelos/repositórios de dados. A principal vantagem na adoção desta arquitetura é a abstração entre a fonte de dados e o seu acesso, devido ao facto de o mediador reformular em tempo de execução a consulta submetida ao modelo de mediação (também designado de modelo global), que por sua vez divide a mesma em várias subconsultas sobre as fontes de dados pertencentes ao sistema. A Figura 15 (Calvanese et al., 2003, p. 2) representa de que forma esta arquitetura é concebida para lidar com as heterogeneidades das fontes de dados.

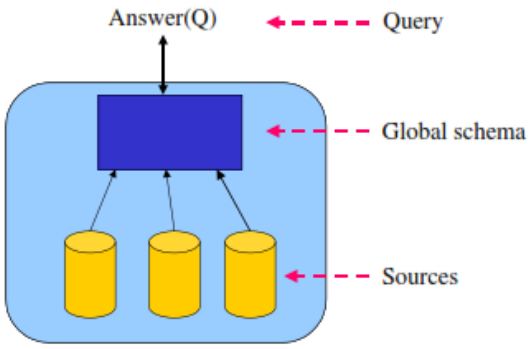


Figura 15 – Arquitetura de mediadores

Para gerar as subconsultas referidas anteriormente são utilizados mapeamentos que servem para obter correspondências entre os modelos de dados das fontes e o modelo global. Para a elaboração dos mapeamentos existem duas abordagens que determinam como será implementado o processo de reescrita.

A abordagem *Global As View* (GAV) consiste na definição de um modelo de dados global que permite o mapeamento entre as instâncias dos repositórios locais e uma vista no repositório global. A principal vantagem é facilitar o processo de reescrita da consulta submetida através da operação de desdobramento da consulta em que certos excertos da consulta são substituídos/traduzidos por referências dos repositórios locais.

Um caso prático da sua aplicação consiste na utilização de vistas no repositório de dados global (vGlobal), correspondente ao elemento *Global schema* da Figura 15, sobre os dados como interface para realizar consultas nos repositórios locais (*Sources*).

Como exemplo temos a seguinte vista:

```
Create View vGlobal as
Select t1.col1 as p1,t1.col2 as p2 from t1
Union
Select t2.col3 as p1,t2.col4 as p2 from t2
```

A partir da elaboração desta interface no repositório de dados global, é possível efetuar uma consulta (elemento Q da Figura 15) que pode ser definida da seguinte forma:

```
Select t1 from vGlobal
```

Internamente a consulta anterior é traduzida na execução das consultas definidas para cada repositório de dados como foi demonstrado no exemplo prático da demonstração das vistas.

A abordagem *Local As View* (LAV) consiste na definição de uma representação da fonte de dados local no modelo global, com o objetivo de permitir o mapeamento entre uma instância da fonte de dados local e uma vista no modelo global. A principal vantagem é o favorecimento da manutenção e de possíveis extensões que o sistema local possa sofrer. Por outro lado, o processo de reescrita é mais complexo porque não é possível aplicar a operação de desdobramento da consulta devido ao facto de esta abordagem ser mais vocacionada para os repositórios de dados locais. Tal limitação existe porque o mapeamento não é realizado como na estratégia anteriormente referida em que existe uma representação global para as instâncias das fontes de dados, sendo necessário recorrer às vistas definidas para cada instância na fonte de dados para realizar as operações de reescrita.

Um caso prático consiste em criar uma vista no repositório local com o formato da vista global pretendida. A tabela original no repositório local tem o seguinte formato:

```
Table1(coluna1,coluna2,coluna3,coluna4)
```

A vista a ser criada consiste na representação estática do repositório de dados global e tem o seguinte formato:

```
Create View lView as
Select coluna1 as p1,coluna4 as p2 from Table1
```

Para além das duas abordagens descritas anteriormente, existem algumas evoluções que resultam da sua combinação, tais como *Global Local As View* (GLAV) e *Both As View* (BAV), de modo a tirar proveito dos benefícios das abordagens base.

A Figura 16 representa de que forma podem ser combinadas as abordagens GAV e LAV para a implementação da abordagem GLAV.

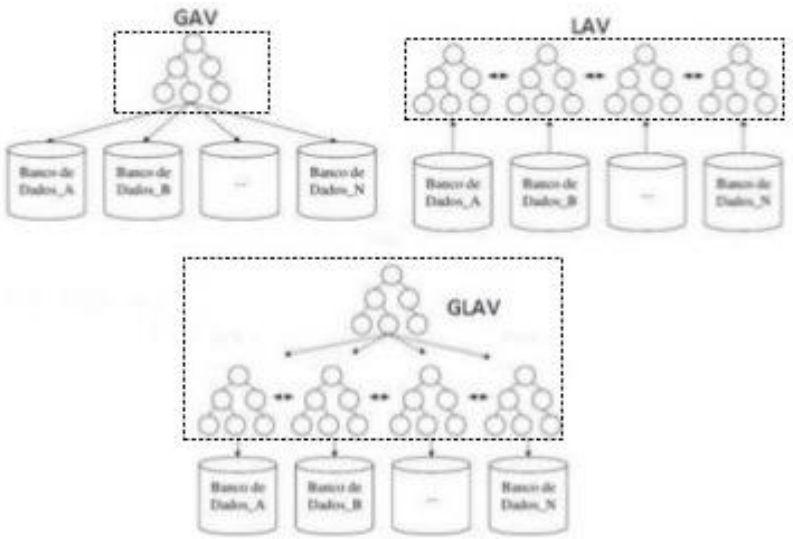


Figura 16 – Representação das abordagens GAV, LAV e GLAV (“Survey on NoSQL integration,” 20:40:28 UTC)

2.5.1.2 Arquitetura P2P

Esta abordagem é baseada no paradigma P2P e é designada por *Peer Data Management System* (PDMS). Este tipo de arquitetura deriva de uma extensão natural dos sistemas baseados na arquitetura apresentada anteriormente, aliada ao conceito de P2P. O motivo da sua criação deve-se à crescente necessidade de integração de dados em sistemas distribuídos.

Uma das principais características desta arquitetura consiste no facto de não existir um modelo global, pelo contrário, cada *peer* que compõe o sistema representa e fornece um modelo que é partilhado com os restantes *peers* do sistema. Esta abordagem é aplicada, por exemplo, em troca de dados entre aplicações (*data exchanging*) e resposta a consultas (*query answering*). A Figura 17 (Calvanese et al., 2003, p. 3) mostra quatro *peers* (P1, P2, P3 e P4) que possuem um conjunto de repositórios locais (*local source*) e um repositório global ao nível do *peer* (*peer schema*). O utilizador ao submeter uma consulta a um determinado *peer*, a mesma pode ser reencaminha para outro devido ao facto de existir um mapeamento entre eles (internamente cada *peer* tem um funcionamento semelhante à arquitetura de mediadores).

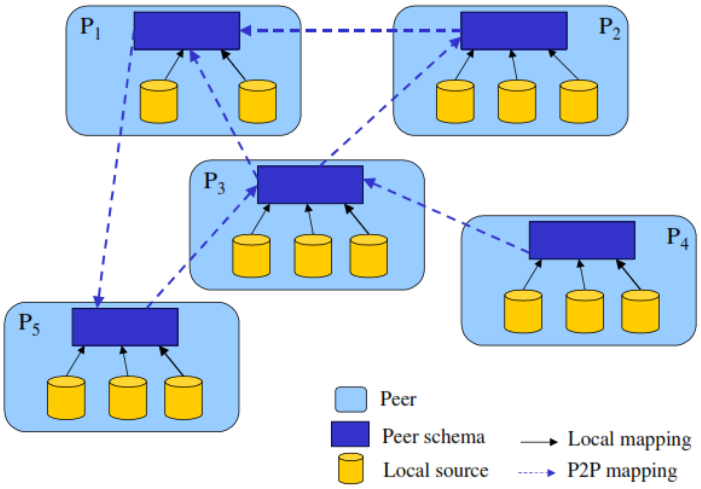


Figura 17 – Arquitetura P2P

Como foi referido anteriormente, existe um mapeamento entre cada *peer* do sistema, tendo esse mapeamento como base o seu repositório de dados global. A partir do repositório de dados global, é possível expor uma "interface" que, pelo facto de ser conhecida pelos outros *peers*, permite uma tradução/reencaminhamento do pedido. A Figura 18 mostra um exemplo de como é feito o mapeamento entre *peers* (P1, P2, P3 e P4), em que cada um possui um repositório global para expor informação relacionada com pessoas. A definição do formato desse repositório global obedece a uma estrutura bastante semelhante (nome e número das colunas).

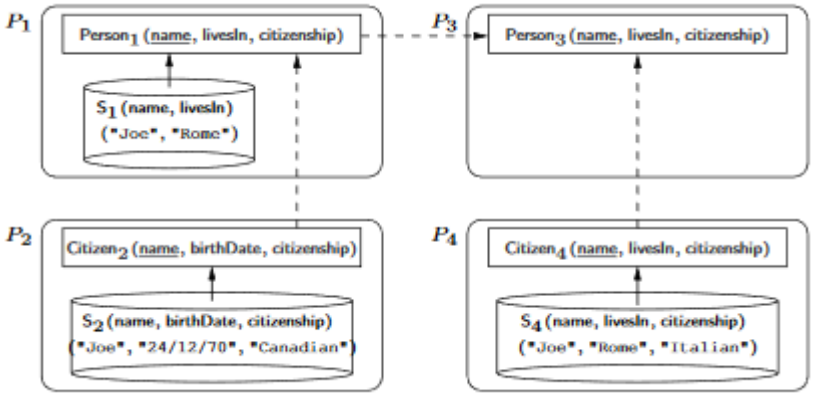


Figura 18 – Exemplo de mapemento entre *peers*

Após a análise destas duas arquiteturas de reescrita, é possível concluir que ambas as arquiteturas permitem a reescrita de consultas entre diferentes modelos/esquemas, utilizando sempre a mesma linguagem. Isto vem confirmar a carência da reescrita entre linguagens. Tendo sido identificada esta carência de reescrita entre linguagens, a secção seguinte tem como objetivo mostrar o estudo efetuado sobre a reescrita de consultas ao nível das linguagens.

2.5.2 Reescrita de consultas entre linguagens

A reescrita de consultas pode ser aplicada em várias dimensões uma delas a reescrita entre linguagens. Atualmente existem diferentes tecnologias de repositórios de dados que se encontram categorizados em dois tipos, relacional e não relacional. Independentemente da categoria onde se enquadram, cada repositório de dados possui um conjunto de particularidades, sendo a linguagem o foco do estudo nesta secção. De acordo com a Figura 14, é abordado o cenário de reescrita entre linguagens diferentes, mas modelos iguais.

Das diferentes linguagens de repositório de dados existentes, é escolhido como caso de estudo a reescrita de consultas entre SPARQL (Arenas and Pérez, 2011) e SQL (Chamberlin and Boyce, 1974). Normalmente a aplicação da reescrita de consultas entre SPARQL e SQL está associado à construção de repositórios de dados em RDF tendo por base as bases de dados relacionais. Este esforço de integração com bases de dados relacionais existe porque sua utilização possui vantagens como a performance relacionada com a indexação de informação que poucos repositórios de dados oferecem, mas a razão principal está relacionada com sua utilização massiva para armazenamento de dados por parte das organizações.

No estudo deste tipo de reescrita de consultas existem várias abordagens, sendo algumas delas apresentadas de seguida.

No trabalho de investigação (Cyganiak, 2005) é descrita a transformação de consultas em SPARQL para álgebra relacional e posterior tradução para consultas em SQL. Durante o processo de descrição das transformações, é especificado de que forma o SPARQL aborda os valores nulos que fazem parte da álgebra relacional e como esses valores devem ser tratados em operações

algébricas. Para além disso são enumeradas limitações entre a tradução de SPARQL para álgebra relacional, tais como:

- Nas instruções "OPTIONAL" existentes em subconsultas e na precedência em que as mesmas são avaliadas, causando resultados diferentes na atribuição dos valores às variáveis.
- Na tradução de instruções do tipo "FILTER" em que é permitida utilização de variáveis que não aparecem noutras partes da consulta, causando uma limitação a nível relacional para o enquadramento das mesmas.

Outra abordagem foi realizada em (Ma et al., 2008) que propõe o processamento de uma consulta SPARQL em dois processos distintos, o processamento das expressões de filtro e as restantes partes da consulta através da tradução dos nós que a compõe. A Figura 19 mostra o fluxo de execução de um processo de reescrita de SPARQL para SQL que começa com uma análise sintática e léxica, resultando na divisão referida anteriormente entre as expressões de filtro e a restante consulta SPARQL que são processadas em separado. A fase seguinte consiste na junção do resultado das duas operações anteriormente referidas, existindo no fim um processo de otimização da consulta SQL com o objetivo de eliminar partes da instrução desnecessárias.

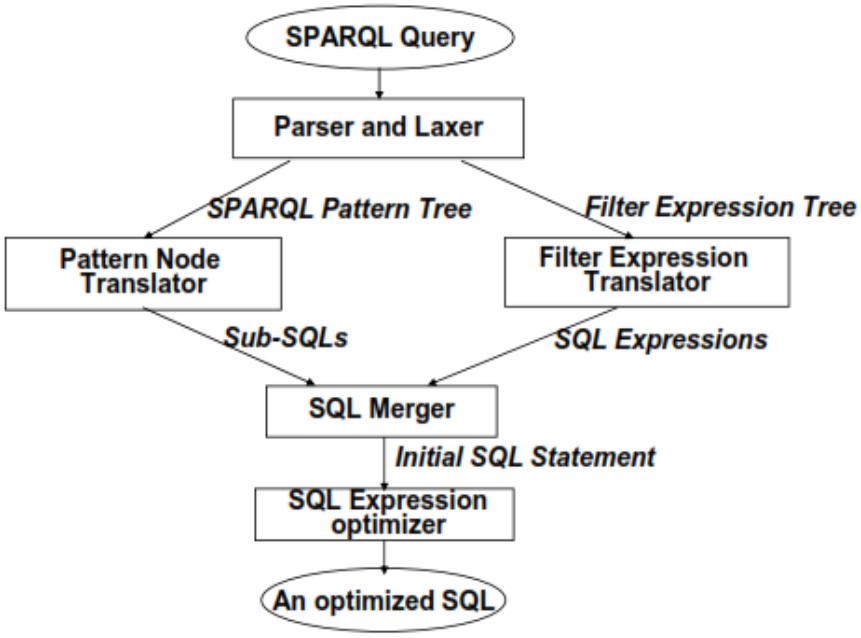


Figura 19 – Fluxograma reescrita de SPARQL – SQL (Ma et al., 2008, p. 7)

Com base nos pontos fracos trabalhos anteriormente referidos, (Chebotko et al., 2009) apresenta um algoritmo que preserva a semântica na tradução de consultas de SPARQL para SQL. Para isso, são realizados dois tipos de mapeamentos designados por α e β em que no primeiro é mapeado o conteúdo da base de dados no que toca à correspondência dos padrões dos triplos armazenados em relação a um dado triplo. O resultado são os triplos que obedecem à correspondência sendo o seu resultado projetado na clausula *SELECT* da instrução SQL. No segundo mapeamento é feita a correspondência entre partes que compõe o triplo (sujeito,

predicado, objeto) retornando um atributo relacional usado para construir a clausula *WHERE* da instrução SQL.

Na execução do processo de reescrita são utilizados os mapeamentos supramencionados em duas funções definidas no trabalho em questão:

- genPR-SQL: dado um padrão de triplo (tp) para realizar correspondências, o mapeamento β e a função Name, esta função retorna uma lista de atributos que correspondem a valores distintos do sujeito, predicado e objeto do triplo, renomeando os atributos para posterior inclusão na consulta.
- genCond-SQL: dado um padrão de triplo para realizar correspondências, o mapeamento β retorna uma expressão booleana que é verdadeiro em caso de o padrão submetido encontrar correspondência.

Depois de apresentados todos elementos que compõe o processo de reescrita, os mesmos podem ser utilizados para traduzir um triplo "tp" para uma consulta SQL sobre um repositório. Dado o padrão do triplo "tp = (?a,email,?e)" é gerada uma consulta que retorna todos os triplos cujo predicado tenha o valor email. O seguinte exemplo mostra onde se posicionam as funções para o processamento da reescrita e que parâmetros utilizam.

```
trans((?a, email,?e),α,β)  = Select Distinct genPR-SQL(tp,β,name) From
α(tp) Where genCond-SQL(tp,β)
```

Depois de executadas as funções, é possível obter uma instrução SQL completa. Os seguintes exemplos (Chebotko et al., 2009, p. 19) mostram qual o resultado retornado pelo processo de reescrita em que as funções "genPR-SQL" e "genCond-SQL" têm o seu valor atribuído:

```
q1 = trans((?a, email,?e),α,β)  = Select Distinct s As a, p As email, o
As e
From Triple Where p = 'email'


q2 = trans ((?a, web,?w),α,β)  = Select Distinct s As a, p As web, o As
w
From Triple Where p = 'web'
```

Para além de realizar operações de reescrita para consultas simples, também é possível reescrever operações SPARQL como *UNION, FILTER, OPTIONAL* e *AND.*

Para além de salvaguardar a semântica da tradução entre formatos, foi tido em conta questões de performance nas consultas obtidas na aplicação desta abordagem, tentando gerar instruções SQL eficientes.

Este trabalho serviu de referência ao trabalho realizado por (Elliott et al., 2009) que colmatou as carências de implementação do operador *GRAPH*, nós em branco, tipos de dados, idiomas suportados pelos literais e por último é também abordada a questão da performance no que toca às consultas criadas no processo de reescrita.

A base do processo de reescrita encontra-se no modelo criado com suporte numa estrutura relacional que tem como objetivo capturar a estrutura dos triplos, bem como os literais e seus

tipos de dados e de um modelo lógico que permite recolher meta dados sobre o modelo de dados que serve de base ao processo. O modelo relacional é constituído por quatro tabelas:

- *Types*: serve para armazenar os tipos de triplos com o objetivo que fazer pesquisas onde o sujeito do triplo é armazenado na tabela *types*, na coluna *member* e o seu predicado tem de ser obrigatoriamente do tipo "rdf:type".
- *LiteralProperties*: serve para armazenar as propriedades dos literais que compõem os grafos. As propriedades armazenadas são o tipo de dados do literal e a língua é referente à língua especificada no literal.
- *Relations*: serve para a armazenar as relações entre recursos.
- *Identifiers*: serve para armazenar os identificadores dos URIs.
- *Literals:* serve para armazenar os identificadores dos literais.

Depois de definida a estrutura que serve ao armazenamento de triplos em formato relacional, o passo seguinte consiste na definição de um modelo SQL que permita o processo de reescrita.

A Figura 20 mostra a diferença entre a abordagem realizada por este autor e a abordagem anteriormente discutida. Uma das principais diferenças está no facto de esta abordagem basear-se num modelo SQL que gera uma só consulta ao contrário da abordagem anterior em que a estratégia passa por gerar subconsultas.

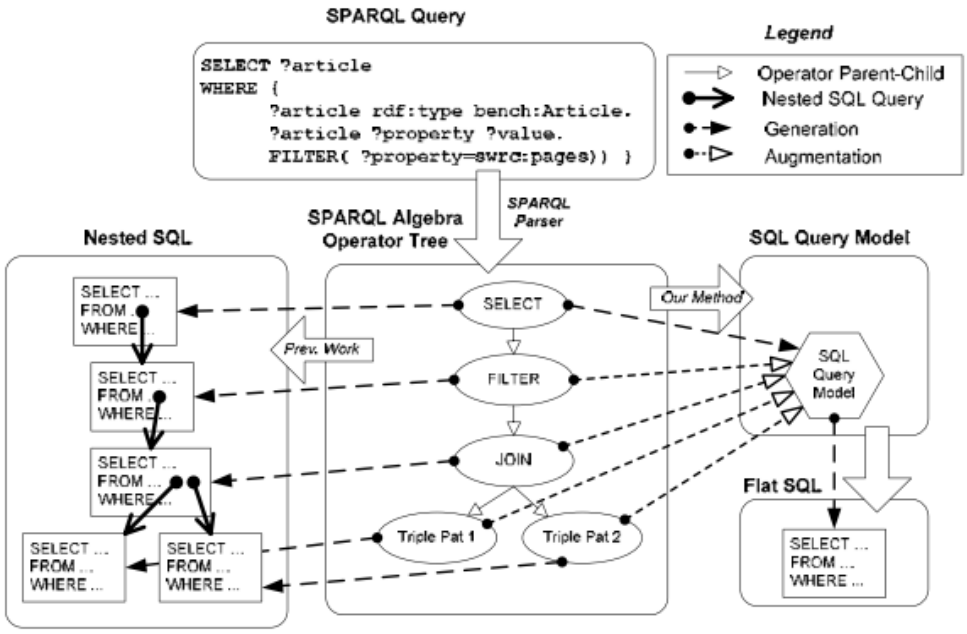


Figura 20 – Comparação da tradução de SPARQL – SQL entre a abordagem referida anteriormente (Cyganiak, 2005) e a abordagem atual (Elliott et al., 2009, p. 4)

O componente "SQL Query Model" pode ser descrito de acordo com um conjunto de propriedades que constam da Tabela 1.

Propriedade	Descrição
ProjectCols	Lista de colunas a projetar na cláusula SELECT.
Tables	Lista de tabelas (ou subconsultas) usadas na cláusula FROM.
Where	Conjunto de condições booleanas num formato conjuntivo na cláusula WHERE.
OrderBy	Lista de nomes de colunas utilizadas na cláusula ORDER BY.
Distinct	Valor booleano que permite determinar se vão ser retornados resultados distintos.
Limit	Valor inteiro que especifica número de tuplos que vão ser retornados.
Offset	Valor inteiro que especifica o número de tuplos que vão ser ignorados até serem mostrados os resultados.

Tabela 1: Propriedades Modelo SQL

Através de uma função em SQL e utilizando o modelo descrito é gerada a respetiva consulta SQL.

Embora esta abordagem não traduza a consulta utilizando mapeamentos entre modelos, ela foca em associar cada um dos construtores de SPARQL com os elementos de SQL, permitindo assim fazer um mapeamento entre linguagens (aqui o foco é nas particularidades das linguagens em vez da reescrita no geral).

Através da descrição das abordagens existentes para a reescrita de consultas entre estas linguagens é possível perceber a complexidade deste processo. Em algumas abordagens o foco é o mapeamento dos elementos que compõe a consulta, sendo assim um mapeamento dos seus elementos, por outro lado algumas abordagens defendem a adoção de construtores específicos que visam resolver determinadas particularidades semânticas das linguagens.

2.5.3 Sumário

A análise do tema reescrita de consultas permite contextualizar o trabalho ao nível das abordagens existentes para esta área, determinando quais as suas limitações tendo em conta os possíveis cenários de reescrita, como também limitações diretamente relacionadas com a tarefa que desempenham.

3 Análise de valor de negócio

Neste capítulo é analisado o valor de negócio onde são abordados os conceitos existentes e respetivo enquadramento com o tema abordado e os processos e seus intervenientes. A análise de valor de negócio inicia com a secção de conceitos de negócio onde é apresentada a ideia de negócio através do modelo de canvas, a definição de qual a proposta de valor subjacente ao problema descrito anteriormente, qual o valor acrescentado para os possíveis clientes tendo em conta os benefícios e sacrifícios que derivam para a sua utilização/aquisição e quais os possíveis cenários de negócio e por último é analisado o valor criado.

Nas seguintes secções são abordados os conceitos referentes à análise do valor de negócio e respetivo enquadramento com o tema abordado.

3.1.1 Proposição de valor

A proposição de valor é a peça central quando analisamos um modelo negócio porque é a partir dela que todas as restantes variáveis são definidas. A sua definição permite determinar qual a mais-valia oferecida ao cliente como por exemplo redução de custos, qualidade, usabilidade e porque que o mesmo vai recorrer ao produto/serviço em questão. Este último ponto é bastante importante porque é necessário determinar de que forma o cliente perceciona a proposta de valor apresentada pelo produto/serviço.

O produto/serviço consiste num módulo que recebe operações de limpeza de qualidade de dados e efetua a respetiva operação de acordo com o repositório definido para a execução da operação. A proposição de valor do produto referido consiste num processo de correção de problemas de qualidade nos dados de uma maneira fácil, rápida e com maior fiabilidade permitindo assim obter resultados com maior credibilidade no momento de análise e poupar tempo e recursos na definição de operações de limpeza porque as mesmas são definidas e executadas independentemente do repositório de dados alvo. Este produto é único porque permite reutilizar conhecimento adquirido para manutenção da qualidade dos dados de um determinado repositório em outros repositórios ou modelos de dados de forma automática reduzindo assim os custos de implementação deste tipo de processos.

3.1.2 Segmentos de clientes e suas necessidades

Relativamente à segmentação dos clientes foram identificados dois segmentos, o primeiro está associado às empresas que desenvolvem solução de integração de dados que possam ter interesse em incorporar este produto como um serviço extra a ser disponibilizado aos seus clientes e o segundo segmento é referente a organizações detentoras de armazéns de dados que precisem de uma ferramenta que execute processos de limpeza que assegurem a qualidade dos dados existentes.

Numa perspetiva longitudinal de valor é necessário avaliar quais os benefícios obtidos pelos clientes na utilização deste produto/serviço, mas também quais os sacrifícios que decorrem da obtenção do mesmo. Esta avaliação permite fazer um balanço geral dos possíveis ganhos em relação às possíveis perdas.

A Tabela 2 seguinte mostra seguinte mostra o conjunto de benefícios e sacrifícios inerentes a cada segmento de cliente identificado.

	Ferramentas de integração de dados	Organizações com armazéns de dados
Benefícios	• Possibilidade de oferecer uma nova funcionalidade aos seus clientes. • Destacar o produto da concorrência.	• Facilidade de especificação e de execução de operações de limpeza de qualidade de dados. • Reutilização de processos definidos anteriormente.
Sacrifícios	• Possíveis incompatibilidades na inclusão da nova funcionalidade no produto existente.	• Necessidade de formação e de esforço inicial para a instalação e configuração.

Tabela 2: Sacrifícios/ Benefícios para a utilização do produto/serviço

De uma forma geral os benefícios identificados estão relacionados com o colmatar de uma necessidade dos clientes (assegurar a qualidade dos seus dados) de forma eficiente e eficaz. Por outro lado, os sacrifícios associados à sua utilização estão relacionados com a falta de informação sobre o seu uso e necessidade de configuração.

3.1.3 Modelo de canvas

Depois de definida a proposição de valor é necessário enquadrar os restantes conceitos que vão ser abordados do modelo de negócio. Para isso é apresentado o modelo de canvas que permite descrever de uma forma breve a ideia de negócio. Através do mesmo conseguimos identificar quais são as diferentes dimensões que servem de base uma análise de modelo de negócio. Os conceitos de proposição de valor e segmentação de clientes foram abordados nas secções, para as restantes é feita uma descrição e de que forma a mesma se encontra enquadrada com o tema abordado.

Os canais utilizados para a divulgação e/ou distribuição do produto são os canais de distribuição dos produtos dos nossos parceiros e a própria página do produto na internet que serve para mostrar detalhes e demonstrações da potencialidade do mesmo como também expor de que forma o mesmo pode ser requerido.

No que toca ao relacionamento com o cliente foi deduzido que é necessário um apoio permanente a implementação do produto dotando os clientes de formação para a utilização do mesmo.

As atividades chaves identificadas são relacionadas com o desenvolvimento do produto de acordo com a proposição de valor, como também de acordo com as principais necessidades dos segmentos de clientes identificados.

Na área das parcerias-chave foram identificados um conjunto de empresas que ocupam uma grande cota de mercado em soluções de ETL e análise dos dados, como também ferramentas de código livre que possam servir de rampa de lançamento ao produto caso as grandes empresas referidas anteriormente não estejam interessadas a incorporar esta solução nos seus produtos.

Os recursos chave identificados são as linguagens e bibliotecas de código livre e aberto que estão na base da implementação do produto.

A estrutura de custos deve-se essencialmente à mão-de-obra para desenvolver o produto, possíveis ações de *marketing* e projetos-piloto para promover o produto e demonstrar de que forma o mesmo cria valor. Quanto às fontes de receita as mesmas provem de cobranças de comissões sobre as vendas efetuadas pelos nossos parceiros como também através da cobrança de uma mensalidade aos nossos clientes disponibilizando o produto como um serviço.

Na Tabela 3 é mostrado de forma breve o modelo de canvas que foi descrito anteriormente.

<table>
<tr>
<td rowspan="2">Parceiros chave
Microsoft/Oracle (Possível integração nos produtos de BI).
Ferramentas ETL de Código Livre</td>
<td>Atividades Chave
Desenvolvimento soluções para aplicação de limpeza de dados em diferentes tipos de repositórios de dados.
Levantamento das principais necessidades dos segmentos de clientes.</td>
<td rowspan="2">Proposição de Valor
Correção de problemas de qualidade nos dados.
Processo realizado com maior facilidade, rapidez e fiabilidade.
Suporte para diferentes tipos de repositórios de dados.</td>
<td>Relacionamento com o cliente
Apoio permanente na implementação da solução.
Realização de ações de formação.</td>
<td rowspan="2">Segmentos de clientes
Ferramentas de integração de dados.
Organizações com armazéns de dados;</td>
</tr>
<tr>
<td>Recursos Chave
Bibliotecas e linguagens de programação de código livre e aberto.</td>
<td>Canais
Incorporação nas soluções dos parceiros;
Página na internet onde é possível testar e adquirir o produto.</td>
</tr>
</table>

Estrutura de Custos	*Fontes de Receita*
• Mão-de-obra de desenvolvimento. • Ações de marketing para divulgação do produto; • Realização de projetos-piloto para angariação de clientes;	• Cobrança de comissões sobre as vendas efetuadas do nosso produto através dos parceiros; • Cobrança de mensalidade por utilização do produto;

Tabela 3: Modelo de Canvas

3.1.4 Criação de valor

O valor criado aos segmentos de clientes identificados pode ser analisado e/ou quantificado da seguinte forma através do modelo conceptual para decomposição do valor para o cliente.

O modelo anteriormente referido assenta em quatro conceitos: formas de valor e posições temporais de valor, redes de valor, ativos endógenos e exógenos da empresa e os benefícios e sacrifícios percecionados pelos intervenientes num processo negocial (tema a abordar na secção seguinte).

Numa primeira fase é necessário construir a rede de valor da empresa identificando as entregas tangíveis e intangíveis como também dos ativos (endógenas e exógenas) construídos e/ou utilizados no fornecimento do produto final. Com isto é possível perceber que o valor para o cliente pode ser dividido em componentes mais simples, integrando o valor percecionado pelos membros da empresa para dada posição no tempo.

Numa segunda fase, o objetivo é obter mais informações sobre a sua perceção de benefícios/sacrifícios junto do cliente num determinado período de tempo e posição, através da seleção dos ativos mais relevantes para perceber que resultados serão utilizados para avaliar a forma como o cliente percebe a proposição de valor do produto/serviço oferecido.

A última fase requer uma análise da perspetiva da empresa e da perspetiva do cliente.

Analisando os ativos/fornecimento de produtos e benefícios/sacrifícios percebidos na perspetiva da empresa em conjunto com a perceção do cliente em relação aos resultados relevantes dos benefícios/sacrifícios percecionados por ambos, permitindo assim o ajuste da proposição de valor.

3.1.5 Abordagem aos segmentos de clientes

Depois de analisar todas as componentes do modelo de negócio é necessário determinar como é que se aborda os potenciais clientes do produto/serviço. Essa abordagem envolve determinar qual a estratégia de negociação a escolher para o efeito e seu respetivo cenário.

Relativamente aos possíveis cenários de negociação existem dois tipos de negociação, a negociação distributiva e a negociação integrativa.

Na negociação distributiva assenta num cenário de ganha-perde em que existe um esforço de cada uma das partes em assegurar os seus benefícios o que não é uma boa estratégia neste caso porque precisamos oferecer determinados benefícios aos possíveis clientes do produto.

Na negociação integrativa as partes envolvidas cooperam entre si para obter o máximo de benefícios através de um cenário de negociação ganha-ganha. Este cenário é o indicado porque é preciso existir um equilíbrio entre os benefícios que procuramos como também os benefícios que os possíveis clientes pretendem obter com o produto/serviço. Neste tipo de negociação é necessário numa primeira fase gerir inicialmente as expetativas e objetivos e determinar quais desses pontos não negociáveis. Durante a preparação do processo deve-se ter em conta possíveis ofertas que podem ser submetidas e como lidar com as mesmas, como também estar preparado efetuar algumas para tentar conquistar a negociação. Para antecipar os cenários anteriormente descritos é necessário um conhecimento de todas matérias que envolvem direta ou indiretamente o processo de negociação. Ao definir quais as possíveis ofertas a propor é necessário definir qual é o seu mínimo e qual o seu máximo a aplicar no processo de negociação, isto permite ter a noção do ponto inicial da negociação, como também qual o limite que a mesma pode atingir no que toca à disponibilização de benefícios. É importante justificar o porque dessas escolhas para a parte interessada conseguir perceber qual a posição da outra parte no processo negocial.

4 Reutilização do conhecimento de limpeza de dados baseada em ontologias

Neste capítulo é apresentada a metodologia de limpeza de dados baseada em ontologias (Almeida et al., 2015) que serve de base a este trabalho. Esta metodologia tem por base os seguintes princípios:

- A especificação das operações de limpeza de dados para um determinado repositório deve ser realizada por um especialista, de forma mais próxima a nível conceptual do ser humano.
- Promover a reutilização de operações de limpeza de dados especificadas para aplicar em diferentes repositórios de dados, independentemente do modelo e esquema de dados subjacente.

Para descrever esta metodologia é realizado em primeiro lugar uma descrição numa perspetiva geral da sua arquitetura, com o objetivo de perceber de que forma se encontra estruturada, passando de seguida para um exemplo prático da sua aplicação.

4.1.1 Perspetiva geral

Nesta secção o objetivo é apresentar esta metodologia através de uma breve descrição da sua arquitetura em camadas.

A Figura 21 representa uma perspetiva geral da metodologia, em particular, dos processos que compõem a mesma e respetivos dados de entrada e saída. Adicionalmente, esses processos estão organizados em três camadas lógicas que são a cama camada de alto nível (*Abstract Data Layer*)*, a* camada intermédia (*Bridge Data Layer*), a cama acesso a dados (*Concrete Data Layer)* e o único ator que interage com o sistema (*Domain Expert*).

Estas camadas podem ser descritas da seguinte forma:

- *Abstract Data sLayer*: definição de operações de limpeza de dados através de um vocabulário específico e da conceptualização do domínio. A execução das operações é iniciada nesta camada, sendo depois despoletado o processo de reescrita da mesma na camada inferior. O vocabulário de limpeza de dados adotado nesta metodologia consiste numa extensão da taxonomia referida anteriormente (cf. secção 2.2.2) designada daqui em diante por E-DQM.
- *Bridge Layer*: permite a interoperabilidade entre a camada "*Concrete Data Layer"* e a camada "*Abstract Data Layer"* através de processos semiautomáticos:
    - Gerar a conceptualização do domínio (*Transformation*): este processo consiste na respetiva transformação do modelo existente no repositório de dados (*DataSource*) para um modelo conceptual (*Domain*) a ser utilizado na camada "*Abstract Layer*".

- Gerar as correspondências ao nível do domínio (*Mapping*): entre as entidades do repositório e a conceptualização obtida no processo anteriormente referido.
- Efetuar o processo de reescrita da operação de limpeza de dados (*Data Cleaning Operation Rewriting Process*): com o objetivo de gerar a respetiva operação de limpeza a ser aplicada no repositório de dados existente na camada inferior.

- *Concrete Data Layer*: responsável por executar os processos de limpeza de dados especificados de acordo com os requisitos definidos para a operação nas camadas superiores.

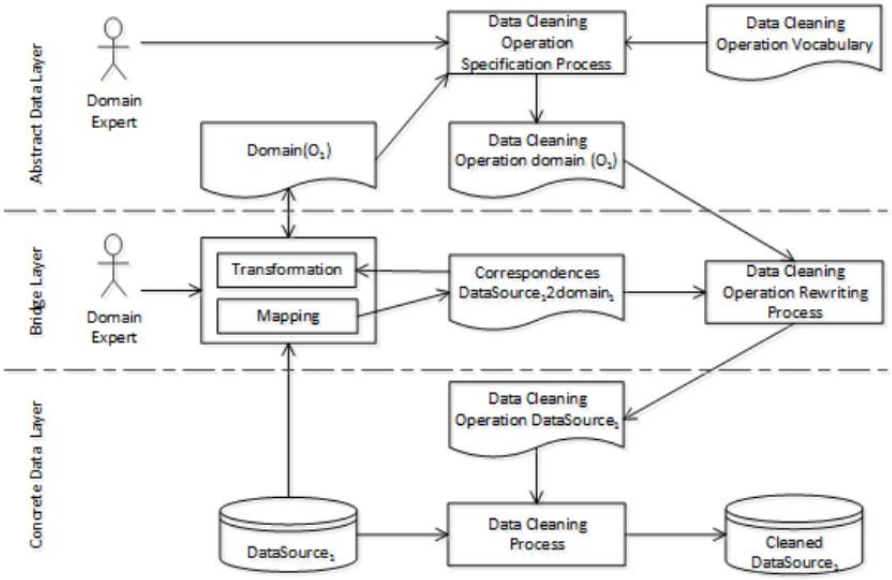


Figura 21 – Representação gráfica da metodologia de limpeza de dados (Almeida et al., 2015)

É possível enquadrar este trabalho na metodologia aqui apresentada através do elemento "*Data Cleaning Operation Rewriting Process*".

4.1.2 Exemplo

Depois de descrita a metodologia, é apresentado um exemplo prático da sua utilização (Almeida et al., 2015) utilizando três repositórios de dados diferentes e uma conceptualização geral dos mesmos através de um domínio. Os três tipos de repositórios de dados existentes na *Concrete Data Layer* são denominados por *DB1, DB2, DB3* e a linguagem de manipulação da informação é SQL, SPARQL e MongoDB *query* respetivamente.

A Tabela 4 permite perceber de que forma os elementos dos diferentes esquemas e o respetivo conceito no domínio estão relacionados entre si.

Domínio	Repositório 1	Repositório 2	Repositório 3
Customer	*Customer*	Buyer	Account
maxDiscount	DiscMax	MaxDiscPerc	MaxDiscount

Tabela 4: Alinhamentos entre o domínio e os elementos dos repositórios de dados

A operação escolhida foi criada utilizando o vocabulário do E-DQM e está relacionada com a limitação do desconto máximo a 20% aos clientes:

```
[] a dqm:LegalValueRangeRule;
dqm:testedClass ex:Customer;
dqm:testedProperty1 ex:maxDiscount;
dqm:upperLimit “20%”.
```

O resultado da reescrita desta operação dada como exemplo para os diferentes repositórios de dados é o seguinte:

- *DB1* (SQL):

```
Select * From DB1.Customer
where DB1.DiscMax > 20%
```

- *DB2* (SPARQL):

```
Select ?c where {
?c a DB2:Buyer;
?c DB2:MaxDiscPerc ?desc;
Filter (?desc > 20%)}
```

- *DB3* (MongoDB *query*):

```
Db.DB3:Account.find({
DB3:MaxDiscount:{$gt: 20%} });
```

4.1.3 Sumário

Nesta secção foi apresentada a metodologia de limpeza de dados baseada em ontologias onde é possível ver a sua arquitetura, os artefactos em que se baseia e exemplos práticos da sua aplicabilidade. É de salientar a importância de um dos elementos da arquitetura designado por *Data Cleaning Operation Rewriting Process*, é a partir dele que é contextualizado este trabalho.

5 Requisitos

Nesta secção é realizada uma análise aos requisitos necessários para a elaboração da solução. Esta análise é importante porque é a partir da mesma que os restantes capítulos como Análise e Design e respetiva descrição da implementação da solução são baseados.

Para o levantamento dos requisitos da solução foi utilizado um modelo designada por FURPS+ ("Appendix B," 2004). Este modelo permite categorizar os requisitos em funcionais e em não funcionais. Ao nível dos requisitos funcionais existem categorias como a usabilidade, confiabilidade, desempenho e suporte. Ao nível dos requisitos não funcionais, os mesmos são alargados a outras áreas através de restrições de *design*, implementação, interface e físicos.

A Figura 22 mostra de que forma é utilizado este modelo para a classificação de requisitos funcionais e não funcionais através de um conjunto de pontos.

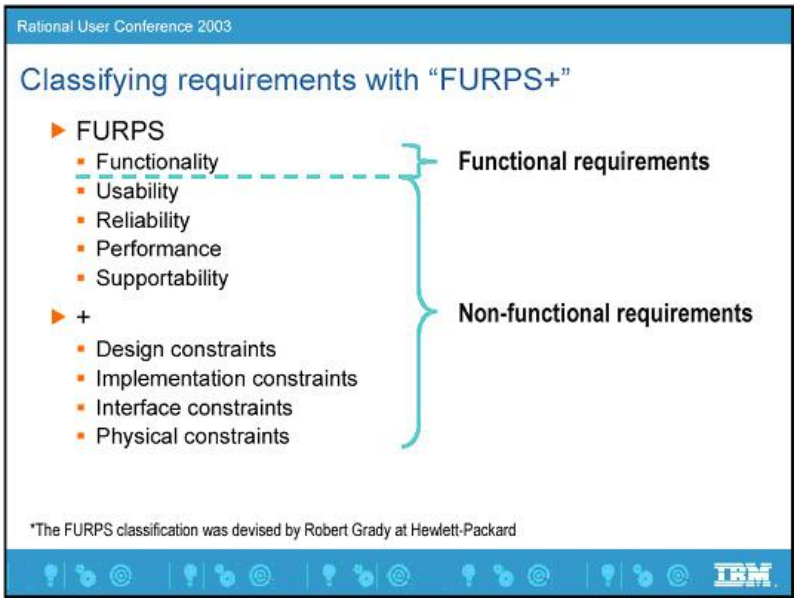


Figura 22 – Classificação de requisitos utilizando FURPS+ ("What, no supplementary specification?," 2004)

A partir das categorias apresentadas anteriormente é feita a identificação de requisitos da solução. De seguida são descritas unicamente as categorias de requisitos aplicáveis a este trabalho.

5.1 Funcionalidade

Nesta secção são descritas as funcionalidades que integram a solução. A sua especificação permite determinar quais são as necessidades diretas dos utilizadores na sua utilização.

Durante elaboração da lista de funcionalidades necessárias, foi concluído que as mesmas podem ser divididas em funcionalidades principais que estão relacionadas diretamente com o problema que a solução pretende resolver e funcionalidades auxiliares que servem de suporte ao seu funcionamento.

Tendo em conta as principais necessidades identificadas para a elaboração deste trabalho, a solução deve disponibilizar as seguintes funcionalidades:

- Reescrever operações de limpeza de dados a partir das operações definidas pelo especialista de domínio na camada mais elevada de abstração dos dados.
- Permitir a reescrita de operações de limpeza de dados entre diferentes domínios e/ou vocabulários.
- Interoperabilidade entre diferentes ferramentas de limpeza de dados.

Para além de permitir a execução das funcionalidades anteriormente referidas, a solução pode possuir um conjunto de funcionalidades que suportem o seu funcionamento relativamente a questões de configuração. A Figura 28 mostra o diagrama de casos de uso da solução onde estão inseridos os dois tipos de funcionalidades anteriormente referidas.

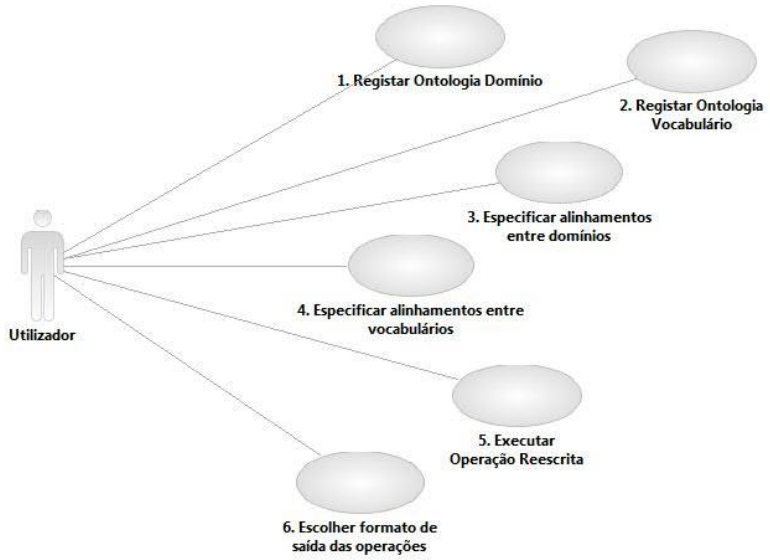


Figura 23 – Diagrama de casos de uso da solução

Os casos de uso numerados de 1 a 4 referem-se à gestão de conteúdo necessário para a realização da operação de reescrita, como por exemplo qual o domínio e vocabulário base em que as operações se encontram escritas, para qual domínio e vocabulário vão ser reescritas e quais os alinhamentos entre domínios e vocabulários que servem de suporte ao processo de reescrita.

O caso de uso 5 refere-se à execução do processo de reescrita tendo por base o conteúdo previamente carregado. Este caso de uso é considerado o principal da solução sendo descrito de forma aprofundada no capítulo seguinte.

Através do caso de uso 6 existe a possibilidade de indicar em que formato as operações de limpeza de dados vão ser serializadas, por outras palavras para que ferramenta de limpeza de dados se vai produzir operações reescritas.

Depois de descritas as funcionalidades da solução a secção seguinte permite categorizar as mesmas quanto à sua facilidade de utilização.

5.2 Usabilidade

Nesta secção são expostos os aspetos relacionados com a usabilidade da solução idealizada. A sua usabilidade é algo tão importante como as funcionalidades que oferece porque não importa possuir um elevado número de funcionalidades e realizar operações de elevada complexidade se as mesmas são difíceis utilizar.

De acordo com as funcionalidades descritas anteriormente, a solução deve disponibilizar um conjunto de operações que permitam a sua configuração, execução e monitorização de operações de reescrita de limpeza de dados. Estas operações aliadas ao conhecimento de ferramentas de limpeza de dados/operações de limpeza de dados devem permitir a sua reutilização.

De acordo com o contexto inicial dado ao problema, a solução necessita de conteúdo específico que é carregado a partir de ficheiros como ontologias de domínio/vocabulário, alinhamentos entre ambas e a gramática de suporte à ferramenta de limpeza de dados que se pretende utilizar.

Devido à complexidade deste tipo de operações é necessária a elaboração de uma interface simples e intuitiva para a sua fácil utilização.

Durante o processo de reescrita a interface da solução deve disponibilizar de uma forma simplista um conjunto de informações ao utilizador:

- Qual o conteúdo carregado para execução da operação.
- Qual a operação que se encontra a processar.
- Em que fase se encontra da operação e o seu respetivo resultado.
- Descrição de possíveis incompatibilidades existentes que leve a operação a abortar.
- Resultado final da operação.

O constante débito de informação por parte da solução ao utilizador, leva a que o mesmo consiga estar consciente do estado de execução do processo, como também permite a resolução de problemas existentes.

5.3 Confiabilidade (Reliability)

Nesta seção são expostos os aspetos relacionados com a confiabilidade dos resultados obtidos através da utilização da solução. A partir desta categoria e do modelo de classificação escolhido,

podemos definir quais os requisitos necessários para garantir a fiabilidade do seu funcionamento no que toca ao processamento efetuado na reescrita de operações, como também a sua capacidade para lidar com situações anómalas.

Um processo de reescrita pode ser considerado fiável de acordo com as seguintes características:

- Obtenção de operações reescritas que obedeçam a sintaxe da respetiva ferramenta de limpeza de dados a utilizar.
- Correta aplicação do mapeamento entre operadores da operação de limpeza de dados e a ferramenta de limpeza de dados.
- Correta aplicação do conteúdo relacionado com o processo de reescrita.
- Perante uma situação anómala durante o processo de reescrita, permitir a sua resolução de forma a concluir o processo ou a mostrar ao utilizador de forma clara quais as causas que impediram o sucesso da execução.

Depois de abordados os pontos que permitem tornar a solução fiável, é desejável abordar quais os requisitos associados à performance que devem ser obedecidos. Para além de uma solução fiável, a mesma deve responder dentro de um intervalo de tempo aceitável.

5.4 Suporte

Esta categoria de requisitos permite determinar de que forma a solução pode ser sustentável relativamente a um conjunto de perspetivas:

- Auditável: a partir desta característica podemos determinar quais as situações anómalas a que a solução foi submetida, permitindo assim a melhoria contínua da mesma.
- Compatibilidade: o facto de o processo de reescrita ser completamente configurável através do conteúdo carregado pelo utilizador, leva a que exista uma elevada compatibilidade com diferentes ferramentas de dados.
- Configuração: necessidade de carregamento de conteúdo relacionado com a reescrita das operações de limpeza de dados e ferramentas suportadas.
- Manutenção: a nível de manutenção é necessário numa situação deste género que a solução permita a atualização da versão do E-DQM que está a utilizar, bem como do serializador de uma determinada ferramenta de limpeza de dados caso a mesma seja atualizada.
- Testabilidade: para testar a solução é necessário modelar o sistema em partes independentes em que seja possível a realização de testes, verificando a assertividade de cada módulo, bem como do resultado do processo na integra.

Depois de apresentadas as principais categorias do modelo FURPS, falta abordar um conjunto restrições (+) associadas aos requisitos.

5.5 Restrições (+)

Para além da categorização realizada anteriormente dos requisitos funcionais, existem um conjunto de restrições que permitem refinar a análise de requisitos de acordo com quatro níveis

que são os requisitos associados ao *design*, implementação, interface e físicos. Para este problema não há necessidade de descrever os requisitos físicos.

As restrições associadas ao *design* relacionam-se com o facto de ser utilizada uma linguagem orientada a objetos. Logo a nível de *design*, a sua elaboração deve ter em conta os paradigmas associados a este conceito.

Nas restrições de implementação podemos encontrar requisitos ditados por componentes externos. Relativamente à linguagem a utilizar, a escolha recai sobre o Java devido ao facto de a mesma oferecer um elevado número de bibliotecas relacionadas com a manipulação de conteúdo de ontologias e elaboração de gramáticas que permitem facilitar o desenvolvimento da solução.

Por fim relativamente às restrições associadas à interface é necessário determinar de que forma é possível incluir o suporte a novas ferramentas de limpeza de dados.

6 Análise e Design

Neste capítulo é descrita a análise realizada ao problema abordado e o respetivo desenho elaborado para sua solução. Na análise do problema, o artefacto utilizado é o modelo de domínio onde são capturadas as entidades que compõe um processo de reescrita e de que forma as mesmas se relacionam para suportar a execução das operações sobre diferentes ferramentas de limpeza de dados e domínios.

Quanto ao *design* elaborado da solução, o mesmo foi realizado sobre duas perspetivas, a arquitetural e a detalhada. No modelo arquitetural é descrita a solução num nível de abstração mais elevado, recorrendo a diagramas de componentes onde é possível visualizar de que forma a solução está estruturada (módulos e suas interações). No modelo detalhado é descrito o fluxo de execução da solução na aplicação de uma operação de reescrita de limpeza de dados. A descrição começa com um breve resumo do fluxo de execução da operação na sua íntegra, passando para a descrição em detalhe de determinados pontos referenciados no diagrama sequência principal.

Estes dois processos são baseados num algoritmo de reescrita que permite ter uma visão geral do conteúdo a receber como entrada, as fases que compõe o processo e a respetiva saída.

Como entrada do algoritmo constam os seguintes artefactos:

- OD: ontologia de domínio base
- O'D: ontologia de domínio objetivo da reescrita
- DCOV: vocabulário base
- DCOV:' vocabulário objetivo da reescrita
- $A(OD,O'D)$: alinhamentos entre os domínios
- $A(DCOV,DCOV')$: alinhamento entre vocabulários
- DCO: operação de limpeza de dados
- FormatoSaida: para qual ferramenta vai ser serializada a operação

O fluxo de execução do algoritmo consiste no processamento dos vários elementos da DCO verificando em primeiro lugar se os domínios estabelecidos como parâmetro são diferentes, caso sejam, é feita a reescrita da operação ao nível do domínio utilizando os alinhamentos entre domínios. Em segundo lugar é verificado se os vocabulários estabelecidos como parâmetro são diferentes, caso sejam, é feita a reescrita ao nível do vocabulário utilizando os alinhamentos ao nível do vocabulário. Depois de reescrita a DCO, a mesma é serializada o que implica a construção da instrução consoante o formato de saída.

```
Entrada:OD,O'D,DCOV,DCOV',A(OD,O'D),A(DCOV,DCOV'),DCO,FormatoSai
da.
Saída: DCOReescrita.
DCOReescrita ← ∅;
ciclo (dcoOD ∈ DCO)
 se (OD ≠ O'D) então // Reescrita ao nível do domínio
        dcoOD ← transformaçãoDomínio(dcoOD, A(OD,OD' ))
 senão
        dcoOD' ← dcoOD
 fim
 se (DCOV ≠ DCOV') então // Reescrita ao nível do vocabulário
        dcoOD' ← transformaçãoVocabulário (dcoOD' , A(DCOV,DCOV' ))
 senão
        dcoOD' ← dcoOD'
 fim
fim ciclo
serializar(DCOReescrita, FormatoSaida)
```

6.1 Modelo domínio

Nesta secção é descrito o modelo de domínio que é utilizado para apresentar quais os conceitos presentes na reescrita de operações de limpeza de dados.

As organizações possuem sistemas de informação que produzem dados que por sua vez são armazenados em um ou mais repositórios. Cada repositório pertence a um domínio em especifico como, por exemplo, medicina, ensino, etc. Num caso prático uma organização que possua dois sistemas informáticos para a mesma área de negócio muito provavelmente terá dois repositórios de dados sobre o mesmo domínio.

Como é do interesse da organização obter uma maior qualidade nos dados produzidos, a mesma sente necessidade de especificar operações de limpeza de dados (DCO). Cada DCO pode ser descrita através de um vocabulário e um domínio.

O processo de reescrita permite que uma operação escrita para um determinado domínio/vocabulário seja reescrita para outro domínio (normalmente associado a um repositório de dados) /vocabulário (normalmente associado às diferentes ferramentas de limpeza de dados existentes na organização). Como foi referido anteriormente, tal processo só é possível devido à existência de alinhamentos entre domínios/vocabulários.

A Figura 24 mostra o modelo de domínio onde se encontram os conceitos descritos relacionados com a reescrita de operações. Tal como na descrição anterior, existe uma organização que possui um conjunto de repositórios/ferramentas de limpeza por um lado e um conjunto de DCOs que querem ver aplicadas por outro.

Dos repositórios existentes na organização, vários partilham um domínio (capturado através de uma ontologia), mas alguns deles podem pertencer a um domínio diferente. Como forma de traduzir esses domínios existe um conjunto de alinhamentos.

Cada DCO possui um vocabulário através do qual foi criada, existindo a necessidade de traduzir esse vocabulário para a ferramenta de limpeza que se pretende utilizar, existindo para isso também alinhamentos entre ambas. Os alinhamentos consistem num conjunto de referências entre duas entidades, neste caso domínios ou vocabulários.

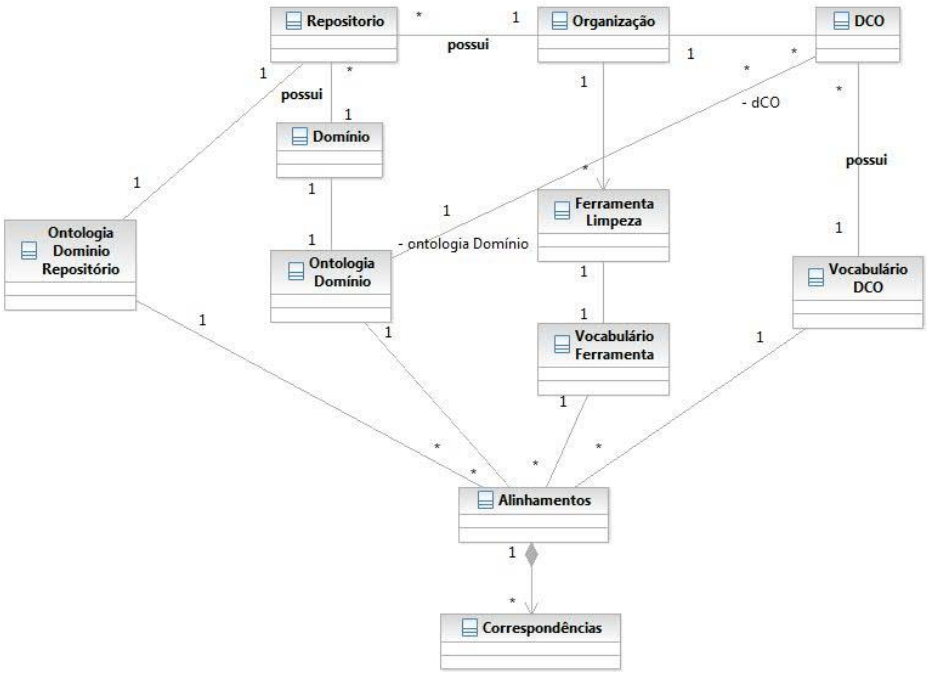


Figura 24 – Diagrama do modelo de domínio

Depois de descrito o modelo de domínio de uma operação de limpeza, é possível ter uma noção de quais os conceitos/entidades que se encontram associados à mesma. O passo seguinte consiste em organizar esses conceitos com o objetivo de obter uma arquitetura que responda à análise realizada ao problema.

6.2 Perspetiva arquitetural

Para apresentar o modelo arquitetural desenhado para a solução foi escolhido o diagrama de componentes, devido ao facto de permitir mostrar de uma forma fácil quais são os componentes em que a solução está organizada e quais as dependências que existem entre eles.

Nesta fase surge um conjunto de questões que influencia diretamente o *design* da solução, sendo necessária uma análise mais aprofundada para determinar uma solução para as mesmas do ponto de vista técnico. As questões são as seguintes:

- Como é que o conteúdo a ser carregado na solução pode ser manipulado durante a execução do processo de reescrita?
- De que forma é possível extrair determinado tipo de conteúdo da ontologia de acordo com o contexto de reescrita que se pretende efetuar (ao nível do domínio ou ao nível do vocabulário)?
- Como é possível efetuar alinhamentos entre ontologias?
- Como é que se serializa uma DCO num determinado formato capaz de ser aceite pela ferramenta de limpeza de dados?

Estas questões levam a que seja necessário um estudo prévio sobre quais as abordagens/ferramentas a adotar para a sua resolução.

A primeira questão está relacionada com um dos requisitos da solução e consiste na necessidade de ler/processar ontologias que servem como forma de representação a artefactos como domínio e vocabulário e DCO. Tendo em conta este facto, torna-se necessária a inclusão de uma API que permita efetuar este tipo de operações. A escolha recaiu sobre a *framework* Apache Jena (Carroll et al., 2004), devido ao facto de fornecer uma interface completa para manipulação de conteúdo de ontologias.

Na abordagem da segunda questão é necessária uma análise ainda mais aprofundada para tentar definir de que forma é possível definir um contexto de reescrita. Num determinado contexto de reescrita, o objetivo é extrair o conteúdo correspondente ao mesmo e efetuar o seu processamento. Dando como exemplo a reescrita ao nível do domínio, o objetivo é extrair da operação unicamente as referências do domínio, deixando de parte as referências do vocabulário. Como o conteúdo de uma DCO está representado na forma de instância de uma ontologia, isto pode através da determinação dos predicados ou objetos dessa instância que estão contidos na ontologia de domínio, e em caso afirmativo, extrair esse conteúdo. Para esse efeito foi estudada a *framework* anteriormente referida na primeira questão, mas a mesma não suportava este tipo de operações. Por isso foi necessário procurar alternativas e foi encontrada uma solução através da OWL API que permite este tipo de validações.

Relativamente à terceira questão, nesta fase o objetivo é obter alinhamentos entre ontologias de uma forma simples de serem especificados/processados. Depois de uma análise sobre as soluções existentes e tendo em conta a linguagem de programação escolhida, foi adotada para efetuar este tipo de operações o Alignment API (David et al., 2011).

Na última questão é necessário estudar que tipo de soluções existem para a construção de instruções associadas a uma determinada linguagem. Depois de analisar várias soluções é concluído que uma ferramenta de gramáticas tornaria esta funcionalidade mais fácil de ser implementada. Como ferramenta para construção de gramáticas é escolhido o ANTLR, devido ao facto de suportar a linguagem de programação escolhida, como também devido à sua flexibilidade no contexto de implementação.

A partir da análise do modelo de domínio e das especificidades técnicas referidas anteriormente, foram identificados cinco componentes:

- Reescrita de operações: neste componente reside um conjunto de artefactos que permitem a execução da reescrita de uma DCO para uma posterior fase da sua serialização.
- DCO: este componente tem como responsabilidade a gestão de DCOs que são carregadas no sistema para serem reescritas.
- Modelos: este componente tem como objetivo providenciar um conjunto de operações para aceder a mapeamentos entre modelos e mapeamentos entre vocabulários, bem como o seu respetivo conteúdo.
- Serializador Ferramenta Limpeza Dados N: este componente tem como objetivo mostrar de que forma cada ferramenta de limpeza de dados é suportada pela solução. Cada ferramenta tem a sua própria implementação para serialização das operações, recorrendo a uma gramática que permite descrever o formato das operações e obedecendo a uma interface especificada através de uma interface para instanciação ao nível do processo de reescrita.
- Acesso a dados: este componente tem como objetivo disponibilizar um conjunto de operações de baixo nível que suportem o carregamento e manipulação das ontologias existentes no sistema (domínios e vocabulários), como também do mapeamento existente entre elas.

A Figura 25 mostra o diagrama de componentes com os componentes descritos anteriormente.

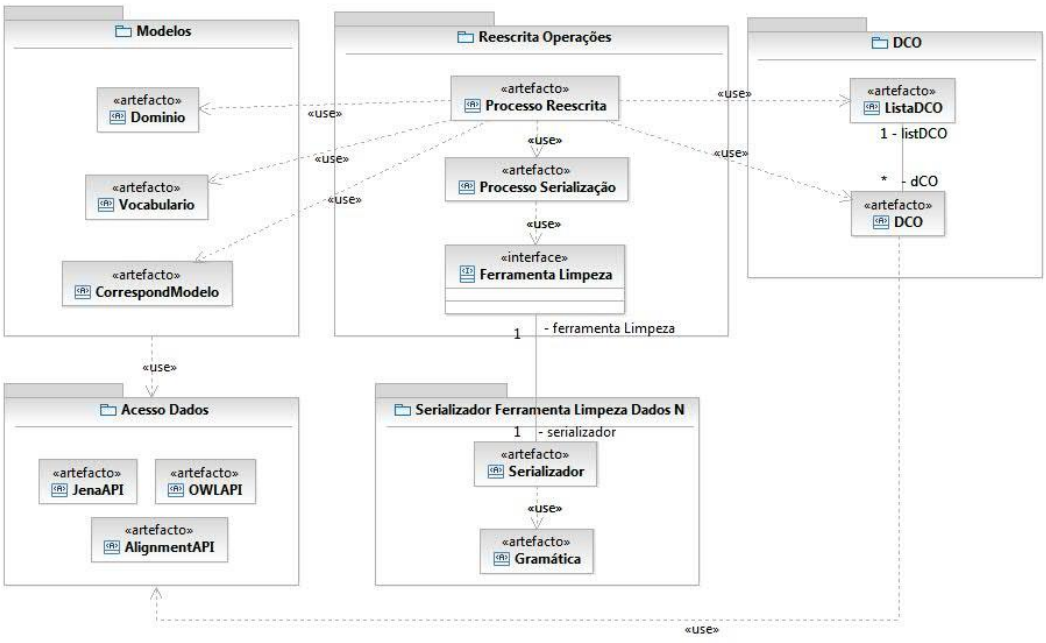


Figura 25 – Diagrama de componentes da solução

A partir do diagrama é possível ter uma visão de como a solução está estruturada, sendo fácil aumentar o suporte de serialização das operações para mais ferramentas de limpeza de dados, devido ao facto de existir uma interface (componente "Ferramenta Limpeza") apropriada para o efeito. Isto permite assim o desacoplamento relativamente à operação de reescrita (adoção do padrão *Strategy*).

O conteúdo existente a nível do modelo, vocabulário e mapeamentos encontra-se também desacoplado do processo de reescrita, existindo para o efeito uma interface apropriada para acesso e manipulação do seu conteúdo.

O componente auxiliar de acesso a dados permite que as operações de baixo nível como leitura/manipulação de ontologias e mapeamentos estejam acessíveis tanto para o carregamento de modelos como para processamento de determinados atributos de uma DCO. É neste componente que estão inseridas algumas das soluções externas referidas no início da secção.

Depois de descrita a arquitetura da solução é necessário descrever detalhadamente o fluxo de execução de um processo de reescrita de limpeza de dados. Neste contexto o foco será no caso de uso cinco identificado no capítulo requisitos.

6.3 Perspetiva detalhada

Nesta secção é descrito de uma forma detalhada o fluxo de execução do processo de reescrita. A descrição começa com a ordem de execução por parte do utilizador, terminando com a serialização das operações reescritas consoante o formato de saída especificado.

Para executar um processo de reescrita de um conjunto de operações de limpeza e dados é necessário que sejam fornecidos um conjunto de parâmetros:

- Operações a reescrever (**ListaDCO**).
- Vocabulário em que as operações estão escritas (**OV**).
- Vocabulário para o qual as operações vão ser reescritas (**OVT**).
- Ontologia de domínio em que as operações estão escritas (**OD**).
- Ontologia de domínio para o qual as operações vão ser reescritas (**ODT**).
- Correspondência entre DCOV e DCOVT (**CorrespondOV**).
- Correspondência entre OD e ODT (**CorrespondOD**).
- Formato para o qual a DCO será serializada (**formato_saida**).

O processo inicia com o carregamento dos parâmetros anteriormente referidos em instâncias de classes da solução. Estas instâncias permitem o acesso ao seu conteúdo através de métodos especificados de acordo com as necessidades do processo de reescrita, permitindo assim uma abstração relativamente ao conteúdo original dos ficheiros carregados (ontologias e alinhamentos entre ontologias).

A segunda fase consiste em carregar as operações a reescrever. Isso é feito criando instâncias da classe DCO que possui estruturas de dados vocacionadas para auxiliar na execução do processo de reescrita. Na prática o processo consiste em carregar em memória as instâncias da ontologia existentes no ficheiro "**ListaDCO"**.

Depois de carregadas as operações de limpeza de dados, é ordenada a reescrita de cada item de acordo com as fases existentes no algoritmo apresentado no inicio do capítulo. Neste caso a descrição seguinte é vocacionada para perceber de forma detalhada o que acontece em cada uma dessas fases:

- Reescrita da operação entre domínios: caso os domínios especificados como parâmetro sejam diferentes, esta operação é executada com vista a substituir as referências de domínio base existentes nos triplos que compõe a DCO pelas referências do domínio que se pretende reescrever.
- Reescrita da operação entre vocabulários: caso os vocabulários especificados como parâmetro sejam diferentes, esta operação é executada com vista a substituir as referências do vocabulário base existentes nos triplos que compõe a DCO pelas referências de vocabulário que se pretende reescrever.

O último passo consiste na serialização da DCO reescrita para o formato de saída especificado no argumento.

Para assegurar o suporte a múltiplas ferramentas de limpeza de dados, é necessário aplicar uma solução que reduza o acoplamento entre a operação de reescrita e o processo de serialização da operação para uma determinada ferramenta de limpeza de dados. A instanciação da classe responsável pela serialização da operação é realizada recorrendo aos padrões de *design*: *Factory+Singleton+Strategy*.

A figura 21 mostra de uma forma simplificada através de um diagrama de sequência o processo de reescrita de operações.

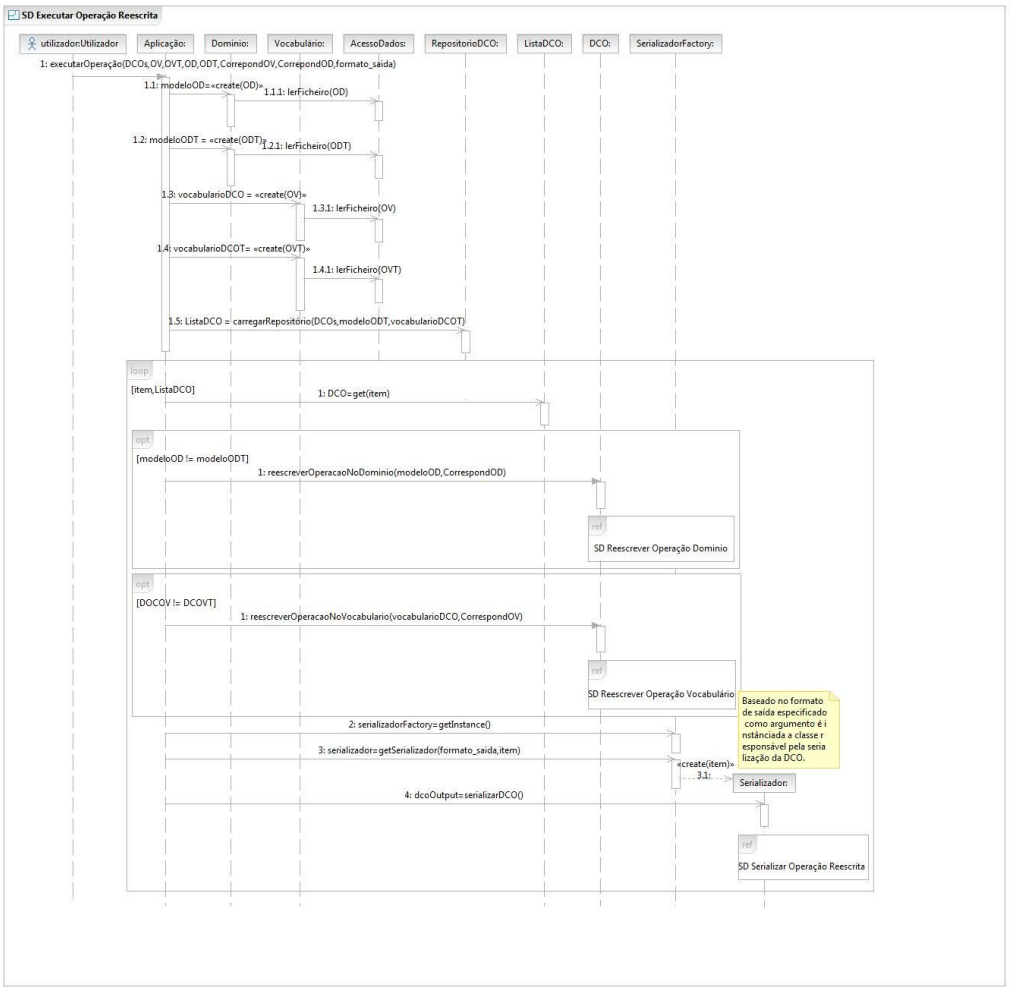


Figura 26 – Diagrama de sequência "Executar Operação Reescrita"

Depois de descrito de forma simplificada o processo de reescrita de operações de limpeza de dados, o passo seguinte consiste em descrever com detalhe cada uma das fases referidas anteriormente.

A fase de reescrita da operação de limpeza de dados consoante o domínio está dividida nos seguintes passos:

1. Obtenção dos elementos existentes nos triplos e que pertencem à ontologia de domínio base.
2. Carregando os alinhamentos entre domínios especificados no ficheiro como parâmetro (**CorrespondOD**). Os alinhamentos são carregados nesta fase porque de acordo com o algoritmo é determinado se a operação necessita ou não de ser reescrita ao nível do domínio, não fazendo sentido carregar logo esta informação no arranque.
3. Procurar as correspondências existentes no respetivo ficheiro de alinhamentos.
4. Atualização do conteúdo dos triplos que compõe a DCO com as correspondências encontradas e persistência do conteúdo reescrito numa estrutura de dados diferente, com o objetivo de preservar a DCO original.

A Figura 27 mostra o fluxo de execução na reescrita da operação relativamente ao domínio.

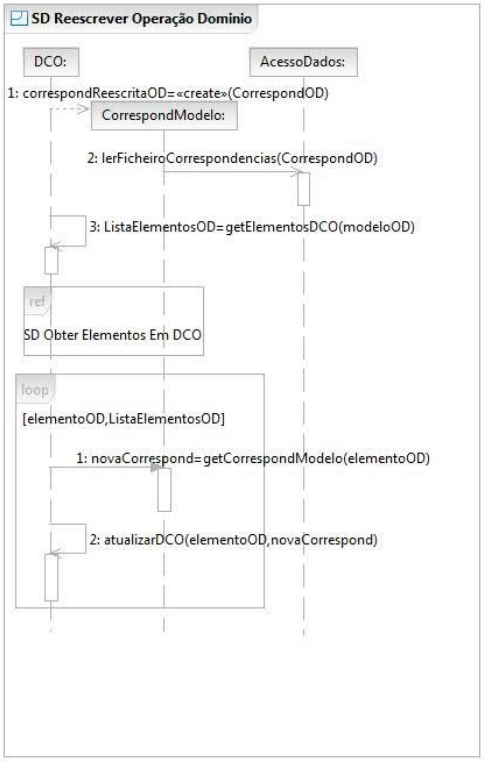


Figura 27 – Diagrama de sequência "Reescrever Operação Domínio"

Como é possível observar na Figura 27, inicialmente são carregados os alinhamentos que vão ser utilizados neste contexto de reescrita. De seguida são obtidos os elementos existentes na DCO e que pertencem à ontologia de domínio base. Esta operação é reutilizada em outros processos, visto que só necessita de uma ontologia base e dos triplos que fazem parte da

operação de limpeza de dados, retornando uma lista de referências que servem de base na pesquisa de correspondências.

Na prática esta operação consiste na verificação da existência do sujeito, predicado ou objeto na ontologia enviada como parâmetro para a função, adicionando o seu URI numa lista a ser retornada no final. Para esta operação são utilizadas as funcionalidades do OWL API encapsuladas numa classe designada por "OntologyManager".

A Figura 28 mostra o diagrama de sequência que ilustra a fase de execução descrita anteriormente.

Figura 28 – Diagrama de sequência "Obter Elementos Em DCO"

O segundo passo consiste na reescrita de operações de reescrita relativamente aos vocabulários utilizados como parâmetro.

Basicamente o fluxo de execução é semelhante à reescrita ao nível do domínio, mudando só a ontologia (**vocabularioDCO**) de onde é feita a verificação para extrair o conteúdo existente na DCO e o ficheiro de correspondências (**CorrespondOV**).

Comparando a Figura 27 com a Figura 29 abaixo apresentada, é possível confirmar a alteração dos parâmetros referidos anteriormente. Isto leva a que o funcionamento de reescrita entre diferentes contextos (domínio versus vocabulário) funcione de forma uniforme ao longo da solução.

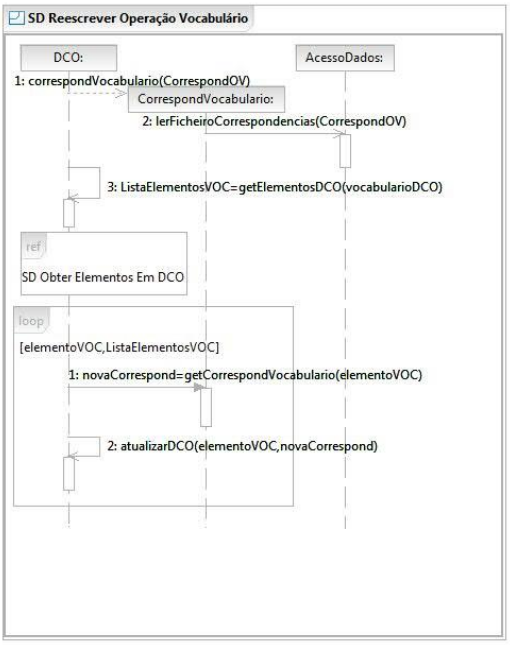


Figura 29 – Diagrama de sequência "Reescrever Operação Vocabulário"

Depois de descrever o processo de reescrita ao nível dos triplos existentes na DCO, o último passo consiste na serialização da mesma consoante o formato de saída especificado como argumento pelo utilizador.

Nesta fase surge um conjunto de dúvidas relativas à abordagem a adotar para a implementação desta parte do processo de reescrita.

- De que forma a gramática deve estar estruturada para permitir a construção de instruções suportadas pela ferramenta de limpeza de dados?
- Qual o conteúdo necessário para a execução de um processo de serialização?
- De que forma esse conteúdo deve estar estruturado?
- Onde é que esse conteúdo deve estar?
- Como pode ser extraído esse conteúdo?
- Como deve ser utilizado esse conteúdo para a construção da operação serializada?

Relativamente à primeira questão, a gramática é definida por *tokens* em que cada *token* possui um significado semântico, podendo ser atribuída a responsabilidade de construir a parte da

instrução que representa. Nesse sentido, é adotada como estratégia de serialização o desenho de gramática orientada aos *tokens* e ao conteúdo necessário para os mesmos puderem construir a parte da instrução que lhes compete.

Relativamente à segunda questão, cada elemento que compõe a DCO e está representado na instrução depois de ser serializada deve conter o respetivo conteúdo de serialização para ser processado pelo *token* da gramática.

Por isso, toda a informação relacionada com a serialização das operações deve estar estruturada de forma a indicar qual o elemento da DCO que representam, em que contexto é utilizada (caso seja necessário) e o valor a ser utilizado na construção da instrução serializada.

No que toca ao armazenamento deste tipo de informação (terceira questão), faz sentido que a mesma se encontre embebida nos ficheiros carregados previamente na solução a par dos elementos do domínio e vocabulário utilizados para a reescrita da operação, visto que existe uma associação entre eles.

A Figura 30 permite perceber de que forma a informação deve ser inserida para reescrita da operação através da representação do diagrama de classes da ontologia criada para representar o vocabulário da ferramenta de limpeza de dados (neste caso o SmartClean). O conteúdo sinalizado a verde corresponde ao das instâncias que devem ser criadas na ontologia para suportar o processo de serialização. Este formato consiste na referência da instância ao tipo *token* e do elemento que se pretende serializar, contendo o respetivo valor de serialização na propriedade "serializetoken".

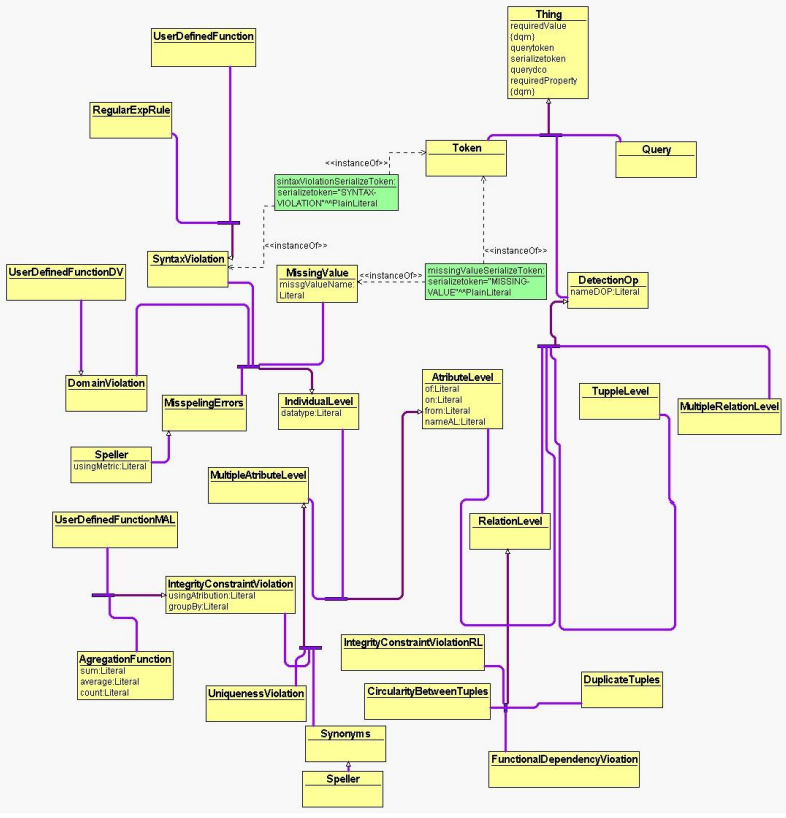


Figura 30 – Diagrama de classes da ontologia de vocabulário SmartClean

Relativamente à forma como podemos obter o conteúdo de serialização desejado para processar determinado *token* da gramática (quarta questão), o conteúdo de serialização está contido em ontologias e devidamente identificado para o contexto em que deve ser utilizado (através da resposta à segunda questão), por isso, é possível a sua obtenção através de uma consulta SPARQL devidamente formulada de acordo com o conteúdo que se pretende extrair. Isto leva a que na prática exista uma consulta SPARQL para cada *token* existente na gramática.

A última questão levantada relaciona-se com a fase terminal deste processo em que é preciso decidir de que forma vai ser utilizado o conteúdo extraído anteriormente para a construção da instrução final. Como a extração de informação é realizada ao nível do *token*, faz sentido que cada um possua a respetiva estrutura de dados para armazenar essa informação.

A Figura 31 mostra o diagrama de classes onde consta as estruturas de dados que armazena as referências de serialização extraídas da DCO e os métodos invocados pela gramática para cada *token*.

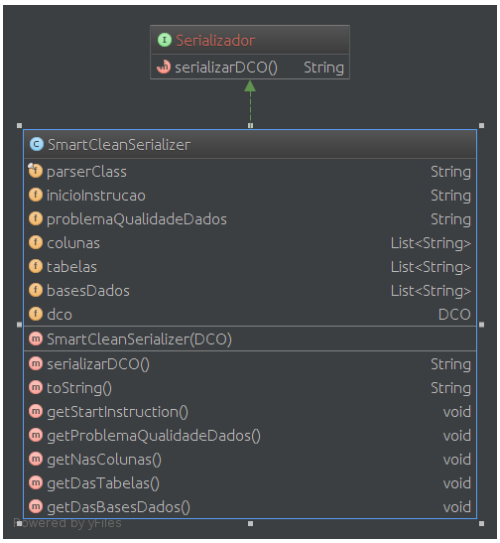


Figura 31 – Classe SmartCleanSerializer

Depois de respondidas as questões levantadas anteriormente, é possível contextualizar o processo de serialização que é descrito de seguida. O mesmo começa por instanciar a respetiva classe responsável pelo processo de serialização. A partir dessa instância é possível navegar/processar cada elemento que compõe as operações suportadas (*token* da gramática). Para cada elemento é solicitado o seu processamento para recolha do conteúdo consoante o seu tipo. Se o conteúdo estiver relacionado com o vocabulário, a implementação da função é realizada na classe que permite instanciar o vocabulário, caso contrário a implementação é feita na classe que permite instanciar o domínio. Essa informação serve de base para a serialização da operação e consiste basicamente em converter o conteúdo existente nos triplos em literais que façam parte da instrução devidamente serializada. Essa informação é armazenada na instância do serializador para no fim ser executado o método que permite construir a instrução final.

A Figura 32 mostra o diagrama de sequência que permite descrever o processo de serialziação de uma DCO de acordo com a descrição textual efetuada anteriormente.

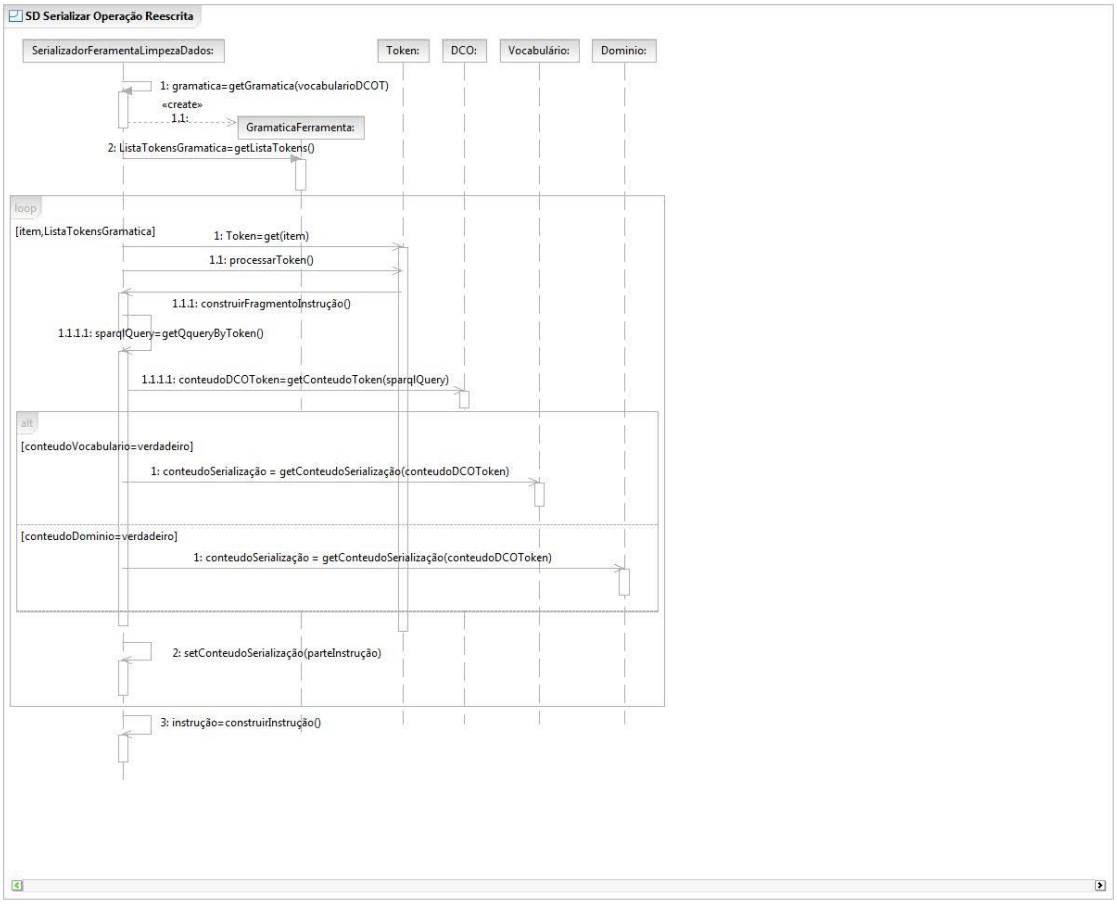


Figura 32 – Diagrama de sequência “Serializar Operação Reescrita”

Depois de concluída a análise do problema e de elaborado o seu *design*, existe uma base para a sua implementação. O capítulo seguinte consiste na descrição do processo de implementação da solução.

7 Implementação

Neste capítulo é descrita a fase de implementação da solução que marca a transição entre uma abordagem conceptual ao problema e a respetiva elaboração do *design*, para a sua resolução sobre o ponto de vista prático.

A descrição da solução é realizada de acordo com um conjunto de etapas que foram identificadas a partir do *design* da solução na perspetiva detalhada, utilizando para isso excertos de código e imagens, com o objetivo fornecer informação importante sobre a implementação da solução sem demasiado detalhe.

Depois de descritas as etapas da implementação, segue-se a validação da solução sob o ponto de vista interno através de testes aos módulos que a compõem.

7.1 Etapas da Implementação

Depois de analisado o problema e da realização do seu respetivo *design*, é necessário definir as etapas de implementação necessárias para construir a solução. De acordo com o *design* efetuado numa perspetiva detalhada, é possível dividir o processo de implementação nas seguintes fases:

1. Carregamento do conteúdo na solução.
2. Execução de um determinado contexto de reescrita.
    a. Obtenção de elementos consoante contexto de reescrita.
    b. Pesquisa de alinhamentos entre ontologias.
    c. Construção da operação reescrita com base nos alinhamentos obtidos na fase anterior.
3. Implementação do processo de atribuição do serializador da ferramenta de limpeza de dados ao processo de reescrita.
4. Definição da estrutura da linguagem suportada pela ferramenta de limpeza de dados
5. Implementação do processo de serialização da operação.

Para cada uma das fases é descrita de uma forma mais aprofundada a sua materialização ao nível da implementação como também caso se venha a verificar possíveis desvios que tenham ocorrido relativamente ao *design* elaborado. As duas últimas fases são descritas juntamente com o exemplo prático devido ao facto de as mesmas abordarem a implementação especifica da serialização das operações para uma determinada ferramenta de dados.

7.1.1 Carregamento do conteúdo na solução

A primeira fase da implementação consiste na implementação do mecanismo de carregamento do conteúdo utilizado na reescrita para as respetivas estruturas de dados. De acordo com o algoritmo apresentado e com o diagrama de sequência apresentado na Figura 26 foi criado um

método estático que recebe os argumentos de entrada e carrega o seu conteúdo através da criação das instancias das respetivas classes, tendo como argumento o ficheiro que vão utilizar para carregar os dados. Os nomes terminados pela letra "T" permitem identificar os argumentos utilizados como objetivo na operação de reescrita (domínio e o vocabulário).

O seguinte excerto de código mostra de que forma é feito o carregamento dos dados, começando pelo carregamento dos dois domínios (base e objetivo), dos vocabulários (base e objetivo), as respetivas correspondências entre domínios e vocabulários e por fim a lista de operações a serem processadas, sendo o seu conteúdo convertido em instâncias da classe DCO.

```
public static void reescreverOperacoes
 (String dominioFile, String dominioTFile,
 String vocabularioFile, String vocabularioTFile,
 String correspondDominioFile, String correspondVocabularioFile, String
 dcosFile, String formatoSaida){
    //Dominio
    Dominio dominio = new Dominio(dominioFile);
    Dominio dominioT = new Dominio(dominioTFile);
    System.out.println("Dominios instanciados");
    //Vocabulario e Correspondencias (são carregados de forma igual aos
domínios)
    //DCOs
    RepositorioDCO rep =
              new
        RepositorioDCO(dcosFile,dominio,dominioT,vocabulario,
        vocabularioT);
        List<DCO> listDCO = rep.getDcoList();
   //Processamento da lista de DCOs…
}
```

Este excerto de código é importante, porque permite mostrar de que forma é feita a ponte entre o *design* e a implementação, sendo a partir do mesmo que são realizadas todas as seguintes fases do processo de reescrita que é descrito daqui em diante.

7.1.2 Execução de um determinado contexto de reescrita

A segunda fase consiste em processar o conteúdo da operação base para contruir a DCO reescrita a nível triplos. Para tal, é necessário implementar uma funcionalidade capaz de extrair o conteúdo de uma determinada DCO consoante o contexto de reescrita que se pretende a aplicar (ao nível do vocabulário ou do domínio).

O seguinte excerto de código permite determinar que contexto de reescrita deve ser aplicado a uma determinada DCO através da comparação dos domínios/vocabulários especificados como parâmetros (caso sejam diferentes, é executado o contexto de reescrita) e qual os parâmetros necessários para a operação ser efetuada (domínio/vocabulário base e respetivos alinhamentos).

```
//reescrever domínio?
if(!dominio.equals(dominioT)){
    dco.reescreverOperacaoNoDominio(dominio,correspondDominio);
}
//reescrever vocabulário?
if(!vocabulario.equals(vocabularioT)){
       dco.reescreverOperacaoNoVocabulario(vocabulario,correspondVoca
bulario);
}
```

Depois de determinado o contexto de reescrita, o mesmo é executado através da obtenção dos elementos a processar através do método "getElementosEmDCO" retornando uma lista de referências. Esta lista é iterada, sendo invocado o método "getCorrespondencia" que tem como argumento a referência a utilizar para a pesquisa de alinhamentos. Caso seja encontrado esse alinhamento, a DCO é atualizada através do método "atualizarDCO" que utiliza como parâmetro a referência atual do elemento que possui e a nova referência obtida através da pesquisa nos alinhamentos. Como exemplo é dado o excerto de código que implementa a reescrita ao nível do domínio. A nível do vocabulário o corpo de método é praticamente o mesmo, mudando unicamente o primeiro argumento que é um objeto do tipo "Vocabulario".

```
public void reescreverOperacaoReescritaDominio
              (Dominio dominio, CorrespondenciasModelo correspond){
   System.out.println("A reescrever dominio");
   List<String>
elementosEmDCO=getElementosEmDCO(dominio.getOntologia());
   for (String elemento: elementosEmDCO) {
     try {
       System.out.println("De:"+elemento);
       String objetivo=correspond.getCorrespondencia(elemento);
       System.out.println("Para:"+ objetivo);
        if(objetivo!=null)
            atualizarDCO(elemento, objetivo);
        else{
            System.out.println("Correspondência     não     encontrada
(Dominio): "+elemento);
        }
     } catch (AlignmentException e) {
         System.out.println("Não foi possível processar de reescrita");
     }
  }
}
```

Para não detalhar demais a descrição da implementação do processo de reescrita, os métodos referidos anteriormente vão ser descritos de uma forma breve, porque já foi identificada a sua

função neste contexto e porque os mesmos abordam aspetos de implementação de baixo nível como o acesso a dados/manipulação de dados.

7.1.2.1 Obtenção de elementos consoante o contexto de reescrita

A primeira fase que foi definida para o processo de reescrita consiste na obtenção dos elementos da DCO a serem processados. Para isso foi implementado um método com a seguinte assinatura:

```
List<String> getElementosEmDCO(OWLOntologia ontologia)
```

Através da ontologia recebida como argumento é feita a verificação dos triplos contidos na DCO para determinar quais os predicados e os objetos que pertencem à ontologia.

7.1.2.2 Pesquisa de alinhamentos entre ontologias

Através da descrição do exemplo anterior foi despoletada uma pesquisa de alinhamentos que consiste na pesquisa de uma determinada referência no formato de URI e retornar o respetivo alinhamento. Como já foi referido, os alinhamentos encontram-se especificados num formato suportado pelo Alignment API.

Cada alinhamento está contido numa célula devidamente identificada, sendo a mesma composta por duas entidades. Cada entidade possui um URI que a identifica, sendo o mesmo utilizado para a pesquisa de correspondências. Por último existe a medida de correspondência que toma valores de 0 até 1 (formando uma escala de 0 a 100%) e o elemento "*relation*" que contém diferentes tipos de operadores de comparação que mostram de que forma se encontram relacionadas as entidades.

Com o objetivo de entender como é feita essa pesquisa é dado um exemplo de utilização do método “getCorrespondencia”:

- Parâmetro: “http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-ontology-20#**Customer**”.
- Retorno: ”http://www.semanticweb.org/fabiosantos/ontologies/2016/4/untitled-ontology-49#**Buyer**”.

Para suportar este exemplo é dado um excerto do ficheiro que contém a configuração de alinhamentos. De acordo com o exemplo este método acha na entidade 1 o parâmetro que recebeu e retorna o valor da entidade 2.

```
<Cell cid='1'>
      <entity1
rdf:resource='http://www.semanticweb.org/fabiosantos/ontologie
s/2016/1/untitled-ontology-20#Customer'/>
      <entity2
rdf:resource='http://www.semanticweb.org/fabiosantos/ontologie
s/2016/4/untitled-ontology-49#Buyer'/>
      <measure rdf:datatype='xsd:float'>1.0</measure
      <relation>=</relation>
</Cell>
```

7.1.2.3 Construção da operação reescrita com base nos alinhamentos obtidos na fase anterior

A última fase do processo de reescrita é a construção da DCO de acordo com o conteúdo obtido na pesquisa do alinhamento, para isso é criado um método com a seguinte assinatura.

```
void atualizarDCO(String oldURI,String newURI)
```

Este método utiliza unicamente URIs para a pesquisar o triplo onde tem de realizar a intervenção e efetuar substituição do elemento em questão.

7.1.3 Implementação do processo de atribuição do serializador da ferramenta de limpeza de dados ao processo de reescrita

A implementação do processo de atribuição do serializador da ferramenta de dados ao processo de escrita é feita recorrendo aos padrões de *design Factory+Singleton+Strategy*.

O seguinte excerto de código mostra de que forma pode ser instanciado o serializador de operações das ferramentas de dados.

```
//Instanciar a fabrica de instâncias através de um método estático
SerializadorFabrica
factory=SerializadorFactory.getSerializadorFabrica();
//Obtenção do serializador consoante o formato especificado (retorna a
devida instância que implementa a interface Serializador)
Serializador serializador = factory.getSerializador(formatoSaida,dco);
//ordem para executar o processo de serialização
String operacaoSerializada = serializador.serializarDCO();
```

Como é possível observar no extrato de código apresentado, em primeiro lugar é instanciada a fábrica de instâncias a partir de um método estático de acordo com o padrão *Singleton*. O seguinte excerto de código mostra de que forma é instanciada.

```
public static synchronized SerializadorFabrica
getSerializadorFabrica (){
        if(instancia==null)
            instancia= new SerializadorFabrica();
        return instancia;
}
```

Depois de obtida a fábrica de instâncias é invocado o método que permite retornar a respetiva instância que implementa a interface do serializador, de acordo com o formato especificado no primeiro argumento e enviando para o seu construtor a DCO a serializar.

O seguinte extrato de código mostra de que forma foi implementado esse método:

```
        public Serializador getSerializador(String formatoOutput,DCO
    dcoIN){
            Serializador serializador =null;
            switch (formatoOutput){
                case("smartclean"):
                    serializador = new SmartCleanSerializer(dcoIN);
                break;
            }
            return serializador;
        }
```

Como é possível observar no extrato de código anterior, é retornada instância do tipo "Serializador" que implementa a seguinte interface para a execução de serialização de operações de limpeza de dados:

```
public interface Serializador {
    String serializarDCO();
}
```

Na secção seguinte é apresentado um exemplo prático que permite suportar a descrição da implementação do processo de serialização (fases 4 e 5).

7.2 Exemplo prático

Depois de descrito o processo de implementação das fases genéricas da solução de uma forma sintetizada na secção anterior, esta secção tem como objetivo descrever o processo, mas recorrendo agora a um exemplo prático para demonstrar de que forma é aplicado.

Para descrever a implementação é utilizada uma DCO em RDF/OWL para ser reescrita em SmartClean. Esta operação permite a deteção de valores em falta num atributo.

```
<owl:NamedIndividual rdf:about="&new;PropertyCompletenessRule1">
    <rdf:type rdf:resource="&dqm;PropertyCompletenessRule"/>
    <dqm:requiredProperty
rdf:datatype="&xsd;boolean">true</dqm:requiredProperty>
    <dqm:requiredValue rdf:datatype="&xsd;boolean">true</dqm:requiredValue>
    <dqm:testedClass rdf:resource="&new;Customer"/>
    <dqm:testedProperty rdf:resource="&new;name"/>
</owl:NamedIndividual>
```

Para executar a reescrita desta operação é necessário definir um conjunto de alinhamentos ao nível do vocabulário e do domínio de acordo com o formato mostrado anteriormente.

Para definir estes alinhamentos é necessário consultar o conteúdo base em que a DCO foi elaborada (domínio base e E-DQM) e o conteúdo utilizado para a sua reescrita (domínio objetivo e vocabulário SmartClean). Depois de consultar esse conteúdo é necessário extrair as referências necessárias e construir o respetivo ficheiro de acordo com o tipo de reescrita que se deseja realizar (ao nível do domínio ou do vocabulário).

Para a reescrita ao nível do vocabulário são necessários os seguintes alinhamentos:

E-DQM	SmartClean	Medida	Relacionamento
PropertyCompletenessRule	MissingValue	1.0	=
testedClass	from	1.0	=
testedProperty	on	1.0	=

Tabela 5: Alinhamentos exemplo prático ao nível do vocabulário

Para a reescrita ao nível do domínio são necessários os seguintes alinhamentos:

Domínio base	Domínio objetivo	Medida	Relacionamento
Customer	Buyer	1.0	=
name	name	1.0	=

Tabela 6: Alinhamentos exemplo prático ao nível do domínio

Depois de terminada a definição dos alinhamentos, é necessário implementar o processo de serialização do conteúdo reescrito. Para isso, é necessário criar uma gramática que permita construir instruções aceites pelo SmartClean.

A gramática é constituída por um conjunto de *tokens* que permitem identificar cada parte da instrução. A título de exemplo é utilizado o tipo de problema de dados que se encontra representado pelo *token* "problemaqualidadedados". Cada *token* para ser processado leva como argumento a classe responsável por serializar a operação (SmartCleanSerializer), sendo executado o respetivo método criado para a serialização daquela parte da instrução.

O seguinte excerto de código mostra como a gramática é constituída ao nível de *tokens* e quais são métodos instanciados na classe responsável pela serialização da operação:

```
instrucao [SmartCleanSerializer serializer]:
inicioinstrucao[serializer]
                                    problemaqualidadedados[serializer]
                                   nascolunas[serializer]
                                   dastabelas[serializer]
                                   dasbasesdados[serializer]
                                    usandocondicao[serializer]
                                    usandodicionario[serializer];
inicioinstrucao [SmartCleanSerializer serializer]: {
    serializer.getInicioInstrucao();
};
problemaqualidadedados [SmartCleanSerializer serializer]:{
    serializer.getProblemaQualidadeDados();
};
```

```
nascolunas [SmartCleanSerializer serializer]:{
    serializer.getNasColunas();
};

dastabelas [SmartCleanSerializer serializer]:{
    serializer.getDasTabelas();
};

dasbasesdados [SmartCleanSerializer serializer]:{
    serializer.getDasBasesDados();
};

usandocondicao [SmartCleanSerializer serializer]:{
    serializer.getCondicao();
};

usandodicionario [SmartCleanSerializer serializer]:{
    serializer.getUsandoDicionario();
};
```

O *token* inicial é designado por "instrucao" e tem como argumento uma instância da classe responsável por serializar a operação. Na definição do *token* inicial são especificados os restantes que compõem a instrução pela respetiva ordem em que aparecem na mesma, tendo o mesmo argumento para a sua execução. Ao especificar individualmente cada *token*, é determinado que método da classe "SmartCleanSerializer" é executado para obter o respetivo conteúdo de serialização a ser guardado numa estrutura interna da classe.

O fluxo de execução do método que processa um *token* da gramática pode ser descrito de uma forma geral a partir das seguintes operações:

1. Extração do conteúdo da DCO que permite pesquisar pelo conteúdo necessário para a serialização do *token* através de uma consulta SPARQL.
2. Extração do conteúdo de serialização contido no vocabulário ou domínio destino, utilizando como parâmetro numa consulta SPARQL o conteúdo extraído no passo anterior.

Como exemplo são escolhidos dois *tokens* da gramática, o primeiro é o tipo de operação para mostrar esta fase do lado do vocabulário e o segundo é o *token* relacionado com as colunas da instrução onde vai ser usado como exemplo a propriedade ”*name*” do lado do domínio.

Na DCO dada como exemplo, a sua reescrita vai resultar na operação do SmartClean designada “MissingValue”. A Figura 30 anteriormente apresentada mostra o conteúdo inserido para esta serialização.

Quanto ao conteúdo inserido ao nível da ontologia de domínio, o formato das instâncias segue a mesma estrutura implementada para a ontologia de vocabulário.

A Figura 33 mostra a instância criada a partir da ferramenta protégé que permite serializar propriedade "name" do exemplo prático dado a nível do domínio. A partir da figura é possível visualizar as classificações atribuidas a esta instâncias (*Types*), como também a propriedade com o valor do token de serialização (*serializatoken*).

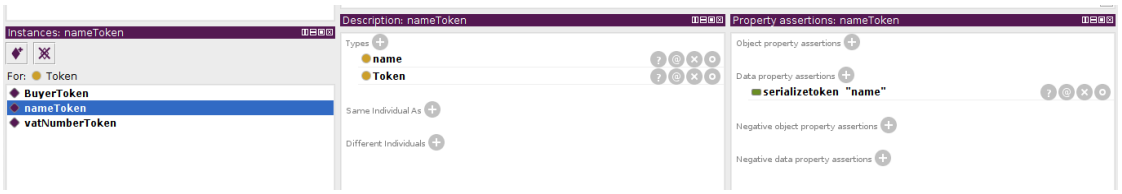


Figura 33 – Instância para serialização da operação ao nível do domínio

Como exemplo do tipo de consultas SPARQL criadas, é descrito o caso do *token* "problemaqualidadedados". O primeiro passo consiste em extrair qual o tipo de problema de dados contido na DCO através da pesquida do triplo que tem como predicado "rdf:type", retornando o valor que está contido no objeto do triplo.

```
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
PREFIX owl: http://www.w3.org/2002/07/owl#
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
PREFIX sc:
http://www.semanticweb.org/ralmeida/ontologies/2016/1/untitled-
ontology-20#
SELECT ?subject ?detectionOp
WHERE { ?subject rdf:type ?detectionOp }
```

No segundo passo é criada uma consulta SPARQL para ser executada sobre o vocabulário. A seguinte consulta mostra de que forma é feita a recolha do conteúdo que permite serializar qual o tipo de problema de qualidade de dados. O conteúdo utilizado para definir a consulta consiste na utilização do parâmetro "**detectionOp**" (obtido na execução do 1º passo) para obter qual a operação correspondente no SmartClean, da especificação do tipo da instância a procurar (*Token*) e do predicado "*serializetoken*" para recolha do conteúdo a utilizar na serialização da DCO.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sc:
<http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-
ontology-20#>
SELECT ?token
```

```
WHERE { ?subject rdf:type sc:Token .
        ?subject rdf:type ?detectionOp .
        ?subject sc:serializetoken ?token
}
```

Depois de recolhido o conteúdo de serialização (no caso da consulta SPARQL anterior o resultado é extraído através da variável ?token), o mesmo é armazenado numa estrutura de dados com o objetivo de construir a instrução final.

Por último, depois de serem processados todos os *tokens* da gramática e preenchidas todas as estruturas de dados necessárias para realizar o processo de serialização da operação, é executado o método "toString" que utiliza essas mesmas estruturas de dados para gerar a instrução final devidamente serializada.

O seguinte extrato de código mostra como essas estruturas são utilizadas para gerar a instrução final.

```
public String toString(){
        String instrucaoFinal= getInicioInstrucao()+
                        " "+problemaQualidadeDados+
                        " ON "+ getColunasInstrucao()+
                        " FROM "+getTabelasInstrucao()+
                        " OF "+ getBasesDadosInstrucao()+
                        getCondicoesInstrucao()+
                        getUsandoDicionarioInstrucao()+
                        getUsandoTabelasInstrucao();
        return instrucaoFinal;
}
```

Como podemos observar neste extrato de código, são utilizados diferentes métodos auxiliares para a construção da instrução final. Neste momento eles podem ser classificados em três tipos distintos:

- Simples: retornam uma única palavra para a construção da instrução
- Lista de elementos: retornam um conjunto de palavras separadas por virgulas
- Construção completa de parte da instrução: retorna uma expressão que permite substituir na integra parte da instrução.

O seguinte excerto de código mostra os três tipos de funções.

```
//Simples
public String getStartInstruction(){
       return "DETECT";
}
```

```
//Lista de elementos
private String getColunasInstrucao(){
        return org.apache.commons.lang3.StringUtils.join(colunas,",");
}
//Construção completa de parte da instrução
private String getUsandoTabelasInstrucao() {
        String tabelasAUsar="";
        if(usandoTabelas.size()>0){
                tabelasAUsar+=" USING TABLES
     "+org.apache.commons.lang3.StringUtils.join(usandoTabelas,",");
        }
        return tabelasAUsar;
}
```

Depois de concluída a descrição do processo de implementação através de um exemplo prático, é necessário validar numa primeira fase se a solução concebida funciona conforme o esperado. Para executar esta fase preliminar de testes foram escolhidos testes unitários como forma de validar o sistema numa perspetiva interna. Este tipo de testes permite avaliar o funcionamento de forma isolada de cada módulo que compõe um processo de reescrita.

7.3 Validação da solução

Nesta secção é descrita a validação preliminar efetuada à solução concebida. O objetivo desta validação é assegurar que os principais requisitos definidos para a solução e o funcionamento dos seus módulos estão a ser obedecidos. Em primeiro lugar é necessário definir quais são os módulos que necessitam de ser validados na solução.

Tendo em conta o fluxo de execução da operação de reescrita, foram identificados os seguintes testes unitários:

- Carregamento dos domínios
- Carregamento dos vocabulários
- Obtenção de alinhamentos ao nível do vocabulário
- Obtenção de alinhamentos ao nível do modelo
- Teste de conteúdo que não pode existir nos alinhamentos

Através dos testes unitários descritos anteriormente é possível agrupá-los em dois grupos distintos:

- Carregamento de conteúdo na solução
- Pesquisa de alinhamentos

A título de exemplo é apresentado o código utilizado para o teste de carregamento do vocabulário objetivo e da pesquisa de um determinado mapeamento.

```
@Test
public void carregamentoVocabularioObjetivo(){
        Assert.assertNotNull("Vocabulario carregado com sucesso", new
Vocabulario(BasicTests.class.getClassLoader().getResource(diretorioBas
e+"sc.owl").toString()););
}


@Test
public void testeAlinhamentoDominioSucesso(){
        String
correspondRes=correspondDominio.getTargetEntity("http://www.semanticwe
b.org/ralmeida/ontologies/2016/1/untitled-ontology-20#Customer");

        Assert.assertEquals("http://www.semanticweb.org/ralmeida/ontol
        ogies/2016/4/untitled-ontology-49#Buyer",correspondRes);
}
```

8 Experimentação e avaliação

Neste capítulo é descrita a fase de experimentação e avaliação da solução anteriormente concebida. O objetivo consiste em mostrar de que forma a mesma responde aos requisitos que lhe foram impostos, tendo em conta um conjunto de parâmetros utilizados para sua avaliação.

Em primeiro lugar é necessário formalizar o processo de experimentação e avaliação tendo em conta um conjunto de questões que por sua vez resultam num conjunto de tarefas:

- O que realmente é necessário avaliar? (Definição de objetivos)
- De que forma se pretende avaliar a solução? (Definição do processo de avaliação)
- O que preciso para efetuar o processo de experimentação e avaliação? (Preparação do ambiente de testes)
- De que forma se pretende classificar os resultados dos testes (Definição do tipo de classificação)
- Quais as conclusões que se pretende extrair? (Definição do tipo de conclusões a extrair de um determinado teste)

As tarefas que resultaram na resposta às questões levantadas anteriormente ditam a estrutura deste capítulo.

Na secção de definição de objetivos é realizada a divisão em categorias. Para cada categoria é feita uma descrição e respetiva justificação da sua existência e quais os objetivos que se pretendem atingir. Na secção de definição do processo são descritos os pré-requisitos necessários para a realização dos testes e formalizada a estrutura adotada para a descrição de cada um. Na secção de ambiente de teste é descrito de uma forma aprofundada de que forma foram obtidos os pré-requisitos referidos na secção anterior, quais as necessidades existentes para a execução dos testes e qual a ferramenta escolhida para o mesmo propósito, justificando a sua escolha. Por último é elaborada uma análise global de todas as conclusões extraídas em cada um dos testes, permitindo assim determinar no geral os pontos fortes/fracos da solução elaborada.

8.1 Definição de objetivos

Antes de definir o processo formal de experimentação e avaliação a adotar, é necessário definir o que realmente é importante avaliar na solução concebida. Realizando este tipo de exercício é possível definir quais os objetivos que se pretende atingir. Os objetivos definidos nesta fase têm um impacto direto no tipo de testes que vão ser efetuados, porque o seu teor depende diretamente deles.

Resultante de uma macro análise aos testes que se pretende realizar na solução, conclui-se que os mesmos podem ser divididos em duas categorias:

- Capacidade de reescrita
- Interpretação da operação/resultado

8.1.1 Capacidade de reescrita

Esta categoria de objetivos surge no âmbito de testar o processo de reescrita de DCOs em duas dimensões diferentes que são tipos/complexidade de DCOs. Quando é referido o termo "tipos" neste contexto, o objetivo é referenciar diferentes tipos de DCOs. Relativamente à sua complexidade, a mesma é inferida através da dificuldade de configurar a ferramenta/serializador das operações para a sua reescrita.

A Figura 34 tem como objetivo mostrar graficamente de que forma é possível enquadrar uma determinada DCO de acordo com as duas dimensões referidas. A escala tem valores entre 0 (nível baixo), 0.5 (nível intermédio) e 1 (nível elevado). Cada ponto representa uma DCO e seu respetivo enquadramento nas dimensões apresentadas. O ponto de cor verde representa uma DCO fácil de reescrever, o ponto laranja representa uma DCO com um nível intermédio de dificuldade de reescrita e os pontos vermelhos significam que as DCOs que se encontram nessa situação têm uma elevada dificuldade de reescrita por parte da solução.

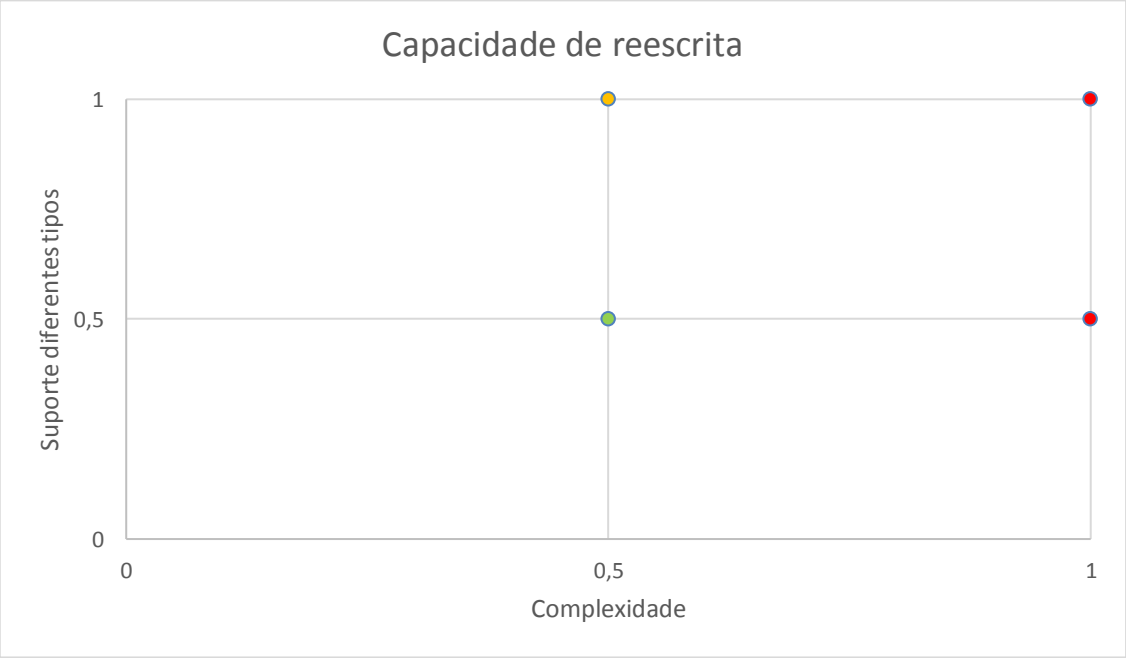


Figura 34 – Gráfico de análise da capacidade de reescrita da solução

Como é possível observar no gráfico, não constam DCOs com complexidade baixa, nem as que se enquadram no baixo suporte de reescrita porque não é o objetivo dos testes abordar esses casos.

A partir da realização deste tipo de testes é possível determinar se o processo de reescrita consegue responder perante diferentes cenários resultantes da conjugação das duas dimensões referidas anteriormente.

8.1.2 Interpretação da operação/resultados

Nesta categoria é avaliada a qualidade da informação gerada durante e no final do processo de reescrita. Este tipo de avaliação é necessário devido à complexidade das operações efetuadas durante a reescrita, sendo indispensável a disponibilização de informação que permita ao utilizador entender o funcionamento da solução.

Ao avaliar esta categoria é pretendido obter as seguintes conclusões:

- Como é que a solução se comporta no caso da existência de anomalias.
- Dificuldade de configuração/utilização.
- Fiabilidade dos resultados obtidos.

Neste momento é percetível quais os objetivos que se pretende atingir no processo de experimentação e avaliação. O passo seguinte consiste em definir de que forma é possível atingir esses objetivos através da formalização do processo.

8.2 Definição do processo

Esta secção tem como objetivo formalizar e descrever a estrutura do processo de experimentação e avaliação a adotar.

A fase inicial do processo está relacionada com o conteúdo necessário para efetuar os testes. De acordo com os objetivos definidos na secção anterior, é necessário definir em primeiro lugar quais as DCOs que vão ser consideradas para teste e o seu respetivo conteúdo de configuração a ser carregado na solução (descrito na secção seguinte).

Depois de definir as DCOs utilizadas para teste, é efetuado um teste para cada uma. A estrutura do teste consiste no seguinte:

- Contextualização: de acordo com os objetivos definidos, determinar quais os objetivos cobertos pelo teste a executar.
- Resultado: determinar o sucesso/insucesso do teste e o grau de automatização do processo de reescrita.
- Conclusão: de acordo com o resultado obtido e com os objetivos propostos, analisar o que foi atingido e o que não foi atingido, sendo no último caso necessária uma justificação.

Depois de realizados todos os testes, é feito um resumo dos resultados obtidos bem como uma conclusão geral sobre os mesmos.

Depois de descrever formalmente o processo de experimentação e avaliação, é necessário descrever qual o ambiente de testes utilizado para a realização dos mesmos.

8.3 Ambiente de teste

Para a execução de experiências é necessário determinar em que condições as mesmas decorrem através da definição de casos de uso, como também das ferramentas a utilizar na sua elaboração.

Nesta secção é descrito qual é o conteúdo utilizado onde se encontra incluído as DCOs e respetivos ficheiros de configuração da solução, como também a ferramenta utilizada para a elaboração das mesmas.

8.3.1 Conteúdo de teste

Os casos de uso definidos para experimentação da solução consistem na utilização de dados relacionados com colheitas de sangue. Esta informação é utilizada para a elaboração de um conjunto de DCOs para reescrita na ferramenta SmartClean.

Antes de definir as DCOs num ficheiro para serem testadas, foram necessários os seguintes passos:

- Conversão das instâncias/esquema relacional numa ontologia de domínio.
- Captura dos conceitos relacionados com saúde numa ontologia de domínio base.

O passo seguinte consiste em criar as DCOs que vão ser submetidas a teste na solução. Para o efeito foi criada uma ontologia que tem como referências externas o vocabulário em que as DCOs são definidas (E-DQM) e a ontologia de domínio criada para este propósito.

Para cada uma DCO testada é necessário efetuar um conjunto de procedimentos, podendo eles ser generalizados da seguinte forma:

- Definição da DCO no ficheiro que contem as DCOs a testar.
- Criação de alinhamentos entre elementos dos dois domínios utilizados no processo de reescrita
- Criação de alinhamentos entre elementos dos dois vocabulários utilizados no processo de reescrita
- Inserção do conteúdo para a serialização da operação no domínio objetivo da reescrita
- Inserção do conteúdo para a serialização da operação no vocabulário objetivo da reescrita

Como forma de permitir a compreensão do conteúdo gerado pela solução na execução dos testes, são apresentados os alinhamentos entre domínios e vocabulários.

E-DQM	SmartClean	Medida	Relacionamento
PropertyCompletenessRule	MissingValue	1.0	=
testedClass	from	1.0	=
testedProperty	on	1.0	=
SyntaxRule	SyntaxViolation	1.0	=
regex	Like	1.0	=
LegalValueSynonymsRule	ExistenceOfSynonyms	1.0	=
trustedClass	usingTable	1.0	=
UniqueValueRule	UniquenessViolation	1.0	=
LegalValueRangeRule	IntegrityConstraintViolation	1.0	=
lowerLimit	LowerLimit	1.0	=
LegalValueRule	MisspellingErrors	1.0	=

Tabela 7: Alinhamentos utilizados na experimentação ao nível do vocabulário

Domínio base	Domínio objetivo	Medida	Relacionamento
Person	dadorestotalaux	1.0	=
date_of_birth	DNasc	1.0	=
Gender	tabauxsexo	1.0	=
gender	Sexo	1.0	=
identityCard	CartaoIdentificacao	1.0	=
weight	Peso	1.0	=

Tabela 8: Alinhamentos utilizados na experimentação ao nível do domínio

Tendo todo o conteúdo para a execução das experiências, é necessário utilizar uma ferramenta para a elaboração dos mesmos. A secção seguinte descreve quais os requisitos que a ferramenta deve conter, qual foi a ferramenta escolhida e de que forma a mesma foi utilizada.

8.3.2 Ferramentas utilizadas

Nesta secção são abordadas as necessidades que existem ao nível de ferramentas que permitem a execução das referidas experiências, qual a ferramenta escolhida como os respetivos critérios que levaram à sua escolha e por fim uma breve descrição de como a mesma é utilizada neste processo de experimentação.

Como principais requisitos para executar as experiências referidas é necessário que exista um carregamento prévio do conteúdo na solução (domínios, vocabulários, alinhamentos e DCOs a testar) e a possibilidade de executar de forma isolada o processo de reescrita para uma determinada DCO de forma automática.

Este tipo de testes está incluído num cariz mais funcional do que os testes que foram elaborados no capítulo de implementação, porque neste caso o objetivo é testar o processo de reescrita como um todo e não as diferentes partes que o compõe de forma individual. Tendo em conta o cariz funcional das experiências a realizar, foi escolhida uma ferramenta de desenvolvimento orientada ao comportamento (*Behaviour Driven Development*) designada por cucumber (Wynne and Hellesøy, 2012).

Esta ferramenta permite a criação de testes a partir de ficheiros especificados na linguagem Gherkin (Wynne and Hellesøy, 2012, p. 4). Num ficheiro é especificada a funcionalidade que é testada (atributo *Feature*), o cenário base a ser utilizado em todos os testes (atributo *Background*), bem como os respetivos casos de uso a testar (atributo *Scenario*).

Cada caso de teste é especificado através de uma história que abrange um conjunto de passos com o seguinte formato: dada (*Given*) uma DCO, então (*Then*) o resultado da sua reescrita tem de ser o seguinte "extrato de texto". Cada passo corresponde a um método Java que é gerado automaticamente de acordo com os argumentos contidos no mesmo (texto ou números), sendo necessário preencher o seu corpo.

O seguinte excerto de código em inglês mostra de que forma estão estruturados os testes efetuados nesta ferramenta:

```
Feature: Reescrita Operacoes SmartClean
Background:
 Given a base domain "health.owl"
 And target domain "ips1.owl"
 And base vocabulary "v.owl"
 And target vocabulary "sc.owl"
 And correspondence domain "alAPI-health-ips1.rdf"
 And correspondence vocabulary "alAPI-v-sc03-05-2016.rdf"
 And DCO list "DCOsTeste.owl"
 And target serializer tool "smartclean"

Scenario: Syntax Violation Rewrite
  Given the DCO "DCO2"
  Then result of DCO2 rewrite should be "DETECT SYNTAX-VIOLATION ON
DNasc FROM dadorestotalaux OF db where DNasc NOT LIKE ^(0[1-9]|[12][0-
9]|3[01])-(0[1-9]|1[0-2])-(19[0-9]{2}|2[0-9]{3})"
```

A partir da análise efetuada e a ferramenta escolhida, é possível concluir que a mesma suprime o conjunto de necessidades referidas para a execução das experiências, como também se encontra adaptada para o tipo de testes a realizar.

Depois de configurada a ferramenta para a elaboração de testes com o conteúdo anteriormente referido, o passo seguinte consiste na elaboração dos mesmos.

8.4 Testes

A primeira categoria de experiências a ser abordada é a capacidade de reescrita, devido à sua importância no âmbito da validação da solução. Nesta categoria vão ser testadas um conjunto de DCOs individualmente.

Na categoria seguinte é abordado um aspeto secundário na validação da solução que consiste na interpretação da operação/resultados. Nesta categoria é definido um conjunto de informações interessantes de se obter num contexto de utilização da aplicação.

8.4.1 Capacidade de reescrita

Ao nível da capacidade de reescrita, os testes são realizados com base em DCOs definidas de acordo com os objetivos a atingir, comparando os resultados desejados com os resultados obtidos pela solução.

O processo de experimentação passa por contextualizar o tipo de experiência a realizar, respetivo resultado e conclusões obtidas através da execução de cada operação testada (caso seja possível) ou através de uma análise aos problemas que a mesma levantou.

8.4.1.1 DCO1

Entrada (E-DQM)	Saída (SmartClean)
:DCO1 a v:PropertyCompletenessRule; v:testedClass s:Person; v:testedProperty1 s:date_of_birth; v:requiredProperty “true” ^^ xsd:Boolean; v:requiredValue “true” ^^ xsd:Boolean.	DETECT MISSING-VALUE ON DNasc FROM dadorestotalaux OF db

Tabela 9: Cenário de teste DCO1

- Contextualização: neste teste o objetivo é mostrar de que forma é possível a reescrita de operações ao nível da propriedade, mais propriamente do tipo "Property Completeness Rule". De acordo com a Figura 34, este teste enquadra-se na classificação (0.5,0.5), o que significa um nível médio de complexidade/suporte na reescrita.
- Resultado: operação reescrita com sucesso de forma automática.
- Conclusão: para operações que utilizem unicamente classes e propriedades a serem testadas, a solução consegue ser configurada de uma forma simples e reescrever as operações com sucesso.

8.4.1.2 DCO2

Entrada (E-DQM)	Saída (SmartClean)
:DCO2 a v:RegularExpression; v:regex “^(0[1-9]|[1|2][0-9]|3[0-1])-(0[1-9]|1[0-2])-(19[0-9]{2}|2[0-9]{3}) v:compliance “ISO/IEC/IEEE 9945:2009” v:appliesSSR v:RegularExpression_dco2_2; v:testedClass s:Person; v:testedProperty1 s:date_of_birth.	DETECT SYNTAX-VIOLATION ON DNasc FROM dadorestotalaux OF db where DNasc NOT LIKE ^(0[1-9]|[12][0-9]|3[01])-(0[1-9]|1[0-2])-(19[0-9]{2}|2[0-9]{3})

Tabela 10: Cenário de teste DCO2

- Contextualização: neste teste o objetivo é mostrar a correta reescrita de operações ao nível da propriedade (neste caso do tipo "Syntax Rule"), utilizando uma expressão

regular como condição. De acordo com a Figura 34, este teste enquadra-se na classificação (1,0.5), o que significa uma elevada complexidade na reescrita e um nível médio de suporte.

- Resultado: a operação foi reescrita com sucesso de uma forma automática. No entanto, ao configurar a solução surgiu uma questão relativamente à aplicação do mapeamento entre “v:regex” e o respetivo operador no Smartclean, porque na maioria dos casos era utilizado o operador “NOT LIKE” mas podem existir casos em que essa não é a opção a ter em conta. Por isso, foi adotado que no caso de reescrita de operações que utilizem expressões regulares, as mesmas são reescritas para "*NOT LIKE*" por predefinição, a não ser que o utilizador especifique no mapeamento que quer utilizar o operador “*LIKE*” predefinição.

- Conclusão: de acordo com a situação referida no resultado, a nível de alinhamento de ontologias do vocabulário, “v:regex” corresponde a “sc:like” ou a “sc:notlike” de acordo com as preferências do utilizador, sendo esta regra aplicada na reescrita de todas as operações que utilizem esse mapeamento. Uma possível melhoria a implementar no futuro consiste em perguntar ao utilizador que operador quer utilizar na reescrita de uma operação em concreto.

8.4.1.3 DCO3

Entrada (E-DQM)	Saída (SmartClean)
DCO3: a dqm:LegalValueRule ; rdfs:label "Valor valido conclusoes"^^xsd:string ; v:referenceClass TrustedClassConclusoes; v:referenceProperty1 TrustedPropertyConclusoes; v:testedClass Conclusoes; v:testedProperty1 conclusoes.Designacaoconclusao.	DETECT MISSPELLING-ERRORS ON conclusoesDesignacaoconclusao FROM conclusoes OF db USING DICTIONARY DicPortugues USING METRIC Jaro-Winkler

Tabela 11: Cenário de teste DCO3

- Contextualização: o objetivo deste teste consiste em verificar o suporte da reescrita de operações que validem um conjunto de valores válidos para uma determinada propriedade. Esta operação podia ser associada ao tipo “*Domain Violation*”, no entanto o objetivo neste teste é testar operações deste género que envolvam uma maior complexidade. No caso prático de reescrita para SmartClean, esta operação leva a que seja necessário configurar o processo de reescrita para que permita a inclusão de dicionários na operação e à análise relativamente à questão de suporte a métricas (algo que é necessário definir neste tipo de operações). De acordo com a Figura 34, este teste enquadra-se na classificação (1,1), o que significa uma elevada complexidade/suporte na sua reescrita.

- Resultado: neste caso o processo de reescrita é semiautomático.

- Conclusão: Não é possível automatizar a reescrita para operações deste tipo, devido à impossibilidade de especificar qual o dicionário/métrica a utilizar na reescrita da operação. Isto acontece porque não existe nenhuma referência da mesma na operação

definida em E-DQM, por isso cabe ao utilizador inserir qual o dicionário e métrica que deseja.

8.4.1.4 DCO4

Entrada (E-DQM)	Saída (SmartClean)
DCO4: a UserDefFunctionDV; v:fName idCardValidation; v:fType “Function”; v:fDescrprition “Function in Java that allows the verification of the number of an identity card”; v:fLocation “define an URI”; v:testedClass s:Person; v:testedProperty1 s:identityCard.	DETECT DOMAIN-VIOLATION ON idCard FROM Dadores OF IPS WHERE PortIdentCardValidation(idCard,nidCard) == False

Tabela 12: Cenário de teste DCO4

- Contextualização: o objetivo neste teste consiste em verificar o suporte de reescrita para funções definidas pelo utilizador para determinados problemas de qualidade de dados que só conseguem ser detetados desta forma. De acordo com a Figura 34, este teste enquadra-se na classificação (1,1), o que significa uma elevada complexidade/suporte na sua reescrita.

- Resultado: não foi possível reescrever a operação.

- Conclusão: neste momento o E-DQM não evoluiu o suficiente para possuir uma forma bem clara de definir operações com funções em que seja possível descrever propriedades como o seu retorno, número e tipo dos argumentos. Para além disso muitas funções são criadas pelos utilizadores para suprimir uma necessidade em especifico, não existindo o conceito de reutilização que está implícito na reescrita de operações.

8.4.1.5 DCO5

Entrada (E-DQM)	Saída (SmartClean)
DCO5: a LegalValueRule_synonyms v:testedClass s:Person; v:testedProperty1 s:gender; v:trustedClass aux:Gender; v:trustedProperty1 aux:desc.	DETECT EXISTENCE-OF-SYNONYMS ON sexo FROM Person OF db USING DICTIONARY Generos

Tabela 13: Cenário de teste DCO5

- Contextualização: neste teste o objetivo é mostrar a correta reescrita de operações relacionadas com a deteção de sinónimos. Esta deteção pode ser feita ao nível da coluna ou ao nível das múltiplas relações e necessita da especificação de um dicionário. De acordo com a Figura 34, este teste enquadra-se na classificação (1,0.5), o que significa uma elevada complexidade na reescrita e um nível médio de suporte.

- Resultado: a operação reescrita com sucesso de forma semiautomática.

- Conclusão: tal como no teste efetuado na DCO3, concluiu-se que não é possível reescrever de forma automática a operação na íntegra, porque não existe um mapeamento direto para o dicionário a utilizar na aplicação em SmartClean a partir do E-DQM.

8.4.1.6 DCO6

Entrada (E-DQM)	Saída (SmartClean)
DCO6: a FuncDepReferenceRule; v:testedClass Class_Colheitas; v:testedProperty1 Colheitas.abo; v:testedProperty2 Colheitas.dador.	DETECT FUNCTIONAL-DEPENDENCY-VIOLATION ON Abo FROM Colheitas OF BDD DEPENDENT ON Dador

Tabela 14: Cenário de teste DCO6

- Contextualização: o objetivo deste teste consiste em determinar a capacidade de reescrita para operações que contenham uma dependência funcional em que o valor de um determinado atributo depende de outro atributo. De acordo com a Figura 34, este teste enquadra-se na classificação (1,1), o que significa uma elevada complexidade/suporte na sua reescrita.

- Resultado: não foi possível reescrever a operação

- Conclusão: como é possível concluir, o E-DQM não tem associado o conceito de dependência como o SmartClean, logo como na operação não existe referência a esta particularidade, ao reescrever para SmartClean, não existe forma de obter essa parte da instrução. Uma possível solução para reescrever esta DCO passaria por alterar o E-DQM para reservar uma propriedade relacionada com valores permitidos a este tipo de operações. Isto leva a que seja possível aplicar um alinhamento e determinar que este tipo de conteúdo da DCO encontra-se associado à área de dependências (DEPENDENT ON) do SmartClean. Neste caso outra questão pode-se levantar a questão de começar a existir acoplamento entre o E-DQM e o SmartClean, sendo isso algo indesejável!

8.4.1.7 DCO7

Entrada (E-DQM)	Saída (SmartClean)
DCO7: a DuplicateInstanceRule; v:testedClass ex:Person; v:testedProperty1 ex:identityCard; v:testedProperty2 ex:name.	DETECT DUPLICATE-TUPLES FROM tabelaExemplo T1 T2 OF DB USING ATTRIBUTES identCard, name WHERE T1.identCard = T2. identCard AND T1. Nome = T2. Nome

Tabela 15: Cenário de teste DCO7

- Contextualização: o objetivo deste teste consiste em determinar a capacidade de reescrita em operações de deteção de duplicados, onde existe um conjunto de particulares a ter em conta como por exemplo a estratégia utilizada para a sua deteção e de que forma os dados são utilizados na mesma. Para este teste foi definido de uma forma simples uma operação em E-DQM e definido a sua reescrita para SmartClean de acordo com uma determinada estratégia vocacionada para este tipo de operações. De

acordo com a Figura 34, este teste enquadra-se na classificação (1,1), o que significa uma elevada complexidade/suporte na sua reescrita.

- Resultado: Não foi possível reescrever a operação

- Conclusão: como é possível observar, a instrução em E-DQM só contem a classe e atributos utilizados para a deteção de duplicados, mas não a forma como eles são utilizados para efetuar a respetiva operação. Uma possível solução para reescrever esta operação passa por alterar o comportamento de serialização tendo em conta este tipo de operação. Neste caso, a operação "DUPLICATE-TUPLES" do SmartClean precisa que a área de condições da operação seja construída de acordo com o tipo de operação, existindo uma estratégia por defeito para a sua execução. Isto leva a que seja levantada a seguinte questão, será que o processo de reescrita deve basear-se unicamente no conteúdo existente na DCO em E-DQM, realizando pequenas alterações para a sua reescrita, ou deve existir uma estratégia bem definida na implementação do serializador para a construção de diferentes tipos de operações de acordo com os requisitos impostos pelo SmartClean? Neste caso em especifico o utilizador deveria especificar de que forma pretende que os duplicados sejam detetados, caso contrário deveria ser adotado o comportamento definido como predefinido pelo sistema para esse tipo de operações.

8.4.1.8 DCO8

Entrada (E-DQM)	Saída (SmartClean)
DCO8: a UniqueValueRule; v:testedClass s:Person; v:testedProperty1 s:identityCard.	DETECT UNIQUENESS-VIOLATION ON CartaoCidadao FROM dadorestotalaux OF db

Tabela 16: Cenário de teste DCO8

- Contextualização: neste teste o objetivo é demostrar a correta reescrita para operações de multi atributo no SmartClean. De acordo com a Figura 34, este teste enquadra-se na classificação (0.5,0.5), o que significa um nível médio de complexidade/suporte na reescrita.

- Resultado: a operação foi reescrita com sucesso de uma forma automática

- Conclusão: devido ao facto de esta operação só necessitar de um tipo de operação, classe a ser testada e o respetivo atributo, isto faz com que a sua complexidade não seja elevada. Logo a sua reescrita é suportada pela solução sem necessitar de algum esforço suplementar.

8.4.1.9 DCO9

Entrada (E-DQM)	Saída (SmartClean)
DCO9: a v:LegalValueRangeRule; v:testedClass s:Pdata; v:testedProperty1 s:weight; v:lowerLimit “50” ^xsd:int.	DETECT Integrity Constraint Violation ON Peso FROM dadorestotalaux OF db where Peso > 50

Tabela 17: Cenário de teste DCO9

- Contextualização: o objetivo deste teste consiste na reescrita de operações que contém condições aplicáveis a atributos numéricos. De acordo com a Figura 34, este teste enquadra-se na classificação (1,0.5), o que significa uma elevada complexidade na reescrita e um nível médio de suporte.

- Resultado: a operação foi reescrita com sucesso de uma forma automática.

- Conclusão: para suportar este tipo de operações aritméticas que não se encontram explicitas no E-DQM sob a forma de símbolos, é necessário um mapeamento para um atributo na ontologia no SmartClean e aplicar o respetivo *token* de serialização onde se encontra o símbolo responsável pela definição da condição.

Relativamente ao conjunto de DCOs testadas, é possível observar que as que possuem uma estrutura simples foram concluídas com sucesso e operações mais complexas como funções definidas pelo utilizador e operações que não incluam informação importante para serialização em SmartClean não foram reescritas com sucesso.

Depois de efetuados os testes de reescrita às DCOs escolhidas, foram experienciadas um conjunto de dificuldades no que toca à interpretação dos processos que compõem a reescrita, como também a análise ao resultado final. A secção seguinte tem como objetivo avaliar um conjunto de pontos associados ao objetivo de interpretação da operação/resultados.

8.4.2 Interpretação da operação/resultados

Nesta secção são avaliados aspetos suplementares ao funcionamento da solução, mas que assumem um papel relevante para o utilizador, porque através dos mesmos é possível perceber o seu funcionamento.

Tal como foi referido no capítulo de implementação da solução, a mesma foi construída por fases através de um desenvolvimento incremental. Tal como o desenvolvimento da solução, o processo de reescrita possuí um conjunto de fases em que é necessário determinar qual é o ponto inicial e em que estado se encontra após a sua execução.

Para isso foi criado um mecanismo de explicação do processo em execução que mostra a seguinte informação:

- DCO no formato E-DQM a ser serializada
- Fases de reescrita:

    - Tipo de reescrita: mostra os elementos extraídos da DCO consoante o tipo de reescrita e a respetiva correspondência obtida para cada elemento a nível dos alinhamentos carregados na solução.
    - Alteração do conteúdo da DCO: mostra de que forma foi alterada parte da DCO, de acordo com a correspondência obtida na DCO.
- DCO reescrita
- DCO serializada

O seguinte excerto de execução mostra um exemplo da informação fornecida pela ferramenta durante um processo de reescrita. Em primeiro lugar é mostrado o nome da DCO que está a ser processada, seguido do conjunto de fases que compõe o processo de reescrita. É mostrado de inicio a DCO no formato original, o conteúdo que foi processado ao nível do domínio, o conteúdo que foi processado ao nível do vocabulário, a DCO devidamente reescrita para serialização e por fim o respetivo resultado serializado para SmartClean. Para facilitar a leitura do excerto de código, são realçadas as partes mais importantes.

Em primeiro lugar é mostrada qual é a operação que está a ser reescrita no momento através do seu nome e respetivo conteúdo.

```
Integrity Constraint Violation Teste (DCO9)

DCO Base:

[http://www.semanticweb.org/fabiosantos/ontologies/2016/7/untitled-
ontology-56#DCO9,

http://purl.org/dqm-vocabulary/v1/dqm#lowerLimit,
"50"^^http://www.w3.org/2001/XMLSchema#integer]

[http://purl.org/dqm-vocabulary/v1/dqm#testedProperty,
http://www.semanticweb.org/fabiosantos/ontologies/2016/6/untitled-
ontology-107#weight]

[http://purl.org/dqm-vocabulary/v1/dqm#testedClass,
http://www.semanticweb.org/fabiosantos/ontologies/2016/6/untitled-
ontology-107#Person]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#type,

http://purl.org/dqm-vocabulary/v1/dqm#LegalValueRangeRule]

---------------------------------------------------------------
```

Depois de devidamente identificada a DCO, o passo seguinte consiste em mostrar se a mesma é reescrita ao nível do domínio. Caso isso aconteça é dada a indicação ao utilizador através de um título de estado da operação de reescrita, seguido dos elementos processados neste contexto e respetivos alinhamentos obtidos.

```
A reescrever domínio:

De:http://www.semanticweb.org/fabiosantos/ontologies/2016/6/untitled-
ontology-107#weight

Para:http://www.owl-ontologies.com/ips1.owl#dadorestotalaux.Peso

De:http://www.semanticweb.org/fabiosantos/ontologies/2016/6/untitled-
ontology-107#Person

Para:http://biostorm.stanford.edu/db_table_classes/DSN_jdbc.mysql.//lo
calhost/ips1#dadorestotalaux

.............................................................
```

Relativamente à reescrita ao nível do vocabulário, o procedimento é semelhante como é possível comprovar pelo seguinte excerto de execução:

```
A reescrever vocabulário:

De:http://purl.org/dqm-vocabulary/v1/dqm#lowerLimit

Para:http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled
-ontology-20#LowerLimit

De:http://purl.org/dqm-vocabulary/v1/dqm#testedProperty

Para:http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled
-ontology-20#on

De:http://purl.org/dqm-vocabulary/v1/dqm#testedClass

Para:http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled
-ontology-20#from

De:http://purl.org/dqm-vocabulary/v1/dqm#LegalValueRangeRule

Para:http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled
-ontology-20#IntegrityConstraintViolation
```

Depois de aplicados os diferentes contextos de reescrita e construída a DCO, a mesma é apresentada num formato semelhante ao de triplos. Isto permite ver de que forma são aplicados os alinhamentos obtidos.

```
DCO reescrita:

[http://www.semanticweb.org/fabiosantos/ontologies/2016/7/untitled-
ontology-56#DCO9,
http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-
ontology-20#LowerLimit,

"50"^^http://www.w3.org/2001/XMLSchema#integer]

[http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-
ontology-20#on,

"http://www.owl-ontologies.com/ips1.owl#dadorestotalaux.Peso"]

[http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-
ontology-20#from,
"http://biostorm.stanford.edu/db_table_classes/DSN_jdbc.mysql.//localh
ost/ips1#dadorestotalaux"]

[http://www.w3.org/1999/02/22-rdf-syntax-ns#type,
"http://www.semanticweb.org/fabiosantos/ontologies/2016/1/untitled-
ontology-20#IntegrityConstraintViolation"]
```

No final da execução do processo de reescrita é indicado que a operação está a ser serializada e apresentado o respetivo resultado.

```
A serializar:

DETECT Integrity Constraint Violation ON Peso FROM dadorestotalaux OF
db where Peso > 50
```

A partir da informação gerada pela ferramenta, é possível determinar que a mesma apoia na tarefa de solucionar os seguintes problemas que podem ocorrer durante utilização da solução:

- Resolução de elementos em falta nas DCOs carregadas na solução
- Resolução de problemas relacionados com a errada classificação de elementos do domínio ou do vocabulário

- Resolução de alinhamentos em falta
- Resolução de problemas na construção da DCO reescrita
- Resolução de problemas que podem ocorrer na serialização da DCO, através da deteção da falta de parte(s) da instrução serializada

Pode-se concluir que além da possibilidade de detetar um conjunto de problemas referidos anteriormente, a informação disponibilizada pela solução descreve o fluxo de execução do processo de uma forma transparente, como também o teor das operações realizadas. Isto vai de encontro ao objetivo do tipo de testes a realizar nesta categoria. Uma das fases que se encontra menos explorada ao nível da informação gerada pela ferramenta é o processo de serialização, devido à dificuldade para tipificar mensagens capazes de mostrar ao utilizador o teor das operações neste contexto.

8.5 Sumário

Através da realização de experiências na solução, é possível neste momento determinar qual é a sua capacidade de reescrita com base nas operações que conseguiu reescrever. No entanto ficam visíveis quais as limitações que existem a este nível. Essa noção é importante, porque através dela é possível retirar conclusões sobre quais foram os objetivos alcançados, quais os que ficaram por atingir e determinar qual é o contributo do trabalho efetuado.

9 Conclusões

Depois de executadas todas as experiências e de se extrair um conjunto de conclusões, torna-se necessário refletir sobre quais foram as conclusões obtidas num cômputo geral.

Essa reflexão é iniciada através da descrição de uma forma sumária os objetivos alcançados, contrapondo com os objetivos que não se conseguiu alcançar, seguindo-se de um balanço entre ambos.

Por fim, com base nas conclusões obtidas neste trabalho são apresentadas linhas condutoras para a elaboração de trabalho futuro.

9.1 Objetivos alcançados

Nesta secção é feita uma exposição dos objetivos alcançados na elaboração deste trabalho.

O principal objetivo consiste na capacidade de reescrita da solução para operações que podem ser reutilizadas em outro contexto. Este objetivo é alcançado através da possibilidade de reescrita de operações simples (normalmente são as operações mais utilizadas) que possuem na sua estrutura a(s) classe(s) a testar, atributos, e condições que seguem também um formato simplista.

Outro objetivo alcançado consiste na fácil interpretação dos dados gerados pela ferramenta através de uma forma clara e objetiva descrever o processo de reescrita a partir dos mesmos. A inclusão deste tipo de informação numa fase de experimentação permitiu corrigir alguns erros que ocorriam, devido ao elevado número de passos necessários para configurar a ferramenta. Isto vem comprovar a utilidade desta funcionalidade numa utilização rotineira da solução.

Por último, mas não menos importante foi alcançado o objetivo de construir uma solução que consegue suportar a reescrita de operações para diferentes ferramentas de limpeza de dados. Isto é possível porque a solução foi desenvolvida por módulos e cada ferramenta de limpeza de dados é considerada como um módulo externo (com uma implementação própria de serialização) que é independente do processo de reescrita, sendo possível escolher no final qual a ferramenta que se pretende utilizar.

9.2 Objetivos não alcançados

Nesta secção é feita uma reflexão sobre os objetivos que não se conseguiram alcançar na elaboração deste trabalho.

Neste momento ficam de fora da capacidade de reescrita da solução as operações complexas que são definidas para suprimir um problema de qualidade de dados muito especifico de um determinado repositório de dados, como por exemplo operações com funções definidas pelo

utilizador. Ficam também de fora as operações que necessitam de um conjunto de requisitos para serialização que não são percetíveis através do conteúdo contido na DCO em E-DQM. Isto levanta a questão da utilização do máximo das capacidades das ferramentas de limpeza através deste processo de reescrita.

Outro objetivo não alcançado consiste na resolução problemas levantados durante o processo de reescrita através da utilização de conteúdo introduzido pelo utilizador. Esta necessidade surge, porque por vezes é necessário indicar algumas opções que precisam de ser definidas para a serialização de uma determinada operação. Não foi possível implementar esta funcionalidade por falta de tempo para a abordar, como também devido ao facto de a necessidade da mesma surgir numa fase bastante avançada deste trabalho.

9.3 Balanço

Nesta secção é efetuada uma comparação entre os objetivos alcançados e aqueles que ficaram por alcançar, permitindo assim ter uma noção dos avanços alcançados em detrimento dos problemas que ficaram por resolver.

Ao traçar os objetivos para este trabalho, eram conhecidos antemão o conjunto de dificuldades para a realização deste tipo de soluções. No entanto, foi adotada uma abordagem simplista para a resolução do problema principal, crescendo gradualmente a complexidade consoante o possível para a resolução dos problemas que foram surgindo posteriormente. A questão que se coloca está relacionada com a aplicabilidade do conceito de reescrita. Normalmente estes processos são uteis quando se o pretendido é a reutilização de uma operação para aplicação noutro contexto, logo a automatização deste processo é uma mais-valia. Apesar de serem abordadas operações com elevado nível de complexidade para reescrita neste trabalho, as mesmas não costumam ser utilizadas para reescrita, servindo assim de uma forma geral para testar a capacidade máxima de reescrita da solução.

Neste momento já é possível reescrever operações por mais simples que sejam, por isso trata-se de um avanço que deve ser tido em conta, mesmo apesar de não se conseguir suportar todo o tipo de operações.

9.4 Trabalho futuro

Nesta fase final do trabalho são referidos um conjunto de pontos que podem servir de linhas orientadoras para evoluir o trabalho em questão com base nos objetivos não alcançados e sugestões de melhoria da solução.

Em primeiro lugar penso que seja pertinente orientar o trabalho futuro para o conjunto de objetivos que não foram alcançados neste trabalho e que abordam questões importantes no funcionamento da solução, como é o caso das operações de elevada complexidade e das operações semiautomáticas que que precisam da ação do utilizador.

No caso das operações com elevada complexidade é necessário determinar até que ponto certa informação deve estar contida no E-DQM para permitir a sua reescrita/serialização ou até que ponto é necessário evoluir o processo de serialização.

Relativamente a operações em que não é possível a reescrita de uma forma automática, é necessário analisar onde é que essa falha pode ser suprimida ou se realmente é uma limitação da reescrita ao nível do vocabulário que não pode ser contornada.

Depois de atingidos os pontos anteriormente referidos, seria interessante testar o processo de reescrita para mais ferramentas de limpeza de dados com o objetivo de comprovar a abordagem adotada para o problema em questão, bem como a nível de implementação testar a flexibilidade da solução para suportar diferentes ferramentas.

Uma possível melhoria da solução passa em primeiro lugar por suportar diferentes tipos de alinhamentos entre ontologias (mais complexos para além do atualmente suportado), como por exemplo o EDOAL, permitindo ao utilizador a possibilidade de escolher em que formato deseja fazer os alinhamentos. Isto permite tornar o processo de reescrita capaz de lidar com situações mais complexas.

Referências

Ranjan, 2009	Ranjan, J., 2009. Business intelligence: Concepts, components, techniques and benefits. J. Theor. Appl. Inf. Technol. 9, 60–70. Survey on NoSQL integration, 20:40:28 UTC.
Hansmann and Niemeyer, 2014	Hansmann, T., Niemeyer, P., 2014. Big Data - Characterizing an Emerging Research Field Using Topic Models, in: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT). Presented at the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), pp. 43–51. doi:10.1109/WI-IAT.2014.15
Hevner et al., 2004	Hevner, A.R., March, S.T., Ram, S., 2004. V y|^^ I^^^^ li^/Research Essay. MIS Q. 28, 75–105.
Kandel et al., 2011	Kandel, S., Paepcke, A., Hellerstein, J., Heer, J., 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11. ACM, New York, NY, USA, pp. 3363–3372. doi:10.1145/1978942.1979444
Raman and Hellerstein, 2001	Raman, V., Hellerstein, J.M., 2001. Potter's Wheel: An Interactive Data Cleaning System, in: Proceedings of the 27th International Conference on Very Large Data Bases, VLDB '01. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 381–390.
Hao and Xing-Chun, 2008	Hao, Y., Xing-Chun, D., 2008. The Design and Implementation of Data Cleaning Knowledge Modeling, in: International Symposium on Knowledge Acquisition and Modeling, 2008. KAM '08. Presented at the International Symposium on Knowledge Acquisition and Modeling, 2008. KAM '08, pp. 177–179. doi:10.1109/KAM.2008.114
Oliveira, 2009	Oliveira, P.J.M., 2009. Detecção e correcção de problemas de qualidade dos dados : modelo, sintaxe e semântica.
Kejriwal and Miranker, 2015	Kejriwal, M., Miranker, D.P., 2015. On the Complexity of Sorted Neighborhood. arXiv preprint arXiv:1501.01696.
Heflin and others, 2007	Heflin, J., others, 2007. An Introduction to the OWL Web Ontology Language. Lehigh University. National Science Foundation (NSF).
Almeida et al., 2015	Almeida, R., Maio, P., Oliveira, P., Barroso, J., 2015. An Ontology-based Methodology for Reusing Data Cleaning Knowledge: SCITEPRESS - Science and and Technology Publications, pp. 202–211. doi:10.5220/0005596402020211
Fürber and Hepp, 2011	Fürber, C., Hepp, M., 2011. Towards a Vocabulary for Data Quality Management in Semantic Web Architectures, in: Proceedings of the 1st International Workshop on Linked Web Data Management, LWDM '11. ACM, New York, NY, USA, pp. 1–8. doi:10.1145/1966901.1966903

Wiederhold, 1992	Wiederhold, G., 1992. Mediators in the Architecture of Future Information Systems. Computer 25, 38–49. doi:10.1109/2.121508
Calvanese et al., 2003	Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R., 2003. Data Management in Peer-to-Peer Data Integration Systems.
Arenas and Pérez, 2011	Arenas, M., Pérez, J., 2011. Querying Semantic Web Data with SPARQL, in: Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '11. ACM, New York, NY, USA, pp. 305–316. doi:10.1145/1989284.1989312
Chamberlin and Boyce, 1974	Chamberlin, D.D., Boyce, R.F., 1974. SEQUEL: A structured English query language, in: Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control. ACM, pp. 249–264.
Cyganiak, 2005	Cyganiak, R., 2005. A relational algebra for SPARQL.
Ma et al., 2008	Ma, L., Wang, C., Lu, J., Cao, F., Pan, Y., Yu, Y., 2008. Effective and Efficient Semantic Web Data Management over DB2, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08. ACM, New York, NY, USA, pp. 1183–1194. doi:10.1145/1376616.1376735
Garshol, 2003	Garshol, L.M., 2003. BNF and EBNF: What are they and how do they work. acedida pela última vez em 16.
Gallier and Hicks, 2014	Gallier, J., Hicks, A., 2014. The Theory of Languages and Computation. Lecture Notes, University of Pennsylvania, http://www. cis. upenn. edu/jean/gbooks/toc. pdf, Accessed December.
Chebotko et al., 2009	Chebotko, A., Lu, S., Fotouhi, F., 2009. Semantics preserving SPARQL-to-SQL translation. Data Knowl. Eng. 68, 973–1000. doi:10.1016/j.datak.2009.04.001
Elliott et al., 2009	Elliott, B., Cheng, E., Thomas-Ogbuji, C., Ozsoyoglu, Z.M., 2009. A Complete Translation from SPARQL into Efficient SQL, in: Proceedings of the 2009 International Database Engineering & Applications Symposium, IDEAS '09. ACM, New York, NY, USA, pp. 31–42. doi:10.1145/1620432.1620437
"What, no supplementary specification?," 2004	What, no supplementary specification? [WWW Document], 2004. URL http://www.ibm.com/developerworks/rational/library/3975.html (accessed 3.12.16).
"Capturing Architectural Requirements," 2005	Capturing Architectural Requirements [WWW Document], 2005. URL http://www.ibm.com/developerworks/rational/library/4706.html (accessed 3.12.16).
"Appendix B," 2004	Appendix B: Architectural Requirements [WWW Document], 2004. URL http://www.ibm.com/developerworks/rational/library/4708.html (accessed 3.12.16).

Carroll et al., 2004	Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., 2004. Jena: implementing the semantic web recommendations, in: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters. ACM, pp. 74–83.
David et al., 2011	David, J., Euzenat, J., Scharffe, F., Trojahn dos Santos, C., 2011. The alignment API 4.0. Semantic web 2, 3–10.
Wynne and Hellesøy, 2012	Wynne, M., Hellesøy, A., 2012. The cucumber book: behaviour-driven development for testers and developers, The pragmatic programmers. Pragmatic Bookshelf, Dallas, Tex.