INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Deteção de Problemas de Qualidade nos Dados

AUGUSTO MANUEL CRUZ GOMES
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Deteção de Problemas de Qualidade nos Dados

Augusto Manuel Cruz Gomes

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimento

Orientador: Paulo Oliveira

Júri:

Presidente:

Vogais:

Porto, 15 de Outubro 2020

Dedicatória

Dedico esta dissertação à minha família, em especial aos meus filhos, que apesar de sentirem as limitações da minha presença, sempre reconheceram o esforço suplementar para atingir esta meta.

Resumo

Na vida das organizações, é muito frequente o acesso a grandes quantidades de dados, imprescindíveis aos processos de tomada de decisão. Por vezes, os dados apresentam problemas de qualidade que afetam a qualidade das decisões. Nos armazéns de dados, durante a manipulação dos dados, para se criar formatos que facilitem os processos de tomada de decisão, podem ser identificados mais problemas de qualidades nos dados (PQD).

Existem diversas abordagens para detetar e corrigir PQD. Nestas abordagens pretende-se classificar os diferentes tipos de PQD que possam ocorrer nos dados e indicar caminhos possíveis de deteção e correção dos PQD. Existem algumas ferramentas que se baseiam em abordagens de deteção e correção de PQD existentes. Estas ferramentas detetam e corrigem PQD, no entanto, normalmente servem para detetar e corrigir PQD específicos e têm custos de aquisição consideráveis.

O processo de deteção e correção pode ser complexo e é moroso. Normalmente estes processos de deteção e correção estão encadeados, e podem assumir nomes diferentes como "limpeza de dados" e "eliminação de dados sujos".

Foram definidos objetivos para desenvolver uma solução de deteção de PQD, freeware, com bom desempenho e de fácil utilização. Fez-se levantamento do estado da arte, apresentando conceitos importantes para a compreensão do tema da dissertação. Estudaram-se diferentes tecnologias uteis no desenvolvimento da solução.

Foi desenvolvida uma solução de deteção de PQD, robusta, de fácil utilização que permite desenhar um *workflow* com uma sequência de operações de PQD escolhida pelo utilizador. O utilizador pode guardar o *workflow* para execução posterior com o mesmo ou com outras fontes de dados. A solução contém uma estrutura facilmente expansível para detetar novos tipos de PQD e com novos motores e algoritmos de deteção.

A avaliação da solução revela que a solução disponibiliza uma interface gráfica facilitadora do processo de desenho de *workflow* e configuração das operações de PQD. A solução apresenta um bom desempenho, utilizando programação de pipelines Java com *streams* paralelas.

Palavras-chave: Problemas de qualidade nos dados, limpeza de dados, deteção de PQD, ETL, *workflow*

Abstract

In organizations life, access to large amounts of data, which are essential to decision-making processes, is very common. Sometimes the data has quality problems that affect the quality of decisions. In data warehouses, during data manipulation, to create formats that facilitate decision-making processes, more problems of quality in the data (PQD) may arise.

There are several approaches to detect and correct PQD. In these approaches it is intended to classify the different types of PQD that may occur in the data, and to indicate possible ways of detection and correction of PQD. There are some tools, which are based on existing PQD detection and correction approaches. These tools, detect and correct PQD, however they usually serve to detect and correct specific PQD, and have considerable acquisition costs.

The PQD detection and correction process can be complex and time consuming. Usually, these detection and correction processes are linked, and may take different names such as “data cleaning” and “dirty data elimination”.

The goals to develop a PQD detection solution were defined. Development of a solution freeware, with good performance and easy to use. A state-of-the-art survey was carried out, presenting important concepts for the understanding the dissertation theme. Different useful technologies were studied in the development of the solution.

A robust, easy-to-use PQD detection solution was developed that allows designing a *workflow* with a sequence of PQD operations chosen by the user. The user can save the *workflow* for later execution with the same or with other data sources. The solution contains an easily expandable structure to detect new types of PQD, with new detection engines and algorithms.

The evaluation of the solution reveals that the solution provides a graphical interface that facilitates designing the workflow and PQD operations configuration. The solution presents a good performance, using Java pipelines programming with parallel streams.

Keywords: Data quality problems, Data cleaning, PQD detection, ETL, workflow

Agradecimentos

Agradeço à minha família que sempre me apoiou, principalmente nos momentos mais difíceis.

Agradeço ao Professor Doutor Paulo Oliveira, pelo apoio, conselhos, disponibilidade e ensinamentos ao longo desta dissertação.

Agradeço ainda o contributo de Colegas e Professores que me ajudaram a atingir esta meta.

“Os homens não desejam aquilo que fazem, mas os objetivos que os levam a fazer aquilo que fazem.”

Platão

Índice

Lista de Figuras

Lista de Excertos de Código

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

QD	Qualidade dos dados
PQD	Problema de Qualidade nos dados
FQD	Fator de Qualidade nos dados
NG	Nível de Granularidade
AIQ	Amplitude Inter Quartil
POC	Prova de Conceito
NCD	Novo Conceito de Desenvolvimento
GUI	*Graphical User Interface*
UI	*User Interface*
ES	*ECMA Script*
NPM	*Node Package Manager*
FEI	*Front End of Innovation*
QFD	*Quality Function Deployment*
HOQ	*House Of Quality*
NPD	*New Product Development*
FFE	*Fuzzy Front End*
IoC	*Inversion of Control*
CSS	*Cascade Style Sheet*
API	*Application Programming Interface*
POJO	*Plain Old Java Object*
JSP	*Java Server Pages*
JSF	*Java Server Faces*

MVC	*Model View Controller*
DOM	*Document Object Model*
SWE	*Standard Widget Toolkit*
WUIP	*Workflow User Interface Pattern*
ETL	*Extract, Transform, Load*
SVG	*Scalable Vector Graphics*
VML	*Vector Markup Language*
TQM	*Total Quality Management*
TDQM	*Total Data Quality Management*
RDBMS	*Relational Database Management System*
OLAP	*Online Analytical Processing*
GREL	*Google Refine Expression Language*
ML	*Machine Learning*
CFD	*Conditional Function Dependency*
FRAQL	*Multi-Database Query Language*
RDS	*Relational Database Service*
RAM	*Random Access Memory*
BPMN	*Business Process Model and Notation*
FURPS	*Functionality, Usability, Reliability, Performance, Supportability*
JPA	*Java Persistence API*
OD	*Operações de Deteção*
CSS	*Cascading Style Sheets*
BD	*Base de Dados*

1 Introdução

Esta dissertação está inserida na realização do Mestrado em Engenharia informática, ramo de Sistemas de Informação e Conhecimento, do Instituto Superior de Engenharia do Porto.

Neste capítulo introdutório pretende-se descrever a contextualização do problema em estudo, bem como apresentar o problema a resolver e os objetivos que se pretendem atingir com a solução proposta.

Com esta dissertação pretende-se analisar problemas de qualidade nos dados e desenvolver uma solução que permita detetar problemas de qualidade nos dados.

1.1 Enquadramento

O valor dos dados é de uma importância muito significativa na vida das organizações. Desde há muito tempo que se tem extraído informações dos dados, imprescindíveis para processos de tomada de decisão nas organizações. Os processos de análise baseados em dados têm evoluído ao longo do tempo. Áreas de *Business Intelligence* e Descoberta de Conhecimento têm vindo a despertar especial interesse no valor dos dados para os processos de exploração e análise de dados.

Os dados são frequentemente processados e armazenados em bases de dados. No entanto, os dados armazenados apresentam problemas que podem afetar os fins a que se destinam. Com o aumento do interesse da exploração e análise de dados, a integração de dados tem sido cada vez mais utilizada para se poder extrair mais informação e conhecimento.

Na literatura, estes problemas nos dados são muitas vezes referidos como problemas de qualidade nos dados (PQD) que vulgarmente se designam por erros ou inconsistências. Estas inconsistências podem ser de diversos tipos como violações de sintaxe, violação de domínio, valores em falta, entre outros. Estas inconsistências prejudicam os processo exploratórios de dados (Oliveira, 2008).

Segundo (Milano, Scannapieco, & Catarci, 2005) limpeza de dados corresponde à deteção dos PQD, e consequente a sua correção. Para corrigir os PQD, primeiro é necessário proceder à devida deteção de PQD (Milano et al., 2005). Com este estudo pretende-se permitir a deteção de PQDs, de forma rápida, em que a sequência de execução das operações é estabelecida pelo utilizador. A ferramenta a desenvolver não terá os custos de aquisição elevados que as ferramentas já existentes possuem.

1.2 Problema

Como referido anteriormente os dados encontram-se frequentemente com PQD. Cada vez mais recorre-se à integração de dados provenientes de diversas fontes em armazéns de dados, para permitir uma melhor procura de informação e conhecimento. Com os processos de integração de dados, surgem mais PQD, o que faz com que a limpeza de tenha um papel cada vez mais importante.

Nos armazéns de dados, usam-se frequentemente ferramentas de extração, transformação e carregamento de dados - "*Extract, Transform, Load*" (ETL) nos processos de integração de dados. No entanto, esta ferramentas não são vocacionadas para a limpeza de dados. As ferramentas de limpeza de dados existentes normalmente são para PQD específicos com custos de aquisição consideráveis (Oliveira, 2008).

Grande parte das correções a efetuar nos dados, para alem de terem custos significativos na implementação, são morosas, para além de que para corrigir é necessário saber primeiro que PQDs existem.

1.3 Motivação

O tema apresentado apresenta um problema comum para quem necessita de diagnosticar a qualidade dos dados, antes de proceder aos processos exploratórios de dados, através dos quais se extrai informação e conhecimento importante para os processos de tomada de decisão.

Uma parte significativa dos possíveis utilizadores da solução, não são especialistas em deteção e correção de problemas de qualidade dos dados. No entanto, necessitam de um diagnóstico sobre a qualidade dos dados a partir dos quais pretendem obter informação/conhecimento.

A motivação do autor é facilitar a vida destes utilizadores, permitindo oferecer uma aplicação que permita de forma fácil saber que tipo de problemas de dados têm, para saber como os corrigir, ou como forma de conseguir avaliar como os problemas detetados nos dados podem impactar a qualidade das informações extraídas.

1.4 Contribuições

A solução a desenvolver baseia-se na construção de um sistema que permita fazer a deteção de PQD. O sistema pretendido, será freeware, e irá disponibilizar uma interface gráfica de fácil utilização que permitirá escolher um conjunto de operação de deteção de PQD pelo utilizador, disponibilizando um relatório dos erros encontrados.

Com a deteção de PQD será possível tomar decisões sobre a possível necessidade de corrigir PQD, e, caso seja necessário, quais os problemas de dados que necessitam de ser solucionados.

1.5 Objetivos

No âmbito deste trabalho e à luz da taxionomia de problemas de qualidade de dados ao nível da instância (Oliveira, Rodrigues, Henriques, & Galhardas, 2005), pretende-se a conceção e desenvolvimento de um sistema open-source que suporte a deteção de alguns dos PQD dessa taxionomia que ocorrem ao nível dos atributos (isoladamente, ao nível da coluna e ao nível da linha). O sistema a desenvolver deve ser passível de ser facilmente estendido para futuramente suportar a deteção dos restantes problemas de qualidade que constam da taxionomia mencionada, noutros níveis de granularidade.

Outro aspeto a ter em conta é o da usabilidade, pelo que a ferramenta deve ser iminentemente gráfica e de fácil utilização. Pretende-se que o conjunto de operações de deteção de PQD sejam realizados num processo e sob a forma de um *workflow* - fluxo de trabalho. Pelo que a interface gráfica pretendida terá a possibilidade de representar graficamente as operações, bem como a sequência de execução das mesmas.

Pretende-se ainda que sejam atingidos os seguintes objetivos específicos:

- Avaliar as *framewoks* de desenho de interfaces com o utilizador baseadas na definição de *workflows* entre componentes gráficos;
- Conceber a arquitetura de forma a que esteja seja facilmente extensível para outras operações de deteção de PQD que futuramente possam vir a ser adicionadas;
- Desenvolver e implementar a solução de deteção de PQD de acordo com o desenho realizado;
- Realizar testes à correta execução das operações de deteção implementadas;
- Analisar o desempenho da execução das operações de deteção de dados, assim como de um *workflow* de operações.

1.6 Estrutura do Documento

A dissertação esta dividida em 7 capítulos.

No primeiro capítulo é feita a contextualização do problema a solucionar. Apresentam-se também as motivações para a escolha do tema, as contribuições obtidas com a solução a implementar, e os objetivos a atingir e ainda a estrutura da dissertação

No segundo capítulo é apresentado o estado da arte, onde se faz uma revisão da literatura sobre as operações de deteção e correção de problemas que afetam a qualidade de dados. São também apresentadas características de algumas *frameworks* existentes para o desenvolvimento de interfaces gráficas.

No terceiro capítulo é feita a analise de valor à solução proposta, utilizando metodologias e ferramentas existentes, e que permitem efetuar esta análise.

No quarto capítulo é apresentado o design da solução. Começa-se por definir requisitos e prossegue com o desenho da solução. A arquitetura de alto nível baseia-se no padrão MVC. São utilizados padrões de desenho de software sempre que considerados adequados para o desenho dos diferentes aspetos da solução.

No quinto capítulo é descrita a implementação da solução. A implementação é efetuada de acordo com o desenho do capítulo 4. Descreve-se a utilização de tecnologias incorporadas e a própria implementação. Utilizam-se figuras, diagramas e excertos de código que ajudam na compreensão da implementação efetuada.

No sexto capítulo é feita a avaliação da solução. São elaborados testes unitários e testes de integração. São apresentadas as métricas utilizadas para verificar o cumprimento de requisitos, as quais são determinadas com recurso a testes funcionais. Finaliza-se com uma abordagem de avaliação da usabilidade da solução.

No sétimo capítulo são apresentadas as conclusões. É apresentado o cumprimento dos objetivos, indicam-se problemas e limitações, e finaliza-se com a sugestão de trabalhos futuros.

2 Estado da arte

Neste capítulo pretende-se esclarecer os conceitos relacionados com problemas de qualidade de dados e evidenciar o seu impacto na informação necessária ao processo de tomada de decisão.

A análise sobre os dados é cada vez mais utilizada em processo exploratórios de dados quer com técnicas de *datamining* (mineração de dados, também conhecida por processos de descoberta de conhecimento em dados), quer em processo de ETL, de forma a permitir criar soluções de *Business Inteligence*, utilizadas no suporte à tomada de decisão.

Pretende-se nesta fase compreender o estado da resolução de problemas de qualidade nos dados. Para se resolver problemas de qualidade nos dados, recorre-se a técnicas de limpeza nos dados. Segundo (Roy & Hossain, 2019), a limpeza de dados consiste no processo de limpeza dados de diferentes erros. Erros estes que permitem classificar os dados como "*dirty*" (sujos). Estes dados afetados por problemas de qualidade dificultam os processos de tomada de decisão, dificultando as decisões estratégicas importantes dentro das organizações. Pelo que os dados contidos nos armazéns de dados provenientes das bases de dados operacionais, tem de estar num estado limpo para permitir informação e conhecimento com precisão.

Nos processos de extração de dados dos sistemas operacionais para os armazéns de dados realizam-se diversas operações de limpeza de dados nos designados processos de ETL. De acordo com (Roy & Hossain, 2019) dados com qualidade significa que estão limpos dos erros após o processo de limpeza de dados. Segundo Erhard Rahm citado por (Roy & Hossain, 2019), "a limpeza de dados, trata da deteção e remoção de erros e inconsistências dos dados, a fim de melhorar a qualidade dos dados". Segundo o mesmo autor são aplicados diferentes algoritmos para a correção dos diversos tipos de erros, aumentando assim a complexidade dos processos de limpeza de dados.

2.1 Qualidade dos dados

Para (Juran & Godfrey, 1999) obter qualidade significa obter uma conformidade com especificações, tendo em conta que os produtos que estão em conformidade com as especificações vão de encontro às necessidades dos clientes.

De acordo com (Singh & Singh, 2010) , consegue-se obter qualidade nos dados quando os mesmos são relevantes, precisos, compreensíveis, abrangentes e adequados. Considera-se que se utilizam os dados de forma eficaz e eficiente, quando os mesmos obedecem a um conjunto de critérios de qualidade nos dados.

2.1.1 Contexto de qualidade nos dados

Segundo (Oliveira, 2008) a qualidade de dados manifesta-se em contextos onde seja necessária a deteção e correção de PQD. Exemplos desses contextos podem ter origem numa fonte de dados única como de base de dados operacional ou ficheiro, ou nas migrações de dados de fontes não estruturadas para fonte de dado estruturada, como as migrações de um conjunto de ficheiros para uma base de dados. Podem ser projetos centrados em dados (*data* mining; *business inteligence*) como referido anteriormente.

2.1.2 Gestão Total da Qualidade de Dados

Nas organizações, os decisores acedem a cada vez maiores volumes de dados, para a tomada de diversos tipos de decisão. Por outro lado, o decisor tem de decidir em espaços de tempo curtos, dada a facilidade de acesso à informação. Torna-se assim de vital importância o fornecimento de dados com qualidade aos decisores. Na literatura existem trabalhos sobre a gestão da qualidade da informação. (Shankaranarayanan & Cai, 2006) apresenta uma *framework* que faz a analogia da informação como um produto e desta forma trata da gestão de qualidade dos dados como um processo análogo à Gestão Total de qualidade ("*Total Quality Management- TQM"*) de um produto.

Existem quatro fases de Gestão total de qualidade nos dados - *("Total Data Quality Management"-TDQM*): A definição de requisito de qualidade e métricas a adotar para medir a qualidade nos dados; Medição da qualidade dos dados em todas os fases de extração de informação a partir dos dados; analise da possibilidade de existência de PQD e identificar os respetivos fatores de falta de qualidade; Processo incremental de melhorar a garantia de qualidade nos dados (Mattioda & Favaretto, 2009). Com a TDQM, existe um processo de melhoria continua para entregar sistemas de informação cada vez com mais qualidade nos dados, e por consequência obter informação e conhecimento com mais eficácia e precisão.

2.1.3 Dimensões de Qualidade de Dados

De acordo com (Akoka et al., 2007) A qualidade nos dados é considerado um conceito multidimensional, complexo e orientado aos objetivos. Na literatura encontram-se identificados várias dimensões de qualidade nos dados. Para (Manjunath T.N, Ravindra S, & Ravikumar G.K, 2010), a qualidade dos dados (QA) verifica-se quando a informação contida nos dados tiver as seguintes dimensões: Conformidade de significado, dados completos ou completude, validade das regras de negócio, eficiência, precisão, não duplicação, integridade, acessibilidade e disponibilidade.

Em (Singh & Singh, 2010) são consideras mais algumas dimensões para se obter qualidade nos dados como fiabilidade, importância, consistência, pontualidade, fineza, compreensibilidade, concisão e utilidade. (Oliveira, 2008) refere que apesar de existirem

diversas dimensões que refletem a complexidade da definição de qualidade nos dados, as dimensões mais aceites são: correção, completude, consistência, atualidade, "interpretabilidade" e acessibilidade.

2.1.4 Passos para a qualidade nos dados

Os dados oriundos de diversas fontes, contêm frequentemente PQD. No entanto, por motivos já argumentados anteriormente, é imprescindível obter qualidade nestes dados. Para (Manjunath T.N et al., 2010), os passos na seguir na obtenção de qualidade nos dados são:

- ***Data Profiling*:** - Corresponde a uma análise dos dados, para aferir o seu estado relativamente ao uso que se pretende. Recorre-se a técnicas estatísticas para analisar dados de carácter individual e nas relações entre tabelas.
- **Limpeza de dados**: - É necessário corrigir os PQD para que as regras de negócio sejam devidamente respeitadas.
- **Padronização**: - Neste passo, os dados provenientes de diversas fontes de dados, são restruturados em estruturas comuns, de tabelas, colunas e tuplos.
- ***Matching***: - Com este passo pretende-se mapear os dados em registos de dados identificáveis, dentro de um conjunto de dados. São feitos vários mapeamentos como por exemplo: moradas, códigos postais, telefones, entre outros.
- **Enriquecimento**: - Aqui interessa suprimir o PQD de dados em falta, recorrendo a outras fontes de dados para o efeito. Exemplos de dados em falta: Nome completo, género, número de telefone;
- **Monitorização**: - A monitorização de dados tem como objetivo detetar e corrigir PQD, para reduzir deterioração da qualidade dos dados.

Em (Oliveira, 2008) é referido que os passos de Data profilling e Análise de dados servem para a deteção de problemas nos dados. Segundo o mesmo autor, os passos mais frequentes de correção dos dados são Transformação de dados e enriquecimento de dados. O processo de obtenção de qualidade dos dados implica a execução dos passos acima definidos, sendo que pelo menos a fase de correção tenha de estar incluída. O processo pode ser iterativo para se atingir o nível de qualidade desejado.

2.2 Problemas de qualidade nos dados

Segundo (Oliveira, 2008) os problemas de qualidade nos dados causam prejuízos muito significativos nas organizações. É muito provável que se encontrem problemas de qualidade em dados (PQD) , sendo que podem-se encontrar vários tipos de problemas de qualidade de dados como: violações de restrições de integridade, violações de domínio, problemas de dados duplicados, sinónimos e violações à restrição de integridade, erros ortográficos , entre outros (Oliveira, 2008). "A falta de dados completos, relevantes, atualizados e precisos acerca de

clientes, concorrentes e tecnologias pode constituir um obstáculo à definição de uma estratégia sólida" (Oliveira, 2008). Segundo (Singh & Singh, 2010), os PQD mais comuns ocorrem em situações de: Procedimentos e processos de tratamento de dados deficientes; falha na introdução e manutenção dos dados; PQD na migração de dados entre sistemas; Dados de terceiros que não estão adequados aos padrões de determinada organização, ou em níveis de qualidade não aceitáveis.

2.2.1 Fatores de falta de qualidade

Os analistas de dados necessitam de saber as causas dos PQD. Para tal é aconselhável o conhecimento dos fatores que afetam a qualidade dos dados. Segundo (Manjunath T.N et al., 2010) uma das principais razões para o fracasso de projetos de *Business Inteligence* é a má qualidade dos dados nos armazéns de dados. Um armazém de dados tem dados de diversas proveniências ou fontes. As fontes de dados contêm dados transacionais, por vezes não estruturados, os quais são transformados e carregados para os armazéns de dados. Existem inúmero fatores que afetam a qualidade dos dados. Estes fatores podem ter origem na fonte dos dados individuais, mas grande parte deve-se ao facto de se ter de combinar dados provenientes de diversas fontes de dados (Manjunath T.N et al., 2010).

Na Figura 1 está apresentada a arquitetura de um armazém de dados, onde se podem ver diversas fontes de dados. As fontes de dados estão sujeitas a processos de ETL, normalmente numa área de staging. Tipicamente, nos armazéns de dados também existem bases de dados dimensionais(*datamarts*[1]) que são usadas em análises OLAP. A partir dos armazéns de dados, também são extraídos dados para processos de *datamining* e para relatórios importantes aos processos de tomada de decisão.

[1] Datamart – subconjunto de dados de um armazém de dados. As tabelas estão num modelo dimensional. Datamart contém tabelas de factos e tabelas dimensionais.(Rauer et al., 2000)

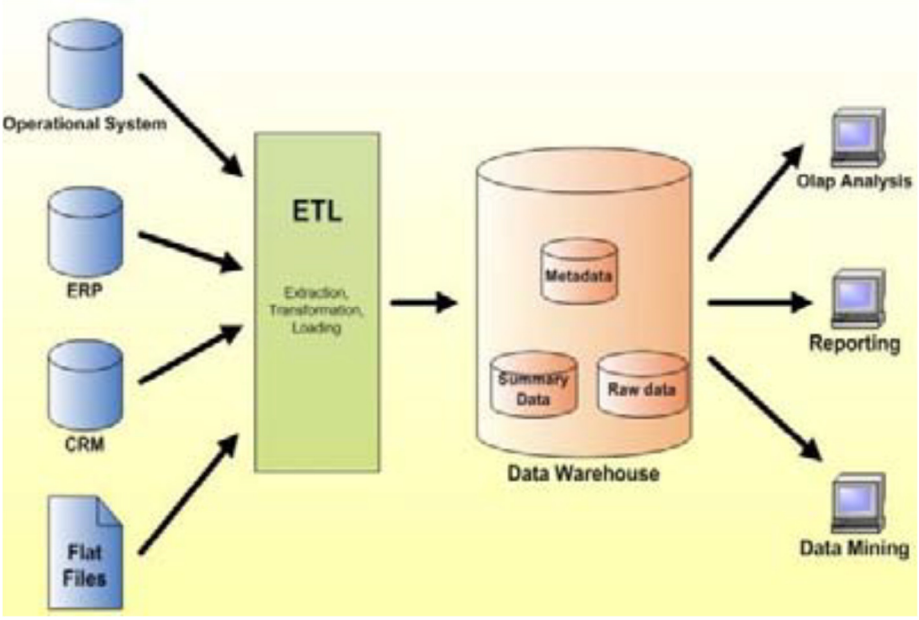


Figura 1: Arquitetura de armazém de dados (Manjunath T.N et al., 2010).

Na área de staging é feito um grande esforço de limpeza de dados, com o objetivo de obter dados consistentes com elevada precisão, e assim aumentar a fiabilidade nos processos de análise baseada em dados, como é exemplo as análises "*Online Analytical Processing*" (OLAP) e o *datamining* (Manjunath T.N et al., 2010). Segundo (Manjunath T.N et al., 2010) os fatores mais comuns que afetam a qualidade dos dados são:

- Procedimentos inadequados de manipulação de dados;
- Fraca relação com os dados originais;
- Erros na migração de dados;
- Dados não estruturados.

Os fatores de qualidade nos dados (FQD) aparecem em diversos níveis. Nos armazéns de dados, aparecem nas fontes de dados de origem, na área de staging e, ainda nos modelos de dados dimensionais. Em (Manjunath T.N et al., 2010) foram identificados 42 FQD nas fontes de dados de origem. Apresentam-se a seguir alguns desses FQD:

- Combinação de dados heterogéneos provenientes de diversos ficheiros;
- Falta de conhecimento das diversas fontes de dado de origem;
- Impossibilidade de gerir dados com antiguidade;
- Atualidade dos dados alterável;
- Processos de validação deficiente;
- Mudanças rápidas nas fontes de dados;
- Várias fontes de dados geram semânticas heterogéneas.

Os FQD na área de staging são menos, mas mesmo assim ainda são um número bastante considerável. (Manjunath T.N et al., 2010) identificou 23 FQD na área de staging. Apresenta-se a seguir alguns desses FQD:

- Diversos *parsings* e diversas regras de negócio provenientes dos dados fonte;
- Falta de criação de regras de negócio;
- Restauração inadequada de dados na área de staging;
- Problemas de reconciliação de dados durante a fase de limpeza;
- Restrições de integridade referencial impróprias;
- Ferramentas de ETL não geram *logs* consolidados;
- Mapeamento de dados inaceitáveis.

Segundo o mesmo autor a qualidade da informação depende, entre outros, da qualidade do modelo de dados, pelo que é recomendável prestar a devida atenção ao modelo de dados no armazém de dados, pois a nível dos modelos de dados dimensionais existem mais alguns fatores que afetam a qualidade dos dados como: Interpretação errada dos requisitos para o modelo dimensional; modelo de dados pouco claro; uso inadequado das relações; falta de documentação do modelo de dados(Manjunath T.N et al., 2010).

2.2.2 Taxionomias de PQD

Existem ainda muitas limitações, na deteção de PQD, pois na maioria das vezes as soluções implementadas são para resolver problemas específicos, no entanto tem-se criado taxionomias para classificar os tipos de problemas de dados (Almeida et al., 2013). As taxionomias permitem capturar os PQD que afetam as bases de dados, permitindo avaliar a qualidade dos dados em relação às taxionomias (Bosu & MacDonell, 2013).

Apresentam-se de seguida algumas taxionomias encontradas na literatura, para classificar os PQD:

- **Taxionomia de Rahm e Do:** – Os PQD estão divididos entre
  PQD de fonte única e PQD de multi-fonte. Ambos PQD são ainda subdivididos em problemas no nível das instâncias de esquema e dados. Os PQD multi-fonte ocorrem devido à integração de dados de diversas fontes de dados. Nos problemas de fonte única, existe distinção entre os problemas que ocorrem ao nível do atributo (valor em falta, erro ortográfico), no registo (violação de dependência funcional), no tipo de registo (registos duplicados) e na fonte dos dados (Ex: referencia errada). Os problemas relacionados com esquema de dados evitam-se com um desenho adequado do esquema de dados, tanto na transformação como na integração de dados (Almeida, Sousa, Deus, Daniel Amvame Nze, & Mendonça, 2013);

- **Taxionomia de Müller e Freytag** – Classificação de anomalias de dados. Os PQD são divididos em anomalias semânticas, sintáticas e de cobertura. Os problemas sintáticos relacionam-se com aspetos de sintaxe e valores usados na representação de entidades (e.g.: violações de sintaxe). Os problemas semânticos impedem os dados de serem uma representação não ambígua e não redundante do mundo real (e.g.: registos duplicados). Os de cobertura estão relacionados com a quantidade de entidades e propriedades do mundo real que são armazenados na tabela (e.g.: valores em falta). Os problemas estão limitados ao nível de uma tabela simples (Almeida et al., 2013);

- **Taxionomia de Kim** – É fornecida uma estrutura para entender como surgem os PQD e quais os aspetos que devem ser considerados na limpeza dos dados. É utilizada uma abordagem hierárquica descendente. Á medida que se desce de nível é aumentado o detalhe dos PQD. É efetuada uma classificação dos PQD em: valores em falta; valores errados; valores corretos, mas não utilizáveis. O último caso ocorre devido à falta de padrão de representação comum para introdução de resultados (Almeida et al., 2013);

- **Taxionomia PQD baseada em regras**– Os PQD são classificados em 37 PQD. A taxionomia fornece um conjunto de PQD mais amplo e mais relacionado com armazéns de dados. PQD são classificados da seguinte forma: regras de entidade de negócio; regras de atributos; regras de dependências nos dados; regras de validação baseadas nos dados (Li, Peng, & Kennedy, 2011);

- **Taxionomia de Barateiro e Galhardas: –** Os PQD são divididos a nível de esquema de dados: problemas que podem ser evitados pelo o sistema de gestão de base de dados relacional ("Relational Database Management System"-RDBMS) ou por um design de esquema de dados aprimorado; e problemas a nível de instancia (ou seja, problemas que não podem ser evitados com aprimoramento de esquema de dados, porque estes PQD estão relacionados com o conteúdo dos dados)(Barateiro & Galhardas, 2005);

- **Taxionomia de Oliveira:** - É proposta uma relação abrangente de 35 PQD. Os problemas são organizados por nível de granularidade de ocorrência, de acordo com modelo de dados relacional. A classificação distingue PQD ao nível de: atributo; registos (tuplos); relação simples; relações múltiplas e múltiplas fontes de dados (Oliveira, 2008).

- **Taxionomia PQD "*time-oriented*": -** Os PQD estão divididos entre fonte única e multi-fonte. Baseada na taxionomia de Kim. Os problemas de fonte única são classificados como: dados ausentes; duplicados; valores improváveis; desatualizados; dados ambíguos. Os problemas multi-fonte dividem-se em: heterogeneidade de sintaxe; heterogeneidade de semântica e referências (Gschwandtner, Gärtner, Aigner, & Miksch, 2012).

Estas taxonomias não permitem classificar a totalidade dos PQD, mas permite obter um conjunto bastante amplo de PQD (Almeida et al., 2013).

Neste trabalho pretende-se desenvolver um sistema de deteção de PQD ao nível do atributo. A taxionomia de (Oliveira, 2008) é bastante completa na descrição de PQD, desde PQD ao nível do atributo até PQD em múltiplas relações e múltiplas fontes de dados. Tendo em vista possíveis extensões futuras do trabalho a desenvolver, optou-se por usar a taxionomia de (Oliveira, 2008) para descrever alguns dos PQD ao nível do atributo:

- **Violação de sintaxe**: - A sintaxe estabelecida para o atributo não é respeitada. Ex: código postal contém valor 4100201 em vez de 4100-201
- **Violação de domínio**: - Ocorre quando o valor não faz parte do conjunto de valores válidos do atributo. Por exemplo: valores que estão fora de um valor mínimo e máximo estipulado. Para valores textuais, a violação de domínio pode ocorrer nos PQD de valor sobrecarregado (contém mais informação do que o pretendido) ou valor incompleto (ex: na designação de um conselho pode encontrar "Paiva" quando o pretendido é "Castelo de Paiva") (Oliveira, 2008).
- **Erro ortográfico**: - Quando é inserido erradamente um erro ortográfico. Ex. palavras em que se troca "z" por "s", exemplo "pezo" em vez de "peso".
- **Valor em falta**: valor em falta num campo obrigatório.

Em caso de multi-valor, ao nível de um atributo, os tipos de PQD, são:

- **Existência de Sinónimos**: - Para determinado atributo, a existência de sinónimos consiste na existência de valores identificados como sinónimos num determinado dicionário definido;
- **Violação de unicidade**: - Quando um valor definido para ser único, é encontrado em mais do que um registo;
- **Violação de restrição de integridade**: - quando o valor do atributo não respeita uma determinada restrição definida (Ex: valores <10).

2.3 Limpeza de dados

A limpeza dos dados, como referido anteriormente na secção 2.1.3, é um dos passos necessários para se obter qualidade nos dados. Segundo (Li, 2012), é nos armazéns de dados, onde a limpeza de dados é mais utilizada devido ao facto de se integrarem dados de varias fontes de dados. Na área de mineração de dados e gestão total da qualidade de dados, a limpeza de dados também é muito importante. Em (Almeida, Maio, Oliveira, & Barroso, 2015), é referido que a limpeza de dados implica um processo para deteção e correção dos PQD. Este processo é realizado com a intervenção de especialistas na especificação de operações de deteção e correção do processo de limpeza de dados.

2.3.1 Protótipos de Investigação

Apresentam-se a seguir algumas abordagens existentes para limpeza de dados:

- **Roda de Potter**: - Protótipo de limpeza de dados interativo, com uma forte integração entre a deteção e correção de PQD. O processo de correção de PQD é efetuado de forma gradual, com adição de nova correção de PQD, ou, recuperação das alterações efetuadas pelas operações de correção anteriores. O resultado da correção, para cada operação de PQD, é mostrado na interface gráfica. A partir de especificações de correção de PQD, efetuadas de forma gráfica, pelo utilizador, existe um mecanismo que mostra de imediato no ecrã o resultado da especificação introduzida. O processo de limpeza é efetuado de forma cíclica, em que os utilizadores podem efetuar correções à medida que os PQD são encontrados. (Raman & Hellerstein, 2001);

- **AJAX**: - facilita a especificação e execução de limpeza de dados, tanto em fontes de dados única, limpando duplicados, ou na integração várias fontes de dados. *Framework* permite várias operações de transformação de dados como: mapeamento, "*matching*", "*clustering*" e "*merging*". Possui linguagem declarativa e expressiva, baseados na semântica de SQL, que permite especificar as operações de limpeza necessárias. A nível da correção toma partido da tecnologia RDBMS existentes (Galhard, Florescu, Shasha, & Simon, 2000).

- **ARKTOS**: As transformações nos armazéns de dados e as tarefas de limpeza de dados são definidos de forma declarativa e gráfica. A qualidade dos dados pode ser medida usando fatores de qualidade específicos. Permite otimizar as sequências complexas de operações de transformação e limpeza. ARKTOS define uma operação de deteção de PQD como uma atividade que corresponde a uma unidade atómica no workflow de processamento de dados através de uma expressão SQL. Cada atividade trata um tipo de PQD específico e uma política. O tipo de erro pode ser por Ex: eliminar duplicados; a política refere-se à forma como os dados são processados, Ex: registo duplicado apagado, ou registo duplicado registado em ficheiro de log. São tratados os PQD ao nível do atributo. É possível definir cenários com listas de atividades a serem executadas. São definidas duas linguagens, uma baseada em XML e outra declarativa semelhante ao SQL para operações mais complexas (Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, & Nikos Karayannidis, 2000);

- **Febrl**: - Utiliza técnicas de ligação de registos ("Record Linkage"). Ferramenta contruída com o objetivo de limpar e ligar dados. Nos dados provenientes de várias fontes usam-se técnicas de ligação de registos para enriquecer os dados. O Febrl tem uma interface gráfica que permite definir as operações de ligação de registos, e limpeza de PQD. O

Febrl é open source, com uma primeira aplicação na área da saúde. Posteriormente o Febrl comtemplou soluções para outras áreas (Christen, 2008);

- **Smartclean**: deteta e corrige PQD. A ferramenta tem implementada uma sequência proposta de operações de limpeza, facilitando a utilização. Tem capacidade de limpeza a nível do atributo, coluna, tuplo, relações e múltiplas relações. Permite interação ao longo do processo de limpeza. As operações de limpeza dividem em operações de deteção e operações de correção. O processo de limpeza obedece às dependências definidas das operações de deteção e correção de cada tipo de PQD. O processo de limpeza inicia com a definição das operações de deteção definidas numa linguagem declarativa baseada na gramática BNF. Posteriormente as operações são analisadas a nível de sintaxe e reportadas para correção. De seguida as operações são envidas para um sequenciador que estabelece a ordem de execução das operações. À medida que as operações são sequenciadas, são enviadas para um motor de execução de operações de limpeza, que grava os PQD detetados numa base de dados. As operações de correção dos PQD podem ser executadas em modo automático. Após a correção de PQD pode-se proceder novamente à deteção de novos PQD. (P. Oliveira, Rodrigues, & Henriques, 2009).

2.3.1.1 Comparação de protótipos

Apresenta-se na Tabela 1 a comparação entre os protótipos apresentados para a limpeza de dados.

Tabela 1: Comparação de protótipos de limpeza de dados

Protótipos / Critérios	Roda de Potter	AJAX	ARKTOS	FeBrl	SmartClean
Processo Iterativo	Sim	Sim	Sim	Não	Sim
Todos Níveis Granularidade	Não	Não	Não	Não	Sim
Âmbito genérico	Só ETL	Só ETL	Não	Não	Sim
Eliminação de duplicados	Não	Sim	Não	Sim	Sim
Deteção de Outliers[2]	Sim			Não	

[2] Outliers - Correspondem a dados anormais que podem sugerir suspeitas relativamente à sua origem (Aggarwal, 2017)

A Roda de Potter é uma ferramenta mais vocacionada para a transformação de dados do que limpeza a nível de instância ou a nível de esquema de dados. Não trata PQD duplicados. (Li, 2012) O Smartclean é uma ferramenta direcionada à limpeza de dados em vários níveis de granularidade, baseado na Taxionomia de (Oliveira, 2008) para a deteção de PQD e respetiva correção.

O principal propósito do AJAX é permitir uma especificação e execução de limpeza de dados. Trata de PQD de uma única fonte de dados; de várias fontes de dados e PQD duplicados. O algoritmo de otimização da limpeza, a nível lógico, depende das escolhas do utilizador sem considerar problemas de domínio. Consequentemente o utilizador pode definir mal os parâmetros dos operadores selecionados, resultados em transformações de limpeza de baixa qualidade (Li, 2012).

Em ARKTOS tal como na roda de Potter as transformações ocorrem a nível de instância e a nível de esquema de dados. O ARKTOS não consegue eliminar duplicados. No ARKTOS, o utilizador define os cenários sem fornecer informação detalhada das atividades, relativamente a questões de domínio e algoritmos envolvidos. Não existe muito conhecimento, sobre o comportamento do ARKTOS, em situações de várias tarefas de limpeza com interdependência (Li, 2012).

Das limitações do *Febrl* indica-se a eliminação de duplicados não ser efetuado num processo de limpeza em conjunto com a limpeza de outros PQD. Ao contrário da *Roda de Potter* o *Febrl* não permite a deteção de *Outliers*. O *Febrl* não faz nenhuma sugestão ao utilizador durante o processo de limpeza, o que dificulta a utilização para utilizadores que não tenham conhecimento das técnicas de limpeza.

Das abordagens existentes comparando o nível de granularidade, o *Smartclean* é a única que permite limpar PQD em 4 níveis de granularidade. Com esta a abordagem é possível um processo iterativo e é aplicável a qualquer domínio (Oliveira et al., 2009).

A maioria das abordagens são especificas para determinado domínio dentro de uma área de negócio. Além disso, grande parte delas tratam de PQD específicos. Por exemplo: Observando a tabela 1 verificamos que nem todas as abordagens permitem eliminação de duplicados. De uma forma genérica, as abordagens de limpeza de dados, necessitam que o utilizador tenha conhecimento especializado na especificação das operações de limpeza. Normalmente existem dependências entre as operações de deteção e as operações de correção, ou seja, certos PQD, só são detetados após a correção dos anteriormente detetados, sugerindo que se adote um processo iterativo na limpeza de dados.

2.3.2 Ferramentas Comerciais

Existem, no mercado, diversas ferramentas de limpeza de dados. Nesta secção vamos apresentar as seguintes ferramentas:

- Open refine
- Data Wrangler
- SCARE
- Cleanix
- NADEEF
- KATARA

2.3.2.1 Open Refine

Open Refine é uma ferramenta para limpeza e transformações de dados. É uma aplicação Desktop com as seguintes características/comportamentos (Ham, 2013):

- Aceita ficheiros de vários formatos (XLS, XML, TSV, CSV);
- Permite Filtros de dados a alterar por data;
- Deteta PQD como: dados duplicados, dados em falta, variações nos dados introduzidos, dados inconsistentes;
- Permite uma análise rápida de dados contidos em ficheiros;
- Permite as ações de UNDO/REDO em todas a operações;
- Utiliza a "Google Refine Expression Language" (GREL) para transformar dados.
- Permite reconciliar dados com fontes fidedignas, através de webservices.

Esta ferramenta é open-source, mas não é adequada para utilizadores que não tenham conhecimentos de programação. Não existe documentação técnica nem suporte à ferramenta, apenas informação dispersa em artigos e tutoriais da internet(Ham, 2013).

2.3.2.2 Data Wrangler

Data Wrangler é uma solução comercial da *Trifacta*. Utiliza um mecanismo de inferência sobre transformações relevantes. Permite a simplificação da especificação das operações de limpeza através de interface gráfica em conjunto com uma linguagem de transformações declarativa. Com a seleção de um conjunto de dados a tratar, a ferramenta sugere as operações de transformação a executar, com base no contexto e no conjunto de transformações previamente escolhidas. (Kandel, Paepcke, Hellerstein, & Heer, 2011).

A linguagem declarativa é e baseada na linguagem de transformação da *Roda de Potter*. Tendo em vista melhorias em tarefas comuns de limpeza de dados, foram adicionados à linguagem, operadores de posição, funcionalidades de agregação, entre outras funcionalidades. *Data Wrangler*, permite o seguinte conjunto de transformações (Kandel et al., 2011):

- **MAP**: - permite transformação de 1 para zero, eliminando dados desnecessários.

As transformações de 1 para 1, permite operações como: extração, corte e divisão de valores em várias colunas. Transformações de 1 para muitos, permite incluir operações para dividir os dados em várias linhas;

- **Lookups e joins**: - Utiliza tabelas externas de lookup para validar os dados durante as operações de transformação de dados provenientes de fontes de dados originais;
- **Reshape**: - Permite manusear as estruturas dos dados de origem com dois tipos de operadores: fold: reduz várias colunas num número mais reduzido de colunas contendo conjunto de chave-valor; unfold: cria colunas a partir dos dados originais.
- **Posicional**: as transformações incluem operação de enriquecimento de dados e de desvio de dados. As operações geram valores baseados em valores vizinhos.

Durante as operações de transformação, o *Data Wrangler*, sugere possíveis operações de limpeza e permite prever resultados de aplicação dessas operações de limpeza de dados. No entanto, os utilizadores não conseguem definir boas estratégias de limpeza quando não conhecem bem o modelo de transformação a realizar. Outro problema ocorre quando o filtro dos dados não é efetuado devidamente, ficando o utilizador bloqueado, não conseguindo prosseguir com as restantes operações de limpeza escolhidas (Kandel et al., 2011).

2.3.2.3 SCARE

O *SCARE* é uma *framework* de limpeza de dados escalável, que se baseia na abordagem de (Yakout, Berti-Équille, & Elmagarmid, 2013), que combina técnicas de *machine learning* com métodos probabilísticos de limpeza de dados. Contém medidas de qualidade baseadas em benefícios probabilísticos e quantificação das alterações efetuadas na base de dados. SCARE combina técnicas de *machine learning* para efetuar previsões de limpeza de PQD. Baseia-se em particionamento horizontal dos dados, pelo que permite que sejam realizadas várias previsões de limpeza num único registo. Contem um mecanismo de combinação de várias previsões para a obtenção de uma previsão final de limpeza de dados (Yakout et al., 2013).

Técnicas estatísticas de *machine learning* (ML) (ex: árvore de decisão, rede de Bayhes) determinam dependências, correlações e *outliers* nos dados. Seguem técnicas de limpeza de dados baseadas em *machine learning* com enfoque principal no enriquecimento de dados e no PQD de duplicação de dados. É de referir, segundo (Yakout et al., 2013), que o processo de treino de dados em grandes bases de dados é custoso. Recorre-se a técnicas para escalar grandes bases de dados. SCARE permite o processamento paralelo de vários blocos de dados, através de um mecanismo de escalonamento horizontal. Os métodos de ML permitem previsões locais de PQD em vários blocos de dados. Existe um mecanismo que combina as previsões dos vários blocos de dados. Várias possibilidade de correção de PQD duplicados, são capturadas numa visualização gráfica, permitindo a associação entre os valores previstos nos vários blocos de dados, conseguindo-se obter uma previsão final da correção dos PQD encontrados (Yakout et al., 2013).

Depois da previsão, é aplicada uma técnica baseada em probabilidades. Primeiro é feita uma aprendizagem de distribuição de dados sem PQD. Depois alteram-se os dados com PQD de forma a maximizar os parâmetros estatísticos da distribuição aprendida (Yakout et al., 2013).

As técnicas de ML normalmente usadas necessitam de conhecimento de relações entre atributos para construir uma rede *Bayesiana*, ficando este tipo de técnica limitada a valores numéricos ou categóricos. No caso do SCARE é feito um esforço de previsão para identificar muitas classes. Conseguem-se identificar PQD em dados do tipo (string, numéricos, ordinal, categóricos). O objetivo do SCARE consiste na previsão do valor de um atributo a partir da previsão de múltiplos atributos. Consegue-se detetar o seguinte tipo de PQD (Valor em falta; Valor incompleto; *Outliers*; Valores inconsistentes). O *SCARE* não contém uma ferramenta gráfica de limpeza de dados, pelo que a seleção das operações de limpeza de dados não pode ser efetuada em tempo real (Yakout et al., 2013).

2.3.2.4 Cleanix

Cleanix é uma ferramenta para efetuar limpeza de bases de dados de grandes dimensões ("*Big Data*"). É escalável, baseada na *framework* Hyracks. Permite fazer a integração de várias fontes de dados originais num único cluster de dados. Cleanix efetua limpeza de vários tipos de PQD como: Dados incompletos; dados duplicados e resolução de conflitos nos dados (Wang et al., 2016). Cleanix apresenta as seguintes funcionalidades principais:

- **Escalabilidade: -** tarefas de limpeza realizadas por um motor de execução paralela;
- **Unificação: -** Unificação de várias tarefas de execução paralela num único *workflow*;
- **Usabilidade: -** GUI de fácil utilização. Permite selecionar regras de forma intuitiva.

Para (Wang et al., 2016) os PQD que aparecem frequentemente nas bases de dados de "Big Data", estão relacionados com a funcionalidades dos 4V (velocidade, Volume, Variedade e Valor). De acordo com (Wang et al., 2016) existem 3 métodos para deteção de PQD :

- **Identificação de entidades**: - encontrar representação comuns, para itens com o mesmo significado;
- **Deteção de PQD de acordo com regras definidas**: - PQD devido à utilização de diferentes tipos de regras. Ex: regras de dependências funcionais e algoritmos baseados em SQL ;
- **Deteção de PQD baseado no dataset principal**: - O *dataset* principal deverá conter dados com elevado padrão de qualidade para visualização consistente dos dados;

De acordo com (Wang et al., 2016) os métodos para corrigir PQD, dividem-se em três grupos:

- **Correção de PQD baseada em regras**: - corresponde à alteração dos dados e colocá-los de acordo com as regras;

- **Descoberta da Verdade**: - usam-se algoritmos de descoberta de verdade, para resolver conflitos nos dados durante a identificação de entidades. Método iterativo de calculo do grau de verdade das fontes e grau de confiança dos dados;
- **Machine learning**: - são usados métodos de ML principalmente para reparar dados incompletos. O Cleanix permite limpar os seguintes tipos de PQD:
- **Deteção e correção de valores anormais**: - Deteção de PQD de acordo com as regras definidas pelo utilizador;
- **Dados incompletos**: - Encontrar atributos sem valor e preenchê-los;
- **Dados duplicados**: - Remover dados duplicados;
- **Resolução de conflitos**: - Encontrar atributos em conflito, que representam as mesmas entidades do mundo real, e encontrar os valores corretos para estes atributos.

O *Cleanix* utiliza o motor *Hyracks* para efetuar a limpeza de dados. Este motor permite extensibilidade das operações através de operadores e conectores; é um motor flexível permitindo um conjunto de políticas para dividir os conectores, permitindo um comportamento do tipo de uma base de dados de processamento paralelo. Apresenta grande eficiência devido às caraterísticas de extensibilidade e flexibilidade (Wang et al., 2016).

A interface com o utilizador do *Cleanix* permite adicionar várias máquinas para realizar a limpeza de dados a partir de fontes de dados diferentes. São definidas as regras de limpeza para encontrar os tipos de PQD. Pode-se definir um "*threshold"* e peso para encontrar duplicados.

2.3.2.5 NADEEF

NADEEF é um sistema de limpeza de dados com uma interface gráfica que permite adicionar um conjunto de regras de limpeza de PQD. Permite definir regras para "conditional functional dependencies" (CFD) e Regras de ETL comuns. Contem uma separação da interface do utilizador com o core de limpeza de dados. Permite a adição de novos algoritmos ao core para se conseguir mais extensibilidade e generalização (Dallachiesa et al., 2013). Segundo (Dallachiesa et al., 2013) as aplicações de limpeza de dados enfrentam os seguintes desafios:

- **Heterogeneidade**: - Negócio baseado em regras de qualidade que são definidas em formatos muito diversificados. A semântica muito diversificada dificulta a criação de sistemas uniformes que aceite regras de qualidade tão diversificadas, de forma a ser possível incluí-las todas numa única *framework* ;
- **Interdependência**: - Os algoritmos de limpeza de dados, na grande generalidade são para resolver problemas específicos;
- **Extensibilidade e deploy**: - dificuldade em encontrar ferramentas de limpeza de dados sem configuração aborrecida;
- **Gestão de metadados**: - as Ferramentas fornecem dados ao utilizados em formatos restritivos. Existe a necessidade de os utilizadores conhecerem bem os metadados antes de se proceder à limpeza de dados;

O *NADEEF* separa a especificação das regras do core que trata da deteção e correção de PQD. Recolhe as regras definidas pelo utilizador, que são compiladas em "constructs". Com base nestes *constructs* é efetuada a detenção de PQD e possíveis formas de os corrigir. Depois os PQD são corrigidos com base na interdependência das regras.

O *NADEEF* ainda não trata base de dados de larga escala; a interface não é muito "user fridenly", para permitir uma introdução de regras de forma mais fácil; Não inclui técnicas de comparações por similaridade. Não permite particionamento de dados para limpar grandes bases de dados e PQD mais complexos. Não tem GUI para limpeza em tempo real, nem sumarização do PQD (Dallachiesa et al., 2013).

2.3.2.6 KATARA

O *KATARA* é um sistema de limpeza de dados baseado em conhecimento. Dada uma tabela, uma base de conhecimento e uma população alvo (dataset com dados de origem), o sistema interpreta a semântica da tabela, para alinhá-la com a base de conhecimento, detetando e corrigindo PQD, gerando top-k possibilidades de correção dos PQD. Dada uma tabela de um *dataset* com dados de uma fonte dados a limpar e uma base de conhecimento, o *KATARA* começa por encontrar padrões na tabela para mapear a tabela com a base de conhecimento. Com os padrões, o *KATARA* anota (marca) tuplos (registos) como corretos ou incorretos. Para os tuplos incorretos, o *KATARA* extrai da base de conhecimento top-k possibilidades de correção. Por outro lado, dados marcados como corretos, que não existam na base de conhecimento são inseridos como factos, enriquecendo a base de conhecimento(Chu et al., 2015). No *KATARA* podemos encontrar os seguintes contributos principais:

- **Definição e descoberta de padrões em tabelas**: - Classe de tabelas padrões para explicar a semântica das tabelas que usam a base de conhecimento;
- **Validação de padrões de tabela por crowdsourcing3:** - Algoritmo para validar o melhor padrão da tabela por crowdsourcing. É usado um algoritmo de escalonamento baseado em entropia para maximizar a redução de incerteza do padrão da tabela candidato;

[3] Crowdsourcing- uso de computação humana para resolver problemas impossíveis ou demasiado caros de resolução por computador"(Franklin, Kossmann, Kraska, Ramesh, & Xin, 2011).

A definição de Crowdsourcing é referida em várias fontes na literatura. Em (Kietzmann, 2017) Crowdsourcing é definido como "o uso de TI para terceirizar qualquer função organizacional para uma população estrategicamente definida de atores humanos e não humanos, na forma de uma chamada aberta."

- **Anotação de dados**: - para um determinado padrão de tabela, os dados são anotados com categorias diferentes: a) correção de dados validados pela base de conhecimento; b) corrigir dados por validação com a base de conhecimento e o *crowd*[4];

Uma Limitação do *Katara* é o facto de necessitar de uma base de conhecimento como ponto de partida para a limpeza de dados. Não permite a integração de várias bases de conhecimento na limpeza de um *dataset*.

2.3.2.7 Comparação de Ferramentas

A comparação das ferramentas é efetuada na Tabela 2.

Tabela 2: Comparação de ferramentas de limpeza de dados

Ferramentas / Critérios	OPEN REFINE	DATA WRANGLER	SCARE	CLEANIX	NAADEF	KATARA
Processo Iterativo	Sim	Sim	Não	Sim	Não	Não
Todos Níveis Granularidade	Não	Não	Não	Não	Não	Não
Âmbito genérico	Só ETL	Só ETL	Não	Não		Não
Preço	Grátis	$419 / mês			Grátis	
Eliminação de duplicados	Sim	Não	Sim	Sim		
Deteção de Outliers		Sim	Sim			

Normalmente as ferramentas existentes são para tratar de PQD específicos em determinadas áreas de negócio. De uma forma genérica, as ferramentas de limpeza de dados, necessitam que o utilizador tenha conhecimento especializado na especificação das operações de limpeza. Uma das razões da necessidade de conhecimento de domínio é a existência de interdependências nas operações de limpeza.

Na secção 3.5, no contexto de analise de valor, serão comparadas ferramentas descritas nesta secção, juntamente com a ferramenta que se pretende desenvolver.

[4] Crowd- Grupo alargado de pessoas ligadas e coordenadas online(Bernstein, 2012).

2.3.3 Metodologias

Segundo (Almeida et al., 2015), os processos de limpeza de dados são dependentes da especificação de especialistas do domínio. Especialistas estes que normalmente definem quais as operações de deteção de PQD e operações de correção de PQD. Além disso a maioria das técnicas são específicas para um determinado modelo. No entanto é aceite pela literatura que existem um conjunto de operações de limpeza que são comuns nos vários modelos de dados. Enumeram-se algumas metodologias utilizadas para efetuar limpeza de PQD (Li, 2012):

- **Parsing** – são efetuados *parsings* para detetar erros de sintaxe. Ex: Verifica se uma dada string faz parte da linguagem em uso;
- **Transformação de dados** – Transforma os dados (estrutura e instância) de acordo com os requisitos das ferramentas utilizadas. As transformações, normalmente são feitas em *batch*. É comum usar uma linguagem "*multi-database query language* (FRAQL[5])" que facilita a limpeza e transformação dos dados;(Li, 2012)
- **Técnicas de aplicação de restrição de integridade**: - corresponde à verificação da consistência dos dados armazenados. São definidas restrições de integridade que consistem em regras que a base dados não permite violar. Aplicando técnicas de restrição de integridade, eliminam-se PQD de restrição de integridade. Pode -se usar a técnica de verificação de restrição de integridade que permite prevenir a ocorrência de uma restrição de integridade no contexto de uma transação. Por outro lado, a técnica manutenção de restrição de integridade permite corrigir a violação à estrição de integridade após a transação, de forma a garantir que os dados respeitam as restrições de integridade estabelecidas;
- **Técnicas de deteção de duplicados**: - Corresponde a identificar quando dois registos são a mesma representação de uma entidade. As chaves são diferentes e os dados, normalmente contêm erros. Pode-se usar uma técnica de aprendizagem supervisionada ou modelos estatísticos. Para tal são necessários dados de treino. Outra possibilidade consiste no uso de técnicas baseadas em regras de conhecimento do domínio; pode-se usar técnicas que usam métricas baseadas nas distâncias observadas nos dados.
- **Métodos estatísticos**: - Métodos que inspecionam os dados para encontrar PQD e realizam a limpeza de dados. Podem detetar *Outliers*. Para este efeito, são usados dados estatísticos, como média, desvio padrão, intervalos de confiança, entre outros;

2.3.4 Operações de deteção de PQD

A deteção de PQD é a primeira fase da limpeza de dados. Na Literatura, existem várias abordagens para a deteção de PQD. Por questões de simplificação do estudo, relacionadas com

[5] FraQL – É uma extensão do SQL que contém definições para "*feder-actions*", acesso a metadados nas queries, restruturação de resultados das queries e resolução de conflitos de integração (Ziegler, 2004).

limitações temporais, escolheram-se apenas três abordagens embora representativas e relevantes: A abordagem de (Oliveira et al., 2005), a de (Li et al., 2011) e a abordagem de (Alves, 2017).

2.3.4.1 Abordagem de Oliveira.

Segundo (Oliveira, 2008) As operações de deteção (OD) de PQD podem ser definidas em vários NG. Apresentam-se a seguir as operações de deteção de PQD ao nível mais elementar (atributo).

2.3.4.1.1 A nível do atributo

Apresenta-se as operações de deteção a nível de atributo, já definidas na secção 2.2.1, que são executadas pela ordem indicada:

- Valor em falta
- Violação de sintaxe
- Erro ortográfico
- Violação de domínio
- Valor incompleto
- Valor sobrecarregado

No contexto multi-valor são detetadas de forma sequencial os seguintes problemas de PQD (definidos na secção 2.2.1):

- Existência de sinónimos,
- violação de unicidade
- violação de restrição de integridade

2.3.4.1.2 A nível do registo

Neste nível de granularidade existe apenas o problema de restrição de integridade. Aqui, o utilizador pode especificar múltiplas OD de violação de restrição de integridade. nos casos em que as OD ocorrem no mesmo atributo, as operações de deteção são realizadas por ordem. Nos casos em que as OD envolvam atributos diferentes, as operações podem ser realizadas em paralelo (P. Oliveira, 2008).

2.3.4.2 Abordagem de Li

Li (2011) define um conjunto de regras para obtenção de qualidade nos dados, apresentados na Figura 2.

Rule Category	Data Quality Rule
1.Business entity rules	R1.1 Entity uniqueness rules
	R1.2 Entity cardinality rules
	R1.3 Entity optionality rules
2.Business attribute rules	R2.1 Data inheritance rules
	R2.2 Data domains rules
3.Data dependency rules	R3.1 Entity-relationship rules
	R3.2 Attribute dependency rules
4.Data validity rules	R4.1 Data completeness rules
	R4.2 Data correctness rules
	R4.3 Data accuracy rules
	R4.4 Data precision rules
	R4.5 Data uniqueness rules
	R4.6 Data consistency rules

Figura 2: Regras de qualidade nos dados

Estas regras estão divididas em 4 categorias de qualidade. A taxionomia de Li (2011) baseia-se nestas regras de qualidade. Para Li (2011) a qualidade dos dados numa organização pode ser obtida como uma medida de conformidade com as estas regras. Na taxionomia de (Li et al., 2011), a deteção de PQD é classificada de acordo com as 4 categorias de regras definidas(regras de entidade de negócio; regras de negócio do atributo; regras de dependência dos dados e regras de validade dos dados).

Os PQD relacionados com as regras de entidade de negócio são:

- **PQD cardinalidade da relação**: Exemplo: caso do número de clientes, contando o número de clientes na tabela e vendas, não é igual o número total de clientes nas tabelas clientes;

- **PQD da relação recursiva**: - Ex: Uma pessoa supervisiona o trabalho de outra pessoa. Cada Pessoa pode ser supervisionada por várias pessoas. Por Exemplo uma Pessoa de

Nome Manuel supervisiona o trabalho de outra Pessoa chamada Rui, enquanto que o Rui supervisiona o trabalho do Manuel;

- **PQD da relação opcional**: - baseado na regra da entidade opcional. Ex: A venda de um produto a um cliente implica a expedição do produto, no entanto na informação do cliente não está registada a morada de envio.

- **PQD de referência definida, mas não encontrada**: - problema de relacionamento em que a chave relacionada não se encontra na tabela relacionada.

Os PQD relacionados com as regras de negócio do atributo são:

- **Violação de conjunto**: - num tipo enumerado, os valores devem estar dentro dos valores permitidos;

- **Valor fora do intervalo de valores**: - Ex. mês 13 num atributo, quando os valores permitidos são entre 1 e 12;

- **Violação de restrição de valor**: - quando se utilizam restrições de valores em determinados atributos: ex. Dados de uma experiência em jovens e aparece um registo com uma pessoa de 100 anos de idade;

- **Tipo de dados errado**: - por exemplo um dado esperado ser do tipo string é preenchido com um número;

- **Violação de sintaxe**: - Ex: formado de hora errado.

Os PQD baseados baseado em regras de dependências de dados aplicam-se a relações entre duas ou mais entidades do negócio. São exemplos destes PQD as violações de restrição de integridade, contradições encontradas nos dados, má derivação de dados a partir de vários atributos.

As regras de validade de dados gerem a validade da qualidade dos dados. Exemplos de validades consideradas são: registo em falta; valor em falta; dados com valores sem sentido; introdução de valores estranhos; falta de elementos nos dados e violação de regras de identidade.

2.3.4.3 Abordagem de Alves

Em (Alves, 2017) recorre-se a regras de validação para efetuar a deteção de PQD. Define regras de validação em função de restrições de integridade. As restrições de integridade consideradas são:

- **Integridade de domínio**: - O valor do atributo tem de pertencer ao domínio definido para o atributo;

- **Integridade de vazio**: Os valores não podem ser nulos;
- **Integridade de coluna**: Valores tem de respeitar a integridade de domínio

  e respeitar as regras definidas para o atributo.

Seguidamente são estabelecidas associações entre as regras de validação e os tipos de PQD, resultando nas seguintes PQD tratados pelo autor: violação de domínio; erro de ortografia; valor em falta e erro de sintaxe.

2.4 Interface do utilizador

A interface do utilizador pretendida será um sistema de fluxo de trabalho, aqui designado por *workflow*, de operações de limpeza de dados. Pretende-se nesta seção descrever o estado de arte relacionado com interfaces gráficas de sistemas de *workflow***.**

As interfaces gráficas com o utilizador - "Graphical User Interface" (GUI) estão disponíveis em diferentes tipos de dispositivos. Dada a natureza do problema em causa, a deteção de PQD, só serão referidas interfaces gráficas para serem utilizadas em computadores ("LapTops"). Podem ser aplicações desktop ou aplicações web. Dada a complexidade da solução de *workflow* pretendida, entende-se não ter interesse explorar as interfaces gráficas em dispositivos móveis.

Dada a imensa diversidade de linguagens e plataformas existentes, escolheu-se abordar apenas tecnologias da plataforma Java, por ser uma plataforma que apresenta soluções nos mais variados domínios de aplicação na indústria. Outro fator que pesou na escolha deste caminho a explorar, foi a familiarização do autor com a plataforma Java em relação a outras tecnologias de desenvolvimento. Não se pretende com este trabalho fazer comparação da plataforma Java com outras linguagens e plataformas de desenvolvimento de aplicações.

Serão abordadas tecnologias como o Swing, e Java FX para desenvolvimento de aplicações desktop. As tecnologias web abordadas serão as mais atuais no contexto do HTML5. Será apresentado o *canvas* em HTML5, o *webgl* e o SVG. Estas tecnologias têm desenvolvimentos web no domínio de aplicações *workflow*, pelo que, neste contexto de *workflow*, serão apresentadas algumas bibliotecas e *frameworks* de desenvolvimento de aplicações.

2.4.1 Sistemas de Workflow

Um *wokflow* permite a automatização dos processos de negócio, permitindo um alinhamento do negócio com as tecnologias da informação (Guerrero García, Vanderdonckt, Calleros, & Winckler, 2008). "um sistema de gestão de *workflow* é utilizado para reduzir a carga de trabalho, organizando processos de negócio, conhecidos como *workflow*" (Alfredsson &

Lundmark, 2015). O mesmo autor acrescenta que um sistema de *workflow* pode ser definido por um "processo de workflow" (PO) ou simplesmente *workflow*, um ponto de partida e um ponto de finalização do processo de *workflow*. O processo de *workflow*, tem um determinado tempo de vida, e pode ter algumas propriedades. Ex.: condições de substituição de tarefas; condições que indicam o progresso de determinado caso de utilização, com indicação das tarefas que foram e as que não foram executadas. Um sistema de *workflow* é estruturado por uma sequência de tarefas. Cada tarefa pode ser executada automaticamente ou ter a intervenção de pessoas ou máquinas. Existe um roteamento no qual se define como as tarefas do *workflow* são realizadas. As tarefas podem ser realizadas em sequência, em paralelo, de forma seletiva, ou de forma iterativa. É definido ainda o conceito de promoção que descreve como a tarefa é desencadeada. Este desencadeamento pode ocorrer de forma automática ou manual (Alfredsson & Lundmark, 2015).

2.4.1.1 Interfaces baseadas em Fluxogramas

Um fluxograma é definido como um desenho gráfico de vários caminhos possíveis de um *workflow* (Klaka et. al, 2013). Segundo (Freeman, 2019) o fluxograma permite representar operações de *workflow*. O fluxograma contém fluxos que representam sequências de operações lógicas necessárias para a resolução de um determinado problema. Utiliza formas gráficas (Ex: quadrados ou outras formas) com a descrição de cada operação. O conjunto de formas e fluxos num fluxograma representam os passos necessários na execução de determinado *workflow*. Em (Copeland, Leffler, Parsons, & Terrill, 2009) é referido que os fluxogramas escondem a complexidade das tarefas executadas nos *workflows*, fornecendo uma visualização instantânea de todo o processo de *workflow*.

2.4.1.2 Padrões de interfaces *workflow*

Na literatura, relativamente ao desenvolvimento de software, existem trabalhos que dedicam esforço na criação de padrões para resolver problemas comuns de desenvolvimento de software. Os sistema de *workflows* também estão incluídos nesse esforço, pelo que se tem desenvolvido "Workflow User Interface Pattern"(WUIP) - Padrões de Interface de Utilizador em sistemas de *workflow* (Garcia, 2012). Existem um serie de padrões para desenvolver sistemas de *workflow*. Em (Russell, van der Aalst, ter Hofstede, & Edmond, 2005) os padrões de *workflow* são divididos em 7 categorias:

- **Padrões criacionais**: - Definem a forma como os itens, após serem criados, são usados, para a execução de determinada tarefa;

- **Padrões "Push":** - Servem para definir como o sistema disponibiliza os itens criados;

- **Padrões "Pull":** - Refere-se a situações que evidenciam a necessidade de se obter itens a partir do sistema, necessários à execução de determinadas tarefas. Ocorre em situações em que a execução de uma tarefa é iniciada pelo recurso e não pelo sistema;

- **Padrões desvio**: - Situações em que os itens de trabalho distribuídos por diversos recursos são interrompidos, alterando o fluxo de estado do workflow;

- **Padrões "auto-start":** - correspondem a situação em que a execução de itens de trabalho é desencadeada por eventos que pertencem ao ciclo de vida dos itens pertencentes ao *workflow*;

- **Padrões de visibilidade**: - Classificam os âmbitos de disponibilidade de um item de trabalho em relação aos recursos;

- **Padrões de recursos múltiplos**. Referem-se a situação de relação de vários itens de trabalho com vários recursos. Mais concretamente situação em que os recursos podem executar mais do que um item de trabalho em simultâneo.

2.4.2 Tecnologias desktop

No desenvolvimento de aplicações desktop de *workflow* em Java o destaque vai para o Swing com a capacidade de desenho 2D e, para o Java FX com a sua capacidade de desenho de 3D. O Swing e o AWT são bibliotecas gráficas incluídas do JRE e JDK. O Swing é uma biblioteca mais recente do que o AWT e com mais componentes. Existe também o *"Standart Widget Toolkit"* (SWE) da IBM usado para criar ambientes gráficos desktop em Java (Caelum, 2020).

JavaFX é uma API Java para criar interfaces gráficas com componentes ricos. De acordo com (Tim Hodkinson, 2020), as GUI do *JavaFx*, surgem para substituir as GUI do SWING e, podem ser usadas em diversos dipositivos. *JavaFX* está incluído com o JDK até ao Java 8. Com versões posteriores do JDK é necessário importar bibliotecas de terceiros. A comunidade *OpenJFX* suporta o desenvolvimento de novas versões do *JavaFX*. Em (openJFX, 2020) é referido que o *OpenJFX* é uma plataforma para desenvolvimento de software de próximas gerações, open-source, para aplicações desktop, mobile e sistemas embebidos construídos em Java.

2.4.2.1 *Frameworks* e bibliotecas

Consideram-se as seguintes bibliotecas para desenvolver aplicações de *workflow*:

- JDiagram: - Uma biblioteca de classes baseada em *Swing* da *Mindfusion*, que permite criar gráficos e diagramas de *workflow*. É necessário licenciamento (MindFusion, 2020);

- JGo: Permite criar gráficos e diagramas com editores de *workFlow* e *flowcharts* ligados. É necessário licenciamento (Northwoods, 2020);

- Yfiles: Esta *framework* existe para *JavaFX* e *Swing*. contem um *toolkit* de componentes para desenhar aplicações de diagramas. É necessário licenciamento (yWorks, 2020);

- FXDiagram: - é uma *framework* baseada em *JavaFX*. Permite escolher os componentes do diagrama, e organiza-los (Koehnlein, 2015).

De acordo com o requisito de ferramenta a criar ser open-source, a única *framework* que podemos considerar, na implementação da solução, é a *FXDiagram*.

2.4.3 *Tecnologias web*

As tecnologias baseadas em HTML são imensas. No mundo Java são diversas as tecnologias que permitem criar aplicações web. Existe um conjunto considerável de *frameworks* e bibliotecas cuja implementação é baseada no Java *Enterprise Edition*. Exemplos: "Java server Faces" (JSF), *Struts*; *Spring*, GWT, *Vaadin* e *Liferay*, entre outras. Estas *frameworks* facilitam a geração do código HTML, CSS e JavaScript, no lado do cliente, que de outra forma seria muito trabalhoso. Além disso facilitam a comunicação da parte HTTP do lado do cliente com o container aplicacional, do lado do servidor, onde reside o código de lógica de negócio.

As tecnologias web foram evoluindo. As linguagens baseadas nas normas *ECMAScript* mais concretamente o JavaScript (JS), foi uma das áreas da web que mais evoluiu. *Frameworks* como *JQuery* simplificam muito a codificação em JS, fazendo com que se tenha de escrever menos código para se ter as mesmas ações nos Browsers. A introdução do *BootStrap* Simplificou muito a construção de sites responsivos para se usar aplicações web em dispositivos móveis. Com o aparecimento do HTML5, surgiram novas capacidades de design gráfico com tecnologias como o *canvas* e a *webgl*.

Nos Browsers modernos, os gráficos são "renderizados" (geração da visualização) em SVG. A biblioteca *HighChart* é uma biblioteca JS utilizada para gráficos dinâmicos na web. *GoJs* é outra livraria JavaSript que permite diagramas interativos com nós, ligações entre componentes, e funcionalidades de "*drag and drop*" (Shahzad, Sheltami, Shakshuki, & Shaikh, 2016). Serão apresentados mais algumas *frameworks* e bibliotecas de JS mais à frente.

2.4.3.1 HTML5 - Tecnologias para construção de *workflows*

Indicam-se de seguida a descrição de algumas tecnologias que potenciam a criação de *workflows*:

- **Canvas**: - Canvas é um elemento HTML que permite gerar a visualização de gráficos compostos por pixéis ou pontos de cores individuais. É possível aceder ao canvas através do JS e desenhar pixéis (Alfredsson & Lundmark, 2015).

- **WebGL**: - WebGL é uma API DOM [6]para desenho de gráficos 3D. WebGL é baseado no OpenGL[7], sendo assim uma API semelhante. WebGL é uma API baseada no DOM e no

[6] DOM- "Document Object Model" É um padrão, com estrutura em árvore, para gerir conteúdo do HMTL (Gupta, Kaiser, Neistadt, & Grimm, 2003).
[7] OPENGL - É um sistema com livraria de desenho gráfico (Segal & Akeley, 1999).

JavaScript (Alfredsson & Lundmark, 2015). *WebGl* pode ser utilizada em elementos canvas do HTML5 (Silva, 2018). Permite criar animações 3D, e obter visualizações com aceleração de hardware de baixo nível. Ex: A biblioteca Three.js permite funções de visualização 3D usando código simples, sem necessitar de compreender as partes complicadas do *WebGL* com Gráficos, detalhes técnicos, e a linguagem complexa de webGL (Liu et al., 2019). Em (Liu et al., 2019) é apresentado um exemplo de um sistema de "*flowchart*"- fluxograma que inclui camada de processamento de dados;

- **SVG**: - "Scalable Vector Graphics" (SVG) é uma linguagem XML utilizada para descrever elementos gráficos na forma vetorial a duas dimensões, como alternativa à representação por cores de pixéis. Cada elemento XML contém as propriedades do elemento gráfico. O vetor de elementos gráficos é convertido pelo browser num bitmap. A biblioteca gráfica mantém uma representação interna dos elementos gráficos do SVG. Quando uma figura do SVG for desenhada, a representação interna dos elementos é atualizada, permitindo que a biblioteca trate da inicialização, gestão, manutenção e limpeza da representação gráfica. A representação interna é feita com elementos do DOM, permitindo interação com o utilizador, através do JavaScript (Alfredsson & Lundmark, 2015);

- **VML**: - ("Vector Markup Language") VML é uma linguagem de marcação baseada em texto. Permite guardar dados específicos no elemento gráfico. Permite formas de *Scripting* através do DOM, para controlar os elementos gráficos das paginas web (Nuttayasakul, 2003);

2.4.3.2 *Frameworks* web *fullstack*

Existem várias tecnologias web que permitem o desenvolvimento de aplicações web. Aqui apresentamos apenas algumas implementações do Java *enterprise edition*, e apenas no contexto de criação de aplicação web:

- Struts: - É uma *framework* open-source da Apache. Baseia-se no padrão de desenvolvimento de software "Model View Controller" (MVC). Contém uma "Aplication Programming Interface" (API) com um conjunto de rich features que facilitam o desenvolvimento web. Contém classes de "action" baseadas em "Plain Old Java Object" POJO, que implementam a logica de negócio; contém "Controllers"- controladores que funcionam como filtros com funcionalidades como intercetar e validar pedidos e respostas HTTP; As visualizações são contruídas com ficheiros Java server Pages (JSP) (Baeldung, 2019);

- JSF: - Java Server faces (JSF) é uma *framework* Java padrão para o desenvolvimento de interfaces do utilizador em aplicação web. Com o JSF, o desenvolvimento das interfaces gráfica é facilitado através de um conjunto de Padrões de desenvolvimento, baseados em experiências adquiridas de tecnologias antecedentes do JSF como Java "servlets" e JSP (Schalk & Burns, 2010). Existe um conjunto de componentes de visualização, que facilitam muito o código HTML e JavaScript. Inclui características do padrão MVC, pelo

que existe uma separação da camada de visualização com a camada de negócio. Existem varias implementações do JSF: Ex. *Mojara* da Oracle, e *MyFaces* da apache (Caelum, 2019);

- Spring MVC: - Spring MVC é uma *framework* Spring baseada no padrão MVC com uma separação por camadas: Camada de apresentação; camada de modelo de domínio ou serviço e a camada de persistência (Ladd, Darren Davison, Steven Devijver, & Colin Yates, 2006). *Spring* MVC baseia-se no container "Inversion Of Control" (IoC) do *Spring*, utilizando os recursos deste container *IoC*. Ex: para simplificar as configurações necessárias nas aplicações (Mak, 2008).

As tecnologias aqui apresentadas contem boa integração com software de terceiros para a integração de *frameworks* que facilitem a implementação de aplicações *workflow*. Existem boas comunidades e documentação para o desenvolvimento de aplicações baseadas nestas tecnologias.

2.4.3.3 *Frameworks* e bibliotecas de JavaScript

JavaScript surgiu como linguagem de Script utilizada pelos Browsers. A linguagem aplicada a elementos HTML, permite interagir com as aplicações web (MDN contributors, 2020). Aplicada aos elementos do DOM do HTML, o JavaScript é a linguagem web core, que é interpretada e executada por um motor de JavaScript presente nos browsers (Gem Dot, Alejandro Martínez, & Antonio González, 2015).

A programação diretamente em JavaScript tem revelado complexidade no entendimento das APIs, além de revelar um dispêndio de tempo significativo na codificação, pelo que desde cedo se começaram a utilizar bibliotecas e *frameworks* de JavaScript para simplificar a vida ao programador. Uma das bibliotecas amplamente usada é o *JQuery* (David Flanagan, 2011).

JQuery foi uma das primeiras bibliotecas JavaScript amplamente utilizada. Uma das grandes vantagens do *JQuery* foi a resolução das incompatibilidades de código JavaScript entre os diferentes Browsers. Outra vantagem foi a simplificação das chamadas AJAX, utilizado para chamadas assíncronas das páginas, sem necessidade de "renderização" (geração da visualização) total da página. É de acrescentar a facilidade com que o *JQuery* permite encontrar e manipular os elementos HTML e propriedades "Cascade Style Sheet" (CSS). O *JQuery* contem ainda um conjunto de funções úteis para trabalhar com objetos e arrays (David Flanagan, 2011).

As implementações do JavaScript seguem as especificações do *ECMAScript*, que definem um conjunto de regras e orientações incluindo as implementações tradicionais do JavaScript. As tecnologias web tem evoluído e, conjuntamente as especificações *ECMAScript*, principalmente as versões 5 e 6 do *ECMAScript* (ES5 e ES6). As alterações do ES5 estão relacionadas com as implementações tracionais do JavaScript, vulgarmente conhecidos na literatura por "Vanilla Script", enquanto que as alterações do ES6 incluem a programação

funcional. A maioria das *frameworks* de JavaScript atualmente utilizam as especificações do ES5 e ES6 (Maria, 2018).

O Bootstrap.js em conjunto com o HTML e o BootStrap.css, permitiram uma grande evolução de aplicações web responsivas, ajustáveis à dimensões de qualquer dispositivo, o que favoreceu a utilização das aplicações web em ambientes mobile (Maria, 2018).

Em 2009 Node.js surgiu como um ambiente de execução em tempo real de JavaScript, permitindo a execução de programas escritos em JavaScript. O node.js permitiu assim a possibilidade de execução de código JavaScript do lado do servidor com funcionalidades relevantes como a execução paralela. Sendo node.js open-source e com uma grande comunidade a contribuir para o seu desenvolvimento, a popularidade da linguagem JavaScript aumentou significativamente. O node.js é uma *framework* modular com muitos pacotes "Node Package Managers" (NPM), o que influenciou o surgimento de diversas *frameworks* de JavaScript para o desenvolvimento de aplicações de "front-end" (Maria, 2018). Apresentam-se a seguir exemplos de *frameworks* e livrarias baseadas em node.js:

- **Angular**: - Angular é uma *framework* de JavaScript que permite contruir aplicações web e mobile. A primeira versão do angular é o *AngularJS*. Segundo (Mathieu Nayrolles, Rajesh Gunasundaram, & Sridhar Rao, 2017), o Angular a partir do Angular2 não é um incremento do *AngularJS*. O angular foi completamente reconstruído e tem novas funcionalidades como injeção de dependências, carregamento dinâmico; *routing*; integração com *TypeScript* e programação funcional com expressões lambdas. Em (Sudheer Jonna & Oleg Varaksin, 2017) é referido que o Angular 2 e versões posteriores são construídas de acordo com as normas de *ECMAScript* 2015/2016 que correspondem às versões ES6 e ES7 do *ECMAScript* respetivamente. A linguagem *TypeScript* apresenta vantagens como: deteção de erros no desenvolvimento; mais fácil de refatorar o código e existem diversos padrões de software para *TypeScript*;

- **React**: - É uma biblioteca de JavaScript para construir interfaces do utilizador. Segundo (Adam Boduch, 2017) *React* não é uma *Framework* . *React* é como a camada de apresentação de uma aplicação. Tal como *JQuery* manipula elementos da interface do utilizador. Os componentes do *React* mudam a visualização do utilizador. Os componentes do *React* coexistem entre os elementos DOM do HTML e os dados. Os componentes do *React* convertem os dados em HTML. *React* está dividido em 2 APIs: *React* Dom e *React Component*. *React* Dom é utilizada para gerar a visualização da página web. O *React Component*, são a parte que o *React* Dom utiliza para gerar a visualização. O *React component* divide-se em 4 áreas: dados; ciclo de vida; eventos e JSX (Adam Boduch, 2017). Segundo Boduch (2017) a simplicidade do *React* permite que se dedique mais atenção a padrões de desenvolvimento de software. De acordo com (Michele Bertoli, 2017) o *React* é rápido devido à forma como lida com o componentes do DOM do HTML.

- **Vue**.js: - Segundo (Andrea Passaglia, 2017) *Vue* é uma *framework* de JavaScript leve e muito poderosa, que permite criar uma aplicação nalguns minutos sem precisar de *setup*. Em (Olga Filipova, 2017), pudemos ver como criar de forma simples, uma pequena aplicação com *node*.js usando *Firebase* e *BootStrap*.css. Segundo Filipova (2017), é relativamente fácil de desenvolver uma pequena aplicação em *Vue* usando *BootStrap* e *Firebase*, sem necessitar de conhecimento aprofundados de *Vue*, *Bootstrap* e *Firebase*[8]. Segundo (Andrea Passaglia, 2017) o *Vue*, tal como o *React*, permite programação reativa, usando o *VueX* que permite o carregamento de dados assincronamente. Para tal usa Mutações que correspondem a uma forma de abstração para decompor as mudanças de estados em unidades atómicas.

A importância desta secção reside no facto da maioria das *frameworks* apresentadas na secção seguinte serem baseadas nas tecnologias descritas nesta secção.

2.4.3.4 Bibliotecas e *Frameworks* de *workflow*

Escolheu-se apresentar as seguintes *frameworks*/bibliotecas de *workflow* em JavaScript:

- NoFlo: - A Biblioteca de JavaScript *NoFlo* comtempla programação baseada em fluxos. A lógica é definida como um grafo, os "nós" são instâncias dos componentes *NoFlo* e os ramos definem ligação entre os componentes. O *workflow* é definido numa estrutura JSON. *NoFlo* é freeware e é usado com Node.js (Persson & Angelsmark, 2015). Os componentes *NoFlo* reagem às mensagens recebidas. Quando um componente recebe mensagens nas suas portas de entrada, executa uma operação predefinida e envia o resultado como uma mensagem para as suas portas de saída.(Henri Bergius, 2020);

- *WorkFlow* ES: - É uma biblioteca de JavaScript ES6 para node.js. Permite que as operações de *workflow* sejam utilizadas no contexto de uma transação, utilizando para o efeito as transações Saga[9]. Esta disponível sobre licença MIT (Gerlag, 2016/2020). No entanto existe muito pouca documentação sobre a biblioteca;

- Metatron JS: - É uma biblioteca de JavaScript de "User interface" (UI), com a qual se consegue criar visualizações de *workflow* (*VirtusaPolarisGTO/MetatronJS*, 2017/2020). Existe pouca documentação, no entanto pode-se consultar um exemplo em: https://virtusapolarisgto.github.io/MetatronJS/demo/;

[8] Firebase - O Firebase é uma base NoSQL de dados de tempo real baseada na Cloud. Os dados são guardados no formato JSON (Moroney, 2017).

[9] Saga - Consiste numa transação de longa duração, que é vista como uma sequência de sub - transações que podem estar interligadas. No entanto cada Sub - transação é uma transação que preserva a consistência da base de dados (Alonso et al., 1996).

- FlowCharty: - *FlowCharty* usa a biblioteca d3.js para gerar dinamicamente o diagrama de fluxo SVG de um determinado *workflow* (Tago, 2018/2019). D3.js É uma biblioteca de JavaScript. Permite ligar dados aos objetos do DOM e assim alterar os documentos. Exemplo com um determinado conjunto de dados, pode-se criar uma tabela HTML ou desenhar um gráfico de barras (Bostock, 2020).

- GoJS: - GoJS é uma biblioteca de JavaScript e *typeScript* para criar diagramas web. Permite criar *flowchart* com técnicas de "drag and drop". Pode ser usada com outras bibliotecas e *frameworks* como Angular, *React* e *vue.js*, mas também pode ser usa sem necessitar de outras bibliotecas de JavaScript. É necessário licenciamento (Northwoods Software, 2020);

- Draw2d: - Draw2d é uma biblioteca de JavaScript que permite criar *workflows* com tecnologia HTML5. A geração da visualização é efetuada recorrendo à tecnologia SVG. Funciona com versões do *ECMAScript* (ES) a partir da versão 6. É biblioteca de código aberto sobre licença MIT (Andreas Herz, 2020).Esta biblioteca não tem muita documentação disponível. Pode-se visualizar um exemplo de aplicação em: http://www.draw2d.org/draw2d_touch/jsdoc_6/#!/example/shape_collapsible;

- jquery.flowchart.js: - jquery.flowchart.js é um plugin de JavaScript open source, que permite desenhar componentes gráficos na forma de retângulos e ligação entre os componentes. Como é mostrado na Figura 3, o componente tem entradas e saídas que permite estabelecer ligações com outro componente.

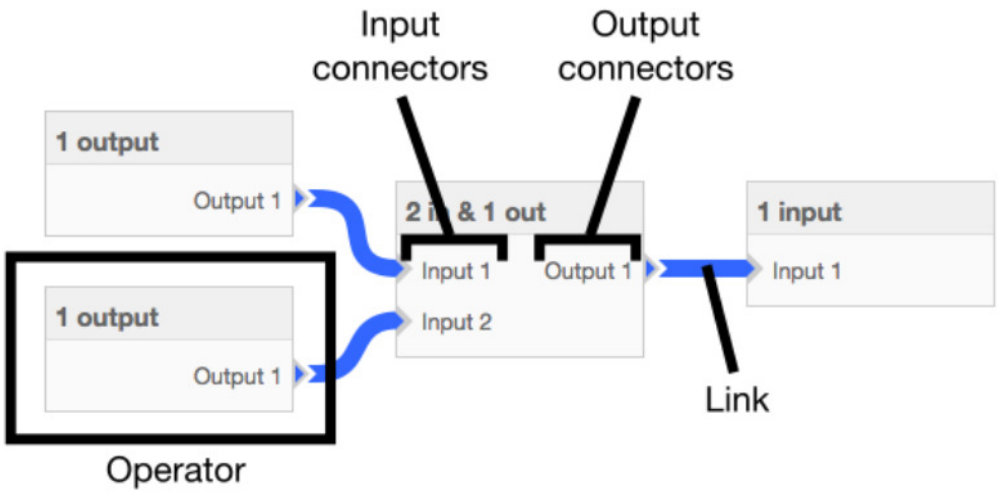


Figura 3: Ligação entre componentes gráficos(Drouyer, 2016/2020)

Os componentes são criados dinamicamente e deslocados para o local pretendido através de "Drag and Drop". Existe um *Hash* que define os componentes no *flowchart*. O campo "Id" define o id do componente e o campo "valor" define a informação de

cada componente. Considera-se que cada componente contém a seguinte informação: valor do topo do componente em relação à *frame* que o contém; valor da posição esquerda do componente em relação à *frame* que o contém; tipo de componente; propriedades: título, corpo do componente; classes CSS adicionadas ao componente DOM; *Hashmap* com os conectores de entrada; *Hashmap* com conectores de saída (Drouyer, 2016/2020);

- jsPlumb: -jsPlumb é uma biblioteca de JavaScript que permite criar ilustrações gráficas e workflows (Gesing et al., 2015). Tem uma versão "Comunity Edition" open-source. Existe o *toolkit jsPlumb* que auxilia na criação de *flowchart*, e tem integração com outra bibliotecas e *frameworks* de JavaScript muito utilizadas atualmente, como o angular, *React* e *Vue*.js (jsPlumb Community, 2020).

2.4.3.5 Comparação de *frameworks*/ bibliotecas de *workflow*

A comparação das *Frameworks*/bibliotecas apresentadas nesta secção está apresenta da tabela 3.

Tabela 3: Comparação de tecnologias JavaScript para *workflow*

Critério / Tecnologia	Open source	Cria componentes dinamicamente	Ligação dinâmica	Boa documentação	complexidade
NoFlo	✓	✓	✓		
Workflow ES	✓				✓
Metratron JS	✓	✓	✓		
Flowcharty	✓				
GoJs		✓	✓	✓	
Draw2D	✓	✓	✓		✓
JQuery flowchart	✓	✓	✓	✓	
JsPlumb C.E.	✓	✓	✓	✓	

Tendo que conta que se pretende uma ferramenta de deteção de PQD freeware, com base na comparação da Tabela 3, o plugin *JQuery flowchart* e a biblioteca *JsPlumb* "comunity edition" aparentam ser as tecnologias baseadas em JavaScript mais adequadas.

2.4.4 Restrições de implementação

O sistema será implementado com *frameworks* e bibliotecas open source. Como o autor é inexperiente nestas tecnologias a solução encontrada poderá não ser a melhor. Existem diversas *frameworks* e bibliotecas que ajudam na implementação de uma aplicação de *workflow*, no entanto várias *frameworks* são dispendiosas, enquanto outras que são open-

source, tem pouca documentação. Será necessária uma análise cuidada das tecnologias a utilizar para desenhar a interface gráfica. Devido á pouca documentação de parte das bibliotecas e *frameworks* existentes, é conveniente realizar protótipos que sirvam de Prova de Conceito (POC) para uma escolha adequada das tecnologias a utilizar. As POC também são importantes para a fase de Design da aplicação na medida em que é necessário ter conhecimento de domínio sobre a forma como a camada de apresentação se liga com a parte de controlo do *workflow*.

3 Análise de Valor

A análise de valor é uma fase imprescindível no processo de criação de um novo produto. Segundo (Nick Rich, Matthias Holweg, & Wirtschaftsing, 2000), a análise de valor corresponde a um processo e não a uma técnica que tem como objetivo aumentar o lucro de produtos e, ao mesmo tempo, desenvolve um conjunto de técnicas para atingir esse mesmo objetivo. Segundo (Nick Rich et al., 2000), a análise de valor é quase universal e pode ser aplicada para analisar produtos e serviços existentes fornecidos por empresas nas indústrias transformadoras, bem como nas empresas fornecedoras de serviços.

Segundo os mesmos autores, o processo de análise de valor é utilizado com o objetivo de fornecer ao cliente um produto ou serviço melhor com um custo mínimo de produção. O processo de análise de valor é um dos recursos principais do negócio e procura atingir a TQM em tudo o que se faz para satisfazer os clientes. Na análise de valor, existe uma revisão sistemática do design do produto para servir de termo de comparação das expectativas do cliente para com o produto.

O valor de uso de um produto e a fonte de valor envolvidos no processo de tentar melhorar o valor de um produto designa-se de valor estimado. Os gestores compreendem a natureza de custos na fabricação de determinado produto, mas não existe nenhuma relação direta entre custo de fabricação e valor estimado para o cliente. São necessárias várias revisões do processo de forma a permitir baixar os custos de produção mantendo o valor do cliente (Nick Rich et al., 2000).

3.1 Inovação

A inovação é necessária no processo de criação ou melhoria de um produto ou serviço, para que seja percecionado valor por parte do cliente, e aumentada a rentabilidade das empresas. Segundo(Roper, Du, & Love, 2008) as atividades de inovação na criação de um novo produto ou processo, correspondem a uma criação de valor que pode resultar numa melhoria do negócio. A Inovação é utilizada para criar ou aumentar o valor dos bens de consumo. Com o processo de invocação adquire-se conhecimento estratégico para a criação de valor de uma empresa (Oliveira & Alves, 2014).

Um dos modelos usados para o processo de inovação é o “Fuzzy Front end” (FFE) – Front-End difuso. Neste modelo a inovação é dividida em três áreas, o FFE, “new product development “ (NPD) - Desenvolvimento de novo produto e a comercialização (Koen, Bertels, & Kleinschmidt, 2014).

O FFE corresponde à primeira parte do processo de inovação. Nesta fase são identificadas oportunidades e definidos conceitos. Contudo nesta fase existe ainda pouca informação acerca de mercados e tecnologias. Tem sido prestada muita atenção nas atividades de FFE para se poder aumentar a probabilidade de sucesso no desenvolvimento e comercialização de novos produtos (Koen, P., Allen Clamen, Albert Johnson, & Seibert, 2002). O FFE tem sido aplicado de diferentes formas nas diversas empresas, pelo facto de não existir uma estrutura ou norma, que permita definir uma linguagem e vocabulário comum.

Segundo (Achiche, Appio, McAloone, & Di Minin, 2012) as incertezas de tecnologias, mercados e recursos, e a pouca geração de ideias com qualidade, contribuem para a baixa eficiência do NPD.

Surgiu um novo modelo designado Novo Conceito de Desenvolvimento (NCD), para fornecer uma terminologia comum para o FFE, e assim diminuir a imprecisão atribuída à fase de FFE, identificando atividades específicas e atributos das organizações (Koen et al., 2002).

3.2 Novo Conceito de Desenvolvimento

O modelo NCD divide-se em três partes que se pode visualizar na Figura 4: motor, a roda e o aro. O motor consiste nos elementos do núcleo e serve para fornecer força ao processo de FFE. Estes elementos são os atributos organizacionais, tais como os valores, estratégia, cultura e a colaboração entre equipas. A roda é constituída por cinco atividades: identificação de oportunidades, análise de oportunidades, geração de ideias, seleção de ideias e definição de conceito. O aro, consiste nos fatores ambientais externos que influenciam o motor e os elementos de atividade (Koen et al., 2014).

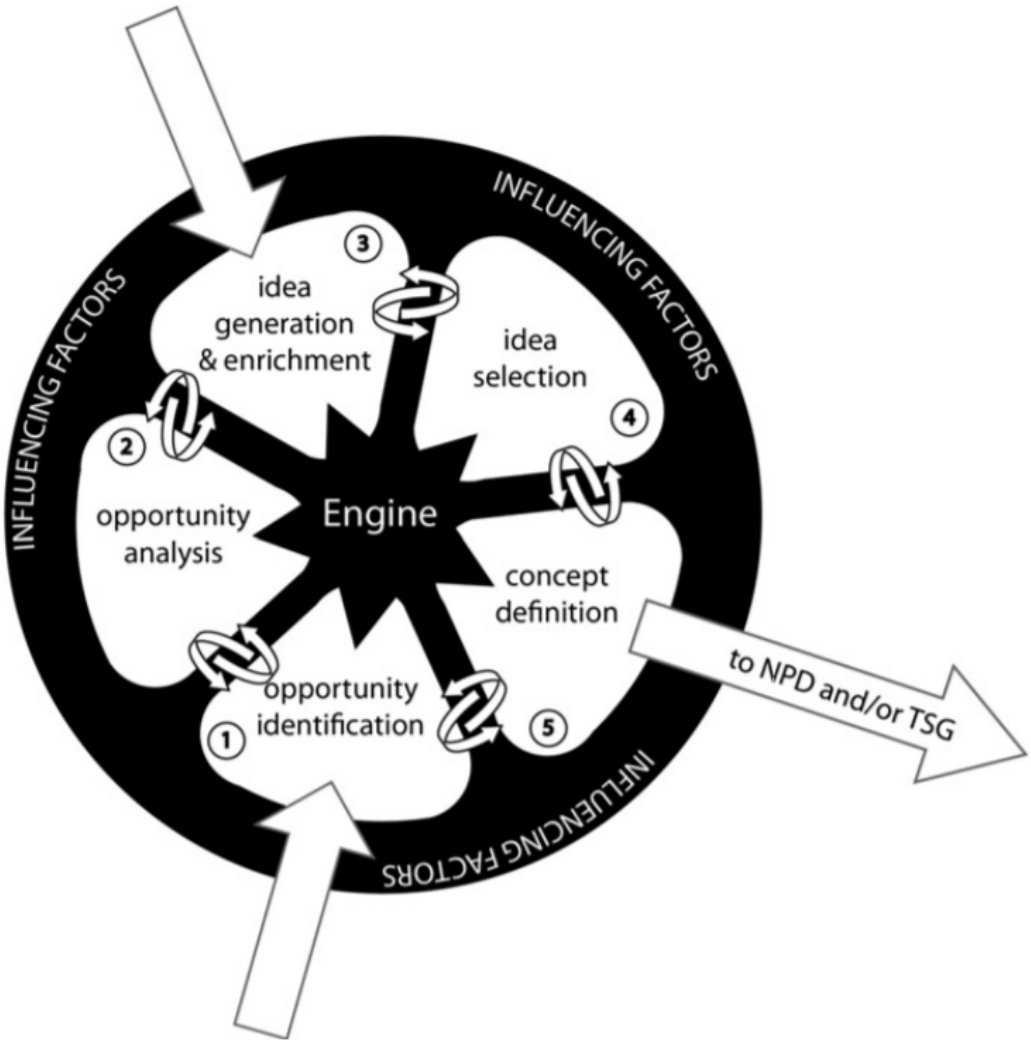


Figura 4: O modelo NCD. (Koen et al., 2014)

A forma circular do modelo mostra o fluxo de ideias e interação entre os cinco elementos. As setas a apontar para o modelo indicam que os projetos podem iniciar com base numa geração de ideia ou na identificação de uma oportunidade (Koen et al., 2014).

Os projetos seguem o sentido de atividade de geração de conceito para o processo NPD, que ao contrário do NCD, são sequenciais. As atividades que constituem o "Front End of Innovation" (FEI)- Front end de inovação, são descritas nas secções seguintes.

3.2.1 Identificação da oportunidade

Este elemento tem como objetivo identificar oportunidades exploráveis, sejam elas comerciais, no sentido de alocar corretamente os recursos em novos segmentos de mercado, otimização de processos ou aumento de eficiência dos métodos existentes. Para executar esta atividade existem diferentes técnicas que possibilitam a sua execução (Koen et al., 2002):

- **Roadmapping:** capturam, graficamente, as forças de um negócio, para permitir um mapeamento que permita melhorar a partilha de conhecimento, e competências dos recursos das equipas.

- **Análise das tendências tecnológicas:** dedica-se à pesquisa e análise de tendências tecnológicas relevantes competitivas para o projeto.

- **Análise de tendências do cliente:** prende-se com a pesquisa e análise das tendências dos clientes. Os clientes que necessitam de obter qualidade nos dados, tendem a procurar ferramentas que o permitam.

- **Análise de inteligência competitiva:** elemento essencial na atividade de identificação de oportunidades, uma vez que permite obter, analisar e comunicar informação de tendências competitivas que ocorrem fora da organização, possibilitando a definição de uma estratégia de negócio.

- **Pesquisa de mercado:** pesquisa de mercados inerentes ao projeto. Neste ponto são considerados potenciais clientes, profissionais que necessitem de correção nos dados, mas não tem conhecimento do domínio. Uma ferramenta que ajude a detetar PQD é uma oportunidade a considerar.

- **Planeamento de cenário:** desenvolvem-se se cenários para visualizar (simular) o futuro. As visualizações fornecem informações importantes acerca do futuro. O confronto de vários cenários permite, às organizações, quais as melhores oportunidades a explorar.

A existência de mercados extremamente competitivos, implica a utilização de técnicas adequadas para explorar as melhores oportunidades que vão de encontro às necessidades dos clientes, tendo em conta o que já existe no mercado. Com as ferramentas de identificação de oportunidades é possível encontrar padrões de atividade que revelam necessidades de potenciais clientes de novos produtos a desenvolver.

No problema em estudo identifica-se a necessidade de detetar PQD. Esta necessidade representa uma oportunidade de criar valor. Como referido no capítulo do estado da arte, existem muitas ferramentas para realizar limpeza dos dados, muitas delas com as suas caraterísticas restritivas já enunciadas anteriormente. As ferramentas existentes são ferramentas que detetam e corrigem PQD. No entanto, não foi encontrada nenhuma ferramenta generalista que faça apenas a deteção de PQD. A complexidade de utilização de

ferramentas de limpeza de PQD torna o processo de limpeza, de difícil execução, em parte, devido à necessidade de conhecimento específico dos utilizadores. A utilização inadequada destas ferramentas resulta numa baixa eficiência na correção dos PQD com os consequentes impactos já referidos anteriormente. Uma ferramenta que permita detetar PQD, permite definir melhores estratégias para a correção adequada.

3.2.2 Análise da oportunidade

Nesta secção é efetuada a análise da oportunidade encontrada, para se estudar a sua viabilidade. Em (Koen et al., 2002) é proposto os seguintes métodos de análises de oportunidades:

- **Enquadramento estratégico:** enquadramento de determinada oportunidade, nomeadamente as forças, fraquezas e ameaças tecnológicas do mercado e da organização;
- **Avaliação do segmento de mercado:** a avaliação do segmento de mercado adequado à oportunidade. Os profissionais de *Business Inteligence* e *DataMining*, necessitam de saber com que tipo de PQD se deparam;
- **Análise da concorrência:** permite identificar quem são os competidores no segmento de mercado alvo. Identifica os tipos de produtos com vantagem competitiva e permite avaliar as estratégias de concorrência. No mercado existem diversas ferramentas que fazem a deteção e correção de PQD. No entanto não foi encontrada nenhuma ferramenta generalista e open- source que permita detetar PQD. Algumas das ferramentas de limpeza de dados são caras e servem apenas para problemas específicos;
- **Avaliação do cliente**: identifica as necessidades dos clientes que não são completamente atendidas pelos produtos atuais.

Para o problema em análise é necessário contextualizar bem a oportunidade com âmbito do problema de forma a identificar claramente as suas forças e fraquezas, bem como oportunidades e ameaças. Para tal temos de conhecer bem o segmento de mercado em que se insere, e de seguida fazer uma análise do que é que a concorrência oferece, e o que não oferece.

O segmento de mercado considerado, corresponde a profissionais das áreas de "Business Inteligence", que necessitem de realizar processos de ETL de dados, e/ou processos exploratórios de dados com recurso a técnicas de "*DataMining*" ou outras técnicas baseadas em inteligência artificial. Estudadas as necessidades dos clientes, considera-se viável a solução proposta para a oportunidade identificada na seção anterior.

3.2.3 Geração de Ideia

Como solução para o problema encontrado, surge a ideia de criação de uma aplicação (ferramenta) com uma interface de fácil utilização, que permita a deteção de PQD por utilizadores que não sejam especialistas em técnicas de deteção de PQD.

3.2.4 Seleção da ideia

O objetivo é selecionar ideias que permitam obter o maior valor de negócio. Ou seja, de entre as novas ideias, é necessário saber quais as que possuem maior relevância para o projeto (Koen et al., 2002). A ideia principal apresentada no ponto anterior, visa resolver um problema de deteção de PQD, numa forma simplificada.

De todas as ideias subadjacentes a esta ideia principal serão selecionadas as que melhor se enquadrem com o objetivo proposto de ter uma aplicação de fácil utilização e extensível para futuros processos de deteção de PQD, que possam eventualmente não ser comtemplados durante a realização deste projeto.

3.2.5 Definição de conceito

Nesta secção é apresentada a informação qualitativa e quantitativa utilizada para convencer os clientes (Koen et al., 2002). A definição de conceito pode ser realizada, através das seguintes metodologias (Koen et al. 2002):

- Abordagens que corresponde ao tempo despendido em definir cuidadosamente os objetivos do projeto e os resultados esperados;
- Definição de critérios descrevendo o que será um projeto atrativo
- Avaliação de inovações com elevado potencial;
- Utilização rigorosa do processo para projetos de alto risco;
- Perceber o limite da capacidade de performance da tecnologia;
- Participação do cliente em testes reais dos produtos;
- Estabelecer parcerias;
- Perseguir abordagens científicas alternativas;

Para a solução preconizada, é necessário tornar evidente as vantagens das funcionalidades principais, identificando claramente a sua correspondência com as necessidades dos clientes, bem como definir critérios de interesse na análise de valor.

3.3 Proposta de valor

A proposta de valor corresponde ao valor criado pelo produto a desenvolver para solucionar o problema identificado, que corresponde à oportunidade de negócio identificada. Esta proposta descreve os benefícios que são entregues aos clientes. Assim sendo, as organizações apresentam os motivos que as diferenciam da concorrência e a razão pela qual os clientes devem dar preferência (Osterwalder & Pigneur, 2003) .

Para a oportunidade identificada, a solução passa por desenvolver uma aplicação que realize operações de deteção de PQD. O utilizador escolherá um conjunto de operações, podendo escolher a sequência de execução das mesmas. A aplicação terá uma interface de fácil utilização que permitira facilmente escolher de entre várias, as operações de deteção PQD e qual a sequência de execução pretendida. Para além da facilidade de utilização, é necessário garantir que as tecnologias utilizadas para implementar a solução permitem que a solução implementada garanta a correta deteção PQD, de forma a se poder definir as melhores estratégias de correção dos PQD detetados.

3.4 Modelo de negócio CANVAS

Nesta secção pretende-se mostrar, de forma simples, uma visualização do modelo de negócio. Apresentamos o modelo CANVAS, através do qual se demonstram várias dimensões de base do modelo de negócio. Deve ser preenchida a seguinte informação no modelo de CANVAS:

- Parceiros-chave- Parcerias que contribuem para a concretização do negócio.
- Atividade-Chave - Atividades que a empresa realiza para entregar valor ao cliente.
- Recursos-Chave: corresponde a ativos fundamentais para que o negócio funcione.
- Proposta de valor- “valor” que o bem entrega ao cliente.
- Relações com o Cliente- Estratégias para conseguir manter clientes.
- Canais- Formas de comunicação e distribuição pelas quais a organização entrega valor ao cliente.
- Segmento de clientes - Clientes que estão dispostos a pagar pelo bem.
- Estrutura de custos - custos do desenvolvimento da solução.
- Fontes de receita- Determina as formas como o cliente pagará pelos bens.

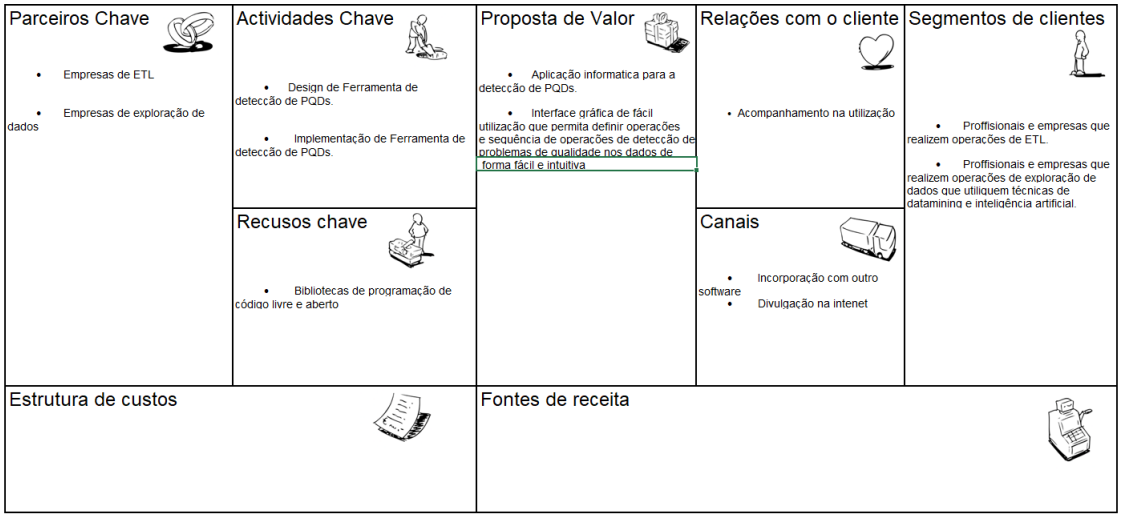


Figura 5: Modelo de negócio CANVAS

A ferramenta a desenvolver será de uso livre, pelo que não serão consideradas fontes de receita

3.5 QFD

QFD ("Quality Function Deployment") é uma técnica adequada para inovação de produtos, orientando as organizações no processo de criação de novos produtos, com um design estruturado direcionado aos requisitos do cliente, através de especificações de engenharia. As especificações de engenharia de QFD são as bases do planeamento do processo produtivo. O QFD permite traduzir os requisitos do cliente em cada fase de produção do produto (Govers, 1996).

O QFD tem sido aplicado em áreas como o marketing, processos de tomada de decisão, gestão de trabalho em equipa, Gestão de métodos e tempos, planeamento, Design e gestão de custos. "Não existe uma fronteira bem definida para os potenciais campos de aplicação do QFD" (Chan & Wu, 2002). O processo de desenvolvimento do QFD envolve um determinado conjunto de matrizes. Num desenvolvimento completo de QFD temos 4 matrizes (Erdil & Arani, 2019):

- Casa da qualidade- "house of quality" (HOQ): - HOQ usado para se referir ao QFD, regista os requisitos de clientes e transforma-os em requisitos técnicos;
- Especificações das partes: - Transforma requisitos técnicos em especificações técnicas das partes;
- Requisitos de processos: - Transforma os requisitos técnicos das partes em especificações técnicas dos processos;
- Requisitos dos produtos: - finalmente obtém-se os requisitos de qualidade dos produtos.

A HOQ é o elemento chave de todo o QFD, pois traduz a perspetiva do cliente em todas a fases e criação do produto. Segue-se a descrição das fases de obtenção de HOQ (Erdil & Arani, 2019):

i. Identificar necessidades dos clientes: - as necessidades dos clientes podem ser obtidas através de inquéritos, entrevista, ou grupos de influência. Toma-se em consideração o grau de importância de cada necessidade.
ii. Identificar requisitos técnicos: - Os requisitos técnicos são estabelecidos com base nas necessidades dos clientes. Estes requisitos técnicos são transformados em requisitos de design, representados por atributos mensuráveis.

Na Figura 6 é mostrada de forma resumida todas as relações e informações evolvidas num processo de QFD.

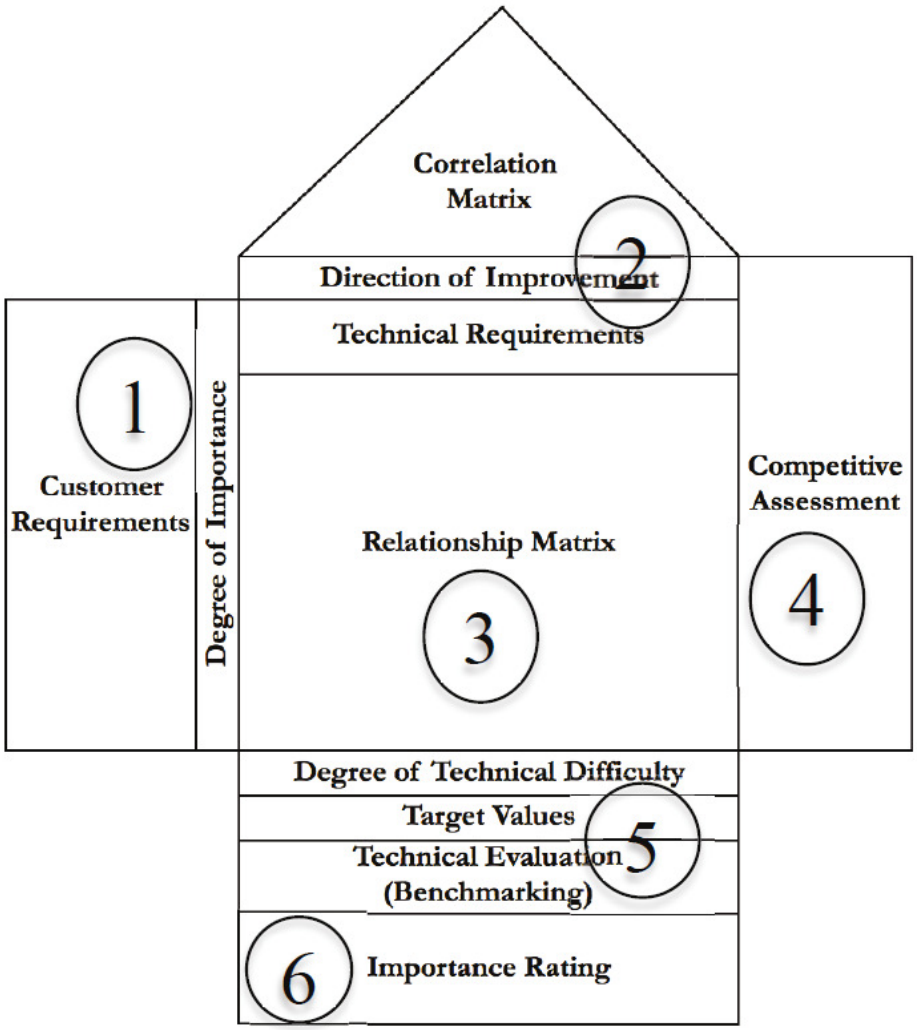


Figura 6: Casa da qualidade(Erdil & Arani, 2019)

O telhado da casa da qualidade, é utilizado para apresentar a correlação entre os requisitos técnicos. No ponto 2 da figura são apresentados os tipos de correlações (símbolos) usadas no telhado da casa de qualidade.

iii. Determinação das relações entre os requisitos dos clientes e os requisitos técnicos. Corresponde ao ponto 3 da Figura 6.
iv. Analise competitiva. É feita a comparação competitiva em relação aos requisitos dos clientes.
v. No ponto 5, apresentam-se as metas a atingir. Usa-se a análise competitiva e os graus de importância para os clientes para se definir metas baseadas nos requisitos técnicos.
vi. Cálculo de classificação de importância. São determinadas quantitativamente várias características, para se identificar quais os requisitos técnicos mais importantes.

Para preencher a matriz de relação entre os requisitos do cliente e os requisitos técnicos (Funcionais), utilizamos símbolos apresentados na Figura 7.

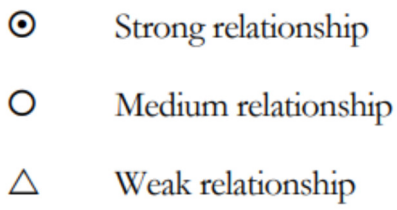


Figura 7: Símbolos da matriz relação

A matriz de correlação, também conhecida por matriz de conflitos entre os requisitos técnicos, é preenchida com os símbolos "+" ou "-" para correlação positiva ou negativa respetivamente.

Depois de preenchidas as matrizes interesse determina-se quais as características técnicas que são mais importantes para o desenvolvimento da ferramenta de deteção de PQD.

Na aplicação de deteção de PQD a desenvolver foram identificados os seguintes requisitos do cliente:

- Interface gráfica de fácil uso;
- Complexidade de especificação das operações;
- Rapidez da capacidade de resposta;
- Âmbito genérico: - aqui interessa incluir o tipo de PQD e as áreas de intervenção da deteção.
- Ferramentas Open-source: - é objetivo a construção de uma ferramenta open-source, pelo que este requisito é importante.

A nível das características técnicas do produto foram identificados os seguintes requisitos:

- Sequência das operações de deteção: - Este requisito refere-se à facilidade da ferramenta em permitir escolher as sequências desejada de PQD. A sequencia tem importância no desempenho global do *workflow*;
- Processamento paralelo: - Este requisito está relacionado com a possibilidade de escalabilidade no caso de bases de dados de grandes dimensões, e com a possibilidade de aumentar a performance do produto a criar;
- Taxa de erros encontrados: - É uma medida de eficiência da aplicação;
- Tempo de deteção: - É medida de desempenho do produto;
- Desempenho do *workflow*: - é uma medida de desempenho global de todas as operações de PQD escolhidas, e resulta da correlação entre outras caraterísticas do produto;
- Existência de "Drag and Drop": - É um requisito que facilita muito a utilização da ferramenta;

- Facilidade de ligar operações: - Refere-se à facilidade de graficamente se ligar as OD na sequência do *workflow*;
- Facilidade de uso da interface: - é uma medida de desempenho global da interface gráfica do utilizador;

Para comparação com a ferramenta a desenvolver, foram escolhidas as ferramentas: *OpenRefine*; *Data wrangler*; *Scare* e *Nadeef*, por serem as ferramentas que possuem características comparáveis com a ferramenta pretendida.

Apresenta-se na Figura 8 a HOQ, onde podemos avaliar quais as características mais importantes para os clientes, no desenvolvimento do produto.

A figura mostra a matriz de relação e a matriz de conflitos preenchidas. Através da análise da Figura 8 pode-se verificar que as características mais importantes para o cliente são; a facilidade de uso da interface gráfica, a generalização da ferramenta, e o custo da mesma.

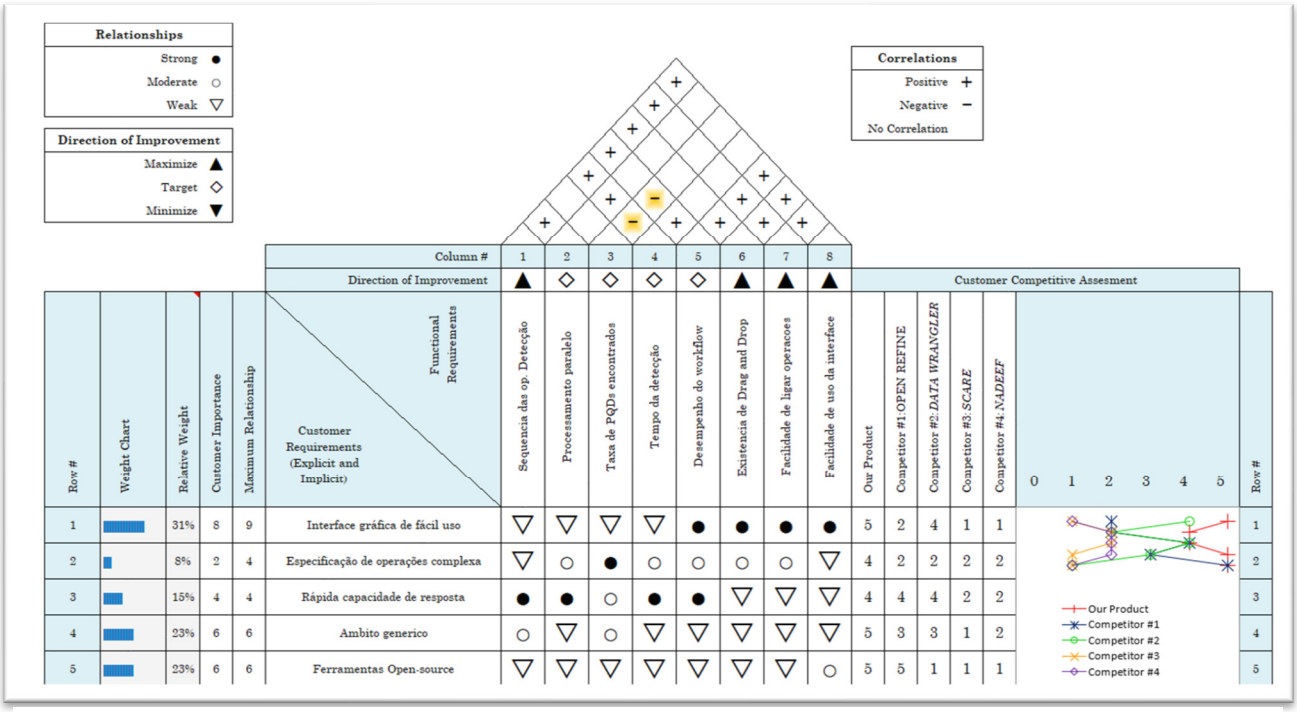


Figura 8: HOQ com avaliação requisitos dos clientes

Esta figura revela uma oportunidade por explorar, uma vez que comparando com outras ferramentas, o produto a desenvolver terá características muito competitivas para o cliente. Na matriz de correlação não foram detetados grandes conflitos entre as características técnicas. Apenas se identificam a correlação negativa da taxa de deteção de PQD com o tempo de deteção e o desempenho global do *workflow*. No entanto como é expectável, esta correlação é mais afetada quando existirem bases de dados com mais PQD.

Com a Figura 9 pretende-se avaliar os requisitos técnicos. Apesar do requisito desempenho do *workflow*, ser o requisito com maior valor de peso relativo (19%), os requisitos técnicos da interface gráfica tem 53% de peso relativo (facilidade de uso da interface + facilidade de ligar operações + "drag na drop"+ facilidade de estabelecer sequências de operações). Este valor evidencia o esforço que se tem de desenvolver para conseguir responder a estes requisitos. A análise competitiva dos requisitos técnicos, torna evidente a oportunidade que representa desenvolver uma ferramenta de deteção de PQD com as caraterísticas técnicas mais destacadas a terem uma valorização diferenciadora no produto a desenvolver.

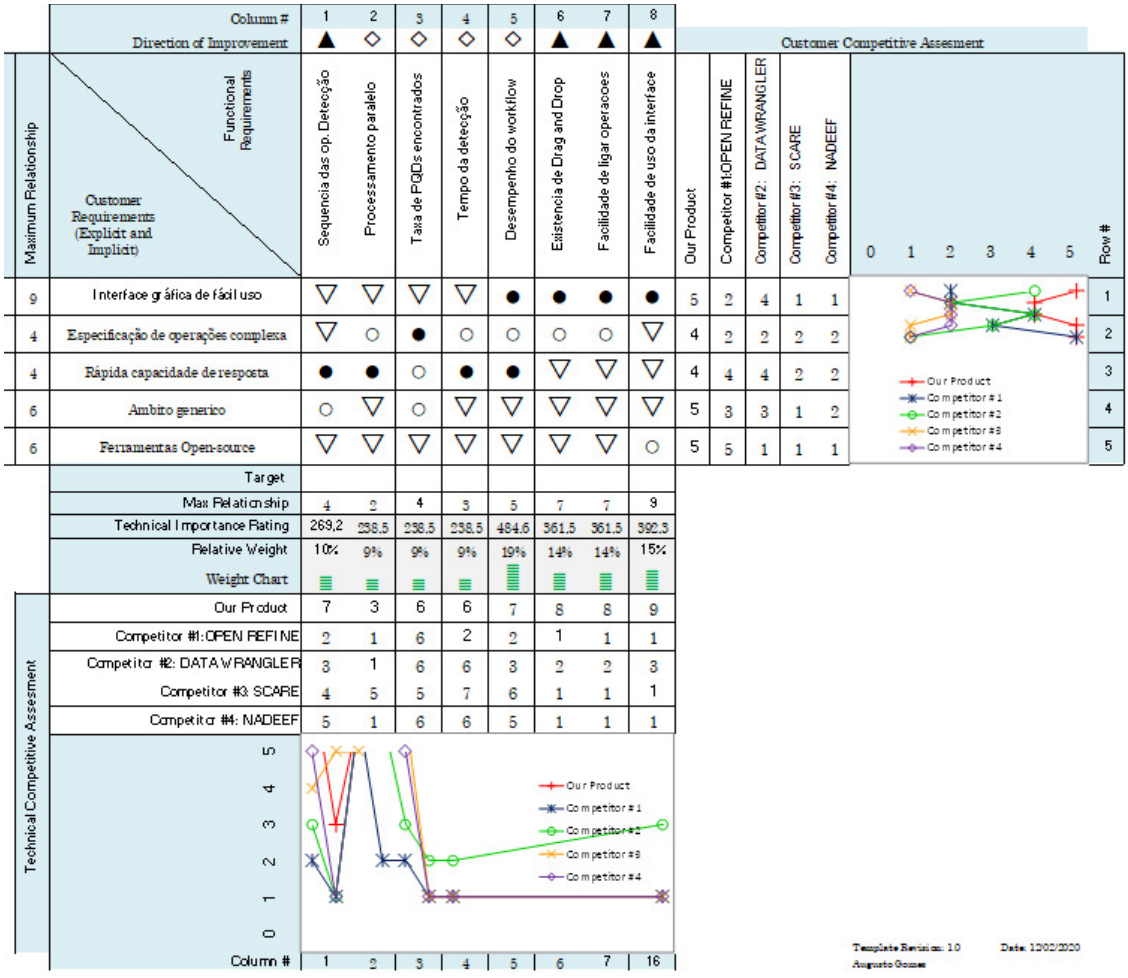


Figura 9: HOQ com requisitos técnicos

4 Análise e Desenho

Este capítulo começa com a análise de requisitos, segundo a metodologia FURPS+, seguindo-se o desenho da solução de acordo com os requisitos definidos. Os requisitos dividem-se em requisitos funcionais e requisitos não funcionais.

O desenho começa por uma arquitetura de alto nível e vai-se aumentando o detalhe do desenho no decorrer do capítulo. No nível de detalhe intermédio optou-se por recorrer a BPMN ("*Business Process Model and Notation* " - Notação de Modelação de Processo de negócio) que demonstra melhor as várias ações integradas da camada de apresentação com a camada de modelo. No caso da camada de modelo, procede-se ao desenho dos dois subprocessos principais.

Prossegue-se com o desenho detalhado das camadas de apresentação e modelo. Na camada de apresentação apresenta-se um fluxograma das ações do utilizador e uma *mockup* da interface gráfica pretendida.

No Modelo são definidos com detalhe os subprocessos identificados na camada de modelo com recurso a BPMN. Utilizam-se padrões de design de software adequados para as fases de criação de objetos e de execução do algoritmo de deteção de PQD (Problemas de Qualidade de Dados).

4.1 Definição de requisitos

A análise de requisitos é essencial para um design adequado da solução pretendida. A análise de requisitos foi efetuada baseada na metodologia FURPS+ (Center of Excellence for Professional Development, 2014), permitindo avaliar os requisitos mais importantes de acordo com as necessidades dos clientes. A metodologia FURPS+ permite classificar os requisitos. A Figura 10 ilustra a classificação de requisitos FURPS+.

Esta secção está dividida em requisitos funcionais e não funcionais. Os requisitos funcionais são os que definem as funcionalidades que se pretendem obter com a solução. Os requisitos não funcionais são aqueles que, apesar de não definirem funcionalidades, definem restrições a ter em conta no design e implementação da solução.

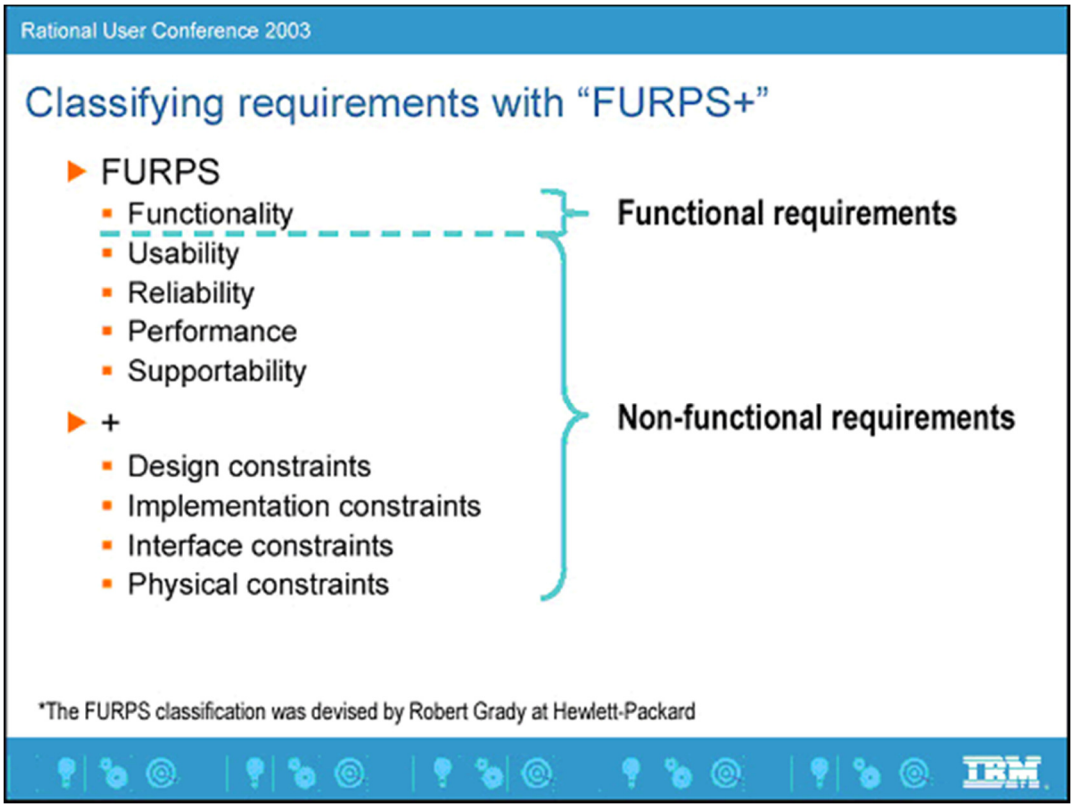


Figura 10: Classificação de requisitos (IBM rational library, 2008).

4.1.1 Requisitos funcionais

Foram identificados os seguintes requisitos funcionais:

- Detetar os seguintes PQD: Valor em falta; Violação de domínio; e, Violação de sintaxe. Estes PQD resultam da taxionomia que consta em (Oliveira et al., 2005)

- Permitir definir *workflow* de PQD a detetar, com a sequência de execução da deteção de PQD a ser definida pelo utilizador;

- Permitir funcionalidade de "*drag and drop*" de componentes que representam as operações de deteção de PQD a realizar;

- Permitir abrir janelas de *pop-up* para configurar os parâmetros dos diferentes componentes (representativos dos PQD a detetar e das origens/fontes de dados);

Apresenta-se, na Figura 11, as ações principais do utilizador, na forma de casos de uso.

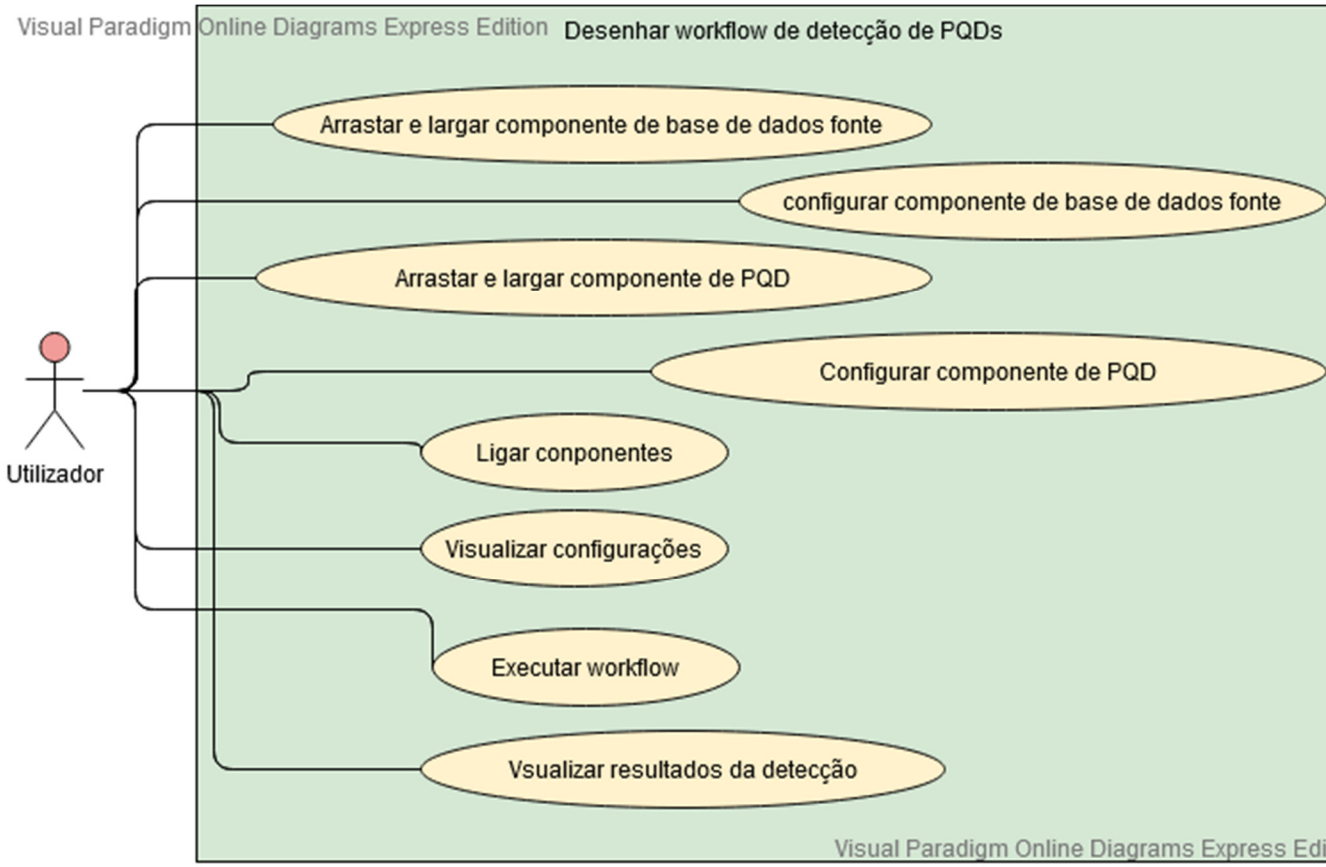


Figura 11: Ações do utilizador no sistema.

A solução deverá permitir ao utilizador escolher um componente, arrastá-lo e largá-lo no local pretendido do painel de *workflow* de operações de deteção de PQD. O utilizador pretende especificar a sequência de operações de PQD a detetar. Esta sequência é importante para uma correta identificação do tipo de PQD a detetar. Com este objetivo, os componentes de operações de PQD a detetar tem de ser ligados de forma visual na interface gráfica. Os componentes colocados no *workflow* têm de ser parametrizados com as propriedades específicas de cada operação de deteção de PQD. O sistema deverá permitir visualizar de forma gráfica os parâmetros definidos em todo o *workflow* desenhado. Depois de configurado, o *workflow* será submetido para execução, sendo executadas as operações de deteção de PQD especificadas, seguindo a ordem/sequência especificada pelo utilizador. Finalmente, o sistema deverá permitir visualizar o resultado da deteção de PQD.

4.1.2 Requisitos não funcionais

Os aspetos relacionados com a interface gráfica, como a acessibilidade e estética são muito importantes para garantir uma boa usabilidade. A solução a desenvolver será de alta disponibilidade e com elevada precisão para garantir uma boa confiabilidade. A taxa de deteção de PQD é uma das medidas essenciais de desempenho da solução. A solução terá de conter

outras características. Deverá ser testável, adaptável, configurável e escalável para permitir uma boa suportabilidade.

Foram definidos os seguintes requisitos não funcionais de acordo com a metodologia FURPS+:

- Usabilidade: Interface gráfica de fácil utilização.

- Confiabilidade: - A frequência de falhas deve ser muito reduzida. Se possível, as falhas de deteção devem ser inexistentes para os PQD suportados. Na ocorrência de uma exceção, a solução deve permitir a possibilidade de recuperação do seu normal funcionamento. A deteção de PQD deve ser executada com exatidão. Tendo, o utilizador, escolhido uma sequência de operações de PQD, a solução deve classificar devidamente os PQD detetados.

- Desempenho: - pretende-se uma solução com taxa de deteção de PQD, para tabelas com 20.000 registos, superior a 10 PQD por segundo; tempo de resposta para tabelas com 100 000 registos inferior a 5 segundos; consumo de memória da solução inferior a 2GB de RAM ("*Random Access Memory*"); Utilização do CPU abaixo dos 70%.

- Suportabilidade: - A solução tem de ser testável; adaptável para novos PQD e novas fontes de dados; de fácil manutenção; configurável; de fácil instalação e escalável.

4.2 Arquitetura

Nesta secção é apresentada a arquitetura. Começa num nível designado de alto nível, que apresenta apenas os componentes principais da solução. Com o decurso da secção, vai-se detalhando mais a arquitetura.

4.2.1 Desenho de padrão MVC de alto nível

Apresenta-se na Figura 12, o diagrama de arquitetura geral da aplicação. A arquitetura de alto nível da solução de deteção de PQD baseia-se no padrão MVC (Leff & Rayfield, 2001).

A camada de apresentação terá um menu com todas as operações que o utilizador pode realizar, e um conjunto de componentes visuais, que representam cada operação de deteção pretendida e os conectores entre componentes. O componente de *workflow* UI ("*User interface*"), corresponde à parte da interface gráfica que representa o workflow propriamente dito, onde são colocadas, com recurso a funcionalidades de "*drag and drop*", os componentes de operações de deteção dos PQD pretendidos. As operações terão uma sequência, através da ligação de componentes. O *workflow* inicia com a fontes de dados (origem). Existem componentes para cada tipo de operação de deteção de PQD suportada. No seguimento, escolhem-se os componentes relativos às operações de deteção pela ordem pretendida pelo

utilizador. O *workflow* é finalizado com um componente que representa os dados limpos, a gravar na base de dados do sistema, depois de terminada a deteção de PQD.

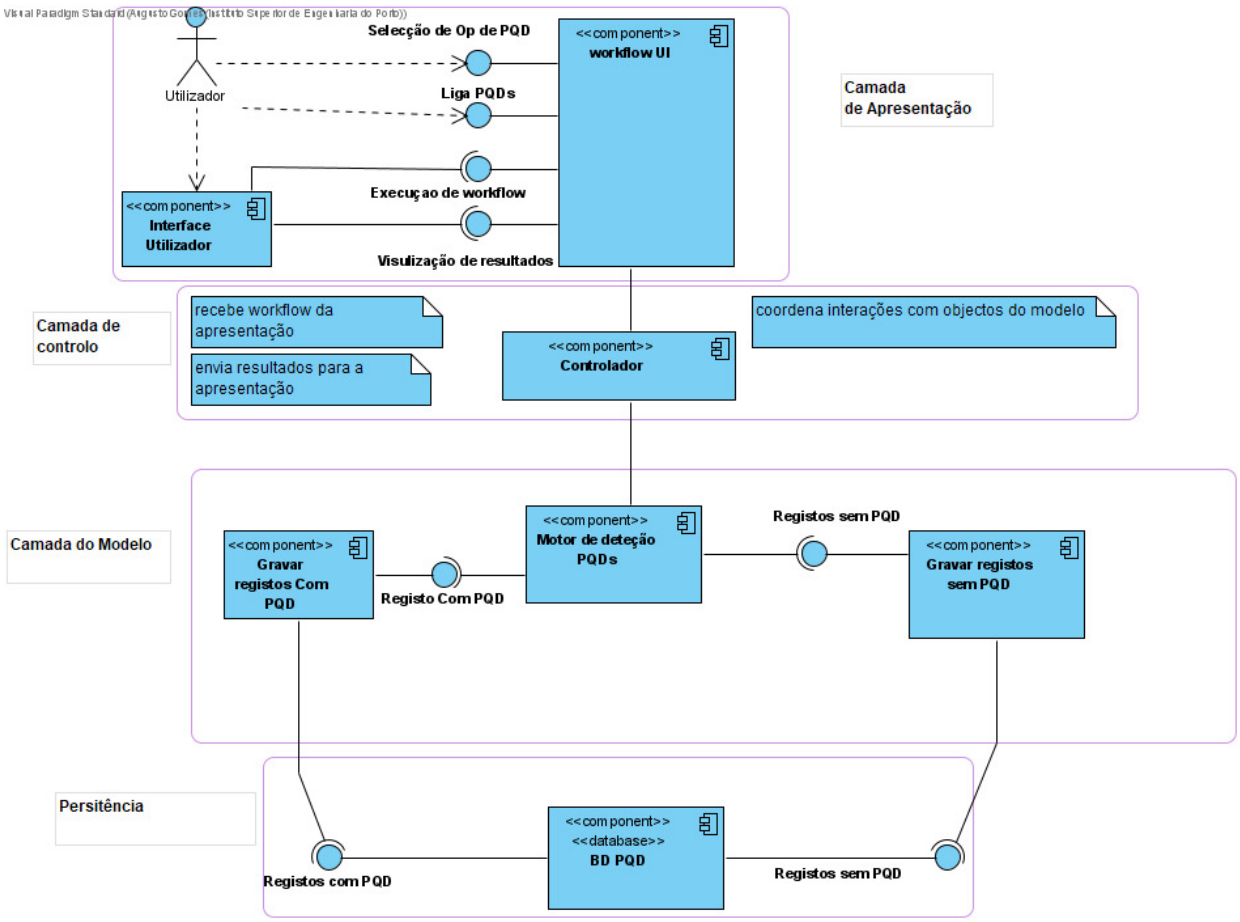


Figura 12: Arquitetura de alto nível

A camada de controlo coordena as interações entre a camada de apresentação e a camada do modelo. A camada de controlo recebe os "*inputs*" da camada de apresentação. A camada de controlo realiza as interações com os objetos da camada do modelo. A camada de modelo tem a lógica de negócio com todos os algoritmos de deteção de PQD relativos a cada operação de deteção suportada. O modelo contém uma camada de acesso a dados que contém os métodos de aceder à camada de persistência. A camada de persistência tem uma tabela de PQD detetados e uma tabela para registar os dados limpos (sem PQD detetados).

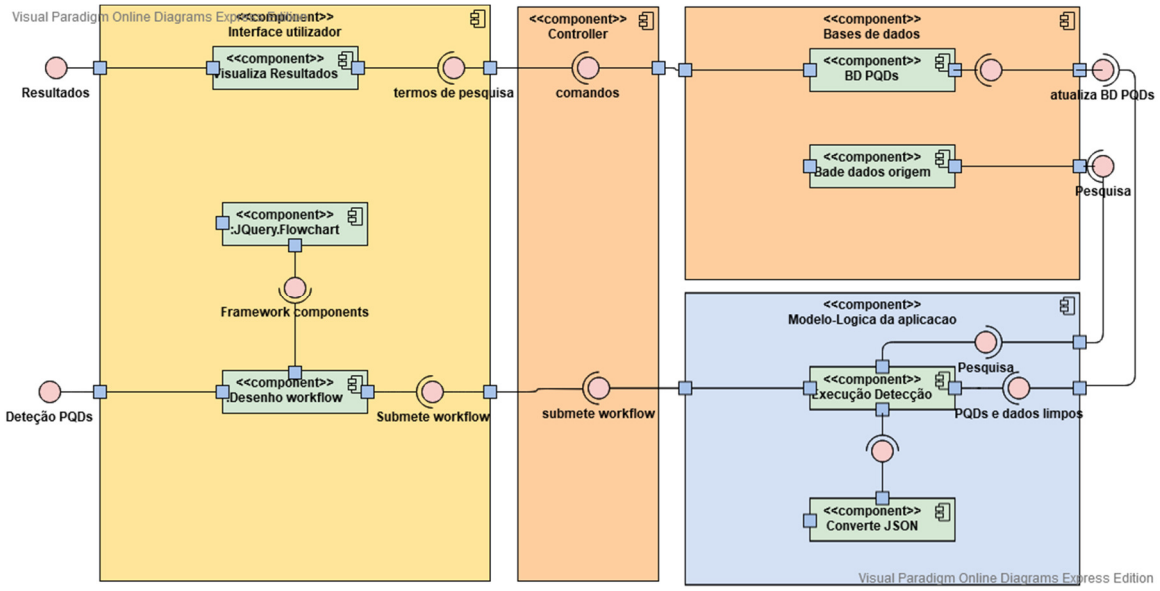


Figura 13: Diagrama geral de componentes

O diagrama da Figura 13 mostra a dependência da interface do utilizador com a *framework JQuery.Flowchart* que permite o desenho gráfico de componentes que representam uma operação de deteção de PQD na interface do utilizador. Neste diagrama, na camada do modelo pode-se ver os dois principais processos que ocorrem: O processo que corresponde à conversão dos dados da camada de interface do utilizador para objetos na camada de modelo e o processo de deteção de PQD. Na camada de persistência, como ilustra a Figura 13, existe uma base de dados de sistema onde se armazenam os PQD detetados e os dados limpos (i.e., registos sem PQD).

A Figura 14 corresponde a um diagrama de sequência de alto nível que identifica: ações principais do utilizador na interface do utilizador; a submissão dos dados da interface do utilizador depois de estar construído o *workflow* de PQD a detetar, para o modelo, através do controlador. As passagens de dados entre processos do modelo são coordenadas pelo controlador. No modelo os dados são submetidos a um processo de conversão para objetos Java, com os quais se executa o processo de deteção de PQD.

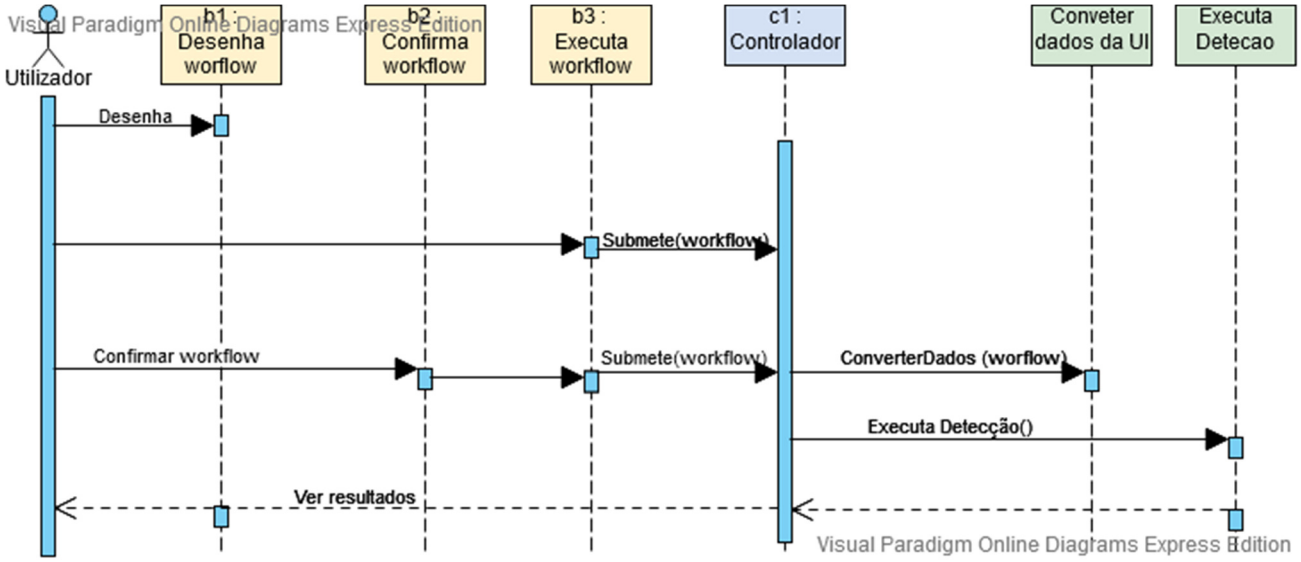


Figura 14: Ações principais do utilizador.

4.2.2 Desenho de processos de negócio

Para se compreender melhor as diferentes fases de deteção de PQD nas três camadas do padrão MVC e a sua interligação, procede-se ao design de processos de negócio com recurso a BPMN. Na Figura 15 apresentam-se os processos de negócio do sistema. Na camada de apresentação estão apresentadas as ações do utilizador, sendo as principais ações, as que correspondentes ao processo de geração do *workflow* de PQD. Finalizado o *workflow*, o mesmo é submetido através da camada de controlo para a camada de modelo. Na camada de modelo apresentada na Figura 15, sob a forma de subprocessos do BPMN, o processo de conversão de dados e o processo de execução da deteção de PQD. Primeiro ocorre o processo de conversão dos dados que converte os dados provenientes da interface do utilizador, para dados utilizáveis na camada de modelo. Depois ocorre o processo de deteção de PQD propriamente dito. É este processo que contém toda a lógica de deteção de PQD.

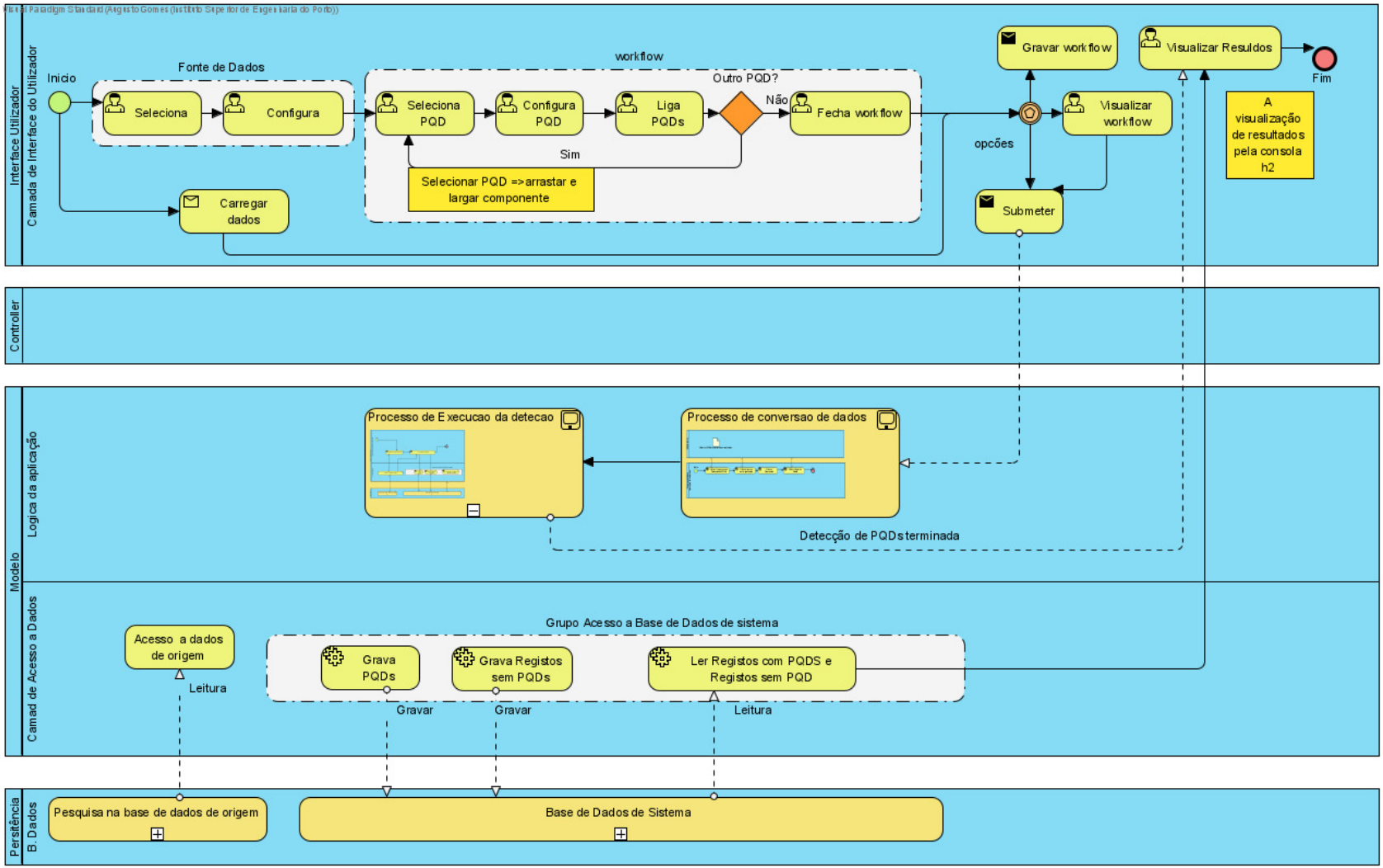


Figura 15: Diagrama geral de processos.

4.2.2.1 Processo de conversão de dados.

A Figura 16 apresenta o desenho do processo de conversão de dados.

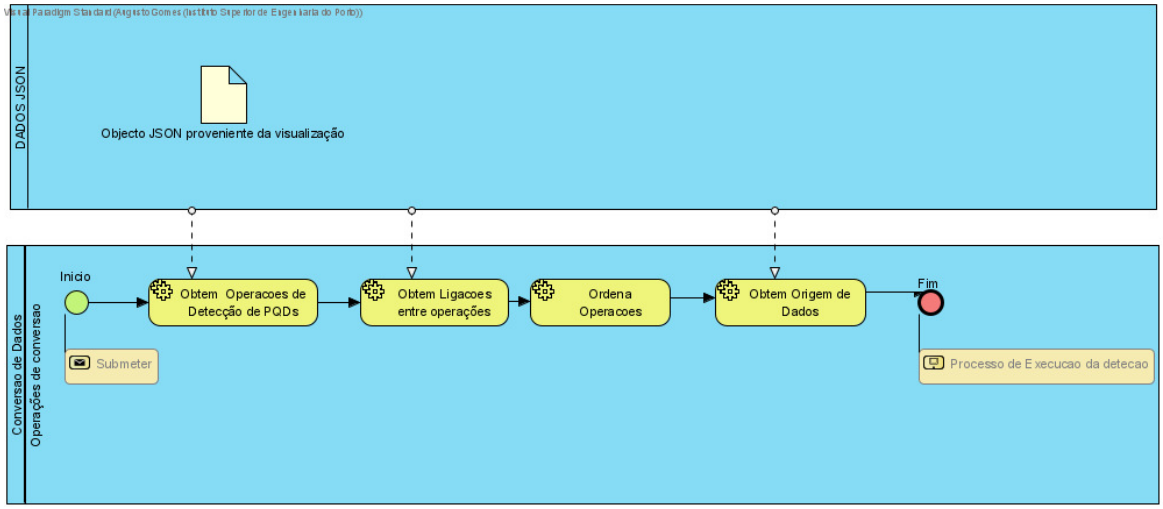


Figura 16: Processo de conversão de dados.

Os dados provenientes da interface do utilizador, estão no formato adequado para a conversão na camada do modelo. Estes dados têm de ser convertidos em objetos Java adequados à lógica da aplicação. Contêm dados de operações de deteção de PQD e informação da sequência de ligação/execução entre as operações. Sendo que existem ainda dados sobre a fonte de dados (origem de dados) para a qual se quer detetar PQD. Na camada de modelo, para se poder executar o processo de deteção de PQD é necessário converter os dados no formato JSON em listas de objetos Java. Na Figura 16 pode-se identificar as várias atividade do processo de conversão de dados: Obtenção de operações de deteção de PQD, que extrai de um ficheiro JSON a lista de operações do *workflow*; Obtenção de ligações, onde é extraída uma lista das ligações entre os PQD do *workflow*; Segue-se a ordenação das operações de deteção dos PQD; Existe ainda a atividade de extração de origem de dados, que contém só parâmetros da base de dados origem/fonte.

4.2.2.2 Processo de execução da deteção

Como mostra a Figura 15, no fim do subprocesso de conversão de dados, os dados são passados para o subprocesso de execução da deteção. É neste subprocesso que reside a lógica de deteção dos PQD. Tendo em conta os limites de recursos dos dispositivos onde a aplicação será executada, divide-se o conjunto de dados em *batchs*. Esta atividade corresponde à primeira atividade do subprocesso de execução da deteção de PQD. A atividade final deste subprocesso, grava registos sem PQD, corresponde à gravação dos dados limpos na base de dados.

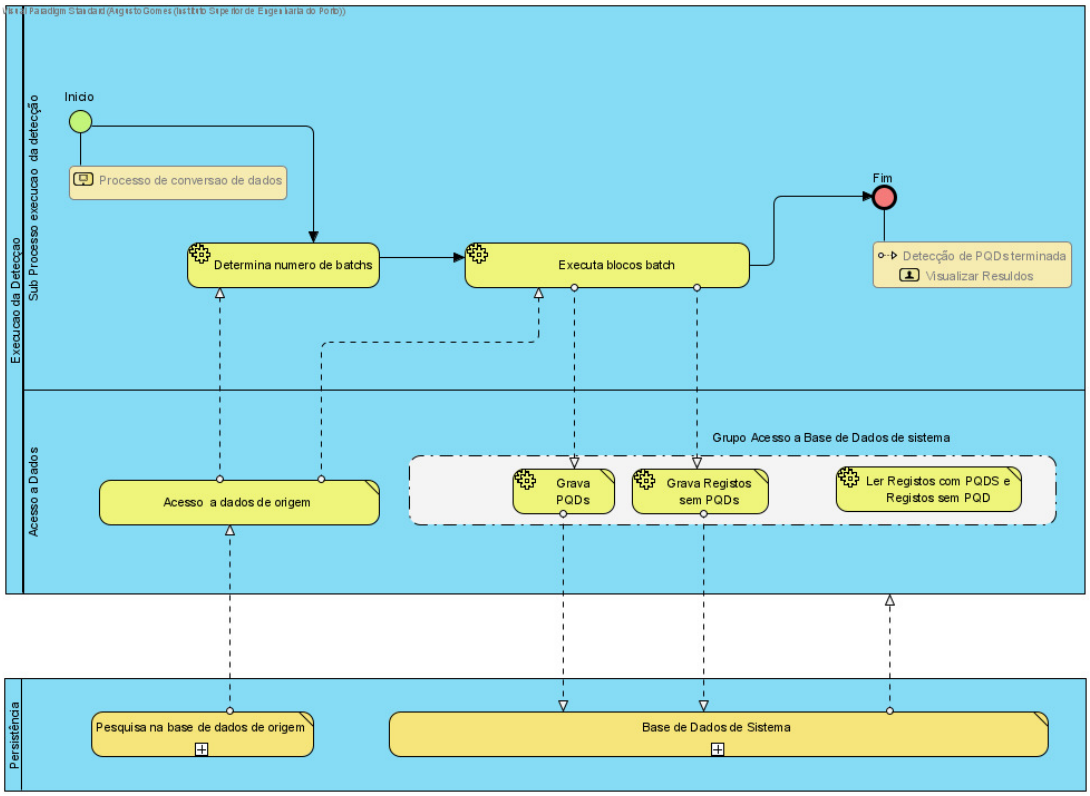


Figura 17: Processo de execução da deteção.

4.3 Interface gráfica

Nesta secção apresenta-se o desenho da camada de apresentação, ou seja, da interface gráfica com o utilizador.

4.3.1 Ações do utilizador

No fluxograma da Figura 18 pode-se visualizar a sequência de ações que o utilizador terá de executar para se poder proceder à deteção de PQD.

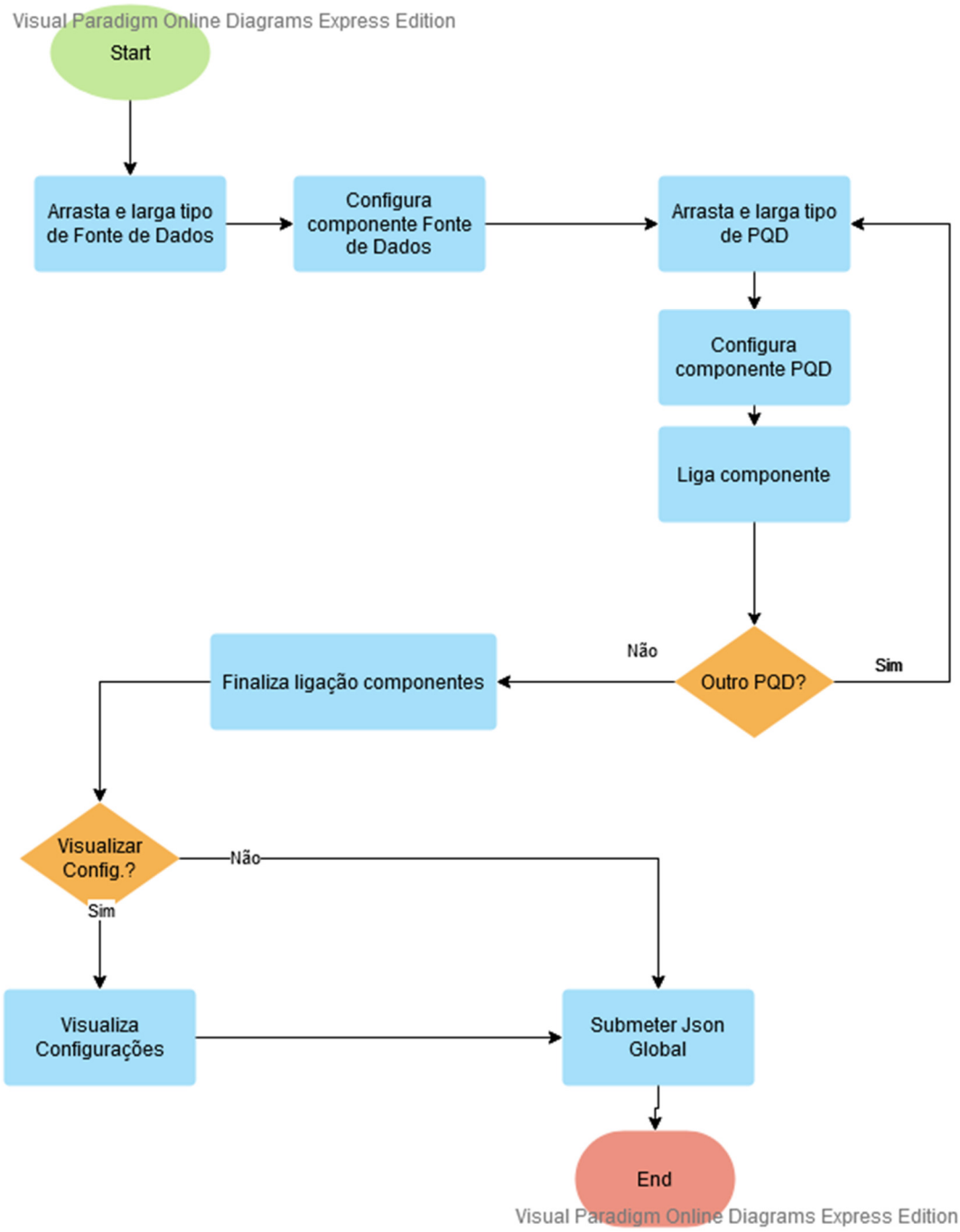


Figura 18: Ações necessárias para detetar PQD.

4.3.2 *Mockup* da interface gráfica

Na Figura 19 está apresentado uma *Mockup* da interface gráfica pretendida. Nesta figura pode-se visualizar um menu com as operações a executar sobre o painel de *workflow* e o botão de submeter o *workflow*. No seguimento vertical de cima para baixo, aparece outro menu que contem componentes *dragable*, com os quais se vai desenhar o *workflow*. O painel de *workflow*

está apresentado com o texto ”arrastar componentes para aqui”, correspondendo à área da interface gráfica para onde são arrastados os componentes dragable. Ao arrastar e largar um componente sobre o painel de *workflow*, é criado o componente gráfico correspondente. Os componentes no painel são colocados pela ordem pretendida pelo utilizador, iniciando com um componente de origem de dados e finalizando com o componente final de dados limpos. Os componentes têm de estar todos interligados para finalizar o desenho do *workflow*.

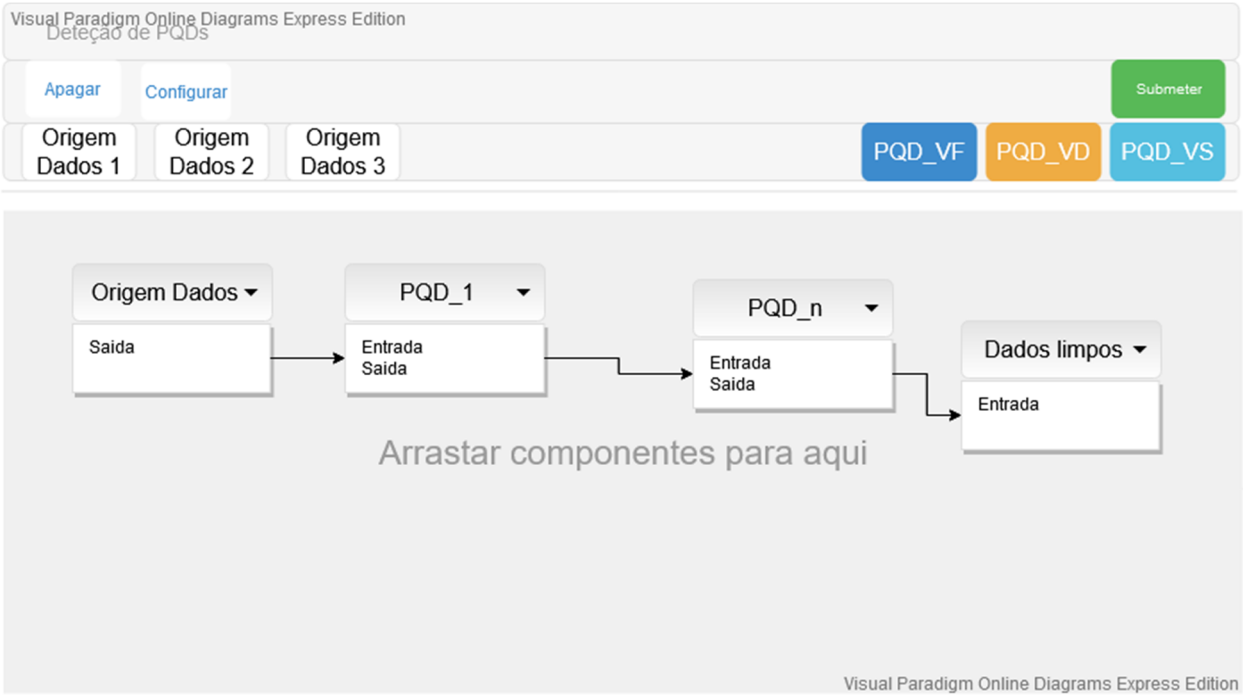


Figura 19: *Mockup* da Interface gráfica.

4.4 Interface origem de dados

A origem de dados pode ser originária de diversos tipos de fontes como base de dados, ficheiros, APIs disponíveis online, etc. O Padrão *Repository* fornece uma camada de abstração entre a camada de acesso a dados e a camada de modelo da aplicação, permitindo interligar vários subsistemas (Prajapati, 2019). Desta forma, consegue-se uma interface única de acesso aos dados, com possibilidade de implementações diferentes de acesso a dados, de acordo com o tipo de dados a que se pretende aceder. A Figura 20 ilustra o desenho do *Repository* de acesso a dados, baseado no padrão *Repository*, que permite aceder a vários tipos de fonte (origem) de dados. Durante o processo de carregamento de dados os dados são obtidos passando como parâmetro uma instância do tipo *OrigemDados*. Este tipo está apresentado na Figura 21, e contem diferentes atributos para diferentes tipos de origem de dados. Nas Figuras 20 e 21 está apresentado o desenho para a origem de dados em base de dados e ficheiros, mas pode ser extensível para outra origem (fonte) de dados, como por exemplo API públicas.

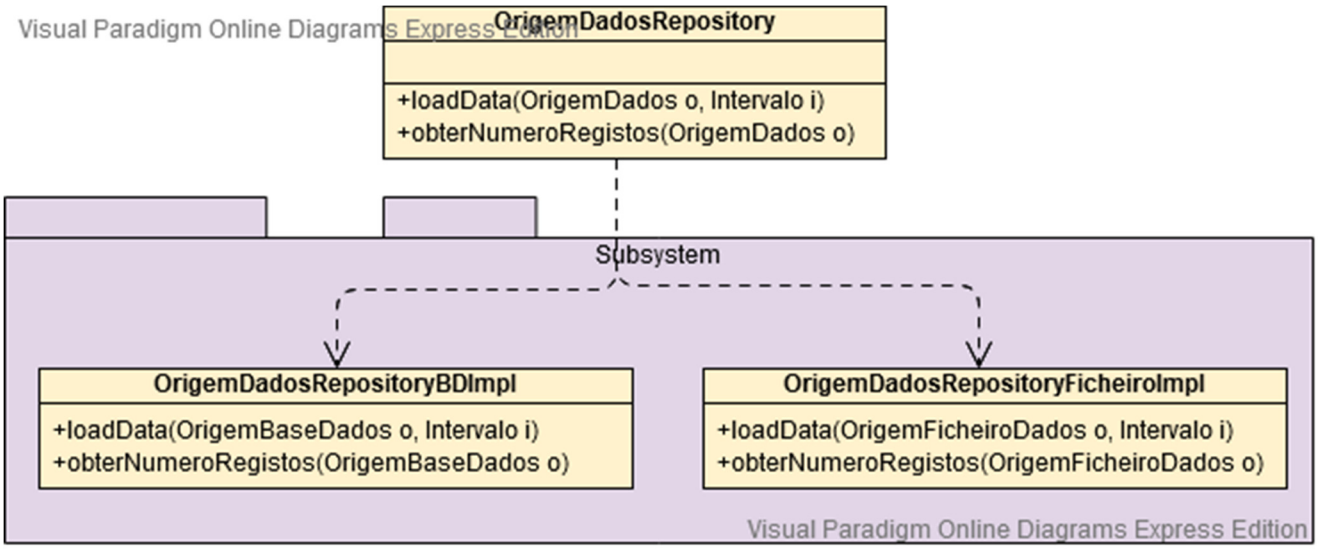


Figura 20: Padrão *Repository* para diferentes tipos de origem de dados.

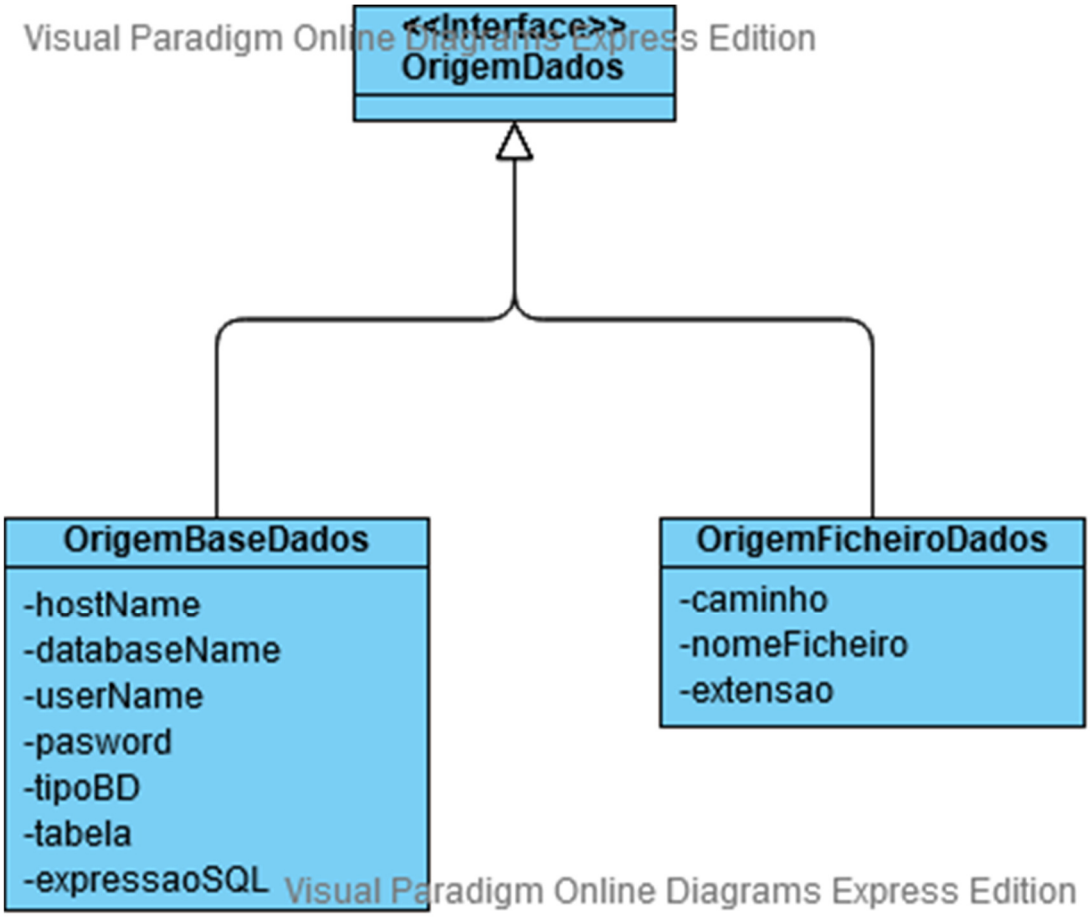


Figura 21: Interface de parâmetros da origem de dados.

4.5 O processo da conversão dos dados

No processo de conversão dos dados recorreu-se ao padrão de design de software *Factory* para criar os objetos do tipo PQD (operações de deteção de PQD) e os objetos do tipo *OrigemDados*, que representam os objetos com parâmetros necessários para acesso a fontes de dados. Apesar do processo de instanciação destes objetos, apresentar alguma complexidade, principalmente os objetos do tipo *OrigemDados*, não existe uma combinação muito diversificada dos parâmetros necessários na criação dos objetos. Nem é necessário criar muitos construtores para cada tipo específico de PQD e origem de dados. No entanto, é necessário criar vários tipos de PQD diferentes com diferentes parâmetros e vários tipos de *OrigemDados* com parâmetros também diferentes. Torna-se assim necessário uma especificação que permita abstrair do tipo de objeto instanciado. Recorre-se ao padrão *Factory*, que permite criar um objeto através da sua interface, abstraindo-se do tipo de objeto instanciado (Purdy, 2002).

4.5.1 AbstractFactory criação de operações de deteção de PQD

Na análise dos tipos de PQD a criar conclui-se ser necessário mais do que uma *factory*, pelo facto dos diferentes tipos de PQD, terem propriedades diferentes e comportamentos diferentes na deteção de PQD. O tipo PQD é uma interface que é implementada por vários tipos de PQD que o sistema suporta. No entanto, os PQD estão relacionados, pois são detetados num *workflow*, evidenciando a necessidade de criar uma biblioteca de PQD suportados pela solução. Acresce o facto da biblioteca de PQD se poder expandir no futuro com mais PQD. Com base no padrão *AbstractFactory*, define-se uma classe *PQDAbstractFactory*, que permite abstração do tipo de *factory* necessária, quando no processo de conversão de dados, surge um PQD a criar (Bulajic & Jovanovic, 2012) .

A classe *PQDAbstractFactory* (refactoring.guru, 2014) da Figura 22 é utilizada para extrair todos os PQD da lista de componentes de operações de deteção provenientes da camada de apresentação. Em tempo de execução, escolhe a *factory* adequada para instanciar o PQD pretendido. Durante o processo de conversão de dados, à medida que os elementos de dados provenientes da apresentação são percorridos, se forem encontrados dados de um componente de tipo PQD conhecida, é invocado o método estático *obtemFactory* da classe *PQDAbstractFactory*, passando como parâmetro o tipo de PQD. Baseado no tipo de PQD é instanciada a *factory* adequada. A classe *PQDAbstractFactory* contém ainda um método abstrato que é implementado por todas as factories que estendem esta classe. O tipo retornado no método *obtemFactory* é *PQDAbstractFactory*, pelo que se faz uso do polimorfismo para imediatamente após a instanciação da *factory* se invoque o método criar para instanciar o PQD correspondente. No método criar, são extraídos os parâmetros específicos do PQD a instanciar, através de um objeto do tipo *JsonNode* passado como parâmetro, cujo valor é imediatamente atribuído aos atributos da classe específica usada na instanciação do PQD.

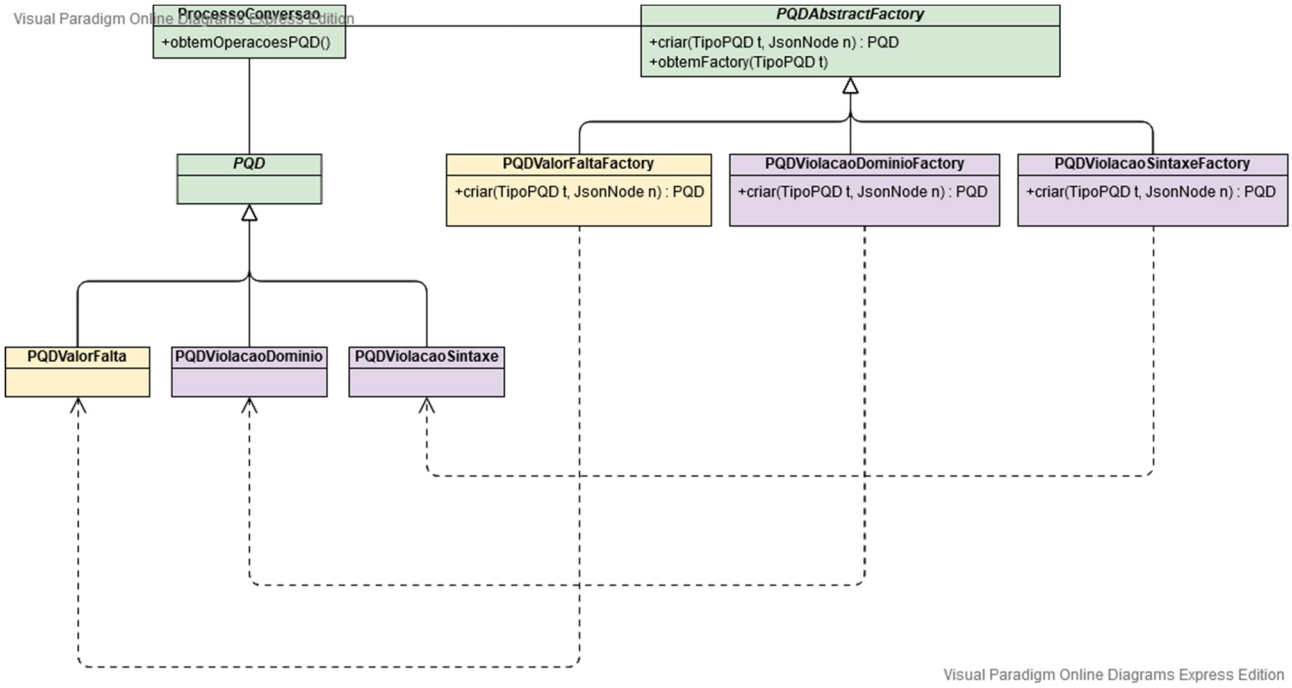


Figura 22: *AbastractFactory* dos PQD

4.5.2 *AbstractFactory* de origem de dados

O processo de criação de objetos de origem de dados pode ser complexo envolvendo parâmetros obrigatórios e parâmetros opcionais (Chaturvedi & Prabhakar, 2014). Nos casos de criação de objetos complexos, em que é possível muitas combinações e parâmetros para criar objetos, utilizando *Factory*, surgem muitos construtores, designados de construtores telescópicos. A existência de muitos construtores telescópicos torna a implementação da *factory* confusa e deselegante. Os construtores telescópicos representam as variadíssimas formas de criar objetos de determinado tipo. Neste caso complexo de criação de objetos, com muitos construtores telescópicos, o padrão *Builder* é mais adequado. A criação de objetos de origem de dados, do tipo base de dados, tem alguma complexidade, devido ao facto de existir um número considerável de parâmetros, e existirem parâmetros opcionais (expressão SQL e *batchsize*). No entanto, considera-se que a complexidade não é suficiente para se usar o padrão *builder*. Em desenvolvimentos futuros com outros tipos de origem de dados, caso a complexidade aumente, sugere-se utilizar o padrão *Builder*.

Os objetos do tipo *OrigemDados* da solução, são criados através de uma *AbstractFactory*, que utiliza a *factory* adequada em função do tipo de origem de dados selecionada pelo utilizador. O desenho do processo de criação do objeto do tipo *OrigemDados* (ver Figura 21), baseia-se no padrão *Factory* e está apresentado na Figura 24. Tal como na Secção 4.5.1, entende-se ser necessário utilizar o padrão *AbstractFactory* para criar uma família de origens/fontes de dados. As razões que levam a esta escolha são: existe a necessidade de ter

mais de uma forma e criar objetos do tipo *OrigemDados*, ou seja, é necessário mais do que um tipo de *factory*, uma vez que existe mais do que um tipo de *OrigemDados* com parâmetros diferentes. Entende-se que a solução deve ser configurada com uma família de vários tipos de dados fonte, de forma a permitir que futuramente sejam incluídas na solução mais tipos de fontes de dados, permitindo desta forma criar uma biblioteca de diversas origens/fontes de dados. A Figura 23 apresenta a aplicação do padrão *AbstractFactory* juntamente com o padrão *Factory* para instanciar diferentes origem de dados.

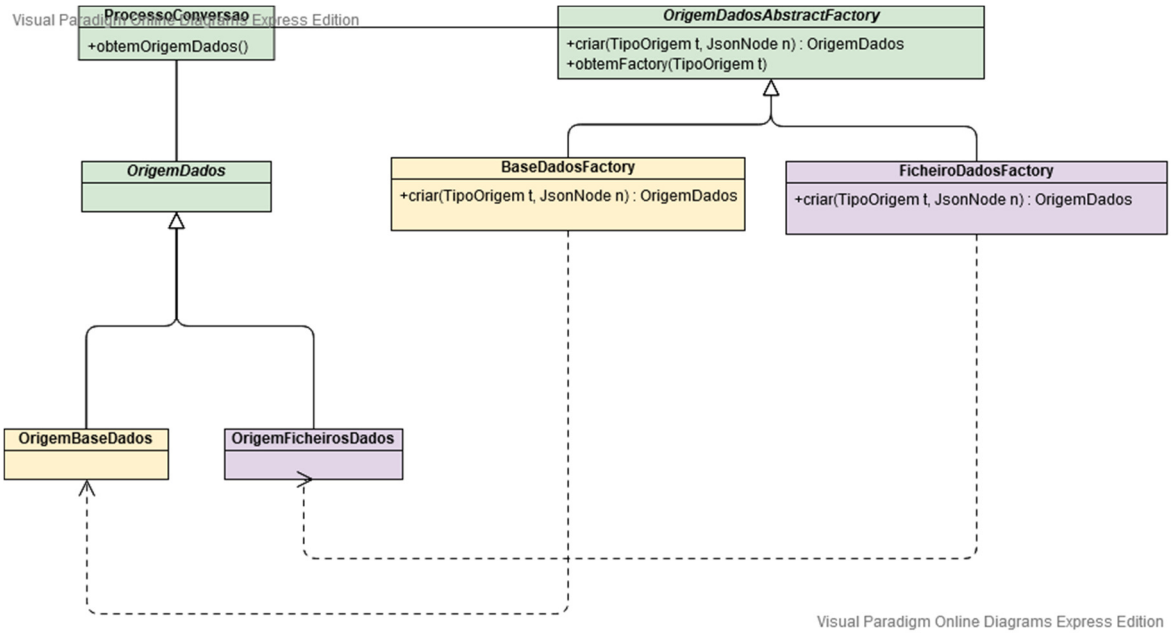


Figura 23: *AbstractFactory* de origem de dados.

No diagrama de classes da Figura 23 o atributo *tipoOrigem* da classe *OrigemDadosAbstractFactory* permite decidir qual a *factory* de criação de origem de dados que será utilizada na instanciação de um objeto do tipo *OrigemDados*. Durante o processo de conversão de dados, à medida que os elementos de dados provenientes da apresentação são percorridos, sendo encontrado um componente de origem de dados, é invocado o método estático *obtemFactory* da classe *OrigemDadosAbstractFactory*, passando como parâmetro o tipo de PQD. Baseado no tipo de PQD é instanciada a *factory* adequada. A classe *OrigemDadosAbstractFactory* contém ainda um método abstrato que é implementado por todas as *factories* que estendem esta classe. O tipo retornado no método *obtemFactory* é *OrigemDadosAbstractFactory*, pelo que se faz uso do polimorfismo para que imediatamente após a instanciação da *factory* se invoque o método criar para instanciar o objeto do tipo *OrigemDados* correspondente. No método criar, são extraídos os parâmetros específicos do objeto origem de dados a instanciar, através de um objeto do tipo *JsonNode* passado como parâmetro, cujo valor é imediatamente atribuído aos atributos da classe específica usada na instanciação do objeto tipo *OrigemDados*.

4.6 Processo de execução da deteção de PQD

Depois de convertidos todos os dados JSON resultantes da criação do *workflow*, procede-se à execução da deteção de PQD. O processo de execução da deteção é executado após ordenação das operações, pela ordem selecionada pelo utilizador. Na Figura 24 está apresentado o padrão *Strategy* que permite implementar um algoritmo específico por cada tipo de operação de deteção de PQD (Jiang & Mu, 2011). No entanto, como estas operações são guardados numa lista de tipos PQD, pode-se facilmente fazer uso do polimorfismo, para de forma iterativa se detetar os diferentes tipos de PQD. Com a utilização do padrão *Strategy*, o algoritmo torna-se extensível para detetar outros tipos de PQD, ou mesmo para implementar algoritmos diferentes que procedam à deteção dos PQD suportados.

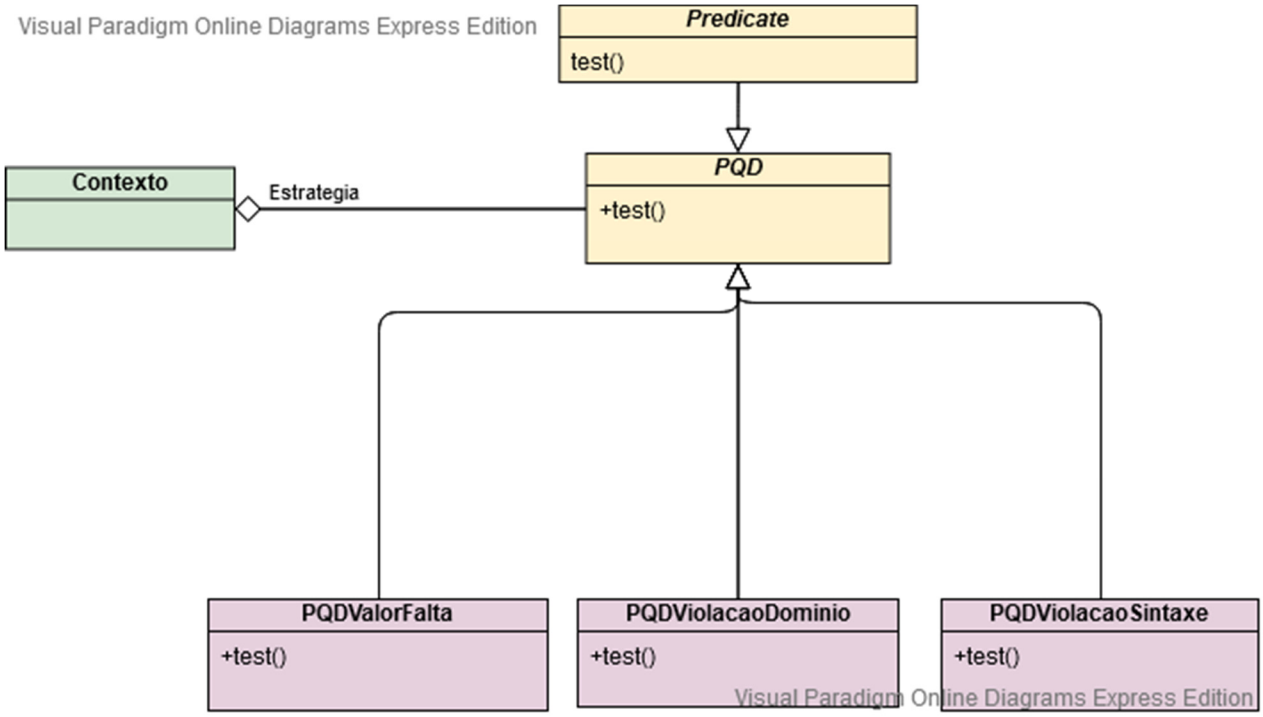


Figura 24: Padrão *Strategy* de deteção de PQD.

Pretende-se tirar partido das construções funcionais que surgiram com o Java 8. As construções funcionais permitem melhorar o encapsulamento de comportamento para posteriormente ser reutilizado (F. Mário Martins, 2017). O Java 8 acrescenta construções funcionais como as expressões lambda, funções e *stream* de dados que simplificam o código e permitem aumento de performance através da utilização dos diversos processadores dos computadores atuais. As operações das construções funcionais podem ser combinadas para construção de pipelines de tratamentos de dados. Para se poder definir os tipos das construções funcionais foram criadas as interfaces funcionais. Uma interface funcional define um método único abstrato a ser implementado por uma expressão lambda. Assim sendo, pode-se criar uma

função do tipo de um interface funcional, que é implementado por um expressão lambda (F. Mário Martins, 2017) .

Uma das Interfaces funcionais que existe no Java 8 é a interface funcional *Predicate*<T>. Esta interface representa funções que recebem um parâmetro do tipo T e devolvem um booleano como resultado da execução do método *test*. A interface PQD é do tipo *Predicate*, pelo que cada implementação desta interface implementa o método *test*. As implementações do tipo PQD passam assim a ser funções que ao serem passadas como parâmetro para o método *filter* da *stream* de dados, são executados no momento de execução da pipeline da referida *stream* de dados e filtradas. Cada implementação da interface PQD devolve *false* caso seja encontrado o tipo de PQD, não passando para a fase seguinte do pipeline.

No parágrafo anterior foi referido o termo *stream* para programação de pipelines em Java. Uma *stream* em Java versão 8, permite criar e operar com sequências de elementos. Os elementos na *stream* podem ser submetidos a operações de tratamento de dados predefinidas. Estas operações podem ser combinadas e executadas em sequência, constituindo um pipeline de operações na *stream*. As operações a executar sobre os elementos das *stream*, podem ser executas em sequência ou com paralelismo. Um número muito significativo das operações admitidas em *streams* são baseadas em construções funcionais, normalmente expressões lambda. As *streams* de elementos representam sequencias de valores de determinado tipo. Os valores podem ser criados ou obtidos através de coleções. As coleções do Java 8 têm o método *stream* que cria uma *stream* de elemento, com uma sequencia de elementos do tipo de dados da coleção (F. Mário Martins, 2017) . Na *stream* pode-se usar paralelismo, através do método *parallel*, tornando a solução com maior desempenho.

4.6.1 Iterações de execução da deteção

A deteção de PQD é efetuada de forma iterativa, carregando para memória um conjunto de registos relativos ao número de *batchs* determinado, de acordo com o desenho da Figura 17. Para determinar o número de *batchs*, existe uma Interface *AlgoritmoDeBatch* que define o método *determinaNumeroBatch*. Implementa-se um algoritmo de determinação do número de *batch* através da classe *AlgoritmoDeBatchImpl* que implementa a interface *AlgoritmoDeBatch*. Este algoritmo divide o número total de registos da base de dados num número fixo de *batch*, por forma a cumprir com os requisitos funcionais definidos. A implementação do algoritmo de batch está ilustrado na Figura 25. Existe apenas uma implementação do algoritmo, mas é possível implementar algoritmos alternativos. Na configuração da fonte de dados é possível introduzir o tamanho de cada *batch* através da introdução do parâmetro opcional *batchsize*. Esta opção permite ao utilizador ajustar melhor a execução da solução com as características do computador e da fonte de dados.

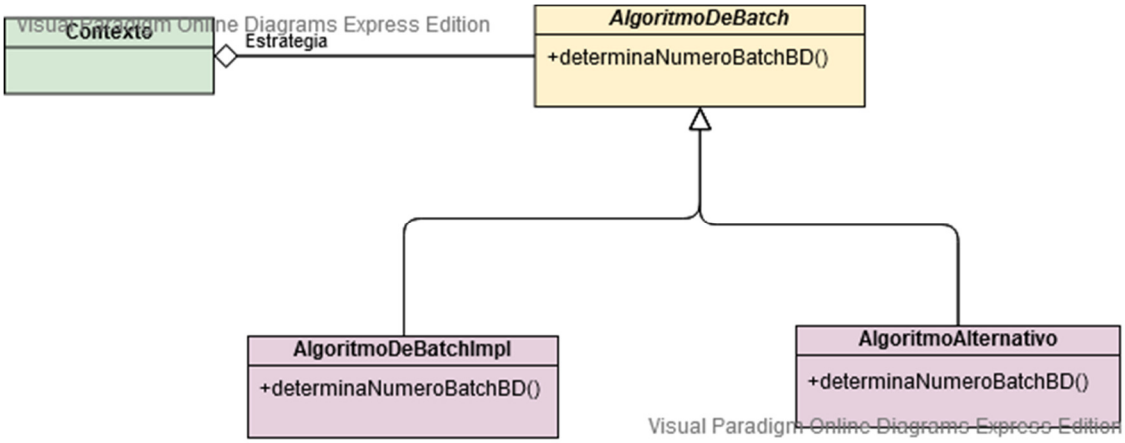


Figura 25: Algoritmo de *Batch*.

Na Figura 17, depois de determinados o número de *batchs*, segue-se a atividade "Executa blocos *Batch*". Para cada *batch* é carregado da fonte de dados origem, o número de registos correspondentes a esse *batch*. O número de registos do *batch* é passado como parâmetro, para o processo de execução da deteção de PQD.

No processo de deteção é percorrida a lista de operações provenientes do sistema pela ordem indicada pelo utilizador. Para cada operação de deteção de PQD é submetido o conjunto de registos do *batch*. É executada a estratégia de deteção específica do PQD, de forma iterativa sobre o conjunto de registos do *batch*. Se for detetado um PQD, o mesmo é gravado na base de dados do sistema e eliminado da lista de registos do *batch*. Depois de verificados todos os registos de todos os *batch*, em todas as operações de deteção escolhidas pelo utilizador, e pela ordem indicada pelo utilizador, os registos que restam são considerados "limpos" e registados na base de dados do sistema, numa tabela de registos que não apresentam PQD.

4.7 Modelo de Dados.

A Figura 26 ilustra o modelo de dados da base de dados de sistema (BD PQD). Tem-se a representação da tabela de PQD (*RegistosComPQD*) e a tabela de dados limpos (*DadosLimpos*). A tabela de dados limpos tem os seguintes campos (colunas): o campo chave *ID* auto - incrementável; o campo *OrigemDados* que representa o nome da fonte de dados (no caso da base de dados é o nome da base de dados); o campo *ConjDados* (no caso de base de dados corresponde ao nome da tabela dos dados fonte ou à expressão SQL); O campo *Registo* guarda uma copia dos registos da fonte de dados original, sobre os quais não foram detetados PQDs.

A tabela de PQD contem os mesmos campos da tabela de dados limpos, acrescentando dois campos adicionais: Tipo, que corresponde ao tipo de PQD detetado; O campo Atributo representa o campo da base de dados fonte no qual se detetou PQD;

Os Dados de ambas as tabelas são alimentadas pelo processo de execução de deteção de um *workflow*, pelo que aparece no diagrama da Figura 25, duas relações de muitos para 1. A relação da tabela de PQD com a tabela de *workflow*; e, a relação da tabela de dados limpos com a tabela de *workflow*.

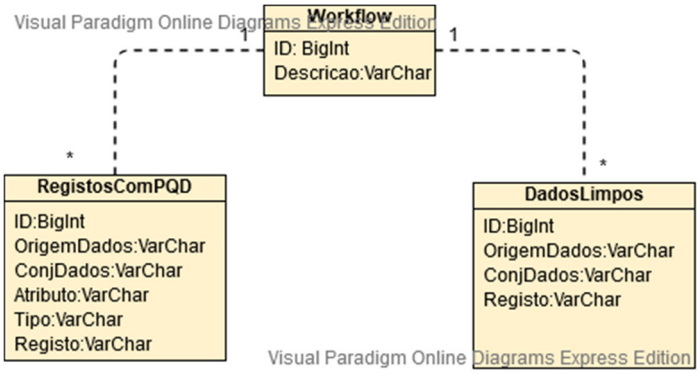


Figura 26: Modelo de dados da BD sistema.

5 Implementação

Neste capítulo será descrito o processo de implementação da solução. A solução consiste no desenvolvimento de um protótipo para detetar Problemas de Qualidade nos Dados (PQD), para os tipos de PQD especificados no capítulo 4 na definição de requisitos.

A descrição de implementação da solução é efetuada por etapas correspondentes ao desenho apresentado para a solução. Serão utilizados diagramas, imagens e excertos de código para explicar o desenvolvimento da solução.

5.1 Etapas do desenvolvimento da solução

Com as etapas de análise e desenho da solução finalizadas, segue-se o desenvolvimento da solução propriamente dita. Procede-se à identificação das etapas de desenvolvimento, de acordo com o desenho efetuado:

1. Integrar implementação da arquitetura MVC.
2. Estruturar o projeto com *SpringBoot*.
3. Integrar componentes de terceiras partes, necessárias à implementação, nas três camadas do modelo MVC.
4. Desenvolver a interface do utilizador.
5. Desenvolver a camada de controlador;
6. Desenvolver a camada de modelo;
    a) Desenvolver o processo de conversão.
    b) Desenvolver o processo de execução da deteção.
7. Desenvolver o acesso a dados.
8. Implantar a solução.

Em cada fase do desenvolvimento é feita uma descrição detalhada, para uma completa compreensão da implementação da solução.

5.2 Integrar implementação da arquitetura MVC

Entende-se por integrar implementação da arquitetura MVC, a criação da estrutura base do projeto de desenvolvimento da solução, seguindo a arquitetura MVC. Para criar a estrutura

do projeto, recorreu-se ao desenvolvimento de um projeto *Maven* (maven.apache.org, 2020) de raiz, baseado na *framework Spring* MVC (Mak, 2008).

Como referido na Secção 2.4.3.2 a *framework Spring* MVC é baseada no padrão MVC, permitindo criar aplicações web *fullStack*. Sendo uma *Framework Spring* , pode-se tirar partido do container "Inversion Of Control" (IoC) do *Spring* , e utilizar mais alguns padrões de software implementados pela *framework Spring* , como a injeção de dependências e *singleton*, uma vez que os *Beans Spring* são *singletons*, permite usar um nível de memória mais baixo, e outras configurações automáticas baseadas em Java *Reflection* [10] que facilitam o processo de desenvolvimento da solução (Ladd et al., 2006) . Os padrões de software utilizados no desenho e na implementação da solução melhoram a qualidade da solução obtida, aumentam a reutilização e a coesão, através do princípio da responsabilidade única dos objetos, e da cooperação dos diferentes objetos intervenientes quando necessários. A *framework Spring* permite inverter o controlo através da utilização do motor de injeção de dependência, para diminuir o acoplamento entre os diversos objetos utilizados na implementação da solução.

Outro aspeto que ajudou na escolha do *Spring* MVC foi a facilidade de integração da *framework JQuery.flowchart* para desenho do *workflow*, através do motor de templates *Thymeleaf*. O código das paginas escritas com *Thymeleaf* é HTML padrão, facilitando a incorporação da *framework JQuery.Flowchart* com os componentes HTML para desenho do *workflow* (thymeleaf.org, 2020) .

Thymeleaf é um motor de templates Java do lado do servidor que permite criar aplicações web em HTML. O *Thymeleaf* é facilmente integrável como *Spring* MVC para realizar a integração dos ficheiros HTML da camada de apresentação com os controladores da *framework* MVC. Sendo a *framework JQuery.Flowchart* baseada na *framework JQuery*, a sua integração nos ficheiros HTML com templates *Thymeleaf* é muito fácil, bastando apenas fazer a inclusão no *header* do ficheiro HTML. Toda a lógica do desenvolvimento da camada de apresentação passa a ser desenvolvida a partir de um ficheiro de base em JavaScript, incluído no ficheiro HTML com templates *Thymeleaf*.

5.3 Estrutura do projeto com *SpringBoot*

Tendo em vista a suportabilidade da solução, relativamente à facilidade de instalação, manutenção e escalabilidade, decidiu-se criar um projeto *SpringBoot*. Com *SpringBoot*, tem-se o *packaging* da solução num ficheiro de extensão *jar*, sendo assim facilmente executável como uma aplicação Java normal.

[10] Reflection: permite fazer uma introspeção do programa em execução, permitindo manipular as propriedades internas do mesmo («Using Java Reflection», sem data) .

Na Figura 27 apresenta-se um exemplo de uma aplicação *SpringBoot* com estrutura *Maven*. Do lado esquerdo vê-se a estrutura do projeto e no lado direito vê-se o código da classe de inicialização da aplicação. A anotação @*SpringBoot*Application permite arrancar a aplicação e carregar automaticamente todas as configurações *SpringBoot*, e todas as configurações padrão das dependências declaradas no ficheiro pom.xml do *Maven*. Com o arranque da aplicação *SpringBoot* entre outros componentes, arranca também a execução do container web padrão embutido, o *Tomcat* (tomcat.apache.org, 2020). No lado esquerdo da Figura 27, a pasta "src/main/java" contem o código da camada de controlo e camada de modelo. A pasta "src/main/resources" contem o código da camada de apresentação.

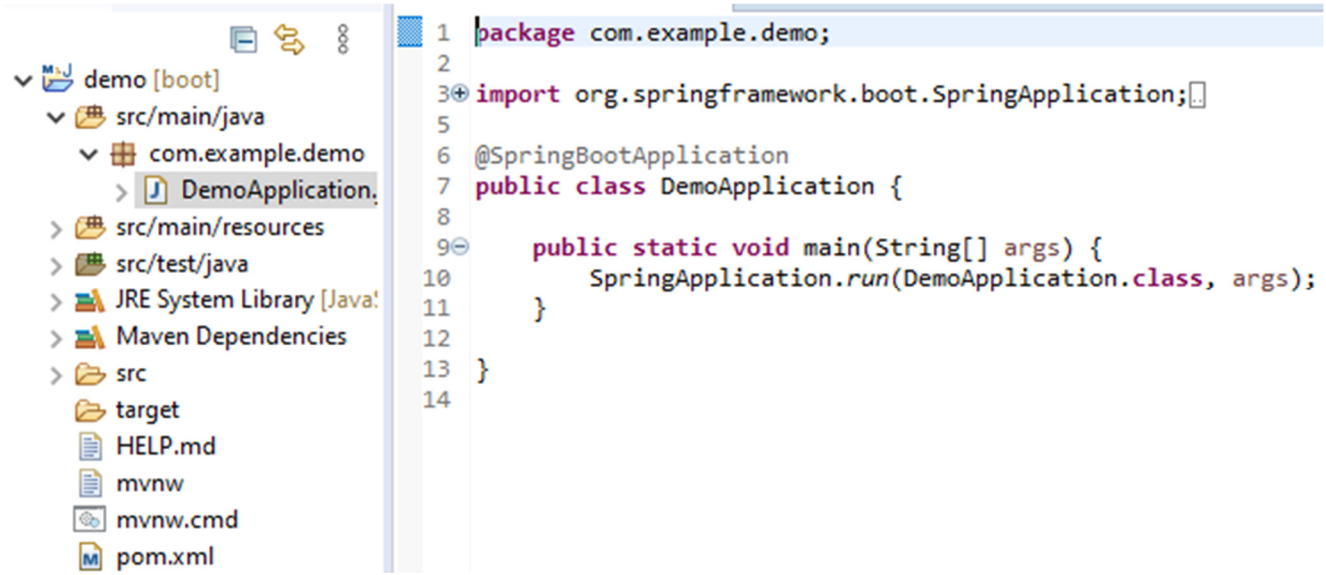


Figura 27: Exemplo de aplicação *SpringBoot*

5.4 Integrar componentes de terceiras partes

Durante o processo de desenvolvimento, é comum utilizar *framework* ou bibliotecas existentes no mercado, para resolver problemas comuns a várias aplicações. As entidades que desenvolvem estas *frameworks*, normalmente tornam-se especialista no domínio dessas partes comuns, pelo que poucas vezes compensa o desenvolvimento de *frameworks* próprias.

Durante a fase de análise e desenho, identificou-se a necessidade de pesquisar uma *framework* muito importante para a implementação da interface gráfica de acordo com a usabilidade pretendida. Tratava-se de pesquisar uma *framework* que permitisse "Drag and Drop", e que por outro lado, desenhasse um componente que representasse uma operação de deteção de PQD. Da pesquisa e análise comparativa de tecnologias JavaScript, através da Tabela 3, Secção 2.4.3, resultaram como melhores tecnologias a biblioteca JsPlumb "comunity edition" e a *framework JQuery.flowchart*. Após uma análise mais cuidada, optou-se por usar a *framework JQuery.flowchart*, pelo facto de ser totalmente freeware, enquanto que a JsPlumb

"comunity edition" não sendo suficiente para a interface gráfica pretendida, podia requerer aquisição de licença. A *framework JQuery.flowchart* está descrita na Secção 2.4.3.3. é uma *framework* totalmente desenvolvida com base na biblioteca de JavaScript JQuery, pelo que basta incluir no template HTML os ficheiros da *Framework JQuery.flowchart*, como mostra o Excerto de Código 1.

```
<script src="/jquery.flowchart/js/jquery.panzoom.min.js"></script>
<script src="/jquery.flowchart/js/jquery.mousewheel.min.js"></script>
<script src="/jquery.flowchart/js/jquery.flowchart.js"></script>
```

Excerto de Código 1: Ficheiros da *Framework JQuery.flowchart*

O 1º ficheiro contém código para a funcionalidade de zoom; o 2º ficheiro contém código para deslocamento do painel de *workflow* e zoom com a roda do rato. O 3º ficheiro contém o restante código para desenhar os componentes e permitir registo de eventos.

O código da camada de apresentação foi implementado com recurso à biblioteca JQuery, pelo que esta *framework* também foi incluída nos ficheiros HTML. Tendo em conta a estética da interface gráfica e a responsividade da mesma, foi integrada a biblioteca *bootStrap*.

A *framework* de templates *Thymeleaf*, foi incluída para permitir a interligação da camada de apresentação com a camada de controlador. Para adicioná-la bastou adicionar a dependência *maven* e adicionar na definição do ficheiros HTML o *namespace*:

```
xmlns:th="http://www.thymeleaf.org"
```

Excerto de Código 2: Tag de namespace do *Thymeleaf*

As restantes *frameworks*/bibliotecas utilizadas, foram adicionadas no ficheiro pom.xml. Apresenta-se as *frameworks* mais importantes nos Excertos de Código 3, 4, 5, 6 e 7

- para implementação do acesso à base de dados de sistema:

```
<dependency>
    <groupId>org.springframework .boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Excerto de Código 3: Dependência Spring Data.

- Para conversão dos dados provenientes da apresentação:

```
<dependency>
    <groupId>org.springframework .boot</groupId>
    <artifactId>spring-boot-starter-json</artifactId>
</dependency>
```

Excerto de Código 4: Dependência de *parser* JSON.

- Driver da base de dados do sistema e consola H2:

```
<dependency>
    <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
```

```
        <scope>runtime</scope>
    </dependency>
```

Excerto de Código 5: Dependência H2.

- Driver da origem de dados suportada MSSQL:

```
<dependency>
        <groupId>com.microsoft.sqlserver</groupId>
        <artifactId>mssql-jdbc</artifactId>
</dependency>
```

Excerto de Código 6: Dependência MSSQL

- Driver da origem de dados suportada MYSQL:

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Excerto de Código 7: Dependência MySQL

5.5 Desenvolvimento da interface do utilizador

A interface gráfica está desenvolvida em HTML com templates *Thymeleaf*. Está definido um layout padrão com recurso ao *thymeleaf*, onde são definidas as importações de bibliotecas e *frameworks* comuns às diferentes vistas da solução. No layout é definido uma secção como se pode ver no excerto de código seguinte:

```
<section layout:fragment="conteudo">
        <p>conteudo principal</p>
</section>
```

Excerto de Código 8: Template da camada de apresentação.

Os ficheiros HTML que importam o layout substituem o conteúdo desta secção. Existem três vistas na solução implementados em 3 ficheiros:

- **resultadosdetecaopqds.html** - onde se pode visualizar o resultado da deteção de PQD. Esta vista inclui um iframe [11]com a consola do H2 (h2database.com, 2020) para visualização de resultados;

- **verparametrizacao.html** - depois de configurados todos os componentes do *workflow*, o utilizador pode visualizar as configurações nesta vista;

desenha_flowchart.html - Este ficheiro contém a implementação da vista principal da solução.

[11] Iframe: O iframe é utilizado para embutir um documento html dentro de outro.

5.5.1 Implementação do *workflow*

O *workflow* da solução está implementado no ficheiro desenha_flowchart.html. Este ficheiro contém todos os componentes HTML necessários para implementar o *workflow*, bem como as "Spring Expression Language"[12] do *Spring*, que o *Thymeleaf* usa para ligação com a camada de controlo. Neste ficheiro é incluído o ficheiro de JavaScript principal.js que contém o código JavaScript necessário para executar todas as ações do utilizador com a interface gráfica. O ficheiro Principal.js inclui diversos ficheiros, de forma a distribuir as responsabilidades por diversas partes. Mostra-se o cabeçalho do ficheiro com os comentários das responsabilidades nos ficheiros importados no Excerto de Código 9.

```
$(document).ready(function() {
	/* obtem determinado elemento nos dados*/
	$.getScript('/aplicacao/js/childs/uteis.js');

	/* abre popup de componentes PQD valor em falta. */
	$.getScript('/aplicacao/js/childs/PQD_DF.js');

	/* abre popup de componentes PQD violação de dominio	*/
	$.getScript('/aplicacao/js/childs/PQD_VD.js');

	/* abre popup de componentes PQD violação de sintaxe	*/
	$.getScript('/aplicacao/js/childs/PQD_VS.js');

	/* abre popup origem de dados	*/
	$.getScript('/aplicacao/js/childs/Origem_dados.js');

	/* cria  componentes do menu drag and drop */
	$.getScript('/aplicacao/js/flowchart/draggableOperators.js');

  /* adiconna funcionalidadede zom ao panel de desenho do flowchart*/
	$.getScript('/aplicacao/js/flowchart/flowchartZoom.js');

	/*	codigo dos eventos dos botoes configurar, submeter,
   gravar, carregar, esconder JSON e botao apagar----- */
	$.getScript('/aplicacao/js/flowchart/botoes_outros.js');


	/*	funcções auxiliares: getNUmeroOperatorInicial,
getNUmeroOperatorFicheiro,
	 *	getNUmeroOperatorRetorno, Configuracao2Flow, Carregar2Flow,
Carregar2Retorno */
	$.getScript('/aplicacao/js/flowchart/auxiliaryJQueryFunctions.js');

	/* eventos dos componentes drag and drop do painel de flowchart */
	$.getScript('/aplicacao/js/flowchart/eventosFlowchart.js');

	/* cria o operador de dados limpos */
	$.getScript('/aplicacao/js/flowchart/OperadorFim.js');
	$.getScript('/aplicacao/js/onWindowLoad.js');
	$.getScript('/aplicacao/js/carregaDados.js');
		$.getScript('/aplicacao/js/gravaDados.js');
```

[12] Spring Expression language: Utilizadas, entre outras funcionalidades, para aceder a propriedades dos beans e controladores Spring (docs.spring.io, 2020).

Excerto de Código 9: Importação de ficheiros JavaScript

O resultado de implementação da vista principal está ilustrado na Figura 28. No topo estão presentes duas barras de menu: a 1ª barra contém as funcionalidades para operar sobre o *workflow*. A 2ª barra de menu contém os componentes "*drag and drop*" a arrastar para o painel de *workflow*. O painel de *workflow* corresponde à área da Figura 28 onde se encontram os componentes MySQL, VF e Dados Limpos.

Apresenta-se, a seguir, as funcionalidades do menu de topo:

- Visualizar: permite visualizar a configuração de todos os componentes do *workflow*, antes do mesmo ser submetido para execução;

- Apagar: permite apagar a ligação ou o componente selecionado no painel de *workflow*;

- Configurar: permite configurar o componente selecionado;

- Gravar: permite gravar todo o *workflow* num ficheiro local, que inclui a parametrização efetuada, parâmetros dos componentes; ligações e posições dos componentes.

- Carregar: permite carregar de um ficheiro um *workflow* anteriormente gravado. O *workflow* é integralmente carregado.

Os itens *dragable* do segundo menu correspondem às origens de dados e PQD suportados pela solução.

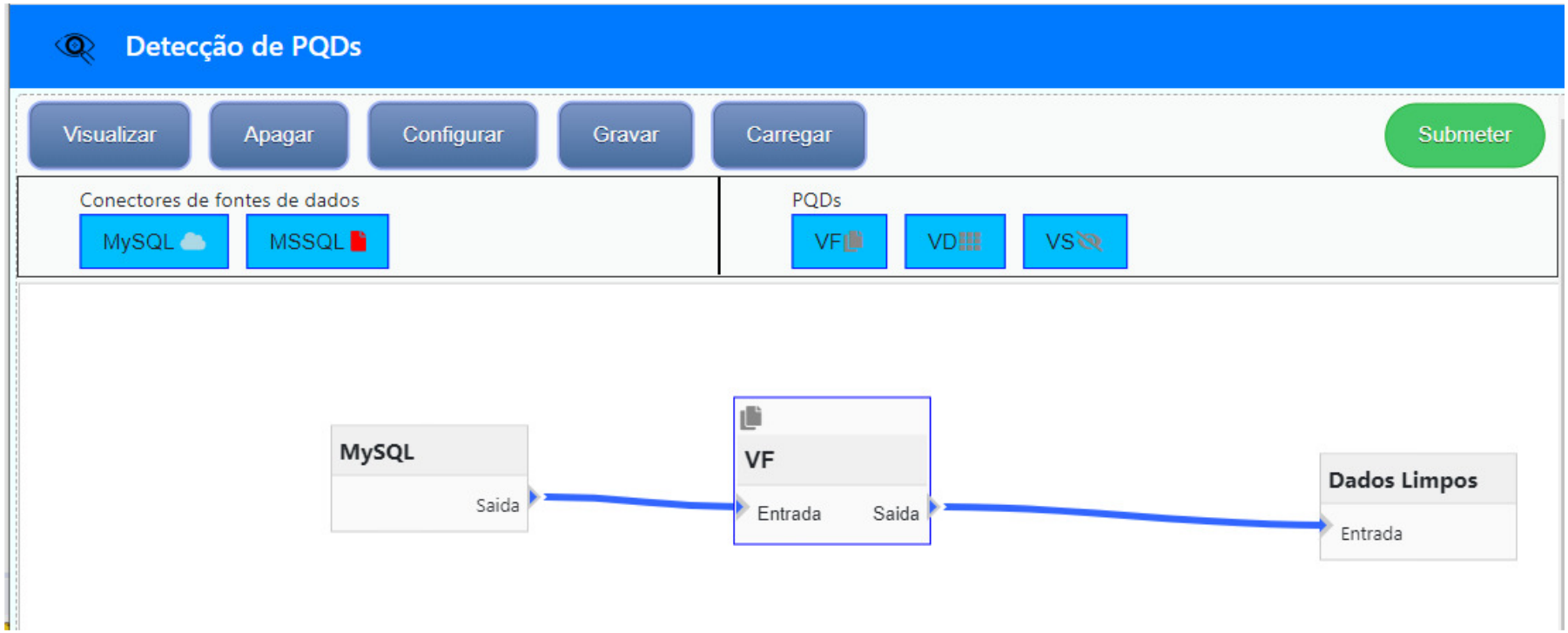


Figura 28: Vista de desenho de *workflow*.

5.5.2 Configuração de componentes

Os componentes do *workflow* são configurados através de janelas de *pop-up*. Estas janelas são abertas através de código JavaScript com *JQuery*, no evento de criação de componentes e no evento do botão "Configurar" que configura o componente selecionado no *workflow*. Apresentam-se a seguir nas Figuras 29, 30, 31 e 32 as imagens das janelas de *pop-up* de configuração dos componentes.

5.5.2.1 Configuração de origem de dados

Na Figura 29 visualiza-se a configuração dos parâmetros da base de dados de origem/fonte numa janela de pop-up. A solução para além de permitir a deteção em apenas uma tabela, permite expressões SQL para obter um conjunto de dados mais refinados numa ou mais tabelas. O utilizador pode definir o parâmetro *batchsize* (tamanho do *batch*), que corresponde à quantidade de registos que são carregados da base de dados de origem em cada iteração do processo de deteção em *batch*.

A solução comtempla a deteção em vários *batchs* para evitar ocupação exagerada de recursos, especialmente a memória RAM. A solução contém a implementação de um algoritmo que permite carregamento parcial dos dados a carregar em *batchs*, para processos de deteção em que o número de registos ultrapasse um determinado valor que, por defeito, são 10 mil registos. No entanto, este valor pode ter impacto diferente em função dos recursos da máquina onde a solução for executada, e também porque a base de dados fonte pode ter conjuntos de dados que ocupam mais espaço em memória, para o mesmo número de registos. Mesmo assim o algoritmo de divisão em *batch* melhora substancialmente a gestão da ocupação de recursos durante o processo de execução da deteção de PQDs. Podem ocorrer situações de deterioração do desempenho da aplicação ou ocorrerem bloqueios, na deteção de PQD em fontes de dados de muito grandes dimensões. Para se puder melhorar o desempenho em fontes de dados de grandes dimensões, opcionalmente, o utilizador pode definir valor para o parâmetro *batchsize* que permite afinar o número de registos ótimo por *batch*. Por exemplo: se o algoritmo de *batch* determinar o tamanho de *batch* em 1 milhão registos, obrigando a uma ocupação de 4GB de RAM na execução da deteção de PQD, com este parâmetro é possível definir o tamanho de *batch* para 500 mil registos e conseguir uma utilização de cerca de 2GB de RAM.

O utilizador será aconselhado a não configurar este parâmetro, exceto em situações estritamente necessárias, de forma a que este parâmetro não afete negativamente o algoritmo de execução da deteção de PQD. Futuramente podem-se desenvolver novos algoritmos de determinação do tamanho do *batch* de forma a ser menos frequente a eventual necessidade de se utilizar este parâmetro.

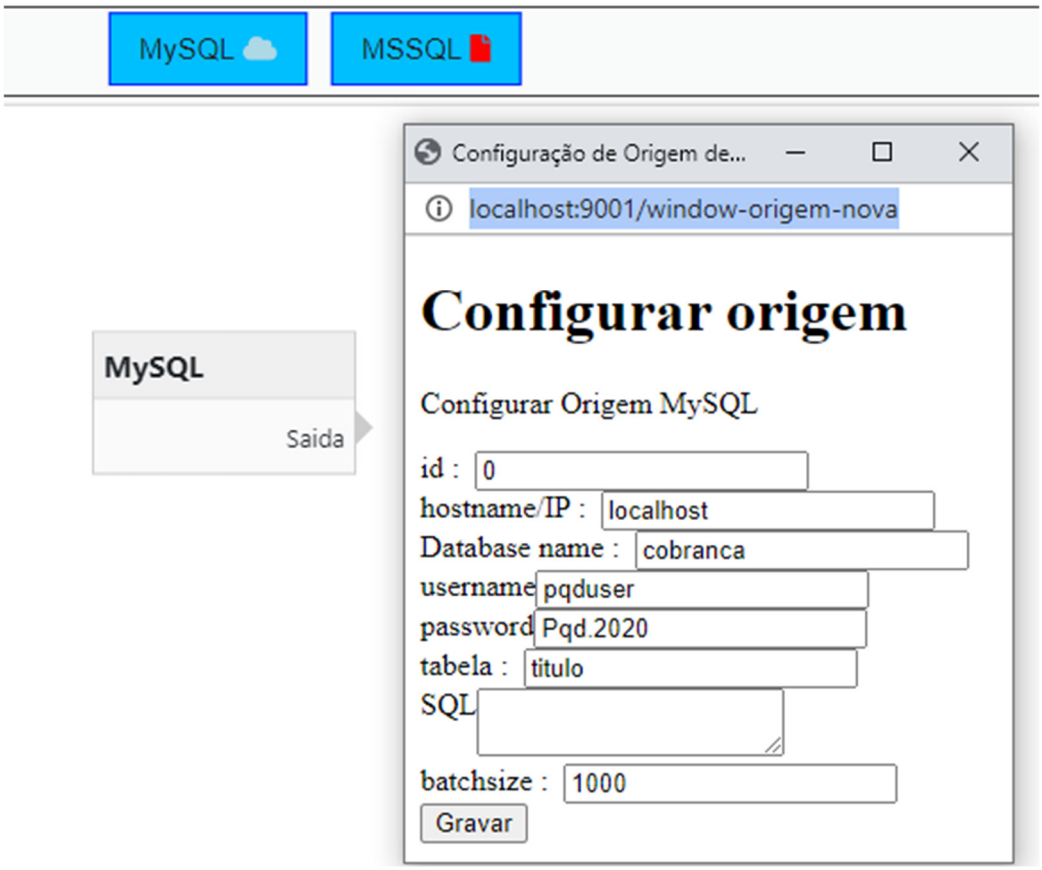


Figura 29: Configuração de origem de dados

5.5.2.2 Configuração de PQD valor em falta

Na Figura 30 apresenta-se a janela de configuração de uma operação do tipo PQD valor em falta. Esta janela é muito simples, contendo apenas o atributo para o qual se quer detetar este tipo de PQD. Idealmente, o componente HTML do parâmetro atributo, deveria ser um componente que liste uma lista de valores para selecionar o atributo, evitando erros de digitação. Este pode ser um melhoramento a considerar numa futura versão da solução.

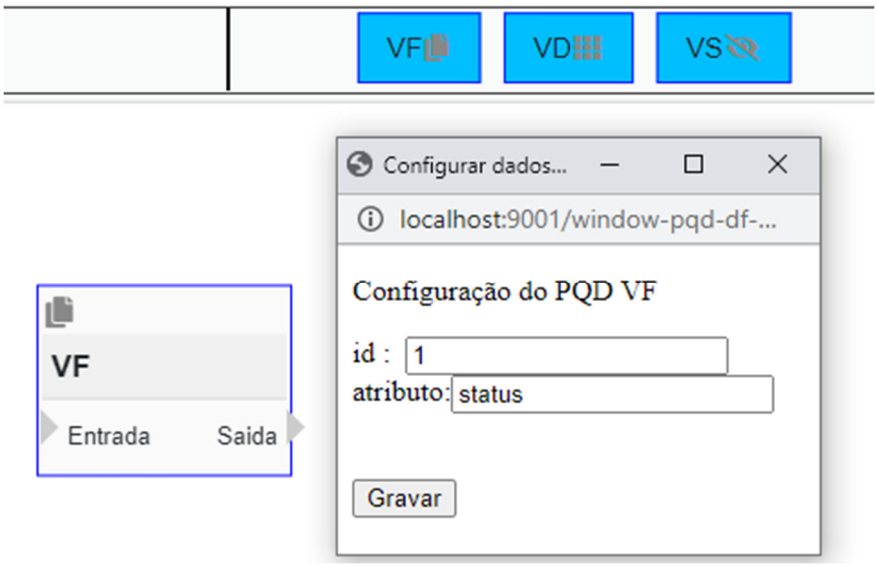


Figura 30: Configuração do PQD valor em falta.

5.5.2.3 Configuração de PQD violação de domínio

Na Figura 31 apresenta-se a janela de configuração de uma operação do tipo PQD violação de domínio. Na configuração desta operação de PQD, além do atributo, tem de se configurar os parâmetros limite inferior e limite superior. Este tipo de PQD deveria permitir introduzir valores enumerados. Uma futura versão da solução, deve considerar valores enumerados.

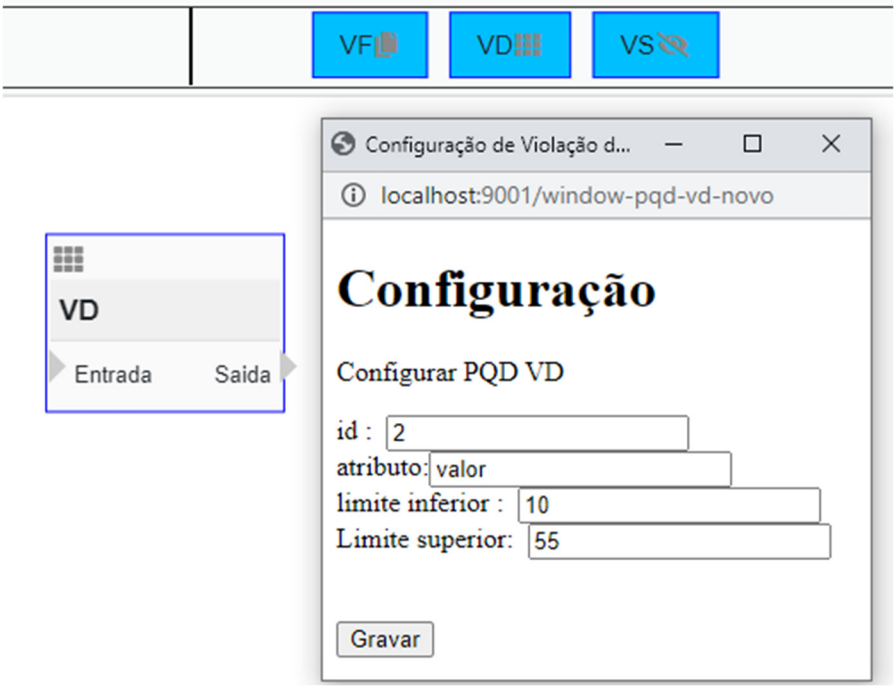


Figura 31: Configuração do PQD violação de domínio.

5.5.2.4 Configuração de PQD violação de sintaxe

Na Figura 31 está apresentada a configuração da operação PQD de violação de sintaxe. Para além do atributo, existe o parâmetro para configurar a expressão regular que permitirá validar os dados respetivos do atributo escolhido. Junto ao campo de introdução da expressão regular tem um pequeno ícone semelhante à letra “i”. Ao clicar neste ícone abre uma nova janela com informação sobre o tipo de expressões regulares que a solução suporta. Este tipo de PQD deveria permitir introduzir outros tipos de expressões regulares. Uma futura versão da solução, deve considerar outro tipo de expressões regulares.

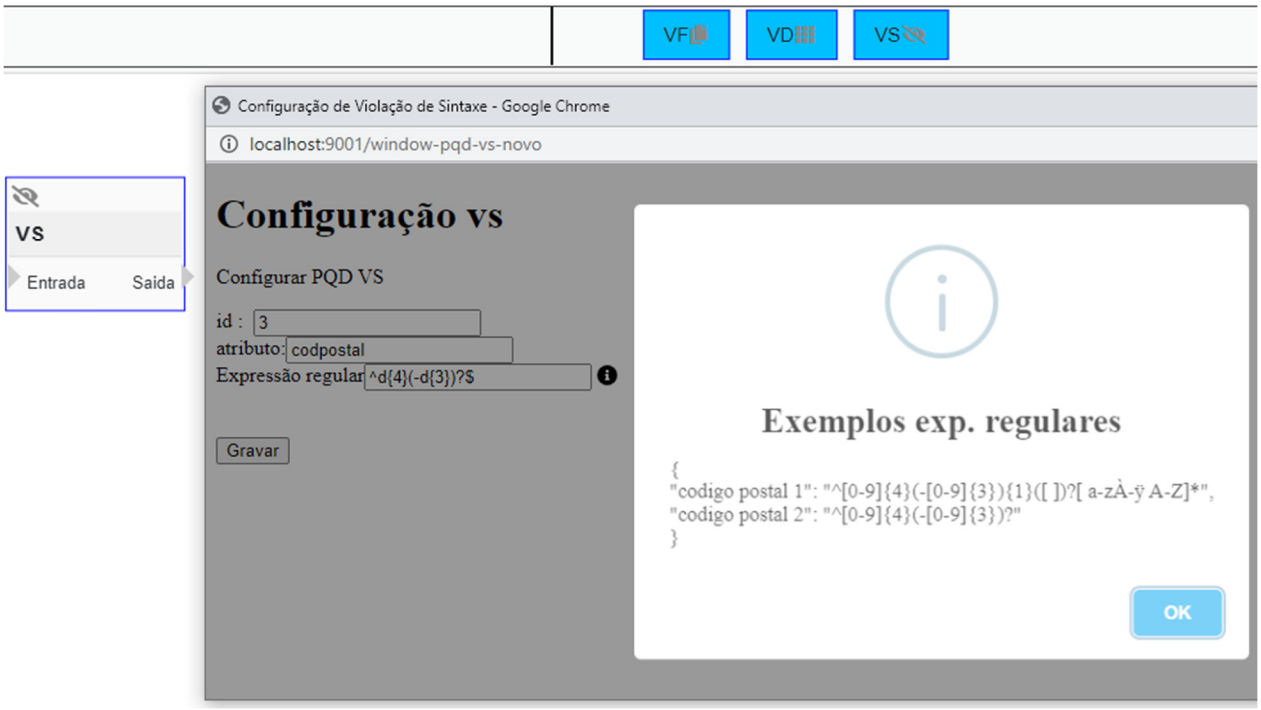


Figura 32: Configuração do PQD violação de sintaxe.

5.6 Desenvolvimento de camada de controlador

Apresenta-se na Figura 33, o componente central do Spring MVC, o *DispatcherServlet*. O *DispatcherServlet* estende *HTTPRequest*, e é uma implementação do padrão *FrontController* (martinfowler.com, 2020). Quando o componente *dispatcher* recebe um pedido dum cliente, consulta o componente *HandlerMapping* para invocar o controlador apropriado. O controlador recebe o pedido e chama o serviço adequado (*HandlerMethod*), que é um método para tratar pedidos HTTP GET ou HTTP POST. Neste serviço, o pedido é processado, invocando métodos nos objetos da camada de modelo. Finalmente o controlador retorna o nome da vista (visualização ou página web da solução) e parâmetros do modelo. Assim que o *dispatcher* resolve o nome da vista, através do componente *viewResolver* num objeto para "renderização". Finalmente, o componente *View* exibe ao utilizador o objeto vista junto com os parâmetros do modelo (Mak, 2008).

Uma classe controladora tem a anotação *@Controller* que indica ao *Spring* que a classe serve como um controlador, e a anotação *@RequestMapping* que serve para mapear o URL.

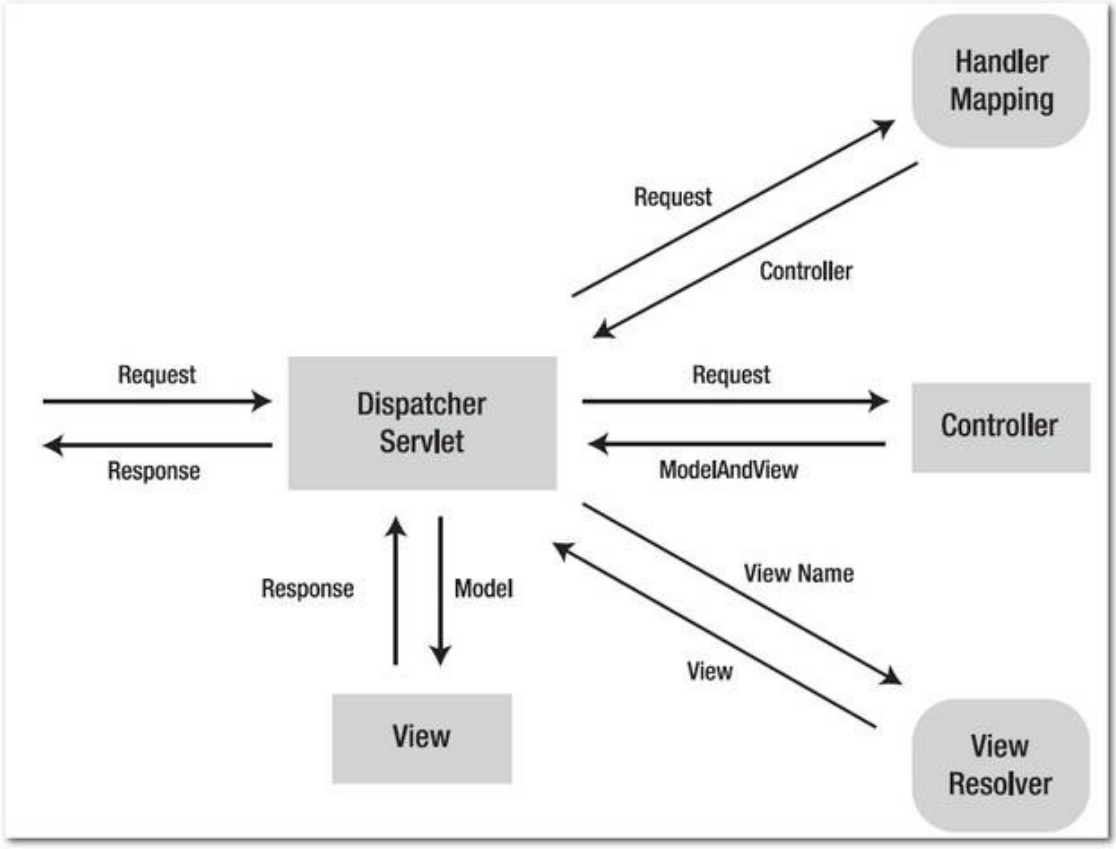


Figura 33: *Spring* MVC *DispatcherServlet* (Mak, 2008).

Na solução de PQD existe o controlador *ControladorPrincipal* que invoca os métodos dos objetos da camada de modelo. Segue-se o Excerto de Código 10 do método *controlaFluxoDeDados* deste controlador:

```
@RequestMapping(method = RequestMethod.POST, params = "action=submitlines")
public ModelAndView controlaFluxoDeDados(@RequestParam("jsonPQDs") String pqdsGlobal,
RedirectAttributes redirectAttributes, Model model) throws IOException {
	ModelAndView redirect ;

	redirect = processoConversao.processoDeConversaoDeDados(pqdsGlobal,
	processoExecucaoDetecao, tuplesPQDService);

	processoExecucaoDetecao.processoDeExecucaoDaDeteccao();

	redirect.addObject("pqdsGlobal", pqdsGlobal);
	return redirect;
}
```

Excerto de Código 10: Método de controlo de fluxo de dados

Os objetos *processoConversao* e *processoExecucaoDetecao* são instanciados pelo motor de injeção de dependências do *Spring*. Estes serviços são anotados com a anotação *@Service*.

O método *controlaFluxoDeDados* recebe como parâmetro, da camada de apresentação, o objeto JSON "jsonPQDs". O objeto *jsonPQDs* é passado como parâmetro para o método *processoDeConversaoDeDados* da classe *ProcessoConversao* onde são extraídos os objetos necessários à execução da deteção. O método *processoDeConversaoDeDados* recebe ainda como parâmetro o objeto *processoExecucaoDetecao*, para o qual são passados os objetos Java depois de extraídos no processo de conversão.

Estando finalizado o processo de conversão de dados, o controlador *ControladorPrincipal* invoca o método *processoDeExecucaoDaDeteccao* da classe *ProcessoExecucaoDetecao* para a execução da deteção de PQD.

Na camada de controlador, para além do controlador *ControladorPrincipal*, existe o controlador *VisualizaParametrosWorkflow*, que permite visualizar a configuração efetuada pelo utilizador, e o controlador *VerResultados*, que permite visualizar os resultados da deteção. Nesta camada existem ainda os controladores necessários à "renderização" das janelas de pop-up para configuração dos parâmetros de PQD.

5.7 Desenvolvimento de camada de modelo

A camada de modelo é a camada responsável pela lógica da solução de deteção de PQD. Existem duas classes principais, a classe *ProcessoConversao* e a classe *ProcessoExecucaoDetecao* que correspondem aos dois processos que ocorrem no modelo de negócio: o processo de conversão de dados e o processo de execução da deteção de PQD, respetivamente.

5.7.1 Desenvolvimento de processo de conversão.

O processo de conversão é responsável por extrair do objeto *jsonPQDs* referido na Secção 5.6, os objetos necessários à deteção de PQD. Segue-se o excerto de código o método *processoDeConversaoDeDados* da classe *ProcessoConversao*:

```
@Override
public void  processoDeConversaoDeDados(String jsonGlobal
        , ProcessoExecucaoDetecao processoExecucaoDetecao) {
        List<PQD> operacoesPQD =
         obtemOperacoesPQD(jsonGlobal,tuplesPQDService);
        List<LigacaoWorkflow> ligacoes = obtemLigacoes(jsonGlobal);
        List<ComponenteWorkflow> componentesWorkflow =
                obtemComponentesWorkflow(jsonGlobal);
        List<PQD> operacoesOrdenadas =
         ordenaOperacoes( ligacoes, componentesWorkflow,  operacoesPQD);
        processoExecucaoDetecao.setOperacoesLambda(operacoesOrdenadas);
        processoExecucaoDetecao.setOrigemDados(obtemOrigemDados(jsonGlobal
                ,tuplesPQDService));
}
```

Excerto de Código 11: Método do processamento da conversão de dados.

Como é mostrado no Excerto de Código 11, no método *processoDeConversaoDeDados*, começa-se por extrair a lista das operações PQD escolhidas pelo utilizador. Segue-se a lista de ligações, e uma lista auxiliar de operações de *workflow* utilizada na ordenação da deteção de PQD. Depois ordenam-se as operações e submetem-se ao processo de execução da deteção. No seguimento extrai-se a componente origem de dados, que é submetido ao processo de execução da deteção. Finalmente são retornados os objetos do modelo para serem devolvidos à vista de resultados da deteção, depois de ocorrer a execução da deteção.

No método *obtemOperacoes*, à medida que o objeto *jsonPQDs* é percorrido, ao encontrar um componente PQD, é utilizada a *AbstractFactory* de PQD que instancia a *Factory* específica do PQD. É invocada o método cria da *Factory* instanciada, para extrair as propriedades JSON do PQD em questão e instanciar o PQD respetivo. No método *obtemOrigemDados*, é utilizada a *AbstractFactory* de origem para instanciar a *factory* adequada. É invocado o método cria da *factory*, onde são extraídas as propriedades da origem de dados de dados e instanciado um objeto do tipo *OrigemDados*, com as propriedades da ligação à fonte de dados.

5.7.2 Desenvolvimento de processo de execução da deteção.

Depois de extraídos todos os objetos necessários à deteção de PQD, procede-se à execução da deteção de PQD no processo de execução da deteção. Neste processo é instanciado o motor de deteção. O motor de deteção é do tipo *IMotor*. Existe uma implementação de *IMotor* na solução através da classe *MotorDeteçãoPQDs* como ilustra a Figura 32. No entanto, futuramente podem-se ter outras implementações do motor de deteção se assim se justificar. Este é um nível de muito alta abstração caso o tipo de abordagens de deteção de PQD permita detetar os restantes tipos de PQD que constam da taxionomia apresentada em (Oliveira et al., 2005), ou caso se pretenda que a solução siga outras metodologias baseadas noutras taxionomias.

O motor contém vários métodos de iterar sobre o conjunto de dados e carregar os dados de origem/fonte. Recorreu-se ao padrão de design de software, o "Template Method", que contém para além da sua própria implementação numa classe abstrata *IteradorDetecao*, a implementação de duas especificações deste "*template method"* , que corresponde a dois algoritmos diferentes de carregamento de dados para deteção de PQD, e também à forma de como os dados são iterados (Zafeiris, Poulias, Diamantidis, & Giakoumakis, 2017) . Com este padrão os métodos que são comuns às várias especificações podem passar para a classe de base, permitindo uma melhor reutilização de código (Hotta, Higo, & Kusumoto, 2012). Desta forma, apenas parte do algoritmo é da responsabilidade das classes específicas e não a totalidade do algoritmo de iteração e carregamento de dados em *batch*.

Na Figura 33 apresentam-se as versões do algoritmo de iteração e carregamento de dados através de duas implementações específicas da classe abstrata *IteradorDetecao*. No método *detectaPQDs* do tipo *MotorDetecçãoPQD*, seleciona-se o algoritmo de iteração e carregamento de dados, baseando-se no facto do utilizador ter definido, ou não, o tamanho dos *batchs* de

dados a carregar para memória RAM. Apresenta-se o excerto de código deste método no Excerto de Código 12.

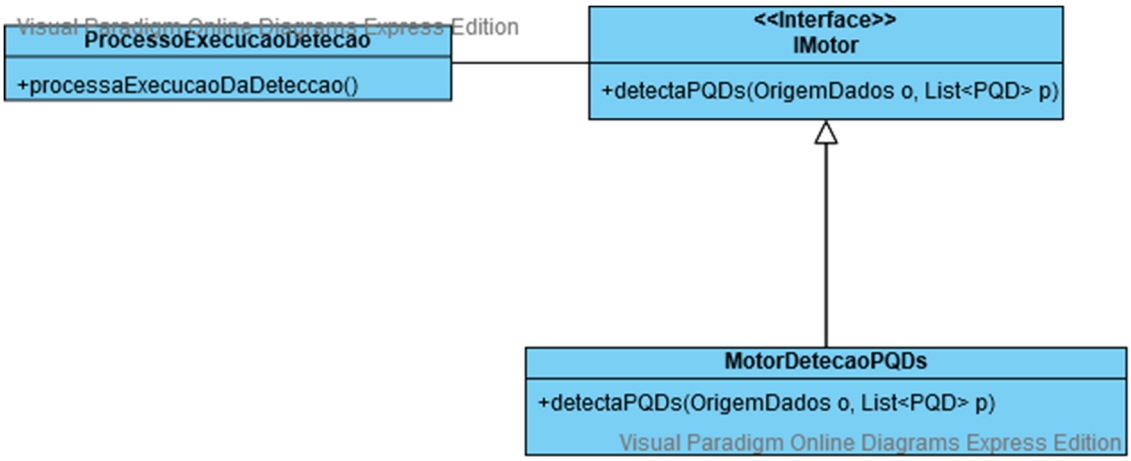


Figura 34: Motor da deteção de PQD.

```
public List<Map<String, Object>> detectaPQDs(OrigemDados origemDados,
List<PQD> operacoesPQDs) {
	this.origemDados = origemDados;
	this.operacoesPQDs = operacoesPQDs;
	List<Map<String, Object>> dadosSaida
		= new ArrayList<Map<String, Object>>();
	long startTime = System.nanoTime();

	Integer tamanhoBatch = ((OrigemBaseDados) origemDados).getBatchsize();

	if (tamanhoBatch != null)
		dadosSaida =
		iteradorBatchSized.executaDetecao( origemDados, operacoesPQDs);
	else
		dadosSaida =
		iteradorBatchFixo.executaDetecao( origemDados, operacoesPQDs);

	long stopTime = System.nanoTime();
	long tempoDecorrido = (stopTime - startTime) / 1_000_000;
	System.out.println("tempo de execucao das operacoes: "
		+ tempoDecorrido + " milisegundos");

	return dadosSaida;
}
```

Excerto de Código 12: Método do processamento da execução da deteção.

O utilizador tem a possibilidade de escolher o tamanho do *batch* de dados carregados em memória em cada iteração do algoritmo do tipo *iteradorDetecao*. No capítulo de testes é aferido um valor típico para este parâmetro, através dos resultados do conjunto de testes funcionais.

Caso o utilizador especifique o tamanho dos *batchs* de dados, existe um algoritmo simples, implementado de acordo com o desenho, na secção 4.6.1, para impedir que os dados sejam carregados para memória de uma única vez, comprometendo os recursos do computador. O

algoritmo implementado separa o número de registos em 5 *batchs*, mas só faz a separação quando o número total de registos é superior a 10000 registos. Este valor é empírico, e influenciado pela utilização do paralelismo. Este valor é aferido a partir da realização de testes funcionais. O paralelismo tem vantagem para grandes quantidades de dados. Em pequenas quantidades de dados, devido aos recursos necessários para a gestão de concorrência, o paralelismo não traz grandes melhorias de desempenho, pelo que não é conveniente carregamento parcial dos dados em *batchs*.

Na classe abstrata *IteradorDetecao* existe um método abstrato *executaDetecao* de execução da deteção de PQD. Este método é executado no algoritmo específico de iteração de dados carregados para memória, instanciado no motor de deteção. A classe *IteradorDetecao* contem ainda os métodos *executaOperacoesEmBatch* e *executaUmaOperacaoEmBatch* que são métodos auxiliares comuns a ambas as classes que estendem a classe *IteradorDetecao*. A classe *IteradorBatchTamFixo* contem o algoritmo de iteração no método *detecaoNosBatch* que, por sua vez, invoca os métodos auxiliares da classe *IteradorDetecao* referidos acima.

O outro algoritmo está contido no método *detecaoNosBatchAlt* da classe *IteradorBatchTamAlteravel*. Este método também invoca os métodos auxiliares da classe *IteradorDetecao* referidos no paragrafo anterior.

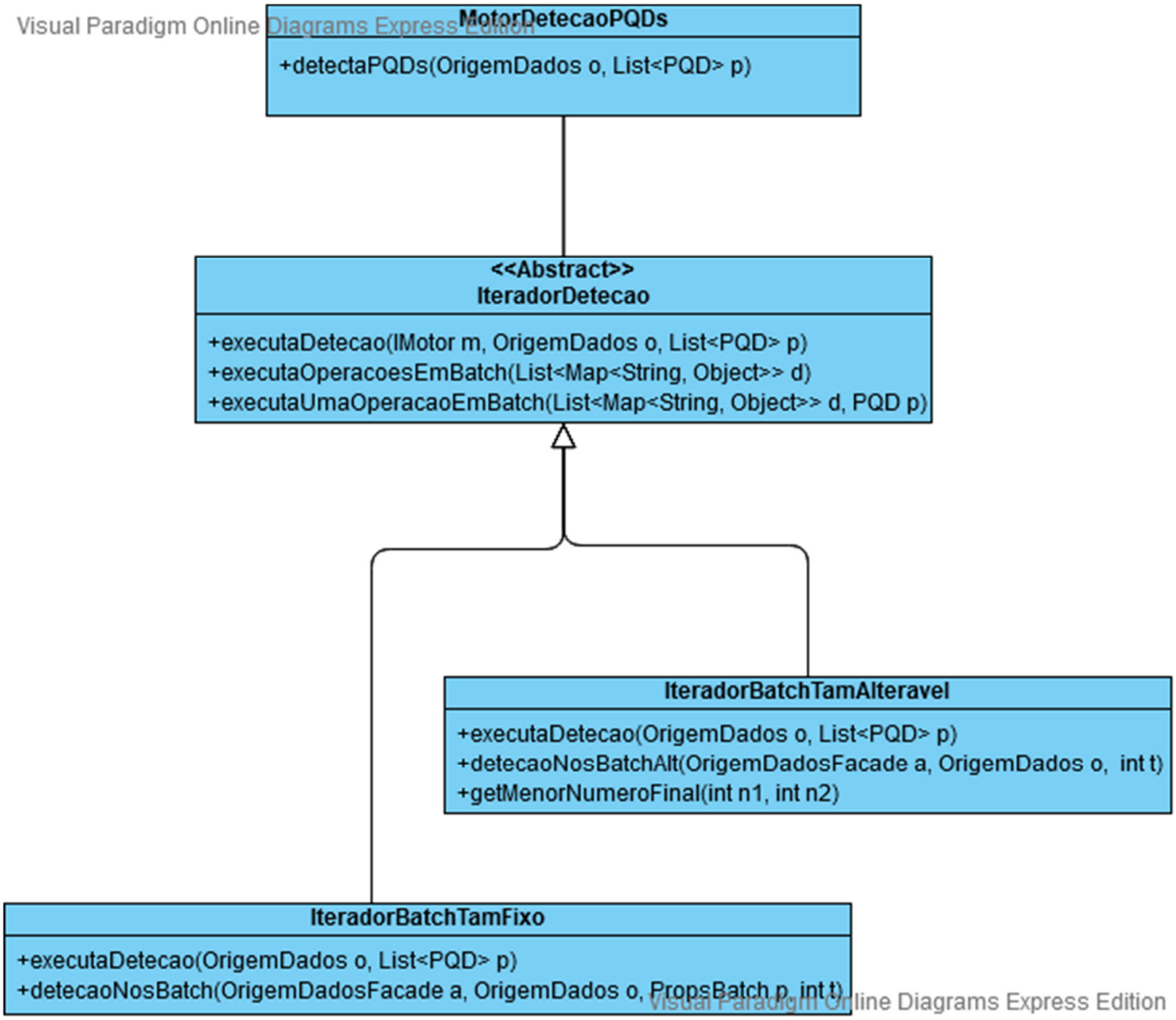


Figura 35: Algoritmos de carregamento de dados em memória.

5.7.2.1 Métodos auxiliares de *IteradorDetecao*

Os métodos da classe *iteradorDetecção*, designados de auxiliares na secção anterior, são:

- `executaOperacoesEmBatch`: - Este método recebe os dados carregados pelo motor da deteção, através do *IteradorDetecao*, e a lista de PQD extraída no processo de conversão de dados. A lista de PQD é percorrida iterativamente e em cada iteração é invocado o método *executaUmaOperacaoEmBatch* para proceder à deteção com recurso a uma *stream* sobre os dados de origem recebidos no método. No fim de cada iteração os dados limpos da iteração anterior são passados para a iteração seguinte. Desta forma, depois de todas as iterações obtém-se os dados limpos para as operações de deteção escolhidas pelo utilizador, os quais, são retornados ao motor de deteção para gravação na base de dados do sistema.

- `executaUmaOperacaoEmBatch`: - Sobre os dados de origem recebidos através do método *executaOperacoesEmBatch*, é criada uma *stream*, que é executada com paralelismo para melhor desempenho. No fim, este método retorna os dados limpos ao método anterior.

5.7.3 Operações de deteção de PQD

As operações de deteções de PQD são do tipo da interface PQD já referida anteriormente. Esta interface estende a interface funcional *Predicate* do Java. As interfaces funcionais *Predicate*<T> representam funções que aceitam um parâmetro do tipo T e devolvem um valor booleano como resultado do método test. Estas funções do tipo *Predicate* são usadas em *streams* de coleções Java, dos dados passados para as operações de deteção de PQD em *batchs*. As *streams* possuem o método *filter* que permite filtrar quais os dados que passam o teste de deteção de PQD (F. Mário Martins, 2017).

As *streams* referidas nesta dissertação, refere-se a *streams* de elementos de coleções Java. As *streams* permitem composição de operações sobre os dados das coleções Java, correspondendo a uma arquitetura de programação de pipeline. Para criar uma *stream*, as coleções a partir do Java 8 contêm o método *stream* que cria um objeto do tipo *Stream*.

Os pipelines Java com *stream* possuem operações intermédias e operações finais de terminação do pipeline. O pipeline só é executado com a definição de uma operação final sobre *streams*(F. Mário Martins, 2017). Na implementação do pipeline com *streams* sobre a coleção de dados carregados da base da fonte de dados de origem, num *batch*, escolhe-se duas operações: a operação intermédia *filter* e a operação final "collect". A operação filter recebe uma função do tipo *Predicate*. As operações PQD são do tipo *Predicate*, pelo que são passadas como parâmetro na operação *filter*. Com a execução do pipeline, o método *test* de cada PQD é invocado, realizando a deteção do PQD. Caso se detete PQD, o mesmo é gravado na base de dados do sistema, e é retornado o valor false. Caso não seja encontrado PQD, o método *test* devolve *true*. Os dados que retornam *false* na operação *filter* não passam para as operações seguintes.

A operação *collect* permite converter a *stream* numa outra estrutura de dados, para se puder voltar a detetar dados noutra iteração, com outra operação de deteção de PQD. Depois de percorridas todas as operações PQD escolhidas pelo utilizador, o resultado da deteção de PQD é uma coleção com os dados que passaram todos os testes de verificação de PQD, sendo considerados dados limpos.

5.7.3.1 Paralelismo em *streams*

A partir de Java 8 a execução de operações paralelas baseadas em *streams*, permite aumentar o desempenho para grandes volumes de dados, dividindo as tarefas em múltiplas

threads. A implementação paralela sobre *streams* é baseada na *framework* "*Fork/Join thread poo*l". A implementação paralela em *streams* sobre coleções de dados de tamanho fixo em memória, durante a execução do pipeline, utiliza todos os processadores disponíveis (Mei, Gray, & Wellings, 2015). A *framework* "*fork join thread poll*" (*ForkJoin*) é uma *framework* de execução paralela, na qual as tarefas são executadas por separação em tarefas menores executadas em paralelo, esperando-se pelo final da execução das mesmas para composição do resultado (Mei et al., 2015).

Existem situações em que a execução paralela pode ser menos eficiente, devido a "*side-effect*" de funções, em casos em que existem dependências entre as operações, provocando bloqueios ou diminuindo o desempenho da execução paralela (Khatchadourian, Tang, Bagherzadeh, & Ahmed, 2019). Na solução desenvolvida, as operações são isoladas, ou seja, não existem interdependências entre as operações a executar.

O paralelismo tem especial importância na deteção de PQD para grande volume de dados, pois permite elevado desempenho na deteção de PQD. Outra razão da escolha do paralelismo com *streams*, é a facilidade de utilização de paralelismo com *streams* a partir do Java 8. A programação *multithread* é, por vezes complexa, mas é muito utilizada, dado o grande desempenho que se consegue. Na solução desenvolvida, como as condições são adequadas para uso de *streams* paralelas, basta usar o método *parallel* da *stream* para se obter a execução paralela do pipeline, com resultados de desempenho notáveis, especialmente para grande volume de dados. Este desempenho pode ser verificado com os testes funcionais no Capítulo 6.

5.7.3.2 O tipo PQD

A interface PQD estende a interface funcional *Predicate<Map<String, Object>.* O *map* passado como tipo na interface *Predicate*, corresponde a um registo da base de dados de origem. A interface PQD subscreve o método *test* da interface *Predicate*, que na interface PQD é um método *default*. Segue o Excerto de Código 13 do método *test* na interface PQD:

```
@Override
public default boolean test(Map<String, Object> dados) {
	return executaAlgoritmo(dados);
}
```

Excerto de Código 13: Método *test* da interface PQD.

Desta forma, com o método *executaAlgoritmo*, tem-se a utilização de polimorfismo utilizando um nome do método mais identificativo da ação a executar. A interface especifica a implementação do algoritmo no método *executaAlgoritmo*, que tem de ser sobrescrito (*override*) por todas as operações de PQD para que cada uma delas execute a sua estratégia de deteção de PQD.

5.7.3.3 Operação de deteção de PQD valor em falta

A operação de deteção de valor em falta contém as propriedades comuns a todos PQD. Pode-se mesmo dizer que a nível de propriedades, o PQD valor em falta tem as propriedades de um PQD *default*. A nível de deteção a operação de valor em falta é representada pela classe *PQDValorFalta* e implementa a interface PQD. As propriedades do valor em falta são:

atributo: - coluna onde se pretende detetar este tipo de PQD;
tipo: - identificação do PQD.

Como referido na secção anterior, todo PQD tem de sobrescrever o método *executaAlgoritmo* definido na Interface PQD. O PQD valor em falta é detetado através da execução do método *executaAlgoritmo* da classe *PQDValorFalta*, como ilustra o excerto de Código 14.

```
@Override
public boolean executaAlgoritmo(Map<String, Object> map) {
	boolean retorno = true;
	for (Map.Entry<String, Object> entry : map.entrySet())
		if (entry.getKey().equals(atributo)) {
			if(entry.getValue() == null) return trataPQDVFalta(map);
			String valor = String.valueOf(entry.getValue());
			if (isBlank(valor))  return trataPQD(map);
		}
	return retorno;
}
```

Excerto de Código 14: Algoritmo de PQD valor em falta.

Apresenta-se a seguir o excerto de código do método *trataPQDVFalta* e do método *isBlank* utilizados no método do Excerto de Código 14:

```
public boolean trataPQDVFalta(Map<String, Object> map) {
	TuplosComPQD tuplesPQD = new TuplosComPQD();
	tuplesPQD.setAtributo(atributo);
	tuplesPQD.setTipoPQD("Valor em falta");
	tuplesPQD.setTuplo(map.toString());
	servicoGravarPQD.save(tuplesPQD);
	return false;
}
```

Excerto de Código 15: Método auxiliar de tratamento de PQD valor em falta.

e

```
public boolean isBlank(String s) {
	return s == null || s.isEmpty() || s.trim().isEmpty();
}
```

Excerto de Código 16: Método verificação *Null*, E*mpty* ou *Blank*.

O método *trataPQDVFalta* grava o PQD na base de dados de sistema e retorna false para o registo ser eliminado do pipeline. O método *isBlank* verifica se o valor é uma *string* nula, vazia ou se tem apenas espaços em branco. São gravados na base de dados do sistema na tabela de PQD o atributo, tipo de PQD detetado e o registo onde foi encontrado. Quando se deteta um PQD deste tipo é retornado *false* e o registo não segue no pipeline, ou seja, é eliminado do fluxo de dados.

5.7.3.4 Operação de deteção de PQD violação de domínio

A operação violação de domínio é representada pela classe *PQDViolacaoDominio*.

Esta classe contém, para além das propriedades da classe *PQDValorFalta*, as seguintes propriedades:

- *limiteInferior*;
- *limiteSuperior*;

Tal como na classe *PQDValorFalta*, é obrigatório sobrescrever o método *executaAlgoritmo*, o qual se apresenta no Excerto de Código 17.

```
@Override
public boolean executaAlgoritmo(Map<String, Object> map) {
       boolean aApagar = true;
       for (Map.Entry<String, Object> entry : map.entrySet())
         if (entry.getKey().equals(atributo)) {
              Class<?> tipo = entry.getValue().getClass();
              BigDecimal valor =
              new BigDecimal(((Number) tipo.cast(entry.getValue())).toString());
              if ( valor.compareTo(new BigDecimal(limiteInferior.toString())) < 0
                || valor.compareTo(new BigDecimal(limiteSuperior.toString())) > 0 ) {
                    TuplosComPQD tuplesPQD = new TuplosComPQD();
                    tuplesPQD.setAtributo(this.atributo);
                    tuplesPQD.setTuplo(map.toString());
                    tuplesPQD.setTipoPQD("Violacao de dominio");
                    servicoGravarPQD.save(tuplesPQD);
                    aApagar = false;
              }
         }
       return aApagar;
}
```

Excerto de Código 17: Algoritmo de violação de domínio.

Neste tipo de PQD o valor é comparado com o limite superior e limite inferior e é considerado PQD se o valor estiver fora do intervalo definido pelos limites inferior e superior. Se detetar PQD, é gravado na base de dados de sistema e é retornado *false* para que o tuplo seja eliminado do fluxo de dados.

5.7.3.5 Operação de deteção de PQD violação de sintaxe

A operação violação de sintaxe é representada pela classe *PQDViolacaoSintaxe*. Esta classe contém, para além das propriedades da classe *PQDValorFalta*, a propriedade *expressaoRegular*. Na janela *pop-up* de configuração deste tipo de PQD junto da propriedade *expressaoRegular*, existe um ícone de *tips* (dicas) que indica ao utilizador os tipos de expressões regulares que a solução suporta. Por exemplo.: "^[0-9]{4}(-[0-9]{3}){1}([ ])?[a-z\u00C0-\u00ff A-Z]*" é uma expressão regular admissível para validar se o código postal é código postal português no formato "9999-999". Apresenta-se, no Excerto de Código 18, a "sobrescrição" obrigatória do método *executaAlgoritmo*.

```
@Override
public boolean executaAlgoritmo(Map<String, Object> map) {
	boolean aApagar = true;
	for (Map.Entry<String, Object> entry : map.entrySet())
		if (entry.getKey().equals(atributo)) {
			Class<?> tipo = entry.getValue().getClass();
			String valor = (String) tipo.cast(entry.getValue());
				boolean validacao =
					valor.matches(expressaoRegular);
			if (!validacao) {
				TuplosComPQD tuplesPQD = new TuplosComPQD();
				tuplesPQD.setAtributo(this.atributo);
				tuplesPQD.setTuplo(map.toString());
				tuplesPQD.setTipoPQD("Violacao de sintaxe");
				servicoGravarPQD.save(tuplesPQD);
				aApagar = false;
			}
		}
	return aApagar;
}
```

Excerto de Código 18: Algoritmo de violação de sintaxe.

Neste tipo de PQD, utiliza-se a expressão regular para verificar se o valor do atributo é válido. Se não for válido, através da verificação com o método *match* do tipo *String*, o PQD é gravado na base de dados de sistema, e o método do algoritmo retorna *false* para que o registo seja eliminado do fluxo de dados.

5.8 Desenvolvimento do Acesso a Dados

A secção de acesso a dados divide-se em duas subsecções. Na Subsecção 5.8.1 descreve-se a implementação do acesso a dados de origem, apenas para leitura. Utiliza-se a *framework Hibernate* (tutorialspoint.com, 2020) para carregar os dados da base de dados.

Na Subsecção 5.8.2 descreve-se a implementação do acesso à base de dados do sistema para leitura e escrita de dados. A implementação utiliza o Spring Data («SPRING», 2020). *Spring Data* é uma implementação do padrão de software *Repository*, que simplifica o processo de persistência e acesso aos dados (Nagilla, 2018).

5.8.1 Base de dados de origem/fonte

Com o objetivo de manter baixo acoplamento no acesso a dados fonte, foi criada a interface *OrigemDadosRepository* para manter o processo de execução da deteção independente do tipo de dados fonte. A definição da interface *OrigemDadosRepository* corresponde à utilização do padrão *Repository*. O padrão de *Repository* é padrão de alto nível e funciona como um mecanismo de encapsulamento do acesso a dados, através de uma interface semelhante a uma coleção de objetos, através da qual se especifica forma de aceder às bases de dados para persistência, recuperação e pesquisas (Bansal, 2020) . Desta forma, o acesso a dados está desenvolvido com recursos a boas práticas, utilizando formas padronizadas de aceder a dados, facilitando a manutenção quando surgirem novos tipos de fontes de dados a integrar na solução.

A solução tem uma implementação para acesso a dados origem/fonte do tipo base de dados, através da classe *OrigemDadosRepositoryDBImpl*. A classe *OrigemDadosRepositoryDBImpl* contem uma propriedade do tipo *OrigemBaseDados*. *OrigemBaseDados* é uma implementação da interface *OrigemDados*, que representa as propriedades da base de dados de origem/fonte.

5.8.1.1 Implementação do acesso a base de dados Fonte

Para o acesso às bases de dados de origem, na implementação *OrigemDadosRepositoryDBImpl*, utiliza-se uma das implementações do JPA ("*Java Persistance API*"), o *Hibernate*, pelo que a classe *OrigemBaseDados* contém duas propriedades adicionais, para além das enumeradas no desenho da solução, a *session* e a *sessionFactory*, ambas da *framework Hibernate*. Na classe existe o método *getConfiguracaoBD* que devolve um objeto do tipo *Configuration* do *Hibernate*. Este objeto do tipo *Configuration* contém todas as propriedades necessárias para iniciar uma sessão com a base de dados. Através do método *buildSessionFactory* de uma instância do tipo *Configuration*, instancia-se a propriedade *sessionFactory* do tipo *SessionFactory*. Com o método *openSession* da propriedade *sessionFactory* obtém-se o objeto *session* do tipo *Session*, que representa a sessão com a base de dados. O Excerto de Código 19 mostra a implementação do início de sessão com a BD (base de dados) fonte.

```
public void iniciaSessao() {
		try {
			SessionFactory sessionFactory =
				getConfiguracaoBD().buildSessionFactory();
			Session session = sessionFactory.openSession();
			this.sessionFactory = sessionFactory;
			this.session = session;

		} catch (Exception ex) {
			ex.printStackTrace();
		}
}
```

Excerto de Código 19: Sessão com BD de Sistema.

A sessão criada é utilizada pelo motor de deteção de PQD para o carregamento de dados na fase de carregamento de dados para os *batchs*.

5.8.1.2 Configuração de parâmetros *Hibernate*.

É necessária uma correta configuração do *Hibernate*, para se poder iniciar sessão e aceder adequadamente às bases de dados de origem. No Excerto de Código 20 mostra-se a configuração de uma das bases de dados suportadas pela solução.

```
case MSSQLSERVER:
        url = "jdbc:sqlserver://"
                + hostname
                +":1433;databaseName="
                +"" +database;
                config.setProperty("hibernate.dialect",
                 "pt.detecao.PQDs.uteis.MySQLServerDialect");
                config.setProperty("hibernate.connection.driver_class",
                "com.microsoft.sqlserver.jdbc.SQLServerDriver");
                config.setProperty("hibernate.legacy_limit_handler",
                 "true");
                config.setProperty("hibernate.connection.url",url);
                config.setProperty("hibernate.connection.username",
                 username);
                config.setProperty("hibernate.connection.password",
                password);
        break;
```

Excerto de Código 20: Configuração de BD tipo MSSQL Server

Na configuração do tipo de base de dados MSSQL Server, como se pode verificar no Excerto de Código 20, um dos parâmetros a fornecer é o dialeto para que o *Hibernate* crie o SQL adequado a cada tipo de base de dados, e para que seja efetuado mapeamento adequado entre as classes Java e as tabelas das bases de dados (Satish Varma, 2020) . Por vezes o mapeamento de tipos nas classes Java e as tabelas das Bases de Dados não é realizado para todos os tipos de dados. Acontece com um dos tipos suportados pela solução, o tipo MSSQL Server da Microsoft. Para ultrapassar a dificuldade é preciso criar uma classe tipo *Dialect* para permitir mapear tipos em falta, pelo para o tipo MSSQL Server foi criada a classe *MySQLServerDialect*, que mapeia os tipos em falta. Apresenta-se, no Excerto de Código 21, a codificação da classe *MySQLServerDialect*.

```
public class MySQLServerDialect extends SQLServerDialect {
        public MySQLServerDialect() {
                super();
                registerHibernateType(Types.NVARCHAR,
                        StringType.INSTANCE.getName());
                registerHibernateType(Types.NCHAR,
                        StringType.INSTANCE.getName());
```

```
                registerHibernateType(microsoft.sql.Types.GEOGRAPHY,
                        StringType.INSTANCE.getName());
        }
}
```

Excerto de Código 21: *Dialect* do Tipo BD MSSQL Server.

5.8.2 Base de dados de sistema

A base de dados do sistema (base de dados de PQD), está configurada no ficheiro *application.properties*. O *SpringBoot* ao arrancar a aplicação cria uma instância da base de dados. Na implementação do acesso a dados com o Spring Data criou-se uma interface *TuplesPQDRespository* que estende a interface *JpaRepository*<*TuplosComPQD*, *Long*>. O *Spring Data* segue a especificação do padrão *Repository* (Thorben Janssen, 2019). O tipo *TuplosComPQD* é uma classe do tipo POJO ("*Plain Old Java Object*"), ou do tipo *Entity*, anotada com *@Entity*. Esta classe contém as propriedades de mapeamento da tabela de registos com PQD. Esta é a classe que representa a tabela *tuplos_com_pqd*, pelo que a classe é anotada com "@Table(name = "tuplos_com_pqd")".

Criou-se ainda uma classe *ServicoGravarPQD* que utiliza a classe do tipo *Repository TuplesPQDRespository* para gravar os PQD detetados na base de dados. A classe *ServicoGravarPQD* está anotada como *@Service*, é instanciada pelo motor de injeção de dependências do Spring, no processo de conversão de dados. No processo de conversão de dados, na fase de criação das instâncias de PQD através da *factory* adequada, o objeto do tipo *ServicoGravarPQD* instanciado é passado como parâmetro para a operação PQD instanciada. Desta forma, todas operações PQD contêm uma propriedade do tipo *ServicoGravarPQD*, pelo que quando se deteta um PQD, este é imediatamente gravado na base de dados do sistema. Na base de dados do sistema, criou-se outra interface, a interface *CorrectPQDRespository* que estende a interface *JpaRepository*<*DadosLimpos*, *Long*>. O tipo *DadosLimpos* é uma classe POJO, anotada com "`@Table(name = "dados_limpos")`", que contem as propriedades de mapeamento da tabela de dados limpos. Esta classe é utilizada pelo motor de deteção, depois da execução da deteção de todos os PQD escolhidos pelo utilizador, para gravar os dados limpos na base de dados do sistema. No momento de persistir registos com PQD (classe *TuplosComPQD*) e registos de dados limpos (classe *DadosLimpos*), em cada instância do tipo *TuplosComPQD* e *DadosLimpos* é preenchida a propriedade *workflow* correspondente, com o valor de *workflow* persistido. Desta forma é efetuada a persistência de *workflow*, PQD detetados e dados limpos de acordo com as relações definidas no modelo de dados do capítulo de desenho.

5.9 Implantação da solução

O empacotamento da solução é feito num ficheiro jar. A estrutura deste ficheiro está de acordo com a estrutura MVC do Spring. O diagrama de implantação da Figura 36 ilustra a

estrutura do conteúdo principal do ficheiro jar. Na pasta "BOOT-INF\classes\" do ficheiro jar encontra-se todo o código gerado na compilação da solução implementada. A pasta "BOOT-INF\lib\" contém todas as bibliotecas que foram utilizadas no desenvolvimento da solução. As restantes pastas do sistema são referentes à estrutura do projeto *Maven* e ao conteúdo *Springboot* para execução da solução.

Descreve-se de seguida o conteúdo da pasta "BOOT-INF\classes\" do ficheiro jar:

- Pasta *static*: - esta pasta corresponde ao conteúdo estático da camada de apresentação, ilustrado na Figura 36. Contem os ficheiros JavaScript para definir o comportamento da interface do utilizador e CSS para definir os estilos de desenho da interface do utilizador.

- Pasta templates: - esta pasta contém os ficheiros HTML da solução. Estes ficheiros contêm os componentes HTML que, conjuntamente com os ficheiros da pasta *static*, constituem a implementação da interface gráfica do utilizador.

- Pasta pt: esta pasta contém as classes java compiladas. O nome "pt" deriva do nome do package da solução que é "pt.detecao.pqd". Esta pasta contém o conteúdo da lógica da aplicação que corresponde às classes Java compiladas da camada de controlador e camada de modelo. Dentro desta pasta encontram-se ainda as classes compiladas da camada de persistência.

O ficheiro jar contem todo o código Spring para executar a solução, com o container *Tomcat* embutido. Além disso o *SpringBoot* instancia todos os módulos necessários através de um processo de auto configuração do *SpringBoot*, baseado no ficheiro *aplication.properties* que contem todas as propriedades necessárias para os módulos da solução, e nas dependências definidas no ficheiro pom.xml. São exemplo destes módulos, o container embutido que lê por exemplo a porta HTTP com que o container *tomcat* atende os pedidos HTTP; as propriedades para criar a instância da base de dados do sistema; e as propriedades JPA para mapeamento das tabelas da base de dados do sistema com as classes entidade da solução. Executar a solução é simples, bastando para isso executá-la como um jar normal. Por exemplo: comando para execução da solução empacotada no ficheiro "pqd.jar": "java -jar pqd.jar". A solução arranca e, quando tiver finalizado o carregamento de todos os módulos aparece a mensagem:

****

Aplicação web a correr na porta 9001

****

Neste caso, o ficheiro *application.properties* está configurado com a propriedade "`server.port=9001`" para arrancar o container *tomcat* (tomcat.apache.org, 2020) na porta 9001.

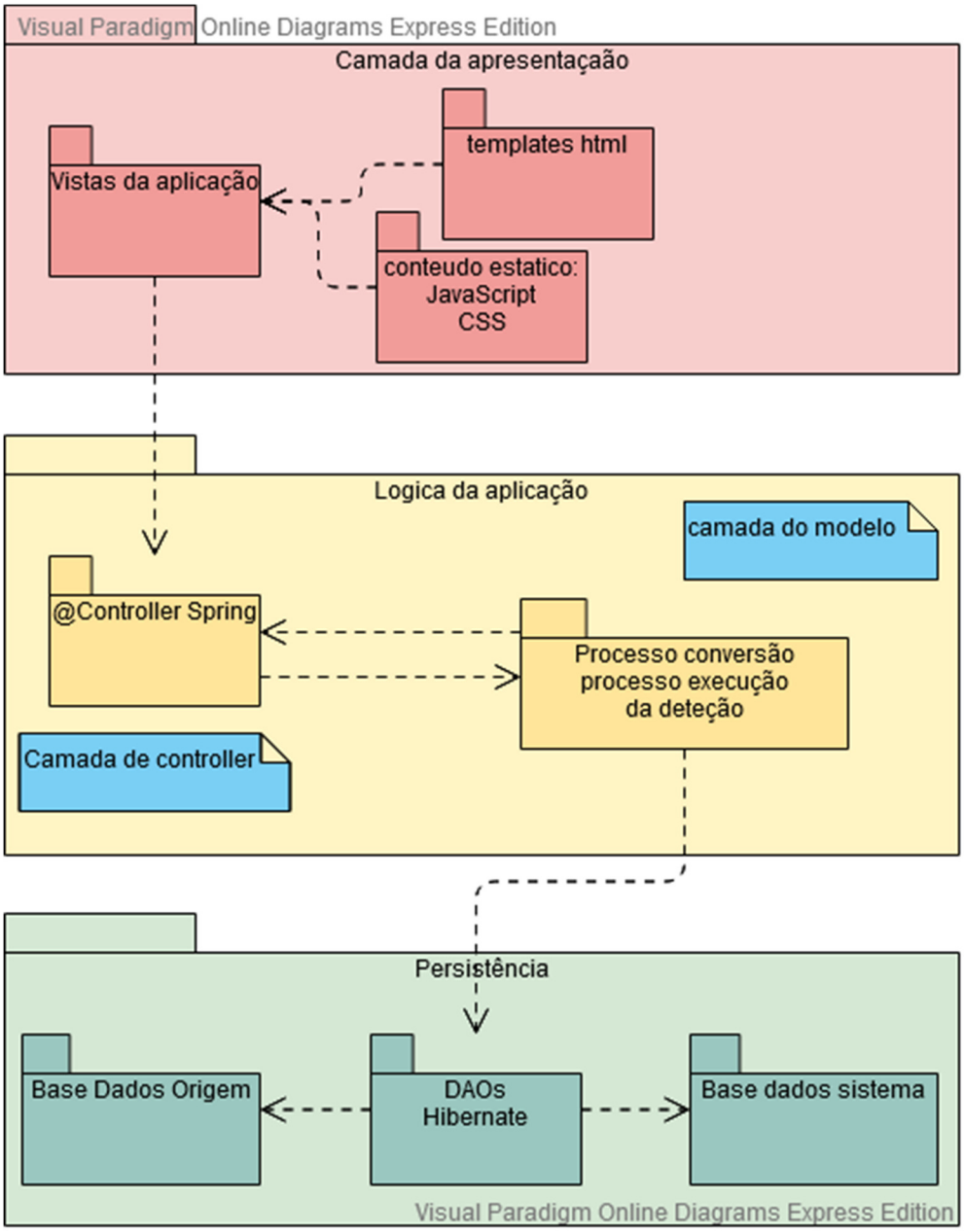


Figura 36: Diagrama de implantação.

Pela análise do público alvo, esta solução inicialmente vai ser mais utilizada em computadores pessoais, pelo que a facilidade de execução é um aspeto muito favorável. No entanto, a solução foi pensada para ser "containerizada". A solução contém um módulo de visualização que integra a consola do H2 embutido. No entanto, para se poder criar uma imagem com a solução e outra com a base de dados, basta devolver um módulo de visualização que substitua a consola H2. Em todo o caso, a solução é escalável para utilização em larga escala, devido ao facto de ser construída com tecnologia *servless*. Por exemplo: pode-se ter a solução escalável com a instalação da ferramenta *kunernetes* (kubernetes.io, 2020) num servidor local para gerir o escalonamento de container; ou num serviço de *cloud*: pode-se criar containers

baseados numa imagem de base de dados do sistema e numa imagem contendo apenas Java e um jar do empacotamento da solução. Desta forma, tanto a base de dados do sistema como a aplicação podem ser escaladas verticalmente. Este escalonamento vertical enquadra-se num contexto de ter muitos utilizadores a utilizar a solução num ponto de acesso centralizado. No entanto, a solução foi pensada para numa fase inicial ser usada em computadores pessoais.

Pode-se recorrer a um fornecedor de serviços de *cloud*. Por exemplo, a Amazon, na qual se pode configurar o RDS ("*Relational Database Service"*) («Amazon», 2020) e deixar a responsabilidade de gestão de redundâncias das bases de dados à Amazon. A solução é uma solução freeware pelo que não contempla serviços de *cloud*. No entanto, tecnicamente essa possibilidade é viável.

Outro aspeto a ter em atenção é a utilização da solução para detetar PQD em Bases de dados de muito grandes dimensões. Apesar da solução estar comtemplada com algoritmo de divisão da totalidade dos registos por *batchs*, pode ser necessário escalonamento horizontal de eventuais servidores que executem a aplicação, para assim se poder ter mais capacidade de processamento e memória RAM, de forma a evitar tempos longos de deteção de PQD.

6 Avaliação da solução

Pretende-se avaliar a solução de deteção de problemas de qualidade nos dados (PQD). Utiliza-se um conjunto de parâmetros para determinar se a solução implementada é viável. A solução é avaliada através de métricas de avaliação de desempenho, fiabilidade e usabilidade.

6.1 Metodologia de avaliação

Para se avaliar a fiabilidade, é necessário efetuar testes unitários, integração, aceitação e testes funcionais que validem que a solução tem o comportamento esperado. São definidas as métricas taxa de deteção de PQD, e tempos de resposta como parâmetros de verificação do bom desempenho da solução. Existem testes unitários que testam a lógica da deteção de PQD. Os testes de integração testam a integração com a camada de persistência e com a camada de controlo da solução. Os testes de aceitação permitem testar os casos de uso da solução, verificando que o software está pronto para ser utilizado («Teste de Aceitação», 2020). São efetuados um conjunto de testes funcionais. Estes testes permitem verificar o comportamento da solução e determinar parâmetros de desempenho como taxa média de deteção de PQD e tempo médio de resposta.

Pretende-se avaliar a usabilidade da solução. Os testes A/B são cada vez mais utilizados para detetar mudança no comportamento do utilizador provocado por alterações no sistema (Fabijan et al., 2019). Pretende-se usar testes A/B para realizar um conjunto de experiências que permitam avaliar a facilidade de utilização da ferramenta de deteção de PQD. Com estas experiências pretende-se comparar os resultados das taxas de PQD obtidos, entre um utilizador que conheça o funcionamento da ferramenta de deteção de PQD, com um utilizador que não tenha conhecimento da ferramenta. Normalmente os utilizadores são profissionais que realizam operações de ETL, ou realizam processos exploratórios de dados.

6.2 Testes Unitários

Foram realizados testes unitários às operações de deteção de PQD, que se apresentam a seguir, por classe de teste:

- `PQDValorFaltaTest`: Dados uma operação do tipo Valor em falta e uma lista com vários registos, em que apenas um deles não contém um PQD valor em Falta. Quando se executa o pipeline de deteção, passando a operação PQD valor em falta, como parâmetro do método *filter* da *Stream*, obtém-se uma lista de dados limpos. Verificar que a lista só contém um registo. Também se verifica que a lista não contém o registo com Valor em Falta no atributo com string nula. Finalmente verifica-se que o único registo da lista é um registo com dados limpos, ou seja,

sem o PQD valor em falta no atributo especificado. O teste do PQD valor em falta é mostrado no excerto de Código 22.

```
@Test
public void testarValoresEmFalta() {
	//Dados
	Workflow workflow = new Workflow();
	List<Map<String, Object>> listaIn =
		new ArrayList<Map<String, Object>>();
	Map<String, Object> map = new HashMap<String, Object>();
	Map<String, Object> mapComAtributoVazio =
		new HashMap<String, Object>();

	/* Parametros da operacao PQD valor em falta*/
	PQDValorFalta vf =
		new PQDValorFalta(1, "title", "atributo",
				servicoGravarPQD, workflow) ;

	/* Valor do atributo string vazia*/
	mapComAtributoVazio.put("atributo","");
	listaIn.add(mapComAtributoVazio);

	/* Valor do atributo string "teste"*/
	map = new HashMap<String, Object>();
	map.put("atributo","teste");
	listaIn.add(map);

	/* Valor nulo no atributo */
	map = new HashMap<String, Object>();
	map.put("atributo",null);
	listaIn.add(map);

	/* Valor do atributo com um ou mais espaços em branco*/
	map = new HashMap<String, Object>();
	map.put("atributo","       ");
	listaIn.add(map);

	//Quando
	List<Map<String, Object>> lista =
		listaIn.stream().filter(vf).collect(Collectors.toList());

	Map<String, Object> mapRegisto = new HashMap<String, Object>();
	mapRegisto=lista.get(0);
	String dadoLimpo = (String) mapRegisto.get("atributo");

	//Então verificar
	assertThat(lista, hasSize(1));
	assertThat(lista, not(hasItem(mapComAtributoVazio)));
	assertThat(dadoLimpo, is("teste"));
}
```

Excerto de Código 22: Teste Unitário de PQD valor em falta.

- `PQDViolacaoDominioTest`: Dados uma operação do tipo Violação de domínio, para um intervalo de valores admissíveis entre 30 e 60, e uma lista com registos, em que apenas um deles não contém um PQD de violação de domínio. Quando se executa o pipeline de deteção de PQD, passando a operação PQD violação de domínio como parâmetro do método *filter* da *Stream*, obtém-se uma lista de dados limpos. Verificar que: A lista só contém apenas um registo

sem violação de domínio no atributo especificado; A lista não contém o primeiro registo inserido na lista (atributo com valor 20); O único registo da lista de dados limpos, no atributo de nome "atributo" contém o valor 40, que está dentro do intervalo especificado de 30 a 60. O teste do PQD é violação de domínio é mostrado no excerto de Código 23.

```
@Test
public void testarViolacaoDominio() {
	//Dados
	Workflow workflow = new Workflow();
	List<Map<String, Object>> listaIn =
		new ArrayList<Map<String, Object>>();
	Map<String, Object> map = new HashMap<String, Object>();
	Map<String, Object> registoComViolacaoDominio =
		new HashMap<String, Object>();

	/* Parametros da operacao PQD violação de dominio*/
	PQDViolacaoDominio vd = new PQDViolacaoDominio(1,
	 "title", "atributo", 30.0 ,  60, servicoGravarPQD, workflow);

	/* Valor do atributo com violação de dominio*/
	registoComViolacaoDominio.put("atributo",20);
	listaIn.add(registoComViolacaoDominio);

	/* Valor do atributo válido para o intervalo
		de valores indicado*/
	map = new HashMap<String, Object>();
	map.put("atributo",40);
	listaIn.add(map);

	/* Valor do atributo com violação de dominio*/
	map = new HashMap<String, Object>();
	map.put("atributo",70);
	listaIn.add(map);

	//Quando
	List<Map<String, Object>> lista =
		listaIn.stream().filter(vd).collect(Collectors.toList());
	Map<String, Object> mapRegisto = new HashMap<String, Object>();
	mapRegisto=lista.get(0);
	Number dadoLimpo = (Number) mapRegisto.get("atributo");

	//Então verificar
	assertThat(lista, hasSize(1));
	assertThat(lista, not(hasItem(registoComViolacaoDominio)));
	assertThat(dadoLimpo, is(40));
	}
```

Excerto de Código 23: Teste Unitário de PQD violação de domínio.

- `PQDViolacaoSintaxeTest`: Dados uma operação do tipo violação de sintaxe, uma expressão regular de validação de valores admissíveis e uma lista com registos, em que apenas um deles não contém um PQD de violação de sintaxe. Quando se executa a deteção de PQD, passando a operação PQD violação de sintaxe como parâmetro do método *filter* da *Stream*, obtém-se uma lista de dados limpos. Verificar que: A lista só contém apenas um registo sem violação de sintaxe no atributo especificado; A lista não contém o último registo inserido na lista (atributo com valor "4000"); O único registo da lista de dados limpos, no atributo de nome

“atributo” contém o valor “4580-221”, que valida a expressão regular especificada. O teste do PQD é violação de sintaxe sendo mostrado no excerto de Código 24.

```
@Test
public void testarViolacaoDeSintaxe() {
	//Dados
	Workflow workflow = new Workflow();
	String expressaoRegular =
	 "^[0-9]{4}(-[0-9]{3}){1}([ ])?[ a-z\\u00C0-\\u00ff A-Z]*";
	List<Map<String, Object>> listaIn =
		new ArrayList<Map<String, Object>>();
	Map<String, Object> map = new HashMap<String, Object>();
	Map<String, Object> RegistoComViolacaoSintaxe = null;

	/* Parametros da operacao PQD violação de Sintaxe */
	PQDViolacaoSintaxe vs =
		new PQDViolacaoSintaxe(1, "title", "atributo",
		 expressaoRegular, servicoGravarPQD, workflow) ;

	/* Valor do atributo de acordo com a sintaxe do atributo */
	map.put("atributo","4580-221");
	listaIn.add(map);

	/* Valor do atributo com violação de sintaxe */
	RegistoComViolacaoSintaxe = new HashMap<String, Object>();
	RegistoComViolacaoSintaxe.put("atributo","1200");
	listaIn.add(RegistoComViolacaoSintaxe);

	/* Valor do atributo com violação de sintaxe */
	RegistoComViolacaoSintaxe = new HashMap<String, Object>();
	RegistoComViolacaoSintaxe.put("atributo","3500");
	listaIn.add(RegistoComViolacaoSintaxe);

	/* Valor do atributo com violação de sintaxe */
	RegistoComViolacaoSintaxe = new HashMap<String, Object>();
	RegistoComViolacaoSintaxe.put("atributo","4000");
	listaIn.add(RegistoComViolacaoSintaxe);

	//Quando
	List<Map<String, Object>> lista =
	 listaIn.stream().filter(vs).collect(Collectors.toList());
	Map<String, Object> mapRegisto = new HashMap<String, Object>();
	mapRegisto=lista.get(0);
	String dadoLimpo = (String) mapRegisto.get("atributo");

	//Então verificar
	assertThat(lista, hasSize(1));
	assertThat(lista, not(hasItem(RegistoComViolacaoSintaxe)));
	assertThat(dadoLimpo, is("4580-221"));
}
```

Excerto de Código 24: Teste Unitário de PQD violação de sintaxe.

Os testes unitários, referentes aos excertos de código 22, 23 e 24, foram codificados com a anotação *@Mock* na propriedade *servicoGravarPQD* do tipo *ServicoGravarPQD* para simular a gravação dos PQD detetados.

6.3 Testes Integração

Foram realizados testes de integração com a camada de persistência e testes de integração aos controladores da solução. Os testes da integração com a camada de persistência apresentam-se a seguir por classe de testes:

`ServicoGravarPQDTest`- Esta classe tem por finalidade testar a integração com a persistência de registos com PQD. Na classe é injetada uma dependência do tipo *ServicoGravarPQD* com a anotação *@Mock* para testar a gravação de PQD. Objetos *Mocks* servem para simular objetos reais para teste (DEVMEDIA, 2020) . Os objetos *Mocks* são injetados antes da execução do teste, como ilustra o Excerto de Código 25.

```
@Before
public void setUp() {
	MockitoAnnotations.initMocks(this);
}
```

Excerto de Código 25: Instanciação de Mocks de gravação de PQD.

Existe outro método para instanciar objetos do tipo PQD a utilizar nos testes da classe *ServicoGravarPQDTest*. Nesta classe são utilizados testes *Mockito*, que permitem verificar quando um método *Mock* é invocado com os devidos parâmetros («Tutorialspoint», 2020). A classe contém os seguintes métodos de teste:

`TesteCriar`: No Excerto de Código 26 verifica-se a persistência de um PQD, utilizando a *framework* Mockito.

```
@Test
public void TesteCriar () {
	servicoGravarPQD.save(pqd);
	Mockito.verify(servicoGravarPQD, Mockito.times(1)).save(pqd);
}
```

Excerto de Código 26: Teste de persistência de PQD.

`TesteProcurarPorId`: - Testa a pesquisa na camada de persistência de um PQD, fornecendo o Id do PQD. Segue o Excerto de Código 27 deste método.

```
@Test
public void TesteProcurarPorId () {
	Mockito.when(servicoGravarPQD.findById(1L))
		.thenReturn(Optional.of(pqd));
	assertThat(servicoGravarPQD.findById(1L), is(Optional.of(pqd)));
	Mockito.verify(servicoGravarPQD, Mockito.times(1))
		.findById(1L);
	}
```

Excerto de Código 27: Teste de método ProcuraPorId.

`TesteProcurarPorAtributo`: - Testa a pesquisa na camada de persistência de um PQD, fornecendo o nome do atributo do PQD. Segue o Excerto de Código 28 deste método.

```
@Test
public void TesteProcurarPorAtributo() {
        //Dados
        TuplosComPQD pqd2 = new TuplosComPQD();
        pqd2.setAtributo("estado");
        pqd2.setId(2L);
        servicoGravarPQD.save(pqd2);

        //Quando
        Mockito.when(servicoGravarPQD.findByAtributo("estado"))
                .thenReturn(Optional.of(pqd2));
        Optional<TuplosComPQD> found =
                servicoGravarPQD.findByAtributo("estado");

        //Entao verificar
        assertThat(found.get(), is(pqd2));
}
```

Excerto de Código 28: Teste de método ProcuraPorAtributo.

Neste método, dado um PQD com um atributo de nome estado, grava-se o PQD. Quando se procura o PQD pelo atributo de nome estado, verificar que o objeto encontrado é igual ao objeto persistido.

Os Testes de integração com os controladores da camada de controlo, tem como finalidade testar a integração com os *servlets* da camada de controlo. Descreve-se a seguir os testes ao controlador principal da solução, implementado na classe *ControladorPrincipalMVCTest*.

A classe *ControladorPrincipalMVCTest* contém diversos testes ao controlador *ControladorPrincipal*. É utilizada a anotação @InjectMocks para criar uma instância do tipo *ControladorPrincipal*, que é utilizada em vários métodos de teste da classe. A classe contém ainda a injeção de um objeto do tipo *MockMvc*, através do qual se fazem os testes aos controladores. Os métodos de testes da classe *ControladorPrincipalMVCTest* são:

`smokeTestLoad`: - Este método contém um teste de fumo para verificar que o objeto do tipo ControladorPrincipal foi instanciado.

`testHomePage`: - É Testada a navegação para a página principal de desenho do *workflow*, a partir da página html de raiz (“/”).

`testSubmit`: - Este teste corresponde ao teste do caso de uso “Execução de Workflow”. Num teste de aceitação global é este teste que é utilizado para verificar o caso de uso “Execução de Workflow”. Apresenta-se a seguir, no Excerto de Código 29 a codificação do método.

```
@Test
public void testSubmit() throws Exception {
        String pqds =
                "{\r\n" + "  \"operators\": {\r\n" +
        "    \"0\": {\r\n" +  "      \"properties\": {\r\n" +
        "        \"title\": \"MySQL\",\r\n" +  " \"inputs\": {},\r\n" +
        "        \"outputs\": {\r\n" + "  \"output_0\": {\r\n" +
        "            \"label\": \"Saida \"\r\n" +       "          }\r\n" +
        "        }\r\n" + "      },\r\n" +
        "      \"left\": 120,\r\n" + "      \"top\": 120,\r\n" +
```

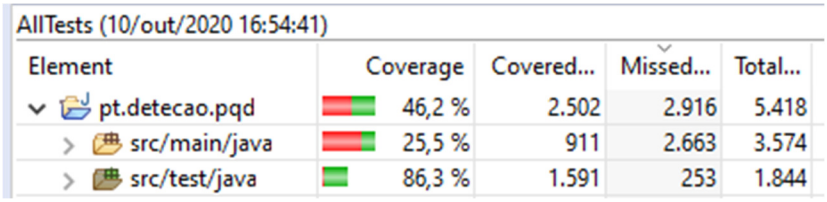
```
"      \"parametros\": {\r\n" +
"        \"IPInput\": \"localhost\",\r\n" +
"        \"database\": \"cobranca\",\r\n" +
"        \"username\": \"pqduser\",\r\n" +
"        \"password\": \"Pqd.2020\",\r\n" +
"        \"tabelanome\": \"titulo\",\r\n" +
"        \"expressaoSQL\": \"\",\r\n" +
"        \"batchsize\": \"1000\"\r\n" +
"      }\r\n" +    "    },\r\n" +
"    \"1\": {\r\n" + "      \"properties\": {\r\n" +
"        \"title\":\"VF\",\r\n" + "\"class\": \"icodf\",\r\n" +
"        \"inputs\": {\r\n" + "          \"input_0\": {\r\n" +
"            \"label\": \"Entrada \"\r\n" +"          }\r\n" +
"        },\r\n" +  "        \"outputs\": {\r\n" +
"          \"output_0\": {\r\n" +
"            \"label\": \"Saida \"\r\n" +
"          }\r\n" + "        }\r\n" + "      },\r\n" +
"      \"left\": 660,\r\n" + "      \"top\": 180,\r\n" +
"      \"parametros\": {\r\n" +
"        \"atributo\": \"status\"\r\n" +
"      }\r\n" + "    },\r\n" +
"    \"final\": {\r\n" + "      \"top\": 460,\r\n" +
"      \"left\": 1260,\r\n" + "      \"properties\": {\r\n" +
"        \"title\": \"Dados Limpos\",\r\n" +
"        \"inputs\": {\r\n" +
"          \"input_1\": {\r\n" +
"            \"label\": \"Entrada\"\r\n" +
"          }\r\n" + "        },\r\n" +
"        \"outputs\": {}\r\n" +
"      }\r\n" +  "    }\r\n" + "  },\r\n" +
"  \"links\": {\r\n" + "    \"0\": {\r\n" +
"      \"fromOperator\": 0,\r\n" +
"      \"fromConnector\": \"output_0\",\r\n" +
"      \"fromSubConnector\": 0,\r\n" +
"      \"toOperator\": 1,\r\n" +
"      \"toConnector\": \"input_0\",\r\n" +
"      \"toSubConnector\": 0\r\n" +
"    },\r\n" + "    \"1\": {\r\n" +
"      \"fromOperator\": 1,\r\n" +
"      \"fromConnector\": \"output_0\",\r\n" +
"      \"fromSubConnector\": 0,\r\n" +
"      \"toOperator\": \"final\",\r\n" +
"      \"toConnector\": \"input_1\",\r\n" +
"      \"toSubConnector\": 0\r\n" +
"    }\r\n" + "  },\r\n" +
"  \"operatorTypes\": {}\r\n" +  "}";
 ObjectMapper mapper = new ObjectMapper();
 JsonNode javaObject = null;
        try {
                javaObject = mapper.readTree(pqds);
        } catch (JsonGenerationException e) {
                e.printStackTrace();
        } catch (JsonMappingException e) {
                e.printStackTrace();
        } catch (JsonProcessingException e) {
                e.printStackTrace();
        }
        String jsonObj= javaObject.toString();

this.mockMvc.perform(post("/desenha_flowchart.html","action=
        submitlines").param("jsonPQDs",jsonObj));
}
```

Excerto de Código 29: Teste integração do caso de uso “Executar Workflow”.

É criada um objeto com dados JSON do tipo String, que corresponde a um *workflow* desenhado na camada de apresentação. Faz-se o *parser* do JSON da String para corrigir a formatação do JSON a passar para o teste. O teste é executado através do método *perform* do objeto do tipo *MockMvc*. Os dados são passados por HTTP POST a partir da página de desenho de *workflow* ("/desenha_flowchart.html"). O que é passado por parâmetros é o objeto JSON *jsonObj*, através do parâmetro HTTP *jsonPQDs*.

O caso de uso "Executar Workflow" é o único caso de uso que invoca diretamente a camada de controlo. Todos os outros casos de usos são executados com código JavaScript que cria e faz a ligação dos componentes da camada de apresentação. Para execução de teste dos restantes casos de usos torna-se necessário integrar na solução uma tecnologia de realização de testes de aceitação com JavaScript, e que permita a interligação com os testes de aceitação do Spring MVC, para assim se conseguir um teste de aceitação global dos diversos casos de uso do utilizador, relativos ao desenho e execução do *workflow*.

Foram realizados mais testes unitários e testes de integração. Não são descritos, uma vez que usam as mesmas metodologias de teste já descritas, e, para não tornar a descrição dos testes exaustiva. No entanto foi realizado a cobertura de testes, com o plugin *EclEmma* para Eclipse («EclEmma - Java Code Coverage for Eclipse», 2020), para informar da percentagem de código cobertos por testes unitários e de integração. A Figura 37 ilustra o resultado da cobertura de testes unitários e testes de integração. Apesar da informação obtida da cobertura de testes indicar que a cobertura está abaixo dos valores recomendáveis de 80%, os testes realizados cobrem os aspetos essenciais da solução.


AllTests (10/out/2020 16:54:41)

Element	Coverage	Covered...	Missed...	Total...
pt.detecao.pqd	46,2 %	2.502	2.916	5.418
src/main/java	25,5 %	911	2.663	3.574
src/test/java	86,3 %	1.591	253	1.844

Figura 37: Cobertura de testes.

6.4 Testes Funcionais

Para validação da solução, relativamente aos requisitos definidos, procedeu-se à execução da bateria de testes funcionais da Tabela 4. Foram utilizadas quatro bases de dados como origem / fonte de dados:

Cobrança: - Base de dados MySQL vazia, obtido de uma aplicação "demo" de Spring MVC (deusyvan, 2017/2020). A aplicação "demo" foi utilizada apara alimentar a base de dados que

se encontrava vazia, e, criar registos com PQD. A finalidade desta base de dados é a deteção de PQD numa base de dados MySQL.

Adventureworks: - Base de dados descarregada da Microsot (MashaMSFT, sem data) .

MaisRitmo: - Base de dados de demonstração utilizada na disciplina de ARPAD, do Mestrado de Engª Informática do ISEP.

MaisSom: Base de dados de demonstração utilizada na disciplina de ARPAD, do Mestrado de Engª Informática do ISEP

Descrevem-se a seguir os testes funcionais da Tabela 4:

Teste número 1: O teste número 1 foi executado com a base de dados de origem *AdventureWorks*, tabela *sales.SalesOrderDetail* e atributo *CarrierTrackingNumber*. Nesta fonte de dados, foi executada uma *query* SQL para determinar o número de registos com valor nulo cujo resultado aparece na coluna PQD esperados. Não se verificaram outro tipo de Valor em Falta no atributo *CarrierTrackingNumber*. Da execução do teste, resultou a deteção de 60398 PQD Valor em Falta, pelo algoritmo de iteração *IteradorBatchSized*, com *batch* de 10000 registos. Foram registados 60918 dados limpos; O mesmo teste também foi executado pelo algoritmo de iteração *IteradorBatchFixo*, tendo-se detetado 60397 PQD e registado 60919 dados limpos. As diferenças ocorridas são devido a erro da solução relacionados com o número inicial da paginação *Hibernate*. O erro foi corrigido para os testes seguintes.

Teste número 2: O teste número 2 foi executado com a base de dados de origem *AdventureWorks*, tabela *sales.SalesOrderDetail* e atributo *UnitPrice*, e com os parâmetros de violação de domínio da Figura 38. Foram detetados 2 PQD de violação de domínio com valor zero no preço unitário.

Tabela 4: Testes Funcionais.

Parâmetros / **PQD**	Número do teste	PQD esperados	PQD detetados
VF	1	60398	60397
VD	2	2	2
VS	3	2	2
VF + VS	4	2	2
VF + VD	5	3	3
VD + VD	6	2	2
VF + VD + VS	7	3	3

As siglas dos PQD da tabela 4 representam: VF – PQD de Valor em Falta; VD – PQD de Violação de Domínio; VS – PQD de Violação de Sintaxe.

Teste número 3: O teste número 3 foi executado com a base de dados de origem *MaisSom*, tabela clientes e atributo *codpostal*. No *workflow* foi definida uma operação PQD violação de sintaxe com a expressão regular: "^[0-9]{4}(-[0-9]{3}){1}([ ])?[a-z\u00C0-\u00ff A-Z]*". Foram detetados dois PQD de violação de Sintaxe: um valor estava com sintaxe errada pois continha apenas os quatro dígitos do código postal; a outra célula do atributo noutro registo, não continha código postal.

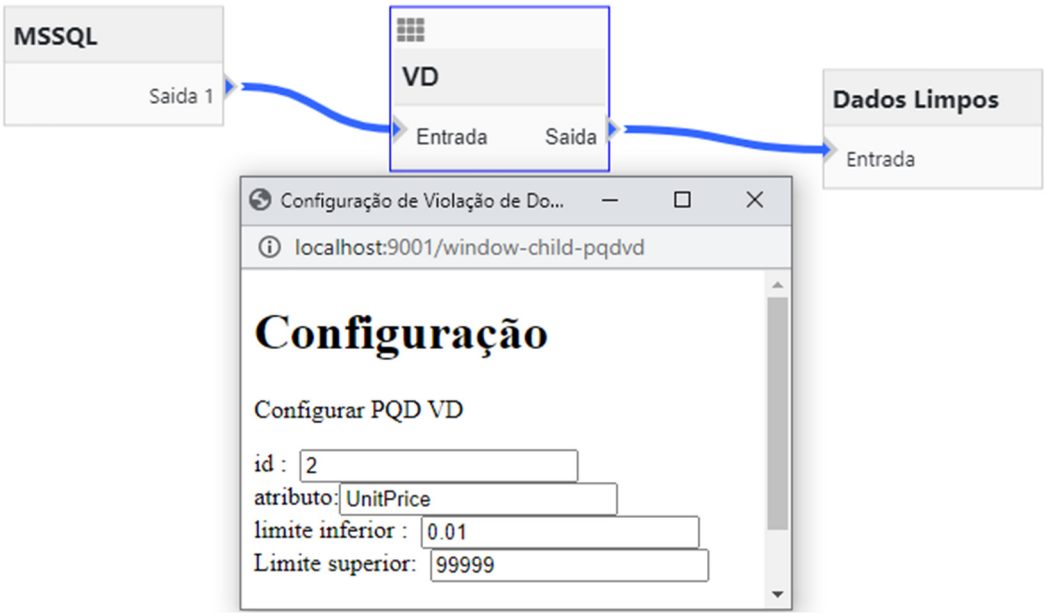


Figura 38: Parâmetros do teste funcional número 2

Teste número 4: O teste número 4 foi executado com a base de dados de origem *MaisSom*, tabela clientes e atributo *codpostal*. Neste teste foi definida a sequência da Figura 39. Ambas as operações de PQD são definidas para o atributo *codpostal*. A operação violação de sintaxe tem definida a expressão regular: "^[0-9]{4}(-[0-9]{3}){1}([ ])?[a-z\u00C0-\u00ff A-Z]*". A execução do teste resultou, pela ordem indicada pelo utilizador, na deteção de um de PQD Valor em falta, e um PQD de violação de sintaxe para o registo que continha apenas quatro dígitos no código postal. Este teste revela que a solução tem o comportamento esperado relativamente à ordem de execução da deteção dos PQD, e à correta identificação dos PQD.

Teste número 5: O teste número 5 foi executado com a base de dados de origem cobrança, tabela título e atributos status e valor. Neste teste foi definida a sequência da Figura 40. A execução do teste resultou, pela ordem indicada na Figura 40, na deteção de um PQD de valor em falta para o atributo status e dois PQD de violação de domínio para o atributo valor, em dois registos com valor inferior ao mínimo admissível (limite inferior do intervalo especificado no PQD violação de domínio), para pagamento de uma prestação do estudo de caso.

Figura 39: W*orkflow* de teste funcional número 4.

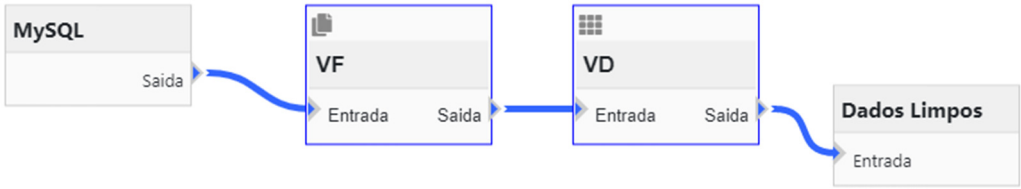


Figura 40: *Workflow* de teste funcional número 5.

Teste número 6: O teste número 6 foi executado com a base de dados de origem *MaisRitmo*, tabela produtos e atributos *precocusto* e *precovenda*. Neste teste foi definida a sequência da Figura 41. A execução do teste resultou, pela ordem indicada na Figura 41, na deteção de um de PQD de violação de domínio para o atributo *precocusto*, e um PQD de violação de domínio para o atributo *precovenda*. Ambos os PQD detetados continham valores negativos de preços de um produto.

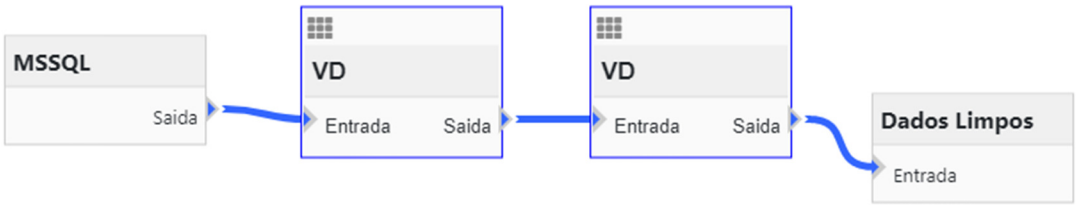


Figura 41: *Workflow* do teste funcional número 6.

Teste número 7: O teste número 7 foi executado com a base de dados de origem *MaisSom*, e na expressão SQL do Excerto de Código 30.

```
select c.codpostal, f.valortotalfinal from dbo.clientes c, dbo.documentosfacturacao f
where f.cliente = c.numero
```

Excerto de Código 30: Expressão SQL de teste número 7.

Neste teste foi definida a sequência da Figura 42, para os atributos *codpostal* e *valortotalfinal*. A execução do teste resultou, pela ordem indicada pelo utilizador, na deteção de um de PQD de valor em falta no atributo código postal, seguido da deteção de uma violação de domínio para o atributo *valortotalfinal* cujo valor encontrado é negativo, e um PQD de

violação de sintaxe para o atributo *codpostal*, num registo com o código postal com apenas 4 dígitos.

Figura 42: *Workflow* de teste funcional número 7.

6.4.1 Avaliação de testes de requisitos funcionais

Através da análise dos testes funcionais da Tabela 4, verifica-se que as funcionalidades definidas nos requisitos funcionais foram cumpridas. É possível ao utilizador parametrizar operações de PQD e origem de dados num processo de desenho de *workflow* que contém uma sequência de operações de PQD, ligadas pela ordem que o utilizador desejar. Estando o desenho do *workflow* concluído é possível ao utilizador acionar a execução da deteção, a qual ocorre pela ordem indicada pelo utilizador, permitindo assim que as dependências entre PQD sejam determinadas corretamente.

6.4.2 Avaliação de testes de requisitos não funcionais

Na avaliação dos requisitos não funcionais da solução, pretende-se avaliar a fiabilidade

e desempenho da solução através de análise estatística de um conjunto de medições dos parâmetros de taxas de PQD e tempos de resposta. Finalmente, será feita uma abordagem à avaliação da usabilidade da solução.

6.4.2.1 Desempenho e fiabilidade

A avaliação de desempenho e a fiabilidade da solução são determinados através de testes estatísticos. São realizados testes estatísticos aos parâmetros taxa de deteção de PQD e tempos de execução de operações de deteção de PQD. Desta forma é efetuada uma análise uni-variada dos valores dos parâmetros indicados. As medidas a considerar são essencialmente a média, mediana, desvio padrão, a variância, e coeficiente de variância. É prestada atenção à possibilidade de existirem *outliers* [13] que podem estar associados a problemas reais de desempenho da solução que mereçam ser analisados, para uma eventual correção. Será

[13] Outlier: - Correspondem a dados anormais que podem sugerir suspeitas relativamente à sua origem (Aggarwal, 2017)

utilizada a medida de dispersão Amplitude Inter Quartil (AIQ), para avaliar os *outliers* que possam eventualmente existir.

Realizaram-se duas experiências, com base no teste número 1 da Tabela 4, com gravação de dados limpos. Na primeira utilizou-se o algoritmo iterador de "*batchsized*" e na segunda o algoritmo padrão que utiliza número fixo de 5 *batch*. O resultado destas duas experiências apresenta-se na Figura 43 e Figura 44. O algoritmo "*batchsized*" foi utilizado com *batchs* de 10000 registos. O acréscimo de utilização de memória RAM, durante a deteção, foi pouco significativo, menos de 300MB, com um tempo de resposta de cerca de 9 segundos. A Experiência com o algoritmo padrão, o acréscimo de ocupação de memória também foi insignificante, menos de 200 MB. Realizaram-se mais duas experiências, com base no teste número 1 da Tabela 4, sem a opção de gravação de dados limpos. Na primeira utilizou-se iterador de "*batchsized*" e na segunda o algoritmo padrão que utiliza número fixo de 5 *batch*. O output destas duas experiências apresenta-se na Figura 45 e Figura 46. Importa referir que a solução em execução no estado de "ready" (pronta), ocupa 700MB de memória RAM, num computador com processador AMD RYZEN 7 e 10 GB de memória RAM. A solução durante o processo de execução da deteção não ocupou mais de 1GB de RAM, valor bem inferior aos 2GB de RAM definida nos requisitos não funcionais. Certamente que num computador com menores recursos a ocupação de memória RAM será superior, contudo o valor de memória RAM que a solução ocupa durante a deteção, revelou-se bastante satisfatório.

Com as experiências relativas às Figuras 45 e 46, a ocupação de memoria RAM foi menor do que com as experiências mostradas nas Figuras 43 e 44. O tempo de resposta é significativamente mais baixo sem gravação de dados limpos. Verifica-se melhor tempo de resposta com o algoritmo padrão, pelo que a determinação das taxas de deteção de PQD e tempos de resposta, é realizada com o algoritmo de iterações *batch* padrão. São utilizados testes estatísticos, com o algoritmo padrão e sem gravação de dados limpos. Desta forma os resultados obtidos não serão influenciados pelo processo de persistência dos dados limpos. Com o resultado destas experiências, é colocada uma informação ao utilizador, sugerindo a introdução do tamanho de *batchs* apenas para conjuntos de dados superiores a 120 000 registos. Caso o número de atributos dos conjuntos de dados seja elevado sugere-se introdução de tamanho de *batchs* a partir dos 100 000 registos.

Para determinar a taxa de deteção e tempo de resposta, apresentam-se na Tabela 5, o resultado de execução de 20 experiências do teste número 1 da Tabela 4. Da análise de resultados da Tabela 5, considerou-se que o resultado da experiência número 1 contém um valor de tempo de resposta significativamente superiror às demais experiências. O tempo excessivo de resposta está associado ao processo de arranque da solução. A cache tendo entrado em funcionamento, os tempos de resposta são mais baixos nas restantes experiências. Com base nos resultados das experiências da Tabela 5, apresenta-se na Tabela 6, o resultado de cálculo dos parâmetros estatísticos, para tempo de resposta e taxa deteção de uma operação de PQD. Com a determinação da amplitude interquartil verificam-se dois *outliers* nos dados obtidos do conjunto de 20 experiências da Tabela 5: A experiência numero 10 contém o

tempo de resposta 2872, que é inferior ao limite inferior de 2887,03, e um valor de 21,03 para a taxa de deteção que é superior ao limite superior de 20,65, apresentado na Tabela 6; Na experiência número 2, a taxa de deteção de 20,76 é superior ao limite superior de 20,65 da Tabela 6.

```
**** Execução de algoritmo IteradorBatchSized com qtd registos fixo por batch. **
Fim gravacao dados limpos no batch em: 105 milisegundos; Nr registos: 8008 registos.
Fim gravacao dados limpos no batch em: 135 milisegundos; Nr registos: 8825 registos.
Fim gravacao dados limpos no batch em: 182 milisegundos; Nr registos: 8689 registos.
Fim gravacao dados limpos no batch em: 167 milisegundos; Nr registos: 7639 registos.
Fim gravacao dados limpos no batch em: 196 milisegundos; Nr registos: 6181 registos.
Fim gravacao dados limpos no batch em: 78 milisegundos; Nr registos: 2323 registos.
Fim gravacao dados limpos no batch em: 230 milisegundos; Nr registos: 5472 registos.
Fim gravacao dados limpos no batch em: 84 milisegundos; Nr registos: 1856 registos.
Fim gravacao dados limpos no batch em: 191 milisegundos; Nr registos: 3673 registos.
Fim gravacao dados limpos no batch em: 149 milisegundos; Nr registos: 2390 registos.
Fim gravacao dados limpos no batch em: 185 milisegundos; Nr registos: 2854 registos.
Fim gravacao dados limpos no batch em: 186 milisegundos; Nr registos: 3009 registos.
Fim gravacao dados limpos no batch em: 0 milisegundos; Nr registos: 0 registos.
Verificados no total: 121317 registos
Tempo total de execucao da deteção : 9319 milisegundos
```

Figura 43: Experiência com gravação dados limpos baseados no algoritmo "*bacthSized*".

```
**** Execução de algoritmo IteradorBatchFixo com 5 batchs. **
tamanho lote : 24263
Fim gravacao dados limpos no batch em: 287 milisegundos; Nr registos: 20814 registos.
tamanho lote : 24263
Fim gravacao dados limpos no batch em: 356 milisegundos; Nr registos: 17054 registos.
tamanho lote : 24263
Fim gravacao dados limpos no batch em: 341 milisegundos; Nr registos: 9740 registos.
tamanho lote : 24263
Fim gravacao dados limpos no batch em: 307 milisegundos; Nr registos: 7403 registos.
tamanho lote : 24265
Fim gravacao dados limpos no batch em: 263 milisegundos; Nr registos: 5908 registos.
Tempo total de execucao da deteção : 5165 milisegundos
```

Figura 44: Experiência com gravação dados limpos baseados no algoritmo padrão.

```
**** Execução de algoritmo IteradorBatchSized com qtd registos fixo por batch. **
Verificados no total: 121317 registos
Tempo total de execucao da deteção : 5852 milisegundos
```

Figura 45: Experiência sem gravação dados limpos baseados no algoritmo "*bacthSized*".

```
**** Execução de algoritmo IteradorBatchFixo com 5 batchs. **
tamanho lote : 24263
tamanho lote : 24263
tamanho lote : 24263
tamanho lote : 24263
tamanho lote : 24265
Tempo total de execucao da deteção : 3561 milisegundos
```

Figura 46: Experiência sem gravação dados limpos baseados no algoritmo padrão.

Pela análise do coeficiente de variância de 6,6 % para o tempo de resposta e 9,7% para a taxa de Deteção de PQD, conclui-se que existe baixa dispersão dos valores das grandezas analisadas (tempo de resposta e Taxa de deteção de PQD) (Marcelo Rigonatto, 2020). Os outliers identificados, têm valores muito próximos dos seus limites, e que podem ser influenciados pela execução de processos no sistema operativo ou outras aplicações durante a execução das experiências e da entrada em funcionamento da cache da Memória RAM . Uma vez que existe baixa dispersão de valores de taxa de PQD e tempo de resposta, considera-se que os outliers tem pouca expressividade na dispersão de valores. (Rafael, 2016) .

Existe uma limitação na análise estatística efetuada, devido ao facto de se ter escolhido uma análise uni-variada. A taxa de deteção de PQD é dependente do tempo de resposta, pelo que seria de considerar uma análise multivariada. Contudo, como a análise estatística é realizadas com apenas duas variáveis (tempo resposta e taxa de deteção PQD), a abordagem univariada é considerada aceitável (EJE, 2019).

Obtemos assim uma taxa média de deteção de 18 PQD por milissegundo com um desvio padrão de 1,8 PQD por milissegundos que é bastante superior à taxa média 10 PQD por segundo definida nos requisitos. O tempo médio de resposta é de 3 segundos com um desvio padrão de 0,2 segundos, inferior ao tempo de resposta de 5 segundos definido nos requisitos. A ocupação da memória durante a execução da deteção de PQD foi inferior a 200 MB + 700 MB (Ocupação de memória RAM com a solução no estado "*ready*"), significativamente inferior ao requisito de 2GB. A utilização do CPU durante a execução da deteção foi inferior a 10%, muito inferior ao requisito de 70%. Com a comparação de valores obtidos através da realização das 20 experiências para determinar a taxa de deteção de PQD e tempo de resposta em comparação com os requisitos definidos, e com base na análise dos resultados dos testes funcionais realizados, considera-se que a aplicação é fiável e tem bom desempenho.

Tabela 5: Resultados de Taxas de deteção de PQD e tempos de resposta.

Experiência	*PQD detetados*	*Tempo resposta(ms)*	*Taxa de deteção*
1	60398	7197	8,392107823
2	60398	2910	20,75532646
3	60398	3260	18,52699387
4	60398	2955	20,4392555
5	60398	3622	16,6753175
6	60398	3468	17,41580161
7	60398	3408	17,72241784
8	60398	3520	17,15852273
9	60398	3554	16,99437254
10	60398	2872	21,02994429
11	60398	3029	19,93991416
12	60398	3454	17,48639259
13	60398	3295	18,33019727
14	60398	3509	17,2123112
15	60398	3303	18,28580079
16	60398	3419	17,66539924
17	60398	3452	17,49652375
18	60398	3353	18,01312258
19	60398	3183	18,97518065
20	60398	3410	17,71202346

Tabela 6: Resultados de parâmetros de testes estatísticos.

PARÂMETROS ESTATÍSTICOS	**TEMPO DE RESPOSTA (MILISSEGUNDOS)**	**TAXA DE DETEÇÃO DE PQD (PQD/MILISSEGUNDO)**
MÉDIA	3314,53	18,31
DESVIO PADRÃO	219,46	1,77
VARIÂNCIA	48164	3,13
MEDIANA	3408	17,72
PRIMEIRO QUARTIL (Q1)	3183	17,42
TERCEIRO QUARTIL(Q3)	3468	18,98
AMPLITUDE INTERQUARTIL	285	1,56
LIMITE INFERIOR	2887	15,97
LIMITE SUPERIOR	3742	20,65
COEFICIENTE DE VARIÂNCIA	6,6 %	9,7 %

6.4.2.2 Usabilidade

Para validar a usabilidade da ferramenta de detenção de PQD, recorre-se a um estimador estatístico para comparar a semelhança entre os resultados obtidos e assim concluir acerca da facilidade de utilização da ferramenta de deteção de PQD por utilizadores sem conhecimento das operações de deteção de dados.

Segundo, (Paese, Caten, & Ribeiro, 2001) ANOVA é um método que permite analisar diferenças significativas entre diferentes conjuntos de dados. Em (Blanca et al., 2017) é referido que a analise de variância de ANOVA assume que as variáveis tem distribuições normais e independentes com iguais desvios entre grupos. Outra possibilidade é utilizar um estimador estatístico baseado nas técnicas de Monte Carlo. Segundo o mesmo autor, as técnicas de Monte Carlo, incluem diversos tipos de distribuições (Blanca et al., 2017).

Em (Preacher & Selig, 2012), é referido que o método de Monte Carlo tem um desempenho comparável com outros métodos, bastante utilizados, de determinação de intervalos de confiança (Exemplo: método de BottStrapping). No entanto o método de Monte Carlo pode ser utilizado em situações em que geralmente outros métodos não podem ser utilizados. No método de Monte Carlo é gerada uma distribuição por amostragem de uma estatística baseada nas estatísticas pontuais dos seus constituintes, junto com a matriz de covariância dessas estimativas. Inclui também suposições de como os constituintes são distribuídos (Preacher & Selig, 2012).

Com um conjunto de experiências de utilização da solução por utilizadores experientes em deteção de PQD, e, utilizadores sem conhecimento das operações de deteção de PQD, a comparação de resultados permitiria concluir acerca da usabilidade da solução desenvolvida. No decurso das experiências, poder-se-ia recolher dados da taxa de deteção de PQD e tipos de PQD identificados. Em função do tipo de distribuição dos resultados, seria utilizado um estimador estatístico adequado para avaliar a semelhança de usabilidade entre o grupo de utilizadores experientes e o grupo de utilizadores não experientes na deteção de PQD.

Na utilização de um estimador estatístico, seria necessário um número bastante considerável de experiências para se conseguir aferir sobre a facilidade de utilização da solução (Lewis, 2006). Outro aspeto a considerar é a dificuldade na determinação do numero mínimo de utilizadores para se aceitar o teste de usabilidade como válido (Cazañas-Gordón, Miguel, & Parra Mora, 2017). Esta tarefa revelar-se-ia demorada. Por outro lado, no capítulo de desenho foi efetuada uma "*mockup*" da Interface gráfica com as características mais importantes identificadas no capítulo de análise de valor. A interface gráfica implementada, em comparação com a "*mockup*" do capítulo de desenho, revelou-se bastante satisfatória, permitindo boa interação com o utilizador. Com boa facilidade de visualização e alteração das propriedades dos componentes do *workflow.* A alteração é facilitada, pelo facto de ser efetuada diretamente no próprio componente, através de janela de pop-up, em vez de se navegar para outra página e deixar de visualizar o workflow desenhado, durante a alteração.

7 Conclusão

Neste capítulo descrevem-se as conclusões desta dissertação. São apresentados os objetivos alcançados, as limitações encontradas, e finaliza-se com sugestões de realização de trabalhos futuros, contextualizados com os objetivos conseguidos no decurso desta dissertação.

7.1 Objetivos alcançados

Com esta dissertação pretendeu-se desenvolver uma solução para o problema de deteção de Problemas de Qualidade nos Dados (PQD), que através de uma interface gráfica de fácil utilização, permitisse detetar PQD, considerando as sequências de tipos de PQD escolhidos pelo utilizador para desta forma permitir a sua correta identificação/correção.

O problema proposto nesta dissertação foi contextualizado com um estudo de estado da arte sobre PQD. Abordaram-se taxionomias de classificação de PQD, técnicas e ferramentas de deteção e correção de PQD existentes. Durante o estudo de estado da arte, identificaram-se técnicas e ferramentas de deteção e correção de PQD, mas não foi encontrada nenhuma solução que permitisse apenas a deteção sem obrigar à respetiva correção. O interesse numa solução que permita apenas detetar PQD, prende-se com custos, e morosidade na correção da maioria das soluções encontradas. Por outro lado, os profissionais que realizam tarefas de análise e tratamento de dados, por vezes, pretendem apenas detetar PQD, de forma rápida, para permitir tomar decisões com celeridade.

Foi efetuada a análise de valor, na qual se realça o caracter inovador de desenvolvimento da solução que permita apenas detetar PQD, de forma rápida e sem custos de aquisição. Foi elaborada uma *mockup* com as características mais importantes, identificadas na realização da análise de valor, as quais foram conseguidas na totalidade.

O Desenho da solução utilizou padrões de desenho de software adequados, que permitiram simplificar a implementação dos diversos processos que compõem a solução de deteção de PQD. A arquitetura da solução é baseada no padrão MVC, com a utilização de tecnologias bem conhecidas. Durante a fase de desenho e desenvolvimento da solução teve-se o cuidado de permitir a evolução e manutenção da mesma, quer através de padrões existentes, quer através de especificações de interligações das diversas partes, de forma a serem alteráveis.

O objetivo de deteção rápida de PQD, foi bem conseguido, recorrendo a programação de pipelines com S*treams* e expressões lambda baseadas em operações de PQD, especificadas pelo utilizador. A programação concorrente com o paralelismo das *Streams* Java, revelou-se robusta e de fácil programação. Os testes funcionais revelaram bons resultados, no desempenho e

confiabilidade da solução, através de determinação da taxa de PQD e tempo de resposta da solução.

7.2 Problemas/limitações

Apesar do desempenho bastante satisfatório da solução na deteção de PQD, não foi possível acesso a bases de dados de dimensão superior a 130 mil registos, para se proceder à deteção de PQD.

Desde o início da realização desta dissertação esteve presente a limitação de tempo no desenvolvimento de uma solução de PQD, com particularidades específicas, quer na camada de apresentação quer na camada de modelo. Na camada de apresentação, foi integrada com sucesso uma *framework* adequada, que permite incorporar design gráfico de componentes que visualmente representem operações de deteção de dados, e ligações entre os componentes que representam a sequência de deteção desejada. Além disso, a *framework* permite também arrastar e largar os componentes dos PQDs, totalizando os requisitos pretendidos para a interface gráfica. No entanto, o cumprimento deste objetivo, dificultou a opção do desenvolvimento da camada de apresentação com tecnologias de *front-end* mais atuais e completas baseadas em Node.js, que permitiriam um desenvolvimento mais estruturado, com melhor utilização de boas práticas, e melhor eficiência no processo de desenvolvimento da camada de apresentação.

A única limitação que surgiu na camada do modelo foi limitação de tempo. Foi necessário despender um tempo considerável a desenvolver a camada de apresentação, restando menos tempo para desenvolver a camada de modelo com toda a lógica da aplicação, e que permitisse tratar mais tipos de PQD. No entanto, a camada de modelo apresenta uma boa estrutura para ser facilmente expansível.

Na camada de apresentação surgiu outra limitação de tempo que impediu o desenvolvimento de um modulo de visualização de resultados adequados. Para resolver esta limitação foi incorporada a consola do H2, uma ferramenta adequada para a pesquisa de resultados da deteção de PQD através de comandos SQL.

7.3 Trabalho futuro

Para trabalho futuro sugere-se:

- Integrar a camada de apresentação com Angular através de *Gulp*, para permitir desenvolvimento da camada de apresentação com tecnologia que permita mais produtividade no processo de desenvolvimento (Wim Deblauwe, 2019) .
- Desenvolver visualização de resultados, para permitir utilizar base de dados em modo não embutido e substituir consola H2 na solução. Desta forma é possível cria uma

imagem Docker com a aplicação e outra com a base de dados, de forma a se poder escalar verticalmente a solução.

- -Tornar a solução numa aplicação completa com login, autorizações e segurança. Melhorar o tratamento de erros de forma a mostrar ao utilizador mensagem melhor entendíveis. Melhorar o CSS principalmente das janelas de *pop-up* de configuração de componentes.
- Ligação com diferentes tipo de fontes de dados (Ex: ficheiros e API Rest, outras API publicas).
- Desenvolver restantes PQD da taxionomia apresentados em (Oliveira et al., 2005) e outras taxionomias apresentadas nesta dissertação que complementem os PQD suportados pela solução.
- Permitir integrar através de serviços Rest, deteção de PQD desenvolvidas noutras tecnologias, como, por exemplo: Integrar deteção de PQD que pela sua natureza se justifique utilizar tecnologias de *machine learning*, desenvolvidas em Phyton.

Referências

Achiche, S., Appio, F. P., McAloone, T. C., & Di Minin, A. (2012). Fuzzy decision support for tools selection in the core front end activities of new product development. *Research in Engineering Design*, *24*(1), 1–18. https://doi.org/10.1007/s00163-012-0130-4

Adam Boduch. (2017). *React and React Native*. Birmingham B3 2PB, UK.: Packt Publishing.

Aggarwal, C. C. (2017). *Outlier Analysis* (Second Edition). New York: Springer. https://doi.org/10.1007/978-3-319-47578-3

Akoka, J., Berti-Équille, L., Boucelma, O., Bouzeghoub, M., Comyn-Wattiau, I., & Cosquer, M. (2007). A FRAMEWORK FOR QUALITY EVALUATION IN DATA INTEGRATION SYSTEMS: *Proceedings of the Ninth International Conference on Enterprise Information Systems*, 170–175. Funchal, Madeira, Portugal: SciTePress - Science and and Technology Publications. https://doi.org/10.5220/0002378301700175

Alfredsson, M., & Lundmark, E. (2015). *Workflow graph editing and visualization in HTML5 and Javascript* (Dissertação, Linkopings universitet). Linkopings universitet, Linkopings, Sweden. Obtido de http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-118001

Almeida, R., Maio, P., Oliveira, P., & Barroso, J. (2015). An Ontology-based Methodology for Reusing Data Cleaning Knowledge: *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 202–211. Lisbon, Portugal: SCITEPRESS - Science and and Technology Publications. https://doi.org/10.5220/0005596402020211

Almeida, W. G., Sousa, R. T., Deus, F. E., Daniel Amvame Nze, G., & Mendonça, F. L. L. (2013). Taxonomy of data quality problems in multidimensional Data Warehouse models. *2013 8th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–7.

Alonso, G., Agrawal, D., El Abbadi, A., Kamath, M., Gunthor, R., & Mohan, C. (1996). Advanced transaction models in workflow contexts. *Proceedings of the Twelfth International Conference on Data Engineering*, 574–581. https://doi.org/10.1109/ICDE.1996.492208

Alves, A. D. C. (2017). *UMA ABORDAGEM PARA DETECÇÃO DE PROBLEMAS DE QUALIDADE DOS DADOS A PARTIR DA AVALIAÇÃO DE REGRAS DE VALIDAÇÃO* (Dissertação). Universidade Federal de Pernambuco.

Andrea Passaglia. (2017). *Vue.js 2 Cookbook*. Birmingham B3 2PB, UK: Packt Publishing.

Andreas Herz. (2020). Draw2D touch [Página web]. Obtido 14 de Fevereiro de 2020, de Draw2D website: http://www.draw2d.org/draw2d/home.html

Baeldung. (2019). A Quick Struts 2 Intro [Página web]. Obtido 9 de Fevereiro de 2020, de Baeldung website: https://www.baeldung.com/struts-2-intro

Bansal, A. (2020, Setembro 10). DAO vs Repository Patterns. Obtido 5 de Outubro de 2020, de Baeldung website: https://www.baeldung.com/java-dao-vs-repository

Barateiro, J., & Galhardas, H. (2005). *A survey of data quality tools*. 7.

Bernstein, M. S. (2012). *Crowd-powered systems* (Tese, Massachusetts Institute of Technology). Massachusetts Institute of Technology. Obtido de https://dspace.mit.edu/handle/1721.1/74888

Blanca Mena, M. J., Alarcón, R., Arnau Gras, J., Bono Cabré, R., & Bendayan, R. (2017). *Non-normal data: Is ANOVA still a valid option?* Obtido de http://diposit.ub.edu/dspace/handle/2445/122126

Bostock, M. (2020). D3.js—Data-Driven Documents [Página web]. Obtido 21 de Janeiro de 2020, de D3js website: https://d3js.org/

Bosu, M. F., & MacDonell, S. G. (2013). A Taxonomy of Data Quality Challenges in Empirical Software Engineering. *2013 22nd Australian Software Engineering Conference*, 97–106. Hawthorne, Victoria, Australia: IEEE. https://doi.org/10.1109/ASWEC.2013.21

Bulajic, A., & Jovanovic, S. (2012). An Approach to Reducing Complexity in Abstract Factory Design Pattern. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(10).

Caelum. (2019). Introdução ao JSF e Primefaces—Lab. Java com Testes, JSF e Design Patterns [Apostilas Caelum]. Obtido 9 de Fevereiro de 2020, de Caelum website: https://www.caelum.com.br/apostila-java-testes-jsf-web-services-design-patterns/introducao-ao-jsf-e-primefaces/

Caelum. (2020). Interfaces gráficas com Swing—Laboratório Java com Testes, XML e Design Patterns [Apostilas Caelum]. Obtido 1 de Fevereiro de 2020, de Caelum website: https://www.caelum.com.br/apostila-java-testes-xml-design-patterns/interfaces-graficas-com-swing/

Cazañas-Gordón, A., Miguel, A., & Parra Mora, E. (2017). Estimating Sample Size for Usability Testing. *ENFOQUE UTE*, *8*, 172–185.

Center of Excellence for Professional Development. (2014, Agosto 5). Business Analyst Training in Hyderabad – COEPD. Obtido 10 de Setembro de 2020, de https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/

Chan, L.-K., & Wu, M.-L. (2002). Quality function deployment: A literature review. *European Journal of Operational Research*, *143*(3), 463–497. https://doi.org/10.1016/S0377-2217(02)00178-9

Chaturvedi, A., & Prabhakar, T. V. (2014). Ontology driven builder pattern: A plug and play component. *Proceedings of the 29th Annual ACM Symposium on Applied Computing -*

SAC '14, 1055–1057. Gyeongju, Republic of Korea: ACM Press. https://doi.org/10.1145/2554850.2555164

Christen, P. (2008). Febrl -: An open source data cleaning, deduplication and record linkage system with a graphical user interface. *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1065–1068. Las Vegas, Nevada, USA: Association for Computing Machinery. https://doi.org/10.1145/1401890.1402020

Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., & Ye, Y. (2015). KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, 1247–1261. Melbourne, Victoria, Australia: ACM Press. https://doi.org/10.1145/2723372.2749431

Conceito: Teste de Aceitação. (2020, Outubro 3). Obtido 3 de Outubro de 2020, de Teste de Aceitação website: https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/concepts/acceptance_testing_12A0F152.html?nodeId=8fe84d80

Copeland, J. L., Leffler, J. J., Parsons, V. L., & Terrill, L. J. (2009). Workflow Challenges: Does Technology Dictate Workflow? *The Serials Librarian*, *56*(1–4), 266–270. https://doi.org/10.1080/03615260802690777

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., & Tang, N. (2013). NADEEF: A commodity data cleaning system. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 541–552. New York, New York, USA: Association for Computing Machinery. https://doi.org/10.1145/2463676.2465327

David Flanagan. (2011). *JavaScript O Guia Definitivo* (6ª Edição). Porto Alegre: O'Reilly.

deusyvan. (2020). *Deusyvan/cobranca* [Java]. Obtido de https://github.com/deusyvan/cobranca (Original work published 2017)

DEVMEDIA. (2020, Outubro 4). Mocks: Introdução a Automatização de Testes com Mock Object. Obtido 4 de Outubro de 2020, de DevMedia website: https://www.devmedia.com.br/mocks-introducao-a-automatizacao-de-testes-com-mock-object/30641

docs.spring.io. (2020, Outubro 1). Spring Expression Language (SpEL). Obtido 1 de Outubro de 2020, de Spring Expression Language website: https://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html

Drouyer, S. (2020). *Jquery.flowchart* [JavaScript, Repositorio github]. Obtido de https://github.com/sdrdis/jquery.flowchart (Original work published 2016)

EclEmma—Java Code Coverage for Eclipse. (2020, Outubro 10). Obtido 10 de Outubro de 2020, de https://www.eclemma.org/

EJE. (2019, Outubro 24). Análise multivariada: O que é e qual a sua importância? Obtido 5 de Outubro de 2020, de EJE Consultoria website: http://ejeconsultoria.com.br/2019/10/24/analise-multivariada-o-que-e-e-qual-a-sua-importancia/

Erdil, N. O., & Arani, O. M. (2019). Quality function deployment: More than a design tool. *International Journal of Quality and Service Sciences*, *11*(2), 142–166. https://doi.org/10.1108/IJQSS-02-2018-0008

F. Mário Martins. (2017). *Java 8- POO+Construções Funcionais* (FCA, Lda.). FCA, Lda.

Fabijan, A., Gupchup, J., Gupta, S., Omhover, J., Qin, W., Vermeer, L., & Dmitriev, P. (2019). Diagnosing Sample Ratio Mismatch in Online Controlled Experiments: A Taxonomy and Rules of Thumb for Practitioners. *Proceedings of the 25th ACM SIGKDD International*

Conference on Knowledge Discovery & Data Mining - KDD '19, 2156–2164. Anchorage, AK, USA: ACM Press. https://doi.org/10.1145/3292500.3330722

Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., & Xin, R. (2011). CrowdDB: Answering queries with crowdsourcing. *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 61–72. Athens, Greece: Association for Computing Machinery. https://doi.org/10.1145/1989323.1989331

Freeman, J. (2019). Flow Chart Definition [Página web]. Obtido 19 de Janeiro de 2020, de Edrawsoft website: https://www.edrawsoft.com/flowchart-definition.html

Galhard, H., Florescu, D., Shasha, D., & Simon, E. (2000). An extensible Framework for Data Cleaning. *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, 312–312. San Diego, CA, USA: IEEE Comput. Soc. https://doi.org/10.1109/ICDE.2000.839429

Garcia, J. (2012). Workflow user interface patterns. Obtido 19 de Janeiro de 2020, de https://www.redalyc.org/pdf/416/41623190009.pdf

Gem Dot, Alejandro Martínez, & Antonio González. (2015). *ANALYSIS AND OPTIMIZATION OF JAVASCRIPT ENGINES*. Technical University of Catalonia (Technical University of Catalonia). Obtido de https://pdfs.semanticscholar.org/98fa/ce148cc1b2b81985cb1c9c7ebac92b6a9804.pdf

Gerlag, D. (2020). *Danielgerlag/workflow-es* [TypeScript, Repositório github]. Obtido de https://github.com/danielgerlag/workflow-es (Original work published 2016)

Gesing, S., Dooley, R., Pierce, M., Krüger, J., Grunzke, R., Herres-Pawlis, S., & Hoffmann, A. (2015). Science gateways—Leveraging modeling and simulations in HPC infrastructures

via increased usability. *2015 International Conference on High Performance Computing Simulation (HPCS)*, 19–26. https://doi.org/10.1109/HPCSim.2015.7237017

Govers, C. P. M. (1996). What and how about quality function deployment (QFD). *International Journal of Production Economics*, *46–47*, 575–585. https://doi.org/10.1016/0925-5273(95)00113-1

Gschwandtner, T., Gärtner, J., Aigner, W., & Miksch, S. (2012). A Taxonomy of Dirty Time-Oriented Data. Em G. Quirchmayr, J. Basl, I. You, L. Xu, & E. Weippl (Eds.), *Multidisciplinary Research and Practice for Information Systems* (pp. 58–72). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-32498-7_5

Guerrero García, J., Vanderdonckt, J., Calleros, J. M. G., & Winckler, M. (2008). Towards a Library of Workflow User Interface Patterns. Em T. C. N. Graham & P. Palanque (Eds.), *Interactive Systems. Design, Specification, and Verification* (pp. 96–101). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-70569-7_9

Gupta, S., Kaiser, G., Neistadt, D., & Grimm, P. (2003). DOM-based content extraction of HTML documents. *Proceedings of the 12th international conference on World Wide Web*, 207–214. Budapest, Hungary: Association for Computing Machinery. https://doi.org/10.1145/775152.775182

h2database.com. (2020, Outubro 1). H2 Database Engine. Obtido 1 de Outubro de 2020, de H2 Database Engine website: https://www.h2database.com/html/main.html

Ham, K. (2013). OpenRefine (version 2.5). Http://openrefine.org. Free, open-source tool for cleaning and transforming data. *Journal of the Medical Library Association : JMLA*, *101*(3), 233–234. https://doi.org/10.3163/1536-5050.101.3.020

Henri Bergius. (2020). Getting started with NoFlo | NoFlo [Página web]. Obtido 13 de Fevereiro de 2020, de Noflojs website: https://noflojs.org/documentation/

Hotta, K., Higo, Y., & Kusumoto, S. (2012). Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph. *2012 16th European Conference on Software Maintenance and Reengineering*, 53–62. https://doi.org/10.1109/CSMR.2012.16

IBM rational library. (2008, Julho 1). Classifying Requirements with FURPS+. Obtido 17 de Setembro de 2020, de https://www.ibm.com/developerworks/rational/library/content/04March/3073/3073_fig3.jpg

Jiang, S., & Mu, H. (2011). Design patterns in object oriented analysis and design. *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, 326–329. https://doi.org/10.1109/ICSESS.2011.5982229

jsPlumb Community. (2020). JsPlumb Toolkit Documentation [Página web]. Obtido 2 de Fevereiro de 2020, de Jsplumbtoolkit website: https://docs.jsplumbtoolkit.com/toolkit/current/

Juran, J. M., & Godfrey, A. B. (Eds.). (1999). *Juran's quality handbook* (5th ed). New York: McGraw Hill.

Kandel, S., Paepcke, A., Hellerstein, J., & Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11*, 3363. Vancouver, BC, Canada: ACM Press. https://doi.org/10.1145/1978942.1979444

Khatchadourian, R., Tang, Y., Bagherzadeh, M., & Ahmed, S. (2019). Safe Automated Refactoring for Intelligent Parallelization of Java 8 Streams. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 619–630. https://doi.org/10.1109/ICSE.2019.00072

Kietzmann, J. H. (2017). Crowdsourcing: A revised definition and introduction to new research. *Business Horizons*, *60*(2), 151–153. https://doi.org/10.1016/j.bushor.2016.10.001

Klaka, E. H., Allen, P., Malek, A., Nelson, M., Arida, P.-J., Cicos, E., & Martin, H. (2013). *United States Patent N. US8621421B2*. Obtido de https://patents.google.com/patent/US8621421B2/en

Koehnlein, J. (2015, Setembro 22). FXDiagram—Diagram Repair [Blog]. Obtido 1 de Fevereiro de 2020, de Jan's Blog website: http://koehnlein.blogspot.com/2015/09/fxdiagram-diagram-repair.html

Koen, P. A., Bertels, H. M. J., & Kleinschmidt, E. (2014). Managing the front end of innovation-part I: Results from a three-year study. *Research Technology Management*, *57*(2), 34–43. https://doi.org/10.5437/08956308X5702145

kubernetes.io. (2020, Outubro 2). Orquestração de contêineres prontos para produção [Orquestração de contêineres]. Obtido 2 de Outubro de 2020, de Kubernetes website: https://kubernetes.io/pt/

Ladd, S., Darren Davison, Steven Devijver, & Colin Yates. (2006). *Expert Spring MVC and Web Flow*. Berkeley, USA: Apress.

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, 118–127. https://doi.org/10.1109/EDOC.2001.950428

Lewis, J. R. (2006). *Sample Sizes for Usability Tests: Mostly Math, Not Magic*. 5.

Li, L. (2012). *Li—Data Quality and Data Cleaning in Database Applica.pdf* (Tese, Edinburgh Napier University). Edinburgh Napier University. Obtido de https://pdfs.semanticscholar.org/f551/918d57f83f4092e291378f567b25bc614e29.pdf

Li, L., Peng, T., & Kennedy, J. (2011). A Rule Based Taxonomy of Dirty Data. *GSTF INTERNATIONAL JOURNAL ON COMPUTING*, 9.

Liu, D., Peng, J., Wang, Y., Huang, M., He, Q., Yan, Y., … Xie, Y. (2019). Implementation of interactive three-dimensional visualization of air pollutants using WebGL. *Environmental Modelling & Software*, *114*, 188–194. https://doi.org/10.1016/j.envsoft.2019.01.019

Mak, G. (2008). Spring MVC Framework. Em G. Mak (Ed.), *Spring Recipes: A Problem-Solution Approach* (pp. 321–393). Berkeley, USA: Apress. https://doi.org/10.1007/978-1-4302-0623-1_10

Manjunath T.N, R. S., Ravindra S, & Ravikumar G.K. (2010). Analysis of Data Quality Aspects in DataWarehouse Systems. *International Journal of Computer Science and Information Technologies*, *2*. Obtido de https://www.researchgate.net/publication/230639906_Analysis_of_Data_Quality_Aspects_in_DataWarehouse_Systems

Marcelo Rigonatto. (2020, Outubro 5). Coeficiente de variação. Cálculo de coeficiente de variação. Obtido 5 de Outubro de 2020, de Mundo Educação website: https://mundoeducacao.uol.com.br/matematica/coeficiente-variacao.htm

Maria, P. (2018). *JAVASCRIPT BEYOND THE BROWSER* (Dissertação). TURKU UNIVERSITY.

martinfowler.com. (2020, Outubro 2). P of EAA: Front Controller [EAA Catalog]. Obtido 2 de Outubro de 2020, de P of EAA: Front Controller website: https://martinfowler.com/eaaCatalog/frontController.html

MashaMSFT. (sem data). AdventureWorks sample databases—SQL Server. Obtido 3 de Outubro de 2020, de https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure

Mathieu Nayrolles, Rajesh Gunasundaram, & Sridhar Rao. (2017). *Expert Angular*. Birmingham B3 2PB, UK.: Packt Publishing.

Mattioda, R. A., & Favaretto, F. (2009). Qualidade da informação em duas empresas que utilizam Data Warehouse na perspectiva do consumidor de informação: Um estudo de caso. *Gestão & Produção*, *16*(4), 645–666. https://doi.org/10.1590/S0104-530X2009000400013

maven.apache.org. (2020, Outubro 2). Maven – Welcome to Apache Maven. Obtido 2 de Outubro de 2020, de Apache Maven Project website: https://maven.apache.org/

MDN contributors. (2020). JavaScript basics [Página web]. Obtido 10 de Fevereiro de 2020, de Modzilla web docs website: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

Mei, H. T., Gray, I., & Wellings, A. (2015). Integrating Java 8 Streams with The Real-Time Specification for Java. *Proceedings of the 13th International Workshop on Java Technologies for Real-time and Embedded Systems*, 1–10. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2822304.2822314

Michele Bertoli. (2017). *React Design Patterns and Best Practices*. Birmingham B3 2PB, UK.: Packt Publishing.

Milano, D., Scannapieco, M., & Catarci, T. (2005). Using Ontologies for XML Data Cleaning. *ResearchGate*. Apresentado na Lecture Notes in Computer Science. https://doi.org/10.1007/11575863_75

MindFusion. (2020). JDiagram Programmer's Guide [Página web]. Obtido 1 de Fevereiro de 2020, de Documentação mindfusion website: https://www.mindfusion.eu/onlinehelp/jdiagram/index.htm

Mockito—Verifying Behavior—Tutorialspoint. (2020, Outubro 4). Obtido 4 de Outubro de 2020, de https://www.tutorialspoint.com/mockito/mockito_verifying_behavior.htm

Moroney, L. (2017). The Firebase Realtime Database. Em L. Moroney (Ed.), *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform* (pp. 51–71). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-2943-9_3

Nagilla, V. (2018, Maio 31). Design Patterns for JPA and Hibernate. Obtido 12 de Outubro de 2020, de Medium website: https://medium.com/@nagillavenkatesh1234/design-patterns-for-jpa-and-hibernate-aca40fb19e2c

Nick Rich, Bs. M., Matthias Holweg, Dipl.-, & Wirtschaftsing. (2000). INNOREGIO: dissemination of innovation and knowledge management techniques. *Lean Enterprise Research Centre*, 0–31.

Northwoods. (2020). JGo Overview. Obtido 1 de Fevereiro de 2020, de https://www.nwoods.com/products/jgo/

Northwoods Software. (2020). GoJS Diagrams for JavaScript and HTML, by Northwoods Software [Página web]. Obtido 21 de Janeiro de 2020, de Gojs website: https://gojs.net/latest/index.html?utm_source=modeling-languages&utm_medium=468x60&utm_term=canvas&utm_content=may2015B&utm_campaign=bsa-gojs

Nuttayasakul, N. (2003). *MathML without Plugins using VML* (Dissertação, San Jose State University). San Jose State University. Obtido de http://www.cs.sjsu.edu/~pollett/masters/Semesters/Spring02/Namon/CS298FinalReport.pdf

Olga Filipova. (2017). *Vue.js 2 and Bootstrap 4 Web Development*. Birmingham B3 2PB, UK.: Packt Publishing.

Oliveira, P. (2008). *Detecção e Correcção de Problemas de Qualidade dos Dados: Modelo, Sintaxe e Semântica* (Tese). Universidade do Minho.

Oliveira, P., Rodrigues, F., & Henriques, P. (2009). SmartClean: An Incremental Data Cleaning Tool. *2009 Ninth International Conference on Quality Software*, 452–457. https://doi.org/10.1109/QSIC.2009.67

Oliveira, P., Rodrigues, F., Henriques, P., & Galhardas, H. (2005). *A Taxonomy of Data Quality Problems*. 15.

Oliveira, S. M., & Alves, J. L. (2014). Influência Das Práticas De Inovação Aberta Na Prospecção De Conhecimentos Para a Criação De Valor Em Ambientes De Alta Complexidade Sob Condições De Incerteza E Imprevisibilidade. *Review of Administration and Innovation - RAI*, *11*(1), 295–295. https://doi.org/10.5773/rai.v11i1.1320

Openjfx comunity. (2020). JavaFX [Página web]. Obtido 1 de Fevereiro de 2020, de JavaFX website: https://openjfx.io/

Osterwalder, A., & Pigneur, Y. (2003). Modeling value propositions in e-Business. *Proceedings of the 5th international conference on Electronic commerce*, 429–436. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery. https://doi.org/10.1145/948005.948061

Paese, C., Caten, C. ten, & Ribeiro, J. L. D. (2001). Aplicação da análise de variância na implantação do CEP. *Production*, *11*(1), 17–26. https://doi.org/10.1590/S0103-65132001000100002

Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, & Nikos Karayannidis. (2000). ARKTOS: A tool for data cleaning and transformation in data warehouse environments. *IEEE Computer Society*, *23*(4).

Persson, P., & Angelsmark, O. (2015). Calvin – Merging Cloud and IoT. *Procedia Computer Science*, *52*, 210–217. https://doi.org/10.1016/j.procs.2015.05.059

Peter Koen, G. A., Scott Boyce, Allen Clamen, E. F., Stavros Fountoulakis, Albert Johnson, P. P., & Seibert, and R. (2002). Fuzzy Front End: Effective Methods, Tools, and Techniques. *stevens-tech*. Obtido de http://www.stevens-tech.edu/cce/NEW/PDFs/FuzzyFrontEnd_Old.pdf

Prajapati, M. (2019). ASP.NET MVC - GENERIC REPOSITORY PATTERN AND UNIT OF WORK. *International Journal Of All Research Writings*, *1*(1), 23–30.

Preacher, K. J., & Selig, J. P. (2012). Advantages of Monte Carlo Confidence Intervals for Indirect Effects. *Communication Methods and Measures*, *6*(2), 77–98. https://doi.org/10.1080/19312458.2012.679848

Purdy, D. (2002). *Exploring the Factory Design Pattern*. 28.

Rafael. (2016, Agosto 26). Identificação de Outliers em Conjuntos de Dados. Obtido 5 de Outubro de 2020, de Aprendendo Gestão website: https://aprendendogestao.com.br/2016/08/26/identificacao-de-outliers/

Raman, V., & Hellerstein, J. M. (2001). *Potter's Wheel: An Interactive Data Cleaning System*. 10. Roma, Itália.

Rauer, A., Walsh, G. V., McCaskey, J. P., Weissman, C. D., & Rassen, J. A. (2000). *United States Patent N. US6161103A*. Obtido de https://patents.google.com/patent/US6161103A/en

refactoring.guru. (2014). Design Patterns. Obtido 17 de Setembro de 2020, de Design Patterns website: https://refactoring.guru/design-patterns

Roper, S., Du, J., & Love, J. H. (2008). Modelling the innovation value chain. *Research Policy*, *37*(6–7), 961–977. https://doi.org/10.1016/j.respol.2008.04.005

Roy, O., & Hossain, A. (2019). An Algorithm For Rule Base Design In Data Cleaning. *International Journal of Advanced Research and Publications*, p. 4.

Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., & Edmond, D. (2005). Workflow Resource Patterns: Identification, Representation and Tool Support. Em O. Pastor & J. Falcão e Cunha (Eds.), *Advanced Information Systems Engineering* (pp. 216–232). Berlin, Heidelberg: Springer. https://doi.org/10.1007/11431855_16

Satish Varma. (2020, Março 12). What is Dialect in Hibernate and List of Dialects. Obtido 5 de Outubro de 2020, de Dialect in Hibernate website: http://javabydeveloper.com/what-is-dialect-in-hibernate-and-list-of-dialects/

Schalk, C., & Burns, E. (2010). *Java Server Faces 2.0. The Complete Reference*. New York, USA: Mc Graw Hill.

Segal, M., & Akeley, K. (1999). *The OpenGLR Graphics System: A Speci cation Version 1.2.1*. Silicon Graphics.

Shahzad, F., Sheltami, T. R., Shakshuki, E. M., & Shaikh, O. (2016). A Review of Latest Web Tools and Libraries for State-of-the-art Visualization. *Procedia Computer Science*, *98*, 100–106. https://doi.org/10.1016/j.procs.2016.09.017

Shankaranarayanan, G., & Cai, Y. (2006). Supporting data quality management in decision-making. *Decision Support Systems*, *42*(1), 302–317. https://doi.org/10.1016/j.dss.2004.12.006

Silva, M. (2018). API de WebGL: Gráficos 2D e 3D para a Web. Obtido 20 de Janeiro de 2020, de MDN Web Docs website: https://developer.mozilla.org/pt-PT/docs/Web/API/WebGL_API

Singh, R., & Singh, D. K. (2010). A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing. *International Journal of Computer Science*, p. 10.

Spring Data JPA. (2020, Outubro 2). Obtido 2 de Outubro de 2020, de https://spring.io/projects/spring-data-jpa

Sudheer Jonna, & Oleg Varaksin. (2017). *Angular UI Development with PrimeNG*. Birmingham B3 2PB, UK.: Packt Publishing.

Tago, A. (2019). *Atago0129/flowcharty* [TypeScript]. Obtido de https://github.com/atago0129/flowcharty (Original work published 2018)

Thorben Janssen. (2019, Janeiro 24). Implementing the Repository pattern with JPA and Hibernate. Obtido 5 de Outubro de 2020, de Thorben Janssen website: https://thorben-janssen.com/implementing-the-repository-pattern-with-jpa-and-hibernate/

thymeleaf.org. (2020, Outubro 2). Thymeleaf. Obtido 2 de Outubro de 2020, de https://www.thymeleaf.org/

Tim Hodkinson. (2020). JavaFX e o futuro das tecnologias Java Client [Página web]. Obtido 1 de Fevereiro de 2020, de InfoQ website: https://www.infoq.com/br/news/2018/04/JavaFXRemovedFromJDK/

tomcat.apache.org. (2020, Outubro 2). Apache Tomcat®—Welcome! Obtido 2 de Outubro de 2020, de Apache Tomcat website: http://tomcat.apache.org/

tutorialspoint.com. (2020, Outubro 2). Hibernate Tutorial—Tutorialspoint. Obtido 2 de Outubro de 2020, de Hibernate Tutorial website: https://www.tutorialspoint.com/hibernate/index.htm

Using Java Reflection. (sem data). Obtido 2 de Outubro de 2020, de https://www.oracle.com/technical-resources/articles/java/javareflection.html

VirtusaPolarisGTO/MetatronJS [JavaScript]. (2020). Virtusa - Global Technology Office. Obtido de https://github.com/VirtusaPolarisGTO/MetatronJS (Original work published 2017)

Wang, H., Li, M., Bu, Y., Li, J., Gao, H., & Zhang, J. (2016). Cleanix: A Parallel Big Data Cleaning System. *ACM SIGMOD Record*, *44*(4), 35–40. https://doi.org/10.1145/2935694.2935702

What Is Amazon Relational Database Service (Amazon RDS)? - Amazon Relational Database Service. (2020, Outubro 2). Obtido 2 de Outubro de 2020, de Amazon RDS website: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

Wim Deblauwe. (2019, Outubro 20). Spring Boot and Thymeleaf with CSS JavaScript processing using Gulp. Obtido 7 de Outubro de 2020, de https://www.wimdeblauwe.com/blog/2019/2019-10-20-spring-boot-and-thymeleaf-with-css-javascript-processing-using-gulp/

Yakout, M., Berti-Équille, L., & Elmagarmid, A. K. (2013). Don't be SCAREd: Use SCalable Automatic REpairing with maximal likelihood and bounded changes. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 553–564. New York, New York, USA: Association for Computing Machinery. https://doi.org/10.1145/2463676.2463706

yWorks. (2020). YFiles for JavaFX [Página web]. Obtido 1 de Fevereiro de 2020, de YWorks website: http://www.yworks.com/yfilesjavafx

Zafeiris, V. E., Poulias, S. H., Diamantidis, N. A., & Giakoumakis, E. A. (2017). Automated refactoring of super-class method invocations to the Template Method design pattern. *Information and Software Technology*, *82*, 19–35. https://doi.org/10.1016/j.infsof.2016.09.008

Ziegler, P. (2004). *User-Specific Semantic Integration of Heterogeneous Data: What Remains to be Done?* (p. 7) [Data Techonology Research Group]. University of Zurich.