UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE COMPUTAÇÃO

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Henrique Fernandes Rodrigues

Nickolas Gomes Pinto

ETL4FAIR: Uma evolução do ETL4LOD+ voltada para dados FAIR

RIO DE JANEIRO

2021

HENRIQUE FERNANDES RODRIGUES

NICKOLAS GOMES PINTO

ETL4FAIR: Uma evolução do ETL4LOD+ voltada para dados FAIR

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Maria Luiza Machado Campos

Co-orientador: Vânia Jesus de Araujo Soares Borges

RIO DE JANEIRO

2021

Rodrigues, Henrique Fernandes
R696e ETL4FAIR: uma evolução do ETL4LOD+ voltada para dados FAIR / Henrique Fernandes Rodrigues, Nickolas Gomes Pinto. – Rio de Janeiro, 2021.

110 f.

Orientador: Maria Luiza Machado Campos.
Coorientadora: Vânia Jesus de Araujo Soares Borges.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2021.

1. Gestão de dados. 2. Princípios FAIR. 3. Dados conectados. 4. Covid-19. 5. Vodan-BR. I. Pinto, Nickolas Gomes. II. Campos, Maria Luiza Machado (Orient.). III. Borges, Vânia Jesus de Araujo Soares (Coorient.). IV. Universidade Federal do Rio de Janeiro, Instituto de Computação. V. Título.

HENRIQUE FERNANDES RODRIGUES

NICKOLAS GOMES PINTO

ETL4FAIR: Uma evolução do ETL4LOD+ voltada para dados FAIR

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 19 de Novembro de 2021.

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
Maria Luiza Machado Campos
Data: 24/11/2021 09:57:30-0300
Verifique em https://verificador.iti.br

_____

Maria Luiza Machado Campos, Ph.D. (UFRJ)

_____

Vânia Jesus de Araujo Soares Borges, M.Sc. (UFRJ)

_____

Giseli Rabello Lopes, D.Sc (UFRJ)

_____

Carolina Felicíssimo, D.Sc. (RNP)

Às famílias e amigos, que sempre
serviram de suporte em todos os momentos de
nossas vidas.

AGRADECIMENTOS

Agradecemos à Maria Luiza, que sempre nos ajudou ao longo de toda a graduação e nos apresentou a projetos incríveis como o VODAN-BR, incentivando cada vez mais a continuar nossos estudos. À Vânia Borges que nos auxiliou da melhor forma possível ao longo de todo o desenvolvimento desse projeto, sempre compartilhando suas ideias e conhecimentos. Obrigado a todas as pessoas que participaram deste estudo e a todos os amigos que, assim como às nossas famílias, nos mantiveram no lugar e torceram por nós durante todo o curso de graduação.

‘Uma coisa que aprendi ao longo de minha vida: a nossa ciência, comparada à realidade, é primitiva e infantil – e, mesmo assim, é a coisa mais preciosa que temos’.

Albert Einstein

RESUMO

A natureza interdisciplinar e o rápido desenvolvimento da Web Semântica levaram à publicação em massa de dados na forma de triplas, utilizando a representação padrão conhecida como *Resource Description Framework* (RDF), serializada em diversos formatos amplamente aceitos. Dado esse grande crescimento, torna-se cada vez mais importante manter todo esse volume de dados acessível, usável, correto e confiável. Esta preocupação, somada a esforços para a Ciência aberta, ou seja, a abertura de dados seguindo padrões de transparência e colaboração, levaram a comunidade científica mundial, a definir um conjunto de princípios a serem seguidos para garantir a facilidade de descoberta, acesso, interoperabilidade e reuso aos dados. Tais princípios foram chamados de Princípios FAIR e estipulam uma série de processos e tratamentos, a chamada FAIRificação, que devem ser aplicados sobre todo o ciclo de vida de dados e metadados, para que se tornem FAIR. Este trabalho apresenta o ETL4FAIR, um conjunto de novas extensões e melhorias para o já existente ETL4LOD+, criado sobre a plataforma *Pentaho Data Integration*, para apoiar a publicação de dados conectados. O ETL4FAIR fornece uma interface intuitiva que permite conectar-se a várias fontes e formatos de dados. As novas extensões, empregando o framework em Java do RDF4J, inserem melhorias que promovem o acesso a repositórios de dados triplificados, a terminais SPARQL e a soluções de banco de dados RDF líderes com suporte a SPARQL 1.1 como, por exemplo, o GraphDB.

Palavras-chave: Gestão de dados; princípios FAIR; dados conectados; COVID-19; VODAN-BR.

ABSTRACT

The interdisciplinary nature and rapid development of the Semantic Web have led to the mass publication of data in the form of triples, using the standard representation known as the Resource Description Framework (RDF), serialized in several widely accepted formats. Given this huge growth, it becomes increasingly important to keep all this volume of data accessible, usable, correct and reliable. This concern, added to efforts for Open Science, that is, the opening of data following standards of transparency and collaboration, led the world scientific community to define a set of principles to be followed to ensure ease of discovery, access, interoperability and data reuse. These principles were called the FAIR Principles and stipulate a series of processes and treatments, the so-called FAIRification, which must be applied over the entire lifecycle of data and metadata, in order for them to become FAIR. This work presents ETL4FAIR, a set of new extensions and improvements to the existing ETL4LOD+, built on the Pentaho Data Integration platform, to support the publication of connected data. ETL4FAIR provides an intuitive interface that allows you to connect to various data sources and formats. The new extensions, employing the RDF4J Java framework, introduce enhancements that promote access to triplestores, SPARQL endpoints, and leading RDF database solutions with SPARQL 1.1 support such as GraphDB.

Keywords: Data management; FAIR principles; connected data; COVID-19; VODAN-BR.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API - Application Program Interface

BDR - Banco de dados Relacionais

CSV - Comma Separated Values

DW - Data Warehouse

ETL – Extract Transform Load

ETL4LOD - Extract Transform Load for Linked Open Data

ETTL - Extract Transform Transport Load

FAIR - Findable Accessible Interoperable Reusable

FAIR DP - FAIR Data Point

IA - Inteligência Artificial

JDK - Java Development Kit

JSON - JavaScript Object Notation

LOD - Linked Open Data

NoSQL - No Structured Query Language

ODBC - Open Database Connectivity

OWL - Ontology Web Language

PDI - Pentaho Data Integration

PHP - PHP Hypertext Preprocessor

RDBMS - Relational Database Management System

RDF - Resource Description Framework

RDF4J - RDF for Java

RDFS - RDF Schema

REST - Representational State Transfer

SGBD - Sistema de Gerenciamento de Banco de Dados

SGML - Standard Generalized Markup Language

SKOS - Simple Knowledge Organization System

SPARQL - SPARQL Protocol and RDF Query Language

SQL - Structured Query Language

TDB - Triplestore

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

VODAN - Virus Outbreak Network

W3C - World Wide Web Consortium

XML - eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com o surgimento da pandemia do coronavírus SARS-CoV 2, necessidades de ações emergenciais de apoio à colaboração na área científica surgiram ao redor do mundo. Dentre elas, ganhou força o movimento internacional GO FAIR[1], voltado para o compartilhamento de dados, obedecendo critérios bem definidos para disponibilização de resultados de pesquisas, de modo que pudessem ter maior reuso e maior agilidade em sua interoperabilidade e exploração. O projeto *Virus Outbreak Data Network* (VODAN[2]) surgiu como uma rede de implementação ligada à iniciativa GO FAIR, com a necessidade de trabalhar dados epidemiológicos adotando representações que possibilitassem o uso de abordagens de Inteligência Artificial para descobrir padrões significativos em surtos epidêmicos. Tal esforço se faz por meio da criação de uma rede federada de dados usando os princípios FAIR - *Findable*, *Accessible*, *Interoperable*, *Reusable* - (WILKINSON, 2017) a fim de permitir sua disponibilização e reuso em novas pesquisas, respeitando as regras de privacidade exigidas.

Os princípios FAIR enfatizam a capacidade de ação da máquina, ou seja, a habilidade dos sistemas computacionais de encontrar, acessar, interoperar e reutilizar dados com nenhuma ou mínima intervenção humana, visto que os humanos dependem cada vez mais do suporte computacional para lidar com o aumento de volume de dados, bem como com sua complexidade e velocidade de criação. Além disso, os princípios referem-se a três tipos de entidades: dados (ou qualquer objeto digital), metadados (informações sobre aquele objeto digital) e infraestrutura. Por exemplo, um dos princípios define que os metadados e os dados são registrados ou indexados em um recurso pesquisável (componente de infraestrutura), que nada mais é do que um repositório que garante a publicação desses dados e metadados, tornando-os conectados.

Para a gênese de tal estrutura mencionada, é importante ressaltar condições de formato e condicionamento dos dados que devem ser alcançadas a fim de se adequar ao padrão exigido na rede do projeto. Um desses possíveis formatos é o *Resource Description Framework* (RDF)[3], que é o padrão fundamental para a Web Semântica (CYGANIAK, uniforma WOOD, & LANTHALER, 2014). RDF é projetado para uso entre recursos

[1] https://www.go-FAIR.org/
[3] https://www.w3.org/RDF/
[2] https://www.go-FAIR.org/implementation-networks/overview/vodan/

mapeados em níveis hierárquicos em uma estrutura baseada em grafos, e agora é amplamente usado em aplicativos da Web Semântica para representar grandes bases e ontologias de dados.

Com o crescimento exponencial da informação, a Web evoluiu de uma rede somente de documentos interligados para uma em que ambos documentos e dados estão conectados. O advento dos dados conectados (Bizer, Heath, & Berners-Lee, 2009), do inglês *linked data*, e suas tecnologias subjacentes, tornam possível a reutilização e a federação de dados. Em particular, os dados representados na forma de RDF têm sido amplamente utilizados na chamada Web de Dados.

1.1 MOTIVAÇÃO

Existem inúmeras ferramentas usadas para a preparação de dados conectados, no entanto, essas ferramentas são separadas entre si, com configurações diferentes e, em certos casos, requerem a criação de um *script* dedicado ou um certo nível de programação para executar a transformação/conversão de dados e, muita das vezes, não oferecem uma capacidade de resolução de conflitos ao lidar com todo esse dado.

A fim de superar esses problemas, foi desenvolvida uma ferramenta de Extração-Transformação e Carga (ETL)[3] que permite o tratamento de dados e a transformação desses em triplas, o ETL4LOD+[4] (DA SILVA, 2018), que se trata de um conjunto de *plugins* associados a ferramenta PDI (*Pentaho Data Integration*)[5], capaz de oferecer soluções para o processamento de dados RDF com uma boa conectividade e escalabilidade. Ele contém uma variedade de *plugins*, cada um com uma funcionalidade determinada, que podem ser combinados arbitrariamente em tarefas funcionais ou específicas.

O ETL4LOD+ lidou muito bem com o ciclo de vida de dados conectados, porém, com o advento dos princípios FAIR e do projeto VODAN-BR, fez-se necessário o uso de um ferramental único com a capacidade de obter os dados, tratar e triplificar, anotando a semântica dos mesmos, assim como realizar a recuperação e triplificação dos metadados. Para atender os quesitos referentes à metadados, devem ser gerados metadados de proveniência, ou seja, o registro dos processos e entidades envolvidas na geração e distribuição de um conjunto

[3] https://en.wikipedia.org/wiki/Extract,_transform,_load
[4] https://github.com/johncurcio/ETL4LODPlus/
[5] https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho.html

de dados, sendo capaz de comprovar a origem e autenticidade desse dado, permitindo sua reprodutibilidade, bem como aqueles referentes ao *dataset* gerado, contribuindo com sua divulgação, localização e acesso. Há de se atentar que proveniência e metadados possuem relação, mas não são o mesmo, havendo a necessidade de os metadados existentes descreverem as características mencionadas anteriormente, permitindo caracterizar origem e processos aplicados a um dado (GIL, 2005). Além disso, é preciso realizar a carga e exposição desses dados e metadados nos repositórios apropriados, como: GraphDB[6] e FAIR Data Point (FAIR DP)[7], que é um sistema que permite a exposição e armazenamento de dados e metadados a respeito de *datasets* e de acordo com os princípios FAIR.

1.2 OBJETIVO

O objetivo deste trabalho é expandir o ETL4LOD+, ampliando suas funcionalidades por meio de novos *plugins* que, integrado aos originais, permitam obter, tratar, triplificar, anotar semanticamente e carregar dados e metadados em um repositório adequado. Essas funcionalidades, operando de forma integrada, promovem a publicação e disponibilização de dados e metadados na Web, contribuindo em ações estabelecidas em diferentes processos de FAIRificação. Para promover essa expansão, os elementos originais do ETL4LOD+ precisam ser revisitados e adequados, por meio da atualização de suas dependências e versões de bibliotecas, e um enriquecimento de suas documentações. Ao complementar a ferramenta com funcionalidades associadas aos princípios FAIR, que não eram trabalhadas no ETL4LOD+, passou-se a denomina-la ETL4FAIR.

Em síntese, este trabalho visa:

1. Integrar o ferramental base (ETL4LOD+) e as novas modificações para a última versão disponível do *Pentaho Data Integration*, 9.2;
2. Atualizar todas as dependências de bibliotecas e *software* para versões mais recentes;
3. Melhorar e expandir a documentação existente.
4. Adicionar novos *plugins* de conexão ao FAIR Data Point para a extração e carregamento de dados e metadados, assim como a publicação dos dados em triplas em um sistema de armazenamento de dados triplificados.

[6] https://graphdb.ontotext.com/
[7] https://www.FAIRdatapoint.org/

1.3 ESTRUTURA

Este capítulo contém uma introdução da necessidade deste trabalho, assim como seu objetivo.

No Capítulo 2, é apresentada uma revisão da literatura utilizada neste projeto, mostrando os conceitos mais importantes e explicando-os em detalhes.

O Capítulo 3 aprofunda os aspectos técnicos dos *plugins* propostos, descrevendo mais sobre as etapas iniciais do processo de implementação e até mesmo as especificações de cada um, assim como todas as modificações realizadas com base nos pontos de melhoria levantados a partir do ETL4LOD+.

O Capítulo 4 exemplifica o uso de cada *plugin* em aplicações reais, mostrando os resultados obtidos ao aplicá-los em alguns conjuntos de dados relacionados à pandemia de COVID-19, especificamente associados ao projeto VODAN-BR.

Finalmente, no Capítulo 5, uma conclusão é apresentada, incluindo as principais descobertas, dificuldades encontradas e trabalhos futuros.

2 REVISÃO DA LITERATURA

Ao longo deste trabalho foram utilizadas diversas tecnologias e conceitos, de modo que se faz necessário uma breve revisão sobre esses pontos e como tem sido suas respectivas evoluções.

2.1 WEB SEMÂNTICA

”A Web Semântica não é uma Web separada, mas uma extensão da atual. Nela a informação é dada com um significado bem definido, permitindo melhor interação entre os computadores e as pessoas”. Através desta afirmação, Berners-Lee (2001) imagina um mundo em que programas e dispositivos especializados e personalizados, chamados agentes, possam interagir por meio da infraestrutura de dados da Internet trocando informações entre si, de forma a automatizar tarefas rotineiras dos usuários. O projeto da Web Semântica, em sua essência, é a criação e implantação de padrões (*standards*) tecnológicos para apoiar este panorama, que não somente facilitem as trocas de informações entre agentes, mas principalmente estabeleça uma língua franca para o compartilhamento mais significativo de dados entre dispositivos e sistemas de informação de uma maneira geral. Nesse sentido, o *World Wide Web Consortium*[8] (W3C) define a Web Semântica como a Web de Dados Conectados, onde os usuários possuem o poder quanto à criação de repositórios de dados na Web, construção de vocabulários e escrita para a interoperabilidade desses dados, além do estabelecimento de uma conexão sobre todos esses elementos por meio de tecnologias como RDF, SPARQL, OWL, SKOS e etc.

Em suma, a Web Semântica tem por objetivo principal o de fornecer a capacidade de adição de metadados a dados na Web, ajudando na inclusão de significado a esses dados, interligando-os de tal forma que uma pessoa ou máquina consiga explorar toda a Web de Dados Conectados (Berners-Lee, 2006).

[8] https://www.w3.org/

2.1.1 *Resource Description Framework* - RDF

O *Resource Description Framework* (RDF), desenvolvido com o suporte da W3C, é uma infraestrutura que permite a codificação, troca e reuso de metadados estruturados. Essa infraestrutura possibilita a interoperabilidade de metadados por meio de mecanismos que suportam convenções comuns de semântica, sintaxe e estrutura. O RDF não estipula a semântica para cada comunidade de descrição de recursos, mas fornece a capacidade para essas comunidades definirem os elementos de metadados conforme necessário. RDF utiliza comumente o XML (*eXtensible Markup Language*) como uma sintaxe para a troca e processamento de metadados. Porém, existem outros formatos para essa sintaxe, como o JSON, Ntriple e Turtle. O RDF impõe uma estrutura que fornece a expressão inequívoca da semântica e, como tal, permite a codificação, troca e processamento por máquina consistente de metadados padronizados.

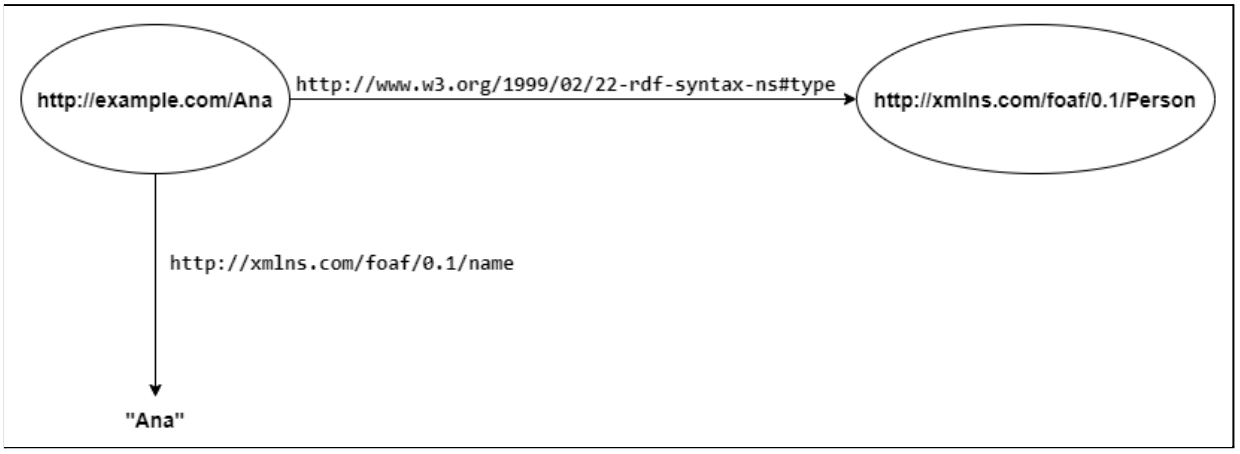


Figura 1 - Descrição do RDF.

A aplicação e o uso de dados RDF podem ser ilustrados por exemplos concretos. Considere as seguintes declarações:

→ “Este trabalho é de autoria de alguém”

→ “Alguém tem a autoria deste trabalho”

Para humanos, essas declarações possuem o mesmo significado, ou seja, este trabalho foi feito por alguém. No entanto, para máquinas, são sequências de caracteres completamente

diferentes. Enquanto humanos são extremamente hábeis em extrair significado de diferentes estruturas sintáticas, as máquinas permanecem bem atrás em comparação nesse quesito. Usando um modelo em triplas de recursos, tipos de propriedades e valores correspondentes (sujeito-predicado-objeto), o RDF tenta fornecer um método inequívoco de expressar semântica em uma codificação legível por máquina.

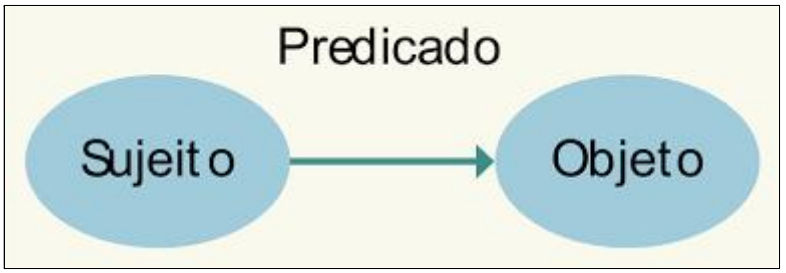


Figura 2 - Modelo RDF de formato em triplas, definindo um Grafo RDF. Fonte: https://ceweb.br/livros/dados-abertos-conectados/capitulo-2/

Uma etapa importante no ciclo de vida de um dado aberto e sua representação em RDF é justamente sua publicação e disponibilização na Web, desse modo, existem várias soluções de armazenamento RDF, como armazenamento em triplas[9], armazenamento particionado verticalmente, armazenamento de linha ou armazenamento de coluna[10]. Para um processamento de consulta eficiente em ambientes orientados à semântica, muitos armazenamentos RDF que interpretam o predicado como uma conexão entre sujeito e objeto foram desenvolvidos - por exemplo, Apache Jena - TDB[11], Amazon Neptune[12] ou AllegroGraph[13]. Os armazenamentos RDF podem ser considerados como uma subclasse de Sistema de Gerenciamento de Banco de Dados (SGBD) de grafos, embora apresentem abordagens específicas que excedem as do SGBD de grafo geral, como suporte à linguagem de consulta SPARQL[14]. SPARQL é a linguagem de consulta padrão para dados RDF.

O GraphDB é um repositório com armazenamento em triplas, semântico, com raciocínio eficiente, em formato de cluster e que oferece suporte de sincronização de índice externo, permite a vinculação de texto e dados em grandes grafos de conhecimento (Bishop et al., 2011). O GraphDB fornece funções como inserir e transformar qualquer tipo de dados em

[9] https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html
[10] https://docs.microsoft.com/pt-br/azure/architecture/best-practices/data-partitioning
[11] https://jena.apache.org/documentation/tdb/
[12] https://aws.amazon.com/cn/neptune/
[13] https://allegrograph.com/
[14] https://db-engines.com/en/article/RDF+Stores

formato RDF, reconciliação de dados, visualização de ontologias e cluster de alto desempenho, além de suportar integração com MongoDB[15]. Como resultado, o GraphDB mostra vantagens competitivas para gerenciamento de dados em grande escala, e pesquisa de similaridade semântica e, por esse motivo, é o triplestore utilizado como padrão no desenvolvimento deste trabalho.

2.1.2 *Linked Open Data* - LOD

Constituído em um conjunto de recomendações visando a publicação e estruturação de dados na Web. Trata-se de uma característica pertencente à Web Semântica, usando seus conceitos e em certos casos se mesclando com essa. Tal conceito especifica que os dados publicados na Web não permaneçam de forma estática, somente armazenados, assim utilizando do modelo de dados RDF a fim de estruturar e propiciar a interligação entre os conteúdos existentes, constituindo os dados interligados ou Linked Data.

Os Dados Abertos Interligados, ou Linked Open Data, ainda mais, trazem consigo a questão de licença aberta nesses conteúdos, visando sua exposição e acesso aberto. Por acomodar os requisitos do RDF e dados abertos, acaba por concordar com termos de formatos não proprietários. Critérios mais bem definidos foram estabelecidos em níveis, caracterizando esses conjuntos de dados:

1. Disponível na Web
2. Disponível em um formato estruturado e processável por máquina
3. Estar disponibilizado em um formato não proprietário
4. Utilizar padrões abertos da W3C (RDF e SPARQL) para identificar o conteúdo, permitindo que esse seja acessado por outros
5. Os dados disponíveis possuem interligação com outros conjuntos de dados, gerando contextos.

Alcançando o nível 5, esses dados estarão devidamente abertos e portando uma anotação semântica, além de, com essas características, estarem interligado a outros dados, formando a nuvem de dados LOD[16].

[15] https://www.mongodb.com/
[16] https://lod-cloud.net/

2.2 FRAMEWORK ETL4LOD+ DE EXTRAÇÃO TRANSFORMAÇÃO E CARGA

Extração, Transformação e Carga (ETL) é o fluxo comum pelo qual os dados de vários sistemas são combinados a um banco de dados ou outro arquivo de destino, sendo responsável por lidar com a homogeneidade do armazenamento, possíveis limpezas e problemas relativos ao carregamento de dados. Soluções ETL surgem do fato de que os dados de negócio da computação moderna residem em vários locais e em muitos formatos compatíveis. É um processo chave para reunir todos os dados em um ambiente homogêneo padrão.

Pentaho Data Integration[17] (também chamado de Kettle ou PDI) é uma plataforma ETL de código aberto líder no mercado e tem sido ligeiramente modificado para quatro elementos - ETTL, realizando: (1) extração de fontes originais de dados; (2) transporte de dados; (3) transformação do dado em um formato e/ou estrutura apropriada; e (4) carregamento dos dados na fonte de destino para um armazenamento de dados operacional ou data warehouse/data mart. Oferece um conjunto de funcionalidades baseadas em Java. É composto por certos componentes principais como o *Spoon* (uma ferramenta gráfica), *Pan* (aplicação de transformação de dados), *Chef* (ferramenta de criação de fluxos de trabalho), *Kitchen* (uma aplicação que auxilia na execução de fluxos em modo batch) e *Carte* (um servidor web para monitoração remota).

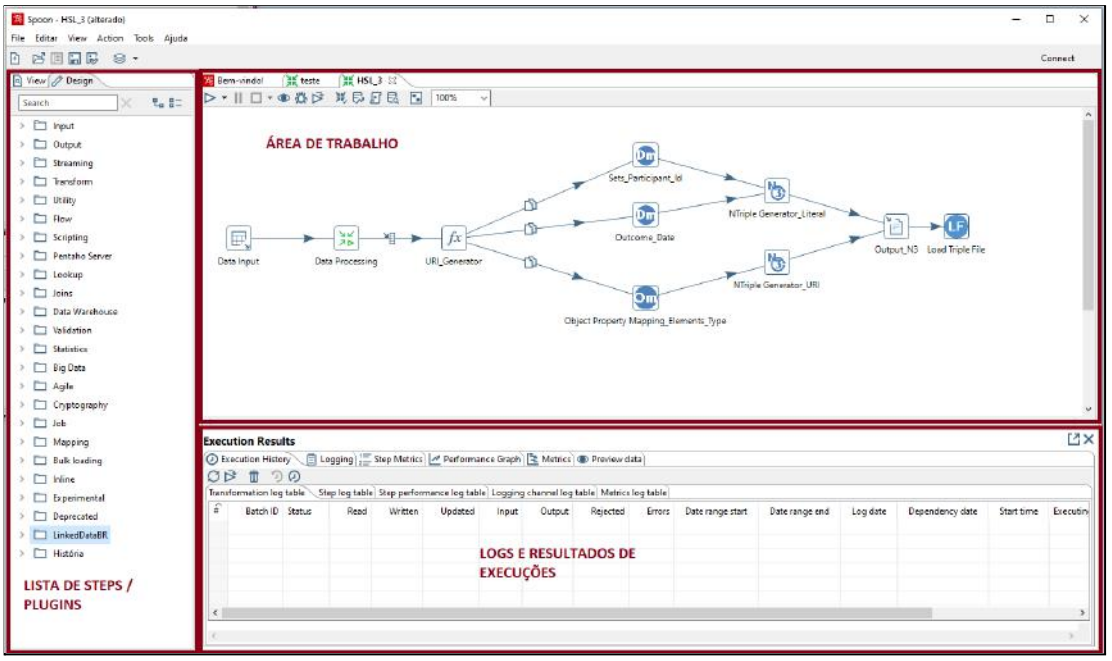


Figura 3 - Componente Gráfico do PDI (Spoon)

[17] https://etl-tools.info/en/pentaho/kettle-etl.htm

Soluções ETL do tipo não-RDF são os principais aportes para operações de ETL em bancos de dados relacionais ou outros formatos de dados (por exemplo, CSV/Excel, arquivos XML). Originalmente, no entanto, não são capazes de processar dados RDF nativamente, como trocar dados RDF entre sistemas, extrair dados RDF de SPARQL endpoints externos, transformar dados RDF de um formato para outro ou carregar dados RDF em um SPARQL *endpoint* externo ou Banco de dados em Grafos. À medida que os dados RDF ganham força, o suporte adequado para seu processamento e gerenciamento tornou-se muito importante.

Originado do esforço inicial do projeto LinkedDataBR[18], que criou o ETL4LOD (CORDEIRO et al., 2011), o ETL4LOD+ foi concebido como um conjunto de *plugins* ETL do PDI focado no processamento de dados RDF, que permite aos usuários transformar, mesclar, transferir, atualizar e compartilhar dados e empregando com máximo proveito o ecossistema da plataforma.

O ETL4LOD+ foi implementado como um conjunto de *plugins* que estendem as funcionalidades do já existente ETL4LOD, ampliando as funcionalidades do PDI para trabalhar com dados conectados.

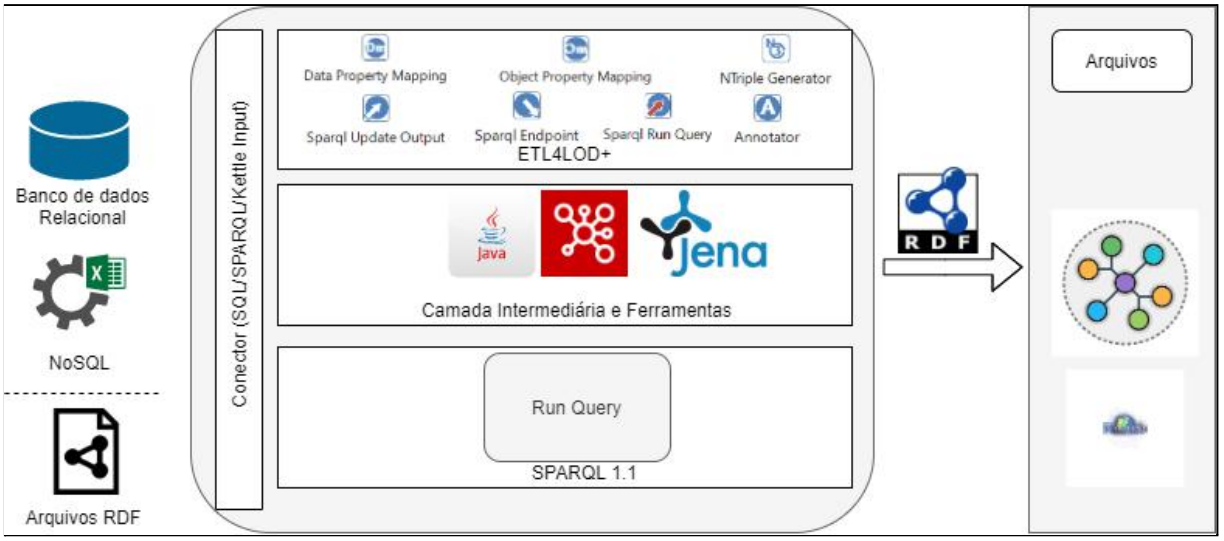


Figura 4 - ETL4LOD+ *Framework*.

A última versão do *framework* apresenta 15 *plugins* de transformação (DA SILVA, 2018) descritos no quadro 1:

[18] https://memoria.rnp.br/pd/gts2010-2011/gt_linkeddatabr.html

Quadro 1: Lista de *Plugins* do ETL4LOD+

Steps	Descrição
Annotator	Gera triplas no formato NTriple, anotadas através da entrada e um mapeamento de-para que aplica os termos da ontologia.
Data Cube	Permite gerar arquivos no formato RDF/Turtle com base no vocabulário Data Cube, fazendo uso de uma tabela obtida de arquivo ou banco. Esse passo, tal qual o vocabulário que usa, visa o trabalho com dados multidimensionais com o fim de publicar na Web.
Property Mapping	Etapa responsável pela definição dos elementos Sujeito, Predicado e Objeto, bem como a aplicação do vocabulário ao atribuir predicados ou objetos específicos, além da definição de tipagem dos elementos. Nessa etapa, os dados obtidos e formatados com suas URIs ou valores literais são encaixados no formato de triplas com elementos de ontologias e vocabulários restritos. Essa se divide em dois tipos: Data Property Mapping e Object Property Mapping.
Data Property Mapping	Na etapa Data Property Mapping, os objetos se restringem aos literais com a discriminação do tipo de valor, ou ao tipo referente ao elemento do sujeito.
Object Property Mapping	Diferentemente do Data Property, no Object Property Mapping, são relacionados somente elementos identificados com uma URI, ou seja, não literais, podendo apontar o tipo do sujeito também nessa etapa.
NTriple Generator	Etapa responsável por serializar o conteúdo no formato de triplas NTriple (n3). Com os dados de Sujeito, Predicado e Objeto definidos, bem como os possíveis dados literais com suas anotações de tipo e linguagem, essa etapa irá gerar uma saída serializada desses dados, concernente ao formato NTriple. Assim gerando uma coluna com todos os dados formatados.
Graph Semantic Level Marker	Possibilita avaliar o nível de expressividade semântica de um grafo RDF. Ao fazer isso, uma nova tripla é gerada anotando o nível avaliado. A avaliação se baseia em um arquivo .xml que estabelece as regras de análise.
Graph Sparql Query	Permite realizar querys SPARQL em um Triplestore.
Graph Triplify	Permite converter um subgrafo RDF, quebrando em três partes: sujeito, predicado e objeto.
Link Discovery Tool	Busca realizar a etapa de encontrar ligações possíveis entre os elementos através do Silk[19], porém integrado à interface do PDI, embora com restrições.

[19] http://silkframework.org/

Owl Input	Permite a busca de termos de ontologia dentro de arquivos de ontologia (owl) ou através do LOV[20]. Após a busca, os resultados dos termos são retornados.
Sparql Endpoint	Possibilita a obtenção de dados de um banco de triplas através de uma consulta SPARQL do tipo SELECT. Como resultado, são retornadas as variáveis definidas na consulta, com seus valores resultantes.
Sparql Run Query	Permite realizar consultas de atualização de um conjunto de dados (UPDATE, DELETE ou DROP), fazendo alterações nos dados do banco de triplas.
Sparql Update Output	Permite inserir e atualizar triplas em um banco de triplas, usando os dados trabalhados no PDI. Esse passo, no entanto, acaba por se limitar a bancos de triplas com o formato de autenticação definido, no caso, ao banco Virtuoso[21].
Turtle Generator	Permite a geração de triplas RDF/Turtle, através de uma entrada CSV e um mapeamento para definição das colunas que geram as triplas, bem como labels e URIs que as anotam de maneira semântica.

Fonte: (DA SILVA, 2018).

2.3 EXTENSÕES DO ETL4LOD

Com a finalidade de trabalhar sobre outros cenários decorrentes de transformações e uso de dados conectados, o Grupo de Engenharia do Conhecimento (GRECO) da Universidade Federal do Rio de Janeiro (UFRJ) vem dedicando esforços no desenvolvimento de uma série de extensões para o ETL4LOD.

2.3.1 ETL4DBPEDIA

Ngomo (2020) criou um *framework* que visa melhorar a completude dos dados na edição portuguesa da DBpedia[22], bem como a sua qualidade. DBpedia é a base de conhecimento da Web Semântica extraída da Wikipedia. DBpedia tem edições (também conhecidas como capítulos) em muitos idiomas. Este *framework*, conhecido como

[20] https://lov.linkeddata.es/dataset/lov/
[21] https://virtuoso.openlinksw.com/
[22] http://pt.dbpedia.org/

ETL4DBpedia[23], visa publicar na Wikipedia, para posterior extração para a DBpedia, os dados ausentes vindos da DBpedia, garantindo que resultarão em recursos e triplas RDF sem inconsistências. ETL4DBpedia é uma estrutura de arquitetura de duas camadas construída pela API Java do PDI e que consiste dos seguintes plugins:

- *Domain Data Transformer*: extrai, limpa e transforma um conjunto de dados de domínio;
- *Templates Maintainer*: obtém os nomes dos modelos que correspondem à string inserida pelo usuário;
- *DBpedia Mappings Maintainer*: retorna uma lista de modelos e um campo indicando se são persistentes ou não;
- *Template Selector*: obtém as classes que estão mais conectadas aos conceitos escolhidos pelo usuário;
- *Template Mapper*: permite ao usuário escolher um modelo e alinhar suas propriedades;
- *Article Checker*: verifica se o artigo em potencial já existe na Wikipédia;
- *Article Content Builder*: gera o conteúdo em potencial do artigo com os dados recebidos da etapa Template Mapper;
- *Article Publisher*: junta os dados em um formato de texto wiki e os publica na Wikipédia.

2.3.2 ETL4PROFILING

Criado por Pacheco (2020), o ETL4PROFILING[24] é um *framework* que utilizou a base estabelecida do ETL4LOD+, com uma instalação à parte e, como tal, uma extensão do PDI. Tem o papel de obter o perfil do banco de dados inserido nos plugins, principalmente DBpedia, servindo como um complemento ao já mencionado ETL4DBpedia e que consiste dos seguintes plugins:

- *Get DBpedia Data*: extrai e gerencia dados da DBpedia, encontrando o perfil de sua estrutura;

[23] https://github.com/JeanGabrielNguemaN/ETL4DBpedia
[24] https://github.com/ingridpacheco/ETL4Profiling

- *Template Property Analyser*: verifica se as propriedades de um template estão sendo utilizadas pelos seus recursos;
- *Template Resource Analyzer*: verifica a integridade de cada recurso de template;
- *Resource Properties Analyzer*: encontra a completude de um recurso individual;
- *Property Analyzer*: estuda a presença de uma propriedade em todos os recursos do template;
- *Template Resource Input Analyzer*: compara os recursos que recebe com os recursos encontrados em um modelo da DBpedia.

2.3.3 ETL4LinkedProv

Ainda, como contribuição importante ao processo de tratamento de dados em RDF, o *ETL4LinkedProv* foi desenvolvido como parte de uma dissertação de mestrado, criado por Rogers Mendonça (MENDONÇA, 2016). Nesse trabalho, a questão da proveniência e metadados associados são abordados no contexto de transformações de dados em *workflows* de triplificação. Com isso, há o intuito de trazer maior facilidade na sua obtenção e no tratamento desse tipo de dados.

O ETL4*LinkedProv* é uma abordagem, que se baseia na captura máxima dos dados de proveniência, realizando um monitoramento de todo o processo de transformação do dado em triplas. Também é definida uma estrutura semântica composta de 4 camadas para a descrição dos metadados do processo, cada uma com uma ontologia associada: PROV-O[25], OPMW[26], Cogs[27] e Ontologia de Aplicação. Após a estruturação e anotação com essas ontologias, a abordagem prevê a disponibilização dos dados de proveniência por SPARQL *Endpoint*.

A fim de concretizar o objetivo proposto, foi criado o chamado Agente Coletor de Proveniência. Um *plugin* do PDI responsável por capturar, interligar e armazenar

[25] https://www.w3.org/TR/prov-o/
[26] https://www.opmw.org/model/OPMW/
[27] https://databus.dbpedia.org/ontologies/vocab.deri.ie/cogs/2020.06.10-212236

temporariamente os dados de proveniência. Os quais devem ser posteriormente submetidos à triplificação através de outros passos do PDI.

2.4 FAIR

Na atualidade, com a imensa quantidade de dados disponíveis, tem-se uma série de problemas e questões derivadas do grande volume de processamento e trabalho com essa massa de dados.

Ao empenhar esforços em certos conjuntos de dados, grupos públicos e privados realizam investimentos financeiros e de tempo com o propósito de obter novas informações derivadas desses dados, sempre armazenando-os para uso futuro. Porém, todo esse processo necessita ser agilizado, a fim de obter um bom reuso e facilidade na compreensão do horizonte de dados, buscando, assim, uma maior economia de recursos e produtividade. Além desses, há outros problemas envolvidos, como a falta de padronização nos formatos e os direitos de uso, seja quanto ao dado ou quanto a um *software* proprietário. Tais problemas constituem grandes desafios para a eScience[28].

Em 2014, no Lorentz Centre, localizado na cidade de Leiden, Holanda, houve um debate sobre como realizar melhorias no contexto de dados para ciência (WILKINSON,2017). Da reunião, foi destacada a importância de um conjunto mínimo de princípios que fosse acordado pela comunidade científica mundial que viabilizasse, assim, possibilitar fácil descoberta, acesso, interoperabilidade e reuso aos dados. Desse modo, foram definidos os princípios que guiam o FAIR e são sua base (HENNING,2019):

- **Localizável (*Findable*)**: Atribuição de uma identificação única e global para os dados, possibilitando sua indexação e, assim, descoberta tanto por humanos quanto por máquinas.
- **Acessível (*Accessible*)**: Definição de um mecanismo para acesso aos dados e metadados, tornando-os disponíveis, estabelecendo um protocolo livre e, também, expondo com clareza as permissões sobre o acesso e uso. Ainda é

[28] https://pt.wikipedia.org/wiki/E-Science

ressaltada a importância da exposição e persistência dos metadados, mesmo não havendo a disponibilidade dos dados.

- **Interoperável (*Interoperable*)**: Definição de uso de padrões para os dados e suas descrições (metadados), sendo representados por vocabulários e ontologias, a fim de poderem ser associados facilmente e de forma automática.
- **Reusável (*Reusable*)**: Definição de facilidades de reuso através da boa descrição dos dados, que devem cumprir as características dos outros tópicos e estar bem anotados, compreendendo seu contexto e apresentando ricos metadados que permitam descrever bem o conteúdo e também retratar a proveniência. Assim o processo de reutilização e combinação com diversos dados de outras fontes torna-se fácil, sendo também automatizada.

2.4.1 FAIR Data Point

A fim de atender às demandas dos princípios FAIR, surge o esforço de criação de uma infraestrutura para expor e armazenar os dados e metadados a respeito de *datasets*, seguindo os mencionados princípios. Isso gerou o chamado FAIR Data Point (FAIRDP), um sistema que permite esses procedimentos sendo composto de três componentes: API do FAIR Data Point, serviço da API e Cliente (FAIR Data Point, 2020).

O primeiro componente se baseia nos padrões de metadados e dados interligados, definindo os preceitos e a estrutura da API por meio do uso. O componente utiliza uma tecnologia REST, possibilitando interoperabilidade e implementações, já que respeita padrões de serviço Web REST, podendo outros meios realizarem requisições para armazenamento ou recuperação dos dados (Red Hat, 2020).

O segundo componente, se refere ao serviço que implementa a API. Ele possibilita os processos mencionados e planejados nas definições. Traz meios de autenticação para controle da inserção e alteração dos dados, já que o acesso do tipo somente leitura é público.

O terceiro componente que constitui o sistema, é o cliente da API. Esse traz uma interface web que permite o registro de metadados através do preenchimento no editor disponibilizado.

O *software*, assim, permite um meio para a exposição de dados e metadados, respeitando os princípios FAIR. Com isso, os dados contíguos nesse repositório, são

estruturados segundo padrões de metadados semânticos, como o DCAT[29] e Dublin Core[30]. Essas ontologias são voltadas para a catalogação de dados e permitem estruturar os conjuntos de dados que forem gerados e armazenados no FAIRDP.

Os dados obtidos são de livre acesso em questão de segurança, mas o armazenamento só ocorre mediante a autenticação e estão estruturados segundo os subgrupos de *Catalog*, *Dataset* e *Distribution*. Esses subgrupos definem o tipo do elemento FAIR armazenado e são registrados como instâncias dos *shapes* no sistema. Além disso, ressalta-se a anotação semântica dos dados que respeita a estrutura apontada na tabela presente nos anexos A, B, C e D[31]. Destacando que compreende a estrutura básica, podendo haver a criação de outros tipos de elementos no FAIR Data Point.

2.4.2 Passos para o FAIR

Uma vez especificados os princípios FAIR, se faz necessária a definição de um procedimento, no qual qualquer conjunto de dados deve passar a fim de se tornar FAIR. Esse processo é conhecido como FAIRificação. Tal ação vem sendo analisada cada vez mais e novos padrões estão surgindo, de modo que uma importante proposta para se alcançar tal nível de dados é a idealização de um Fluxo de Trabalho (*Workflow*) genérico.

[29] https://www.w3.org/TR/vocab-dcat-2/
[30] https://dublincore.org/
[31] https://github.com/FAIRDataTeam/FAIRDataPoint-Spec

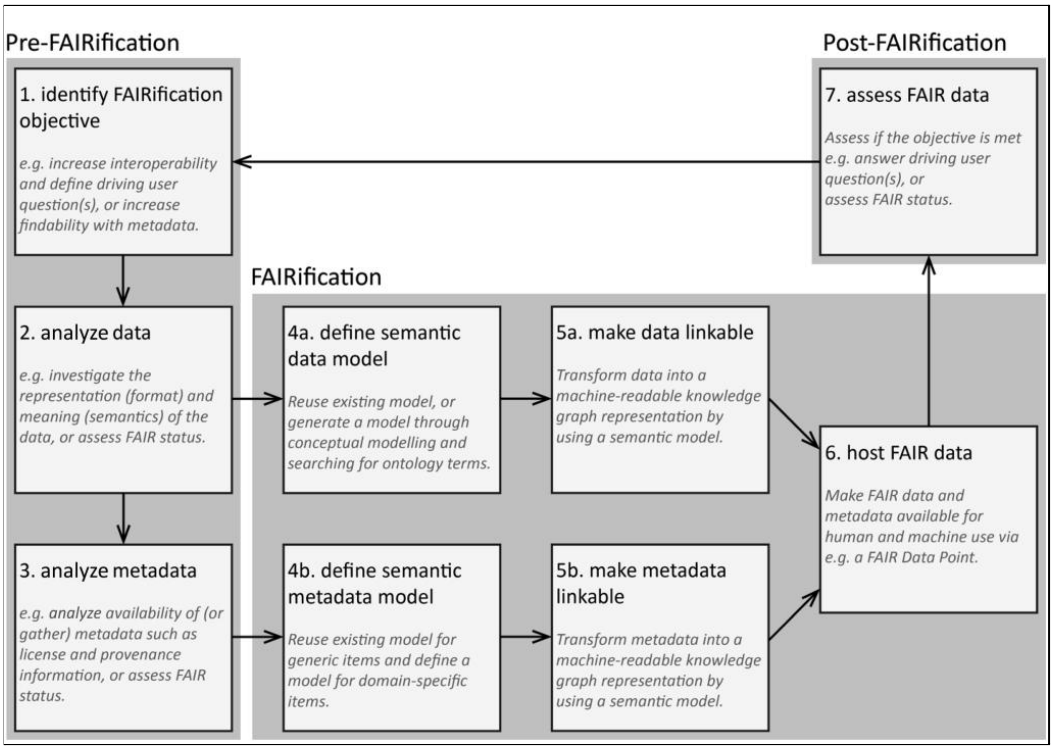


Figura 5 - Diagrama de *workflow* genérico de FAIRificação. Fonte: JACOBSEN, 2020

De forma geral, o fluxo se estabelece em três principais partes: Pré-FAIRificação, FAIRificação e Pós-FAIRificação.

2.4.2.1 Pré-FAIRificação

Na primeira, o processo depende principalmente do agente de FAIRificação, ou seja, dos envolvidos responsáveis pelo processo, de modo que o centro de ação se dá pela análise inicial dos dados. Assim, há a análise do nível FAIR dos dados, de sua estrutura, definindo objetivos finais, construindo questões de competência, bem como fazendo tal avaliação em relação aos metadados associados a esses dados.

Nessa primeira etapa, também é realizado um esforço inicial sobre a proveniência, importante para a divulgação final e reuso. Dados como fonte do *dataset*, seus metadados iniciais e seu contexto, devem ser mantidos e bem descritos para momentos posteriores, visando o bom aproveitamento e uso dessa massa de dados.

2.4.2.2 FAIRificação

Essa etapa é o cerne do processo que visa, de fato, transformar os dados em FAIR. Em um primeiro momento é construído um modelo semântico condizente com os dados, para que se possa definir como serão anotados os dados, utilizando ontologias apropriadas.

Após essa construção, os dados são triplificados, sendo trabalhados para se adequar ao formato recebido do modelo tornando-os conectados. Tal esquema se alinha com os preceitos FAIR, por suas definições exigirem elementos básicos para um dado com essa qualidade. Esses passam a ser identificados unicamente por URIs, possibilitando fácil obtenção e acesso. Além disso, se tornam padronizados e descritos por termos bem definidos em ontologias, viabilizando reuso e fácil combinação com dados da mesma forma tratados.

As duas atividades mencionadas: modelagem e triplificação, devem ser também aplicadas aos metadados. Ainda mais, há de se mencionar que ao triplificar, surge uma camada de metadados que descrevem os dados iniciais, compondo os metadados do modelo. Mas o próprio método gera novos metadados da execução e transformação, compondo outra parte da proveniência.

Após o condicionamento dos dados, esses devem ser expostos, armazenados em um repositório apropriado, de forma que possam ser recuperados por humanos e consumidos por máquinas.

2.4.2.3 Pós-FAIRificação

Por fim, a etapa final visa avaliar a estrutura de dados e metadados produzida, contrapondo com a idealizada inicialmente. Para tal, os objetivos e alvos inicialmente propostos são verificados, validando se foram cumpridos. Uma das avaliações é quanto às questões de competência levantadas na pré-FAIRificação.

2.4.2.4 Suporte por ferramentas

Em algumas das etapas do processo de FAIRificação, há a possibilidade de serem mediados por ferramentas, possibilitando maior facilidade, semi-automatizando os processos e minimizando os riscos de erros.

Uma delas é a referente ao levantamento e organização dos metadados através de ontologias. A análise de ontologias pode ser realizada por meio de ferramentas como Protégé [32], permitindo a visualização e exploração necessárias para decisão da modelagem dos dados. Outra forma é o auxílio na construção de um esquema semântico para metadados, que oferece suporte por aplicações, como a plataforma CEDAR[33] , que é um administrador de esquema de metadados e auxilia nessa tarefa. Além desses, também há o FAIRifier, que, em sua nova versão, agrega o OpenRefine com uma extensão[34], mas limitando a conexão e criação de metadados por meio do preenchimento manual de um formulário. Similar, há o Metadata Editor[35] que permite a geração de metadados FAIR através do preenchimento manual e exportação em arquivo.

Para a atividade de triplificação, várias ferramentas estão disponíveis, mas dentre elas, a em voga neste trabalho, é o *framework* ETL4LOD+, um conjunto de extensões para o PDI. Assim, apoia não só a limpeza como a anotação dos dados, unificando diversas manipulações em um ambiente, e sendo capaz de oferecer a exportação desses dados, com certa limitação quanto ao destino desses dados, como bancos de triplas, em que existe suporte somente ao Virtuoso[36], e como arquivo, somente em texto.

Das ferramentas mencionadas, muitas se concentram na atividade de triplificação somente. A exemplo, o Triplify[37] da Universidade de Leipzig, o software Karma[38] da Universidade do Sul da Califórnia, que também traz certos adicionais, como a limpeza dos dados e aplicação de ontologias.

Já na carga e manutenção de dados e metadados, algumas ferramentas devem ser destacadas, como GraphDB, FAIR Data Point e DATAVERSE. A primeira responsável por

---

[32] https://protege.stanford.edu/
[33] https://metadatacenter.org/
[34] https://github.com/FAIRDataTeam/OpenRefine-metadata-extension
[35] https://editor.fair-dtls.surf-hosted.nl/#!/
[36] https://virtuoso.openlinksw.com/
[37] https://www.w3.org/2001/sw/wiki/Triplify
[38] https://usc-isi-i2.github.io/karma/

armazenar dados e seus respectivos esquemas em triplas RDF, permitindo o acesso através de serviços de consultas por SPARQL *Endpoint*. A segunda permite a carga de metadados estruturados nos padrões FAIR e sua manutenção, possibilitando a recuperação do que foi armazenado e indicação de conteúdos externos, como arquivos em endereços e interfaces de consulta. A última, por fim, se trata de um portal de conhecimento, permitindo carga e estruturação de conjuntos de dados com descritores, seguindo principalmente a ontologia DCAT[39] para catalogação de datasets.

2.4.2.5 Implementações

Esquemas para FAIRificação admitem uma estrutura genérica, para que tenham capacidade abrangente, podendo esses serem utilizados em diversos contextos e tipos de dados.

Em vista disto, é importante que meios de concretizar ainda mais o *Workflow* de FAIRificação surjam, trazendo, assim, formas mais diretas de tornar dados FAIR e, com isso, estipular passos mais definidos. Possibilitando um maior controle e definição do processo e seus resultados de acordo com o desejado. Assim, trabalhos como (OLIVEIRA 2021 MTSR), fazem esforços nesse sentido, modelando mais profundamente e apresentando uma abordagem mais prática do *Workflow* de FAIRificação, a fim de facilitar sua implementação. Contudo, isso é proposto e construído visando uma estruturação e amadurecimento desses processos e seu reuso em outros contextos, não perdendo a abrangência, mas sim adicionando métodos menos generalistas e mais focados na produção.

O ETL4FAIR foi planejado para atuar na necessidade de um instrumental para apoio ao processo de FAIRificação, que auxilie e permita a melhor implementação desse processo, sendo capaz de apresentar uma melhor interconexão entre todas as etapas da FAIRificação, unificando os ambientes e oferecendo facilidade de uso, compondo toda a proveniência.

[39] https://www.w3.org/TR/vocab-dcat-2/

3 ETL4FAIR FRAMEWORK

Motivados pela importância dos princípios FAIR, se verificou que o ETL4LOD+ podia ser estendido de modo a oferecer apoio ao processo de FAIRificação.

Nesse sentido, faz-se necessário o desenvolvimento de um ferramental capaz de suprir todos os passos para, verdadeiramente, trabalhar o ciclo de dados conectados para FAIR, servindo como base para a arquitetura de dados do VODAN-BR, apoiando o processo de transformação de dados, e realizando a publicação em repositórios desses dados e seus metadados. Para isso, foi desenvolvido o *framework* ETL4FAIR.

O ETL4FAIR foi concebido como uma extensão do ETL4LOD+[40] e, por conseguinte, é um *framework* que reúne um conjunto de *plugins* do PDI ETL focado no processamento de dados RDF, que permite ao usuário transformar, mesclar, transferir, atualizar, compartilhar dados abertos, com o apoio do ecossistema ETL do PDI. A Figura 6 ilustra a estrutura geral do *framework*.

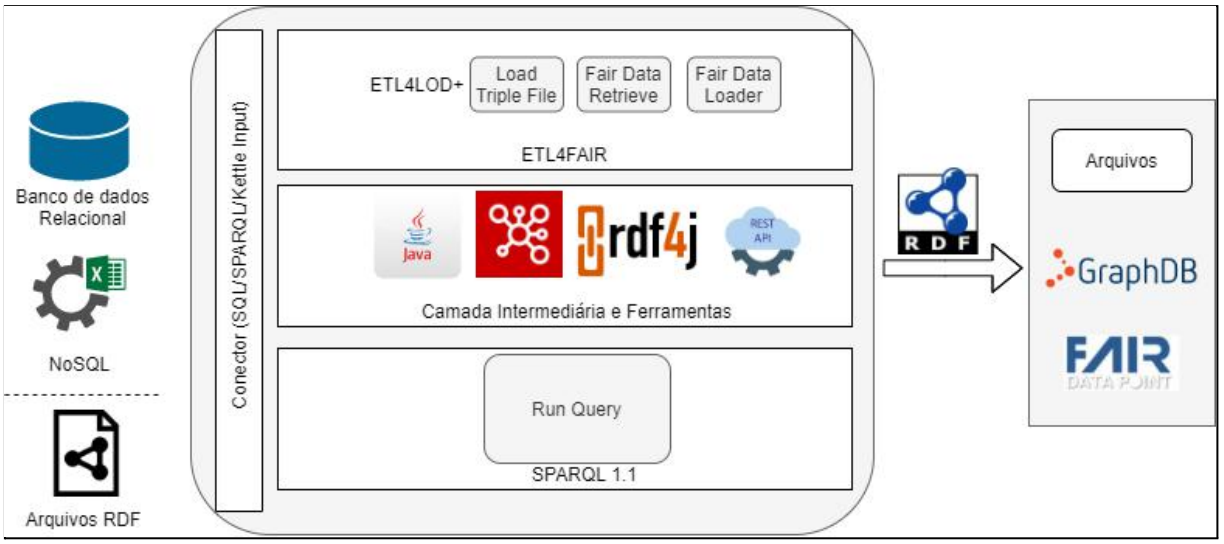


Figura 6 - Arquitetura do ETL4FAIR

Diferentemente do ETL4LOD+, que utiliza o Apache Jena[41] como camada intermediária, o conjunto de *plugins* do ETL4FAIR, utilizam uma estrutura Java de código

[40] https://github.com/johncurcio/ETL4LODPlus
[41] https://jena.apache.org/

aberto, o RDF4J [42](antiga Sesame API), como camada intermediária para realizar o acesso a repositórios de dados com suporte a SPARQL 1.1 (Broekstra, Kampman, & Harmelen, 2002). Importante ressaltar que ao substituir o Jena pelo RDF4J obtém-se uma API de fácil uso e acesso para todas as principais soluções de banco de dados RDF (também conhecido como *Triplestores*). Além disso, essa solução permite conectar-se a terminais SPARQL remotos e criar aplicações que potencializam o poder dos dados conectados e da Web Semântica, já que é um *framework* aberto muito mais robusto, com diversos métodos já em bibliotecas e com ampla compatibilidade, vide Figura 7. Com base no ecossistema do ETL PDI, o conjunto de *plugins* é implementado pelo back-end de programação Java.

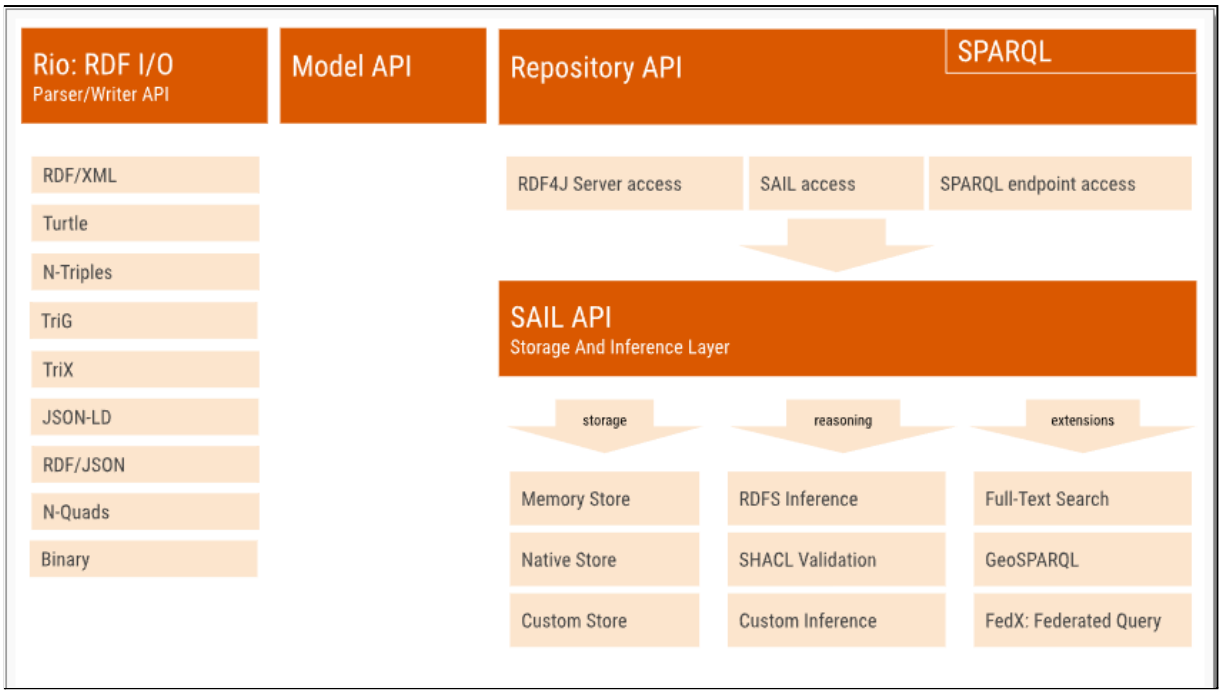


Figura 7 - Arquitetura do RDF4J

Além das modificações nos já existentes *plugins* do ETL4LOD+ e em todos os problemas encontrados (comentado mais abaixo), essa primeira implementação do ETL4FAIR buscou atuar nos seguintes pontos do modelo de arquitetura do VODAN-BR(figura 8):

[42] https://rdf4j.org/

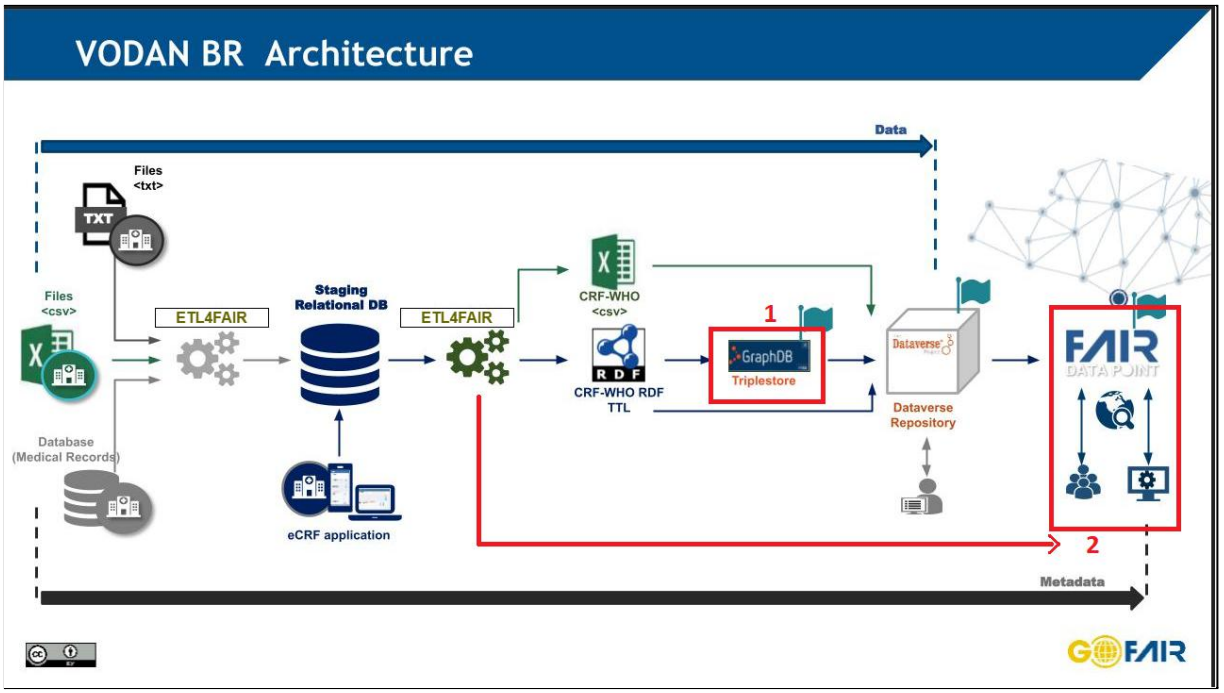


Figura 8 - Modelo arquitetural do VODAN-BR

1. Carga de dados conectados no padrão FAIR em um *triplestore*
2. Recuperação de metadados em um FAIR Data Point
3. Carga de metadados em um FAIR Data Point

Para atingir esse objetivo, foram criados três plugins específicos no ETL4FAIR, que são determinados pela combinação com o protocolo RDF4J e SPARQL 1.1, assim como REST API , bem como suas finalidades pretendidas:

- ***Load Triple File***: Um *plugin* que tem por objetivo receber como entrada dados no formato RDF e realizar o carregamento desse conjunto de dados em um *triplestore*. Ele suporta dados RDF de entrada do sistema de arquivos, sendo compatível com diversos formatos (RDFL/XML, Turtle, N-Triples, Trig, Trix, entre outros) e *triplestores*.
- **FAIR *Data Retriever***: Um *plugin* que tem como objetivo receber uma URL de um artefato armazenado no FAIR Data Point, como o próprio repositório ou um *dataset* e realizar a recuperação desses dados. A saída consiste em três colunas com o sujeito, predicado e objeto do que foi recuperado na serialização N-Triples.

- **FAIR *Data Loader***: Um *plugin* que tem como objetivo receber uma URL de um FAIR Data Point, o tipo de armazenamento a ser feito, ou seja, o objeto a ser criado no repositório, e os metadados triplificados a serem armazenados. Ainda mais, é necessária a entrada de um usuário e senha para autenticação do processo.

Apesar de ser uma boa melhoria quanto ao ETL4LOD (CORDEIRO et al., 2011), o ETL4LOD+, com os seus *plugins*, procurava cobrir todo o fluxo de vida da publicação de dados em RDF (Figura 9) e verdadeiramente abertos, contudo, uma série de problemas o impediam de atingir esse objetivo, como:

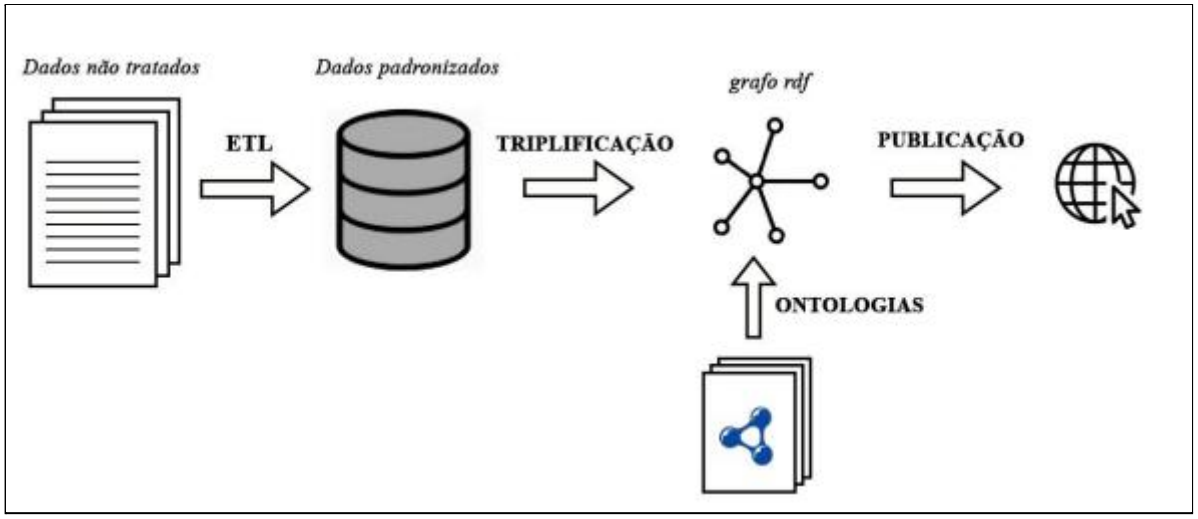


Figura 9 - Ciclo de Vida da publicação de dados em RDF

Versão desatualizada do *Pentaho Data Integration*: Atualmente o PDI se encontra na versão 9.2 e o ETL4LOD+ foi criado para a versão 8.1, impossibilitando seu uso com a versão mais recente.

Versão defasada das dependências de todos os *plugins*: Todos os *plugins* utilizam uma série de bibliotecas que são necessárias para o correto funcionamento e, essas bibliotecas, estavam completamente defasadas.

***Framework* não cobria a parte final do ciclo de vida de dados abertos, a Publicação**: O *plugin*, Sparql Update Output, tinha como objetivo inserir triplas em um banco de dados de triplas, porém, ele estava limitado ao formato de entrada, somente N-TRIPLE (n3), ao repositório de dados, exclusivamente Virtuoso[43] e, acima de tudo, seu código fonte apresentava certas inconsistências que impossibilitaram completamente seu correto uso.

[43] https://virtuoso.openlinksw.com/

Como por exemplo, dependências a bibliotecas que já foram descontinuadas, dados de conexão em codificação rígida (hard-coded, Figura 6) e outros.

```
SparqlUpdate su = new SparqlUpdate("http", "1[redacted]6.164.[redacted]",
        "sparql-auth", 8890, "lodbr", "123456");

// File file = new File("C:\\[redacted]\\Expedito\\Downloads\\download.rdf");
File file = new File("D:\\User\\LodBr\\Kettle\\rdf\\lattes.rdf");
// File file = new File("D:\\User\\LodBr\\Kettle\\rdf\\teste.rdf");
```

Figura 10 - Codificação Rígida

Não estava alinhado com os princípios FAIR e de Ciência Aberta[44]: Nenhum dos *plugins* cobria os princípios FAIR e conexão com FAIR Data Point para enriquecimento de dados e metadados.

Esse trabalho tem por objetivo atuar sobre todos esses pontos levantados, enriquecendo o conjunto de *plugins* com novas funcionalidades, garantindo que o mesmo tem capacidade de suprir todo o ciclo de laboração com dados verdadeiramente abertos e FAIR.

O ETL4FAIR foi desenvolvido levando sempre em consideração a facilidade de uso e a eficiência, mantido limpo e direto, sem que o usuário tenha que realizar qualquer tipo de implementação a parte ou desenvolvimento de codificação. A camada intermediária do RDF4J possibilita acesso direto a *triplestores* como Virtuoso[45] e GraphDB[46], de modo que sua conexão é a mais robusta o possível e, além disso, oferece todo suporte a conexão com FAIR Data Points via REST API e estrutura de Parsing[47] para análise sintática e carregamento de dados e metadados pretendidos para o FAIR Data Point.

3.1 IMPLEMENTAÇÃO DO ETL4FAIR

O ETL4FAIR herda muitas funções do *Pentaho Data Integration* e do ETL4LOD+, tendo a maioria dos recursos necessários para processar dados RDF de uma forma altamente intuitiva. Do ponto de vista do usuário, ele é livre para instalação e fácil de usar. Sua implementação foi realizada no ambiente Windows[48], com o Java *Development Kit* (JDK), em

[44] https://en.wikipedia.org/wiki/Open_science
[45] https://virtuoso.openlinksw.com/
[46] https://graphdb.ontotext.com/
[47] https://en.wikipedia.org/wiki/Parsing
[48] https://www.microsoft.com/pt-br/windows/

conjunto com o Apache Maven[49] para construção do projeto, além de fornecer uma interface interativa para o usuário no *Spoon* do PDI, capaz de adicionar e iniciar diversas tarefas. Uma tarefa é composta de:

- **Nome da Tarefa**: Breve resumo do propósito da operação ao preparar e criar uma tarefa de processamento de dados, ajudando mais no gerenciamento do fluxo de tarefas, conforme ilustrado na figura 11 (a).
- ***Steps***: Os componentes principais implantados no fluxo para fazer uma tarefa de processo de dados. Contém entradas, *plugins* ou saídas desejadas. Para tornar a operação mais fácil e amigável, os modelos de configurações são definidos em um modelo de formulário, com campos para mapeamento que foram personalizados para os requisitos de configurações do *plugin* e podem ser editados diretamente pelos usuários, conforme ilustrado na figura 11 (b).
- ***Hops***: São responsáveis por vincular os objetos (nós) escolhidos e indicar para qual direção os fluxos de dados vão. Uma vez que os Hops são definidos as tarefas são validadas e prontas para serem executadas, conforme ilustrado na figura 11 (a).

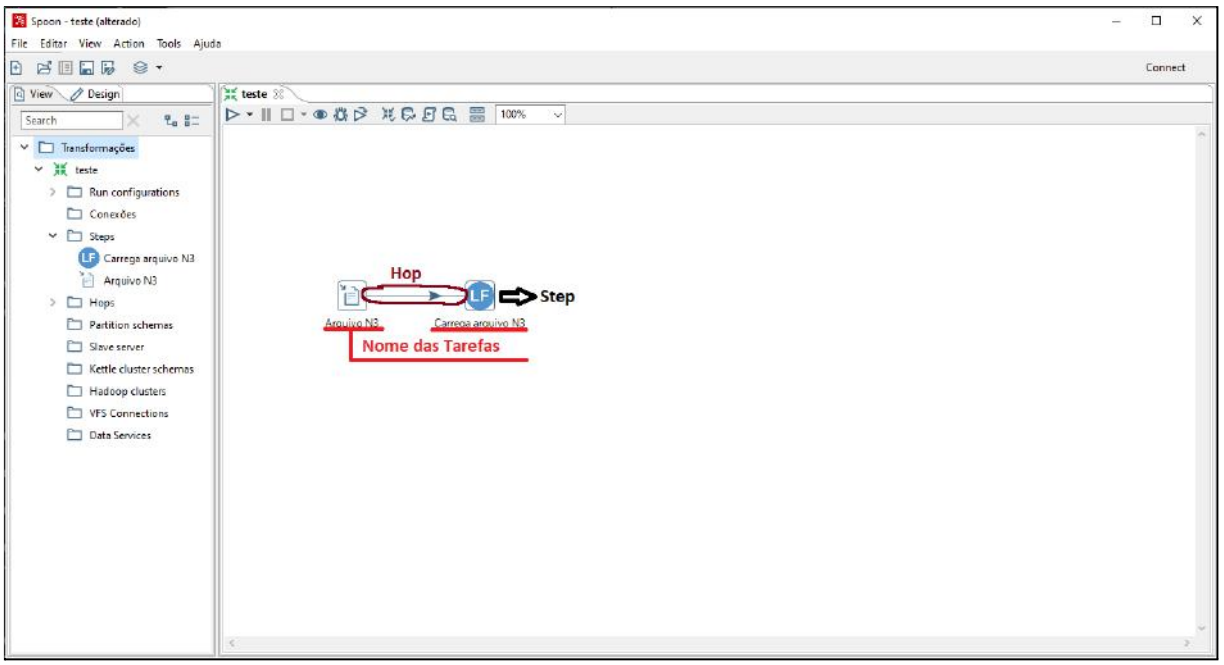


Figura 11 (a) - Um exemplo da interface Gráfica

[49] https://maven.apache.org/

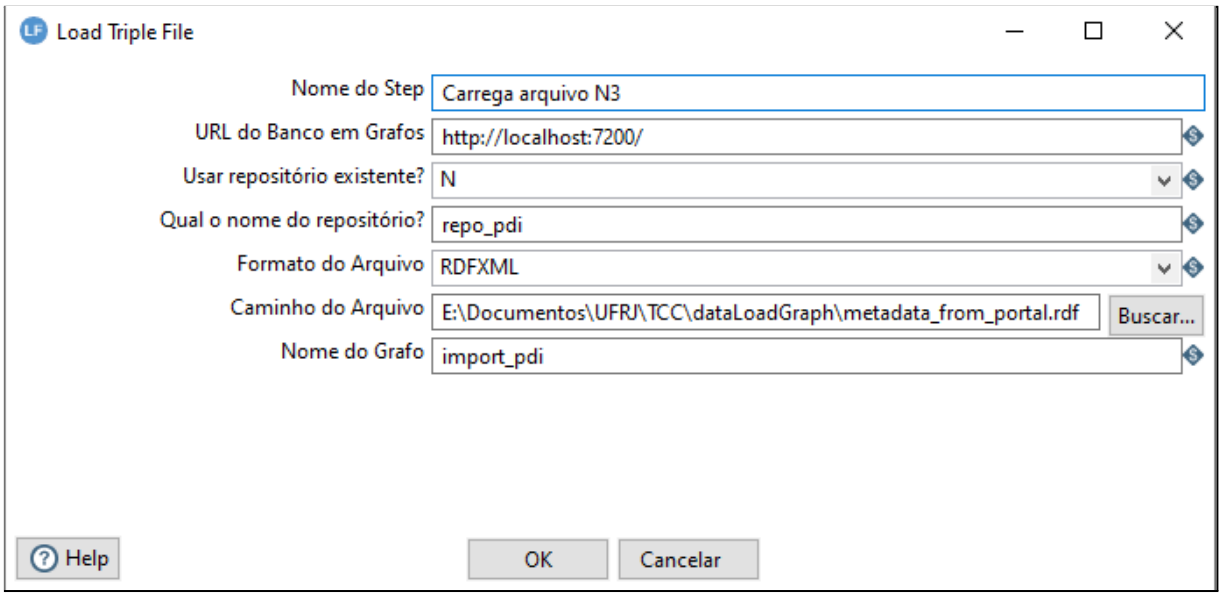


Figura 11 (b) - Front-End de um *Plugin*

3.1.1 Preparação do Ambiente de Desenvolvimento

O ETL4LOD+ estava bem defasado quanto àversão de todas as dependências de seus *plugins* e, por isso, foi necessário uma equalização geral dessas dependências, para garantir uma correta atualização e um ambiente de desenvolvimento mais estável e com mais recursos. Faz-se necessário pontuar que a única dependência que não foi atualizada foi a do próprio JDK (Java *Development Kit*), visto que certas funcionalidades desta versão foram depreciadas nas versões seguintes, funcionalidades essas que são utilizadas em todo o conjunto de *plugins*. Em virtude da facilidade em encontrar e executar o download da Versão 8 do JDK no site oficial da Oracle[50], ele foi mantido. Todas as outras dependências foram alteradas conforme o quadro 2:

Quadro 2: Lista de dependências atualizadas

Dependência	Mudança
PDI	Da versão 8.1 para a versão 9.2.0.0-290
xstream	Da versão 1.4.10 para a versão 1.4.18

[50] https://www.oracle.com/br/java/technologies/javase/javase8-archive-downloads.html

httpclient	Da versão 4.5.6 para a versão 4.5.13
httpcore	Da versão 4.4.10 para a versão 4.4.14
jena	Da versão 3.8.0 para a versão 3.17.0
commons-compress	Da versão 3.8.0 para a versão 3.17.0
libthrift	Da versão 0.10.0 para a versão 0.15.0
any23	Da versão 2.2 para a versão 2.4
json	Da versão 20180718 para a versão 20210307
jackson	Da versão 2.9.7 para a versão 2.12.5
json-ld	Da versão 0.12.1 para a versão 0.13.3
mysql	Da versão 5.1.35 para a versão 8.0.25
slf4j	Da versão 1.7.6 para a versão 1.7.36

Fonte: própria

Após essa equalização de ambiente, toda a documentação do conjunto de *plugins* foi revista, visando uma melhora na clareza dos processos e facilidade para a integração e instalação por parte do usuário.

3.2 *LOAD TRIPLE FILE*

Porquanto o *plugin* Sparql Update Output do ETL4LOD+ não estava cobrindo corretamente a etapa de publicação dos dados na Web e, mesmo se estivesse cobrindo, estaria limitado ao *triplestore* Virtuoso. Foi necessário a implantação de um novo ferramental para cobrir essa importante fase no ciclo de vida dos dados conectados. Parte da implementação pode ser vista no Apêndice D.

O *Load Triple File* visa substituir o Sparql Update Output e melhorar o mesmo, a fim de oferecer uma interface de simples uso para inclusão de dados triplificados em *triplestores* genéricos e com suporte a diferentes formatos e não só N-Triples. Com esse intuito, o plugin apresenta uma interface com poucas configurações necessárias e que seguem uma ordem lógica quanto a inserção de dados em um Triple Store, sendo essas configurações imediatas, conforme a figura 11 (b).

Note que as configurações necessárias são:

- **Nome do Step**: necessário para todos os *plugins* do PDI.
- **URL do *triplestore***: URL de conexão com o banco em grafos que irá armazenar os dados do arquivo de input.
- **Validação de repositório existente**: A fim de proporcionar uma experiência mais personalizada ao usuário, foi incluído a possibilidade fazer uso de um repositório (*dataset*) existente ou não. Ao selecionar que não irá utilizar um *dataset* existente, a aplicação cria um repositório com o nome "repo_pdi" e configurações padrões.
- **Formato do arquivo**: Procurando uma maior robustez no processo e tendo em mente os diversos formatos de dados RDF, foi adicionado a compatibilidade com os formatos abaixo:

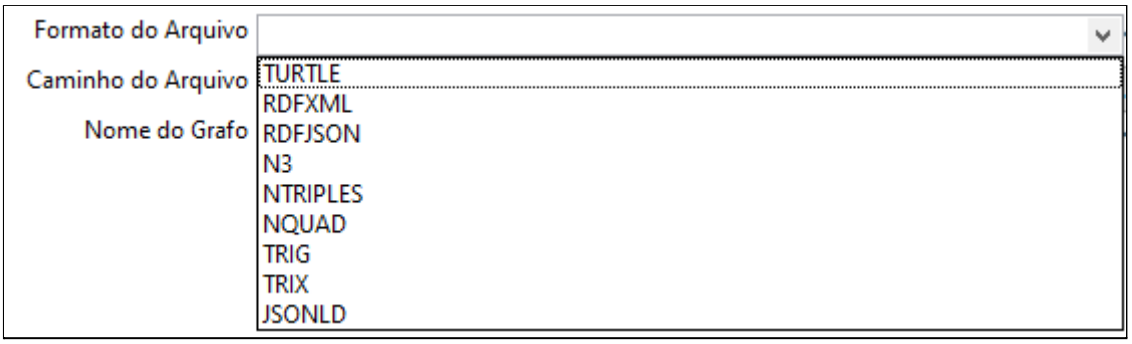


Figura 12 - Formatos Suportados pelo *Load Triple File*

  - Onde:
    - TURTLE: arquivos .ttl
    - RDFXML: arquivos .rdf, .rdfs, .owl e .xml
    - RDFJSON: arquivos .rj
    - N3: arquivos .n3
    - NTRIPLE: arquivos .nt
    - NQUAD: arquivos .nq
    - TRIG: arquivos .trig
    - TRIX: arquivos .trix
    - JSONLD: arquivos .jsonld

- **Caminho do arquivo**: caminho absoluto do arquivo (como, C:\Users\User\Documents) ou variável recebida de step anterior.

- **Nome do Grafo**: cria grafo nomeado no triplestore para armazenar os dados do arquivo, cria sempre no formato http://etl4FAIR.com/nome_do_grafo_infornado

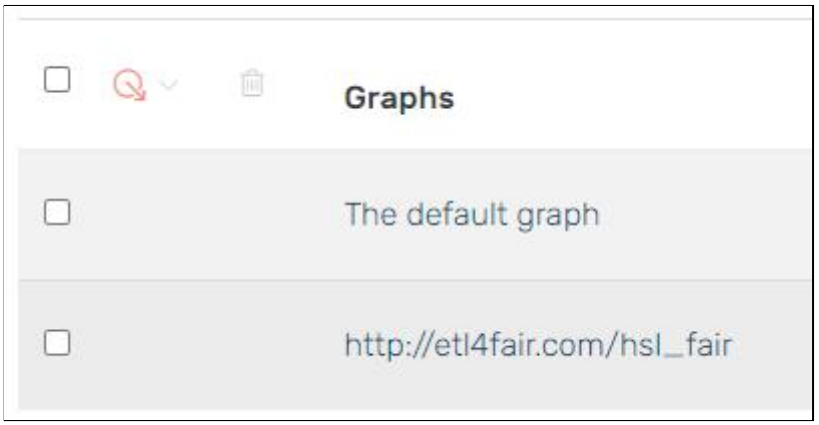


Figura 13 - Exemplo de grafo nomeado criado no GraphDB

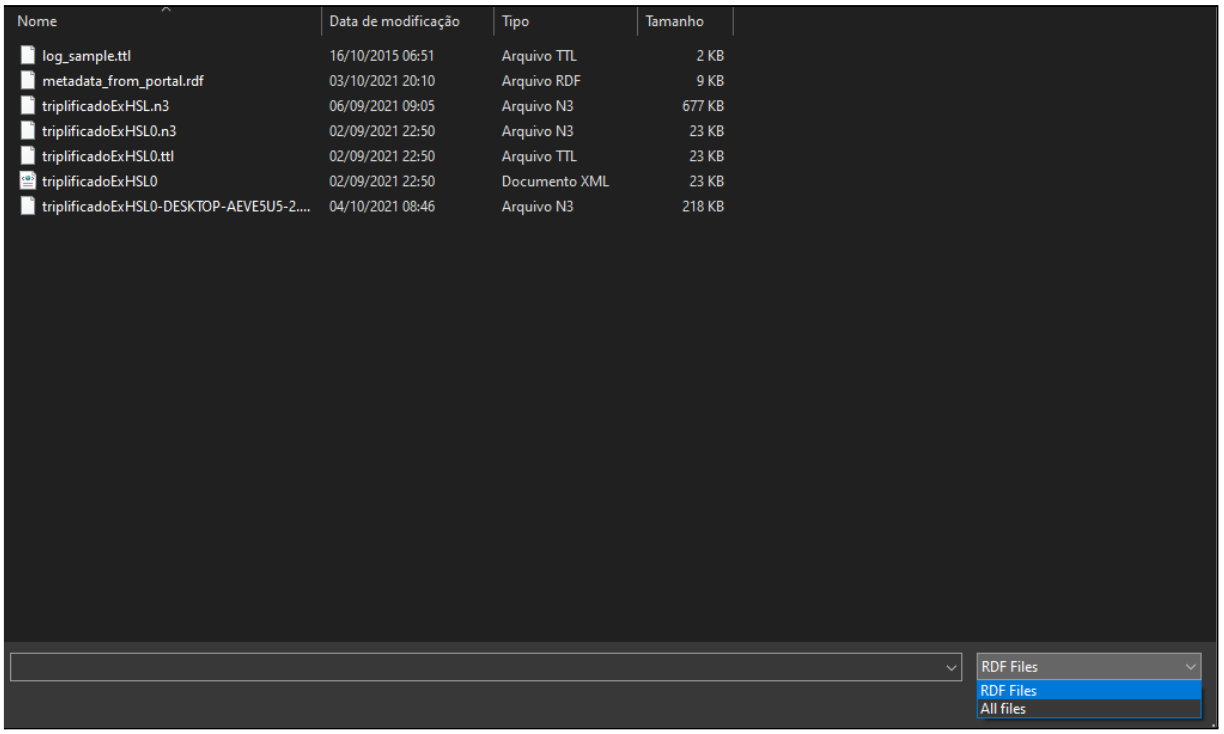


Figura 14 - Exemplo de busca de arquivos na máquina do usuário

3.2.1 Conectividade e Dependências

O *Load Triple File* foi desenvolvido tendo como base a camada intermediária proporcionada pelo RDF4J, visto que a maioria dos mais utilizados *triplestores* da atualidade oferecem total suporte ou foram desenvolvidos também sobre o RDF4J. Com isso, tem-se uma vasta gama de conectividade a *triplestores* e capacidade de lidar com situações adversas, pois o RDF4J possui uma lista extensa de métodos[51] para trabalhar com RDF.

Salienta-se que, mesmo com essa capacidade de fazer uso a diferentes *triplestores*, o mesmo apresenta um maior desempenho e nível de integração com o GraphDB, pois ele foi inteiramente desenvolvido a partir do RDF4J. Portanto, sua incorporação com o GraphDB é notável e faz-se necessário recomendar seu uso.

Ademais, foram utilizadas duas dependências padrões para o *Load Triple File* apresentar o melhor funcionamento.

Quadro 3: Lista de dependências do *Load Triple File*

Dependência	Descrição
RDF4J	Na versão 3.7.1, que oferece uma série de métodos como capacidade de conexão e carga a Triple Stores, módulos de Parser para arquivos RDF, criação de grafos nomeados e outros
GraphDB Free Runtime	Incluído pensando na melhor otimização ao uso de recursos do GraphDB.

3.2.2 Funcionalidade

Ao se utilizar do *plugin*, uma série de procedimentos são realizados no *backend* da aplicação, procedimentos esses que seu correto entendimento é necessário para o melhor uso do mesmo. Pode ser resumido em duas frentes de atuação:

- Usuário
- Sistema

[51] https://mvnrepository.com/artifact/org.eclipse.rdf4j

O usuário deve inserir todas as configurações descritas na seção 3.2 e, com base nesses parâmetros, o sistema realiza uma série de ações. Focando nos procedimentos do sistema, tem-se:

1. Recupera variáveis fornecidas pelo usuário na aplicação gráfica
2. Se conecta ao *triplestore* com base na URL fornecida
3. Valida se vai utilizar um repositório de dados existente ou não e:
    a. Se não, cria um repositório com base nas configurações padrões listadas no arquivo repo-defaults_test.ttl, que está localizado na pasta lib da instalação do *plugin* no PDI (data-integration\plugins\steps\LoadTripleFile\lib). Esse arquivo cria um repositório com o nome “repo_pdi”.
    b. Se sim, segue para 4.
4. Conecta no repositório informado
5. Usa o parâmetro que contém o nome do grafo nomeado informado para realizar a carga dos dados sobre esse grafo. Destaca-se que, caso o grafo não exista, será criado um novo no padrão http://etl4FAIR.com/nome_do_grafo_infornado
6. Lê arquivo fornecido, seja acessando por um caminho absoluto na máquina do usuário ou de um *step* anterior, e salva conteúdo do arquivo em memória
7. Realizar *parser* do arquivo para validar o formato
    a. Se não for um formato compatível com o informado, notifica o usuário e aborta o fluxo
    b. Se for, segue para 8
8. Insere conteúdo do arquivo no grafo nomeado
9. Finaliza fluxo

A figura 15 ilustra de maneira geral todo o caminho percorrido tanto pelo usuário quanto pela aplicação durante o uso do *Load Triple File*. Vale evidenciar que, em cada nó dessa arquitetura são realizadas uma série de validações de ambiente para garantir o correto uso e funcionamento. Caso detectado algo que impacte nessa normalidade, o fluxo é abortado e uma mensagem de erro descritiva contendo as informações necessárias para *debug* é exibida.

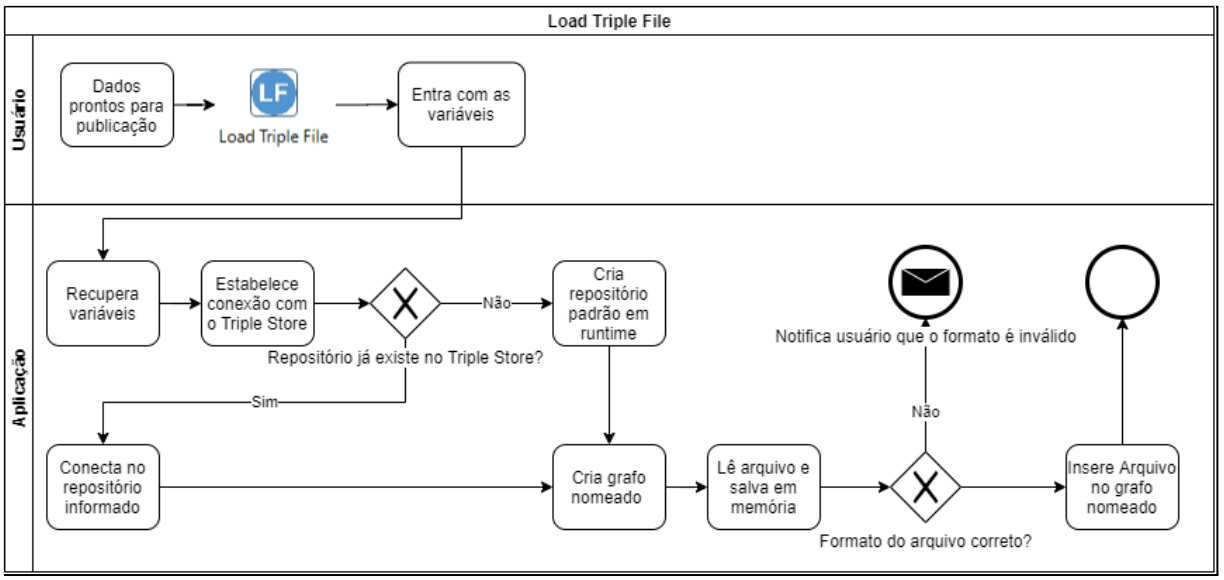


Figura 15 - Fluxograma geral do *Load Triple File*

3.3 FAIR *DATA RETRIEVER*

Com o intuito de promover maior conectividade com o FAIR Data Point, foi realizado um esforço visando a conexão, manipulação dos metadados em triplas e seu futuro carregamento. Nesse sentido, esforços foram realizados para garantir a comunicação, processamento de triplas e recuperação dos metadados contidos em um FAIR Data Point. Assim, obtendo o plugin FAIR *Data Retriever*.

Seu desenvolvimento se deu de uma forma bem compacta. A parte central do código responsável pela recuperação pode ser vista no Apêndice E. Dessa forma, requisitou poucas configurações, generalizando o tratamento dos dados recuperados. Afinal, esses respeitam uma estrutura similar como pode ser observado nos apêndices A, B, C e D. Com isso, a configuração necessária para seu uso é:

- **URL do artefato FAIR Data Point**: Entrada da URL do artefato a ser recuperado do repositório FAIR Data Point, *Catalog*, *Dataset* e *Distribution*.

Figura 16 - Visão do painel do FAIR *Data Retriever*

3.3.1 Conectividade e Dependências

O FAIR *Data Retriever* realiza uma conexão com o FAIR Data Point, necessitando de uma instância ativa, local ou remota, para realizar o processo de requisição e obtenção dos dados. Sendo assim, foi utilizada uma base de funções de requerimentos HTTP, compatível com a estrutura REST API existente no FAIR Data Point. Além disso, também foi estruturado um módulo com o intuito de processar os dados recebidos no formato N-Triple para uma saída correta, usando a biblioteca Rio Parser, do *framework* RDF4J.

Quadro 4: Lista de dependências do FAIR *Data Retriever*

Dependência	Descrição
RDF4J/Rio	Na versão 3.7.1, oferece uma série de métodos como capacidade de conexão e carga a *triplesrotes*, módulos de *parse*r para arquivos RDF, criação de grafos nomeados e outros. Foco no *parser*, submódulo Rio que permite a manipulação e destaque das triplas sendo recuperadas.
FAIR Data Point	REST API que permite a realização de processos e armazenamento de metadados FAIR.

3.3.2 Funcionalidade

Após o usuário inserir a URL do artefato que se deseja obter, certos processos são realizados para a recuperação dos dados:

1. *GET Request* - Primeiramente uma requisição através de protocolo Web HTTP do tipo *GET* é definida com uso do endereço fornecido na entrada da aplicação. Em tal, já é

configurado o tipo de dado a ser recuperado, no caso, triplas serializadas no formato N-Triples.

2. Organização dos resultados - O retorno da requisição *GET*, contendo os dados desejados, é recuperado via linhas de retorno. Onde esses dados são acumulados no *buffer* da aplicação, para o processamento do resultado final na saída do plugin.
3. Processamento das saídas - Após os resultados estarem organizados e acumulados no formato correto, o *parser* da biblioteca Rio é executado sobre o conjunto de dados, quebrando-os em triplas e processando no formato coluna desejado para a saído, ou seja, cada uma das colunas como sendo: sujeito, predicado e objeto.

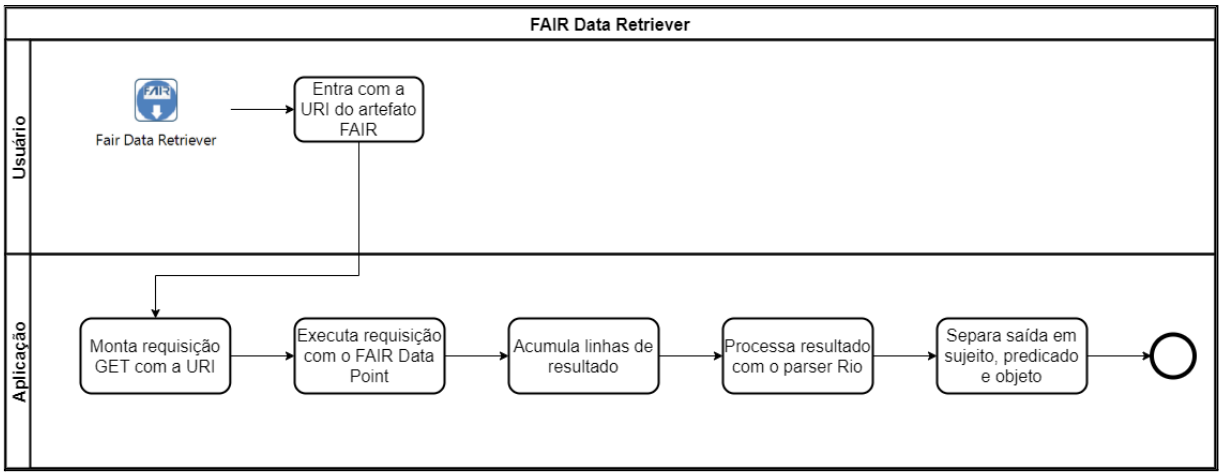


Figura 17 - Fluxograma geral do FAIR *Data Retriever*

3.4 FAIR *DATA LOADER*

Uma das necessidades apresentadas, é a carga de metadados nos FAIR Data Points existentes. Com a praticidade e existência do ambiente *Pentaho*, a unificação dos processos se faz importante tanto na geração dos metadados e carga como na documentação do processo. De tal forma, foi desenvolvido um *plugin* com a intenção de criar *Catalog*, *Datasets* ou *Distributions* e carregar os metadados de cada.

O *plugin* tem como base de funcionamento as requisições *REST API* do tipo *POST*. O artefato em questão também realiza a autenticação do usuário, por meio de um *login* e senha definidos, para realizar as operações em seu momento de execução. Como entrada, recebe as triplas que definem o elemento FAIR a ser criado. Como resultado, retorna o endereço do

elemento criado para que sub-elementos possam ser criados, como *Datasets* que pertencem a um *Catalog*. Uma parte central do código pode ser vista no Apêndice F.

De forma geral a configuração se dá pelas entradas:

- **URL do artefato FAIR DP:** Endereço do FAIR Data Point
- **Usuário:** Usuário registrado no FAIR Data Point para autenticação
- **Senha:** A senha do usuário cadastrado para prosseguir com a autenticação
- **Tipo de carregamento:** Caixa de seleção com três opções: *Catalog*, *Dataset* e *Distribution*. Define o tipo de processo e elemento FAIR a ser criado de acordo com a opção.
- **Publica:** Marcação para que os metadados sejam publicados diretamente e não carregados como *Draft*.

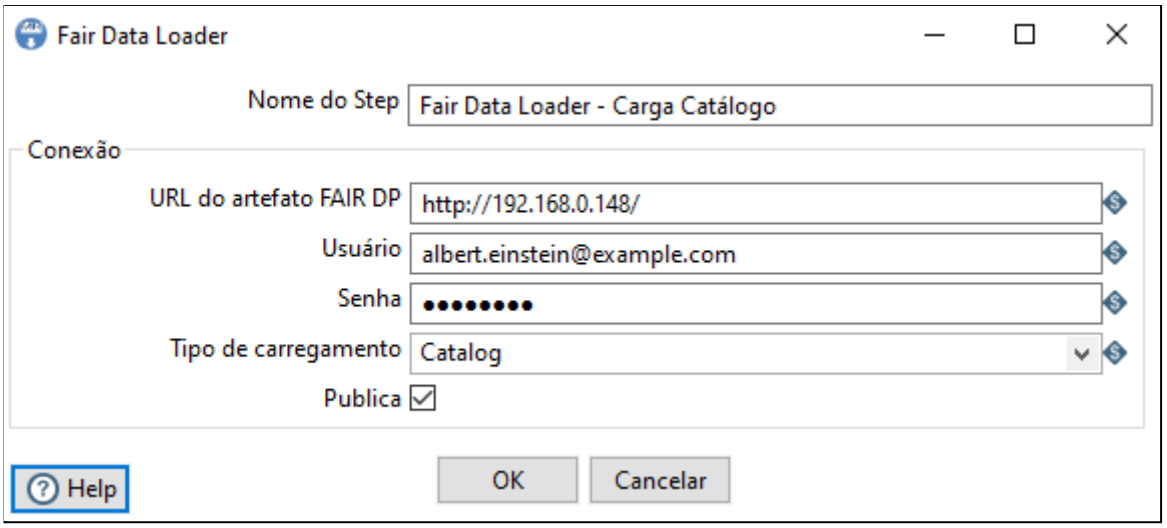


Figura 18 - Visão do painel do FAIR *Data Loader*

3.4.1 Conectividade e Dependências

O FAIR *Data Loader* realiza uma conexão com o FAIR Data Point, necessitando de uma instância ativa, local ou remota, para realizar o processo de autenticação e armazenamento dos dados. Dessa forma, foi utilizada uma base de funções de requerimentos HTTP, compatível com a estrutura REST API existente no FAIR DP.

Quadro 5: Lista de dependências do FAIR *Data Loader*

Dependência	Descrição
JSON	Necessário para processar o envio e o recebimento de mensagens no formato JSON pelas requisições
FAIR Data Point	REST API que permite a realização de processos e armazenamento de metadados FAIR.

3.4.2 Funcionalidade

Uma vez inseridos os dados de configuração, o *plugin* necessita da entrada das triplas, que devem ser geradas nos passos anteriores ao uso desse *plugin*, no PDI. E, então, são passadas através do fluxo até a entrada do FAIR *Data Loader*. O processamento do *plugin* pode ser separado em:

1. Preparação: A partir desse ponto, as triplas recebidas no fluxo são acumuladas no *buffer* e preparadas para a possível carga.

2. Autenticação: Consiste no processo realizado através de uma requisição *GET* autenticada com usuário e senha, obtendo um token de autorização e, assim, concedendo permissão para realizar alterações. Caso não autorizado, um erro é propagado e notificado no *log* do PDI.

3. Criação e carga: Uma vez autorizado, o comando *POST* de requisição da criação do elemento e carga de dados, é montado e executado, compondo o token de autorização e todo o conteúdo que irá compor o elemento.

4. Publicação: Em caso de sucesso na carga dos dados, se foi solicitada a ação de publicação direta, uma nova ação é feita. Um comando *PUT* é montado com o token de autenticação e é executado para alterar o estado do elemento criado. É notificado o sucesso ou falha da publicação. Caso essa ação não tenha sido solicitada, o processo segue para a última parte.

5. Resultado: Em caso de sucesso, são retornadas as triplas, o endereço do elemento, bem como a mensagem de sucesso (Resposta 201). Caso contrário, mensagens de erro são geradas e notificadas pelo *log*.

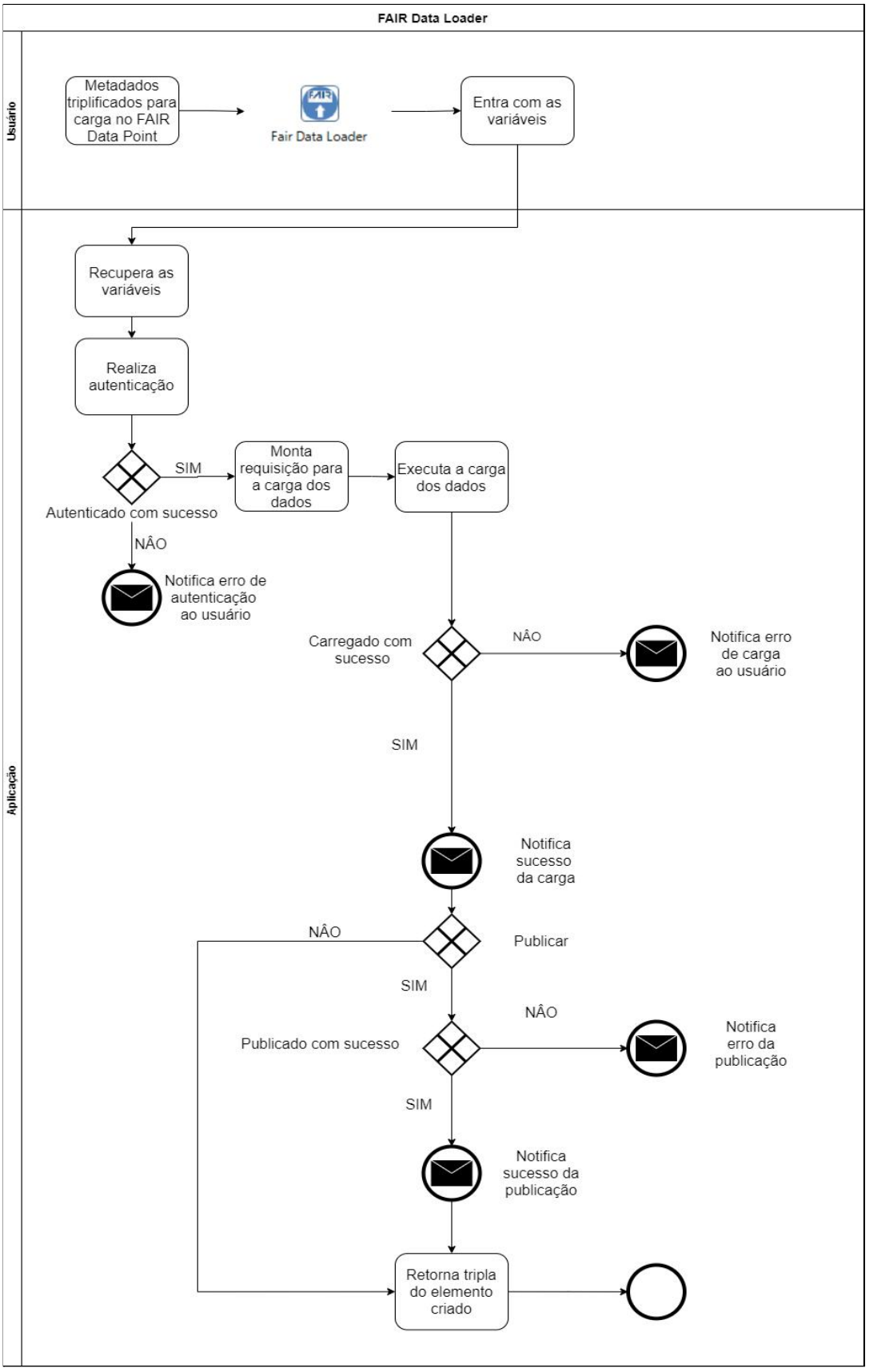


Figura 19 - Fluxograma geral do FAIR *Data Loader*

4 EXEMPLOS DE APLICAÇÃO

Durante a pandemia de COVID-19, que teve início no fim de 2019[52], e em ocasiões anteriores, vimos um gerenciamento e reutilização de dados abaixo do ideal e, além disso, o acesso a dados de extrema importância sobre epidemias anteriores nem sempre é acessível para diferentes populações e países afetados. Por exemplo, os dados das últimas epidemias de Ebola são muito difíceis de encontrar, acessar e, se forem acessíveis, não são interoperáveis, muito menos reutilizáveis (J. LEGRAND et al., 2006).

Sob a necessidade urgente de tornar esses dados epidemiológicos verdadeiramente abertos, garantindo que os mesmos estejam de acordo com os princípios FAIR, o projeto VODAN-BR[53], em parceria com o Hospital Sírio Libanês[54], vem utilizando amplamente a estrutura fornecida pelo ETL4FAIR, com o propósito de normalizar os dados de internações de pacientes com COVID-19 para os padrões FAIR, seguindo a estrutura do Formulário de Relato de Casos Clínicos Global de COVID-19 (Case Report Form - CRF) da Organização Mundial da Saúde[55], o COVIDCRFRAPID[56], possibilitando o uso desses dados em futuras pesquisas. Isso é alcançado pelo processo de triplificação, aplicando o modelo construído, um recorte no Apêndice E, baseado na ontologia da OMS COVIDCRFRAPID (Apêndice F).

Na primeira parte deste capítulo, uma análise mais aprofundada foi realizada em todos os dados selecionados. Além disso, também foi feita uma verificação dos resultados, esclarecendo mais informações sobre as descobertas. Por fim, alguns comentários adicionais foram expostos.

4.1 ESTRUTURA

Com o propósito de realizar o processo de FAIRificação (adequar os dados com base nos princípios FAIR) sobre os dados de internações de pacientes com COVID-19 no Hospital

---

[52] https://www.paho.org/pt/covid19/historico-da-pandemia-covid-19
[53] https://portal.fiocruz.br/en/vodan-brazil
[54] https://www.hospitalsiriolibanes.org.br/Paginas/nova-home.aspx
[55] https://www.who.int/publications/i/item/global-covid-19-clinical-platform-case-report-form-(crf)-for-post-covid-conditions-(post-covid-19-crf-)
[56] https://www.who.int/publications/i/item/WHO-2019-nCoV-Clinical_CRF-2020.4

Sírio Libanes (HSL), disponibilizando-os no devido formato para que possam ser utilizados em pesquisas, foi desenvolvido um módulo de transformações ETL no PDI, fazendo uso do ETL4FAIR, seguindo o modelo de arquitetura do VODAN-BR (figura 8).

Primeiramente, neste processo, tem-se os dados do HSL em um arquivo, no formato CSV, que são obtidos via portal da FAPESP[57], por meio de um *download* manual. Esses dados então são inseridos em um Banco de Dados Relacional da *engine* PostgreSQL, permitindo a criação de consultas (*Queries*) avançadas para a devida análise. Esse volume de dados é separado em três visões distintas, uma com os dados do paciente, outra com os dados do desfecho da internação e, por fim, uma visão sobre todos os exames realizados. Essas visões são salvas nas determinadas tabelas:

Quadro 6: Lista de visões por tabelas

Tabela	Descrição
tb_paciente	Contém os dados básicos do paciente como seu numéro de identificação, e, também, os de caráter demográfico como: ano de nascimento, sexo, cidade, UF, município e CEP
tb_paciente_covid	Contém os dados iniciais dos pacientes internados no HSL, dada a condição de teste positivo de COVID-19
tb_exames	Contém os dados dos exames realizados durante a internação do paciente, apresentando valor de resultado e data
tb_desfecho	Contém os dados referentes ao desfecho da internação, apresentando os dados da conclusão da assistência como a data, a unidade e como foi o desfecho, se houve alta médica ou outro

Exercida essa separação, com os dados obtidos e a modelagem construída, é necessário iniciar o processo de anotação semântica dos dados, a fim de obter os dados no padrão FAIR, de acordo com o estipulado. Essa transformação, por sua vez, irá realizar a concretização da modelagem semântica, aplicando os termos da ontologia, descrevendo os dados com o significado que possuem. Tal processo é realizado com o suporte do ferramental oferecido pelo PDI com o *plugin* ETL4FAIR.

[57] https://repositoriodatasharingfapesp.uspdigital.usp.br/

Com isso, foi construído um *workflow* com o conjunto de processos de tratamentos e adequação dos dados iniciais para alcançar o estado final planejado. Tal fluxo de dados foi separado em quatro módulos.

1. **Módulo de Controle**: define o fluxo geral de transformação dos dados;
2. **Módulo de Internação**: responsável pela FAIRificação dos dados iniciais de internação do paciente, como dados pessoais e demográficos;
3. **Módulo de Acompanhamento**: responsável pela FAIRificação dos dados que são gerados enquanto o paciente está internado, representando uma visão evolutiva;
4. **Módulo de Desfecho**: responsável pela FAIRificação dos dados referente a saída do paciente;
5. **Módulo de Metadados**: responsável por realizar a adequação e carga dos metadados pertinentes, segundo modelo, para um FAIR Data Point.

4.2 MÓDULO DE CONTROLE

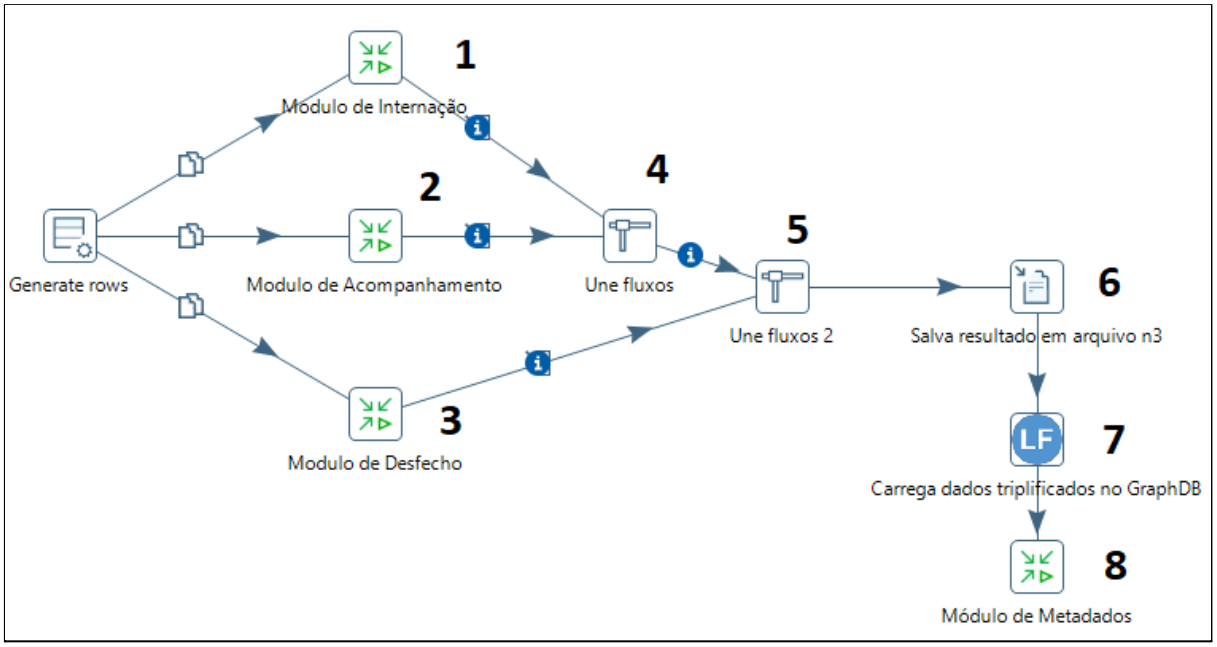


Figura 20 - *Workflow* geral de transformação dos dados

Como era necessário controlar todos os outros módulos e seus respectivos fluxos de dados, assim como consolidar todos os resultados das transformações em um único conjunto

de dados e, após isso, criar um arquivo único com esse volume e realizar sua carga em um triplestore ou FAIR Data Point, esse módulo foi criado. Seu maior objetivo é o de servir como controlador sobre os outros módulos, de modo que é possível desabilitar um componente se não é desejado sua execução.

4.3 MÓDULO DE INTERNAÇÃO

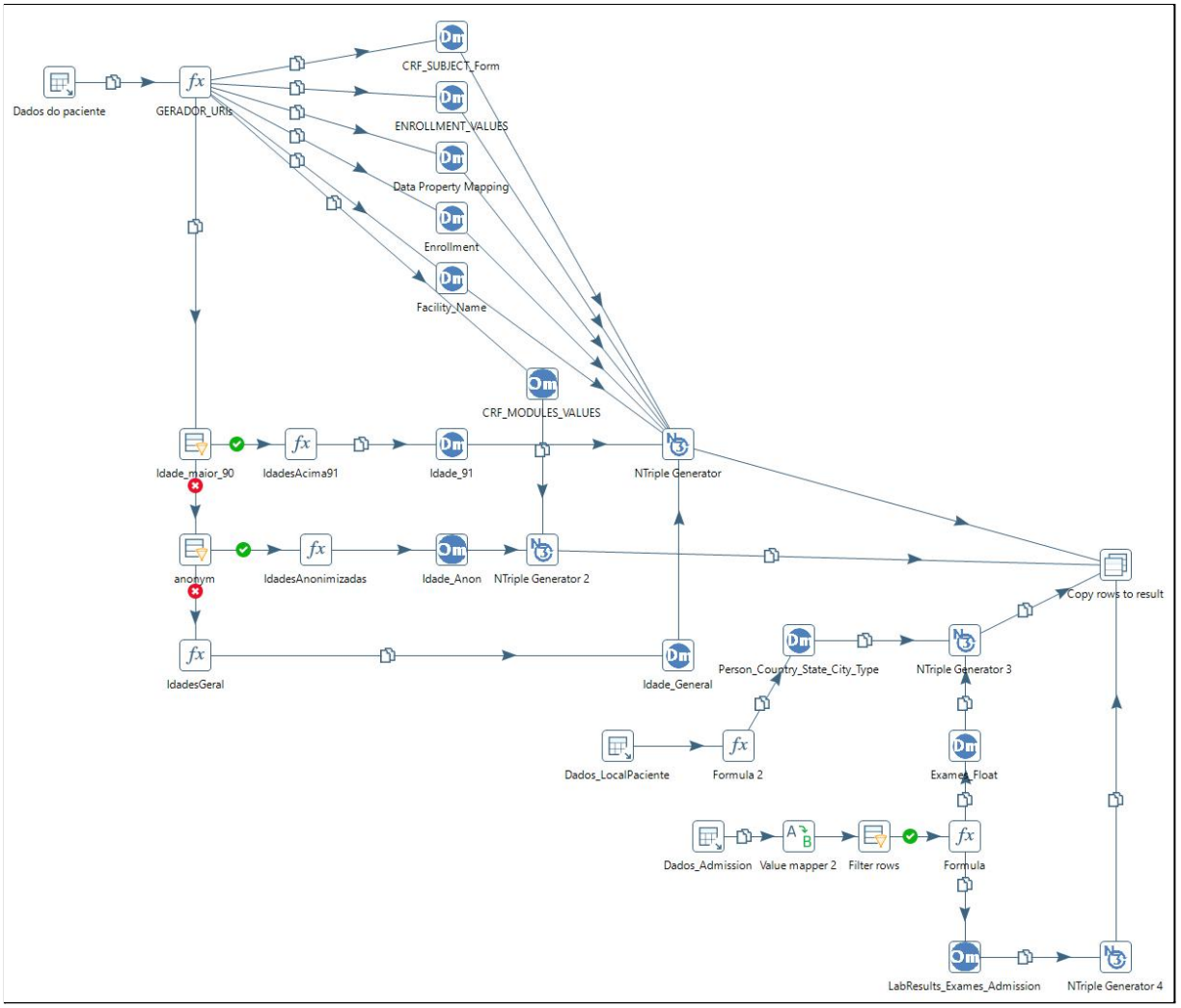


Figura 21 - Fluxo de dados do Módulo de Internação

O objetivo dessa transformação é adequar a massa de dados referentes a internação do paciente à ontologia COVIDCRFRAPID[58], tornando-os FAIR.

[58] https://bioportal.bioontology.org/ontologies/COVIDCRFRAPID

4.3.1 Dados do Paciente

Ao dar entrada no hospital, o paciente preenche um formulário com seus dados de admissão, seguindo o padrão estabelecido no CRF (módulo 1 do COVIDCRFRAPID[59]), o que constitui o primeiro grande volume de dados. Essas informações são:

1. Dados demográficos
    a. Nome (anonimizado como identificador único de data Instituição, nesse caso, o Hospital Sírio Libanes substitui os nomes por ID's)
    b. Sexo
    c. Ano de Nascimento
    d. Idade
    e. País
    f. Estado
    g. Município
2. Dados do Hospital
    a. Nome da Unidade Hospitalar
    b. Número de registro do paciente no HSL
    c. Número de registro da internação
3. Dados laboratoriais
    a. Resultados de exames pelo o qual o paciente teve de realizar

O primeiro passo neste módulo é a recuperação dos dados iniciais do paciente, por meio de uma consulta na tabela tb_paciente_covid, e a criação de URIs para todas as colunas seguindo o padrão da ontologia, identificando os recursos.

59 https://apps.who.int/iris/bitstream/handle/10665/333229/WHO-2019-nCoV-Clinical_CRF-2020.4-eng.pdf?sequence=1&isAllowed=y

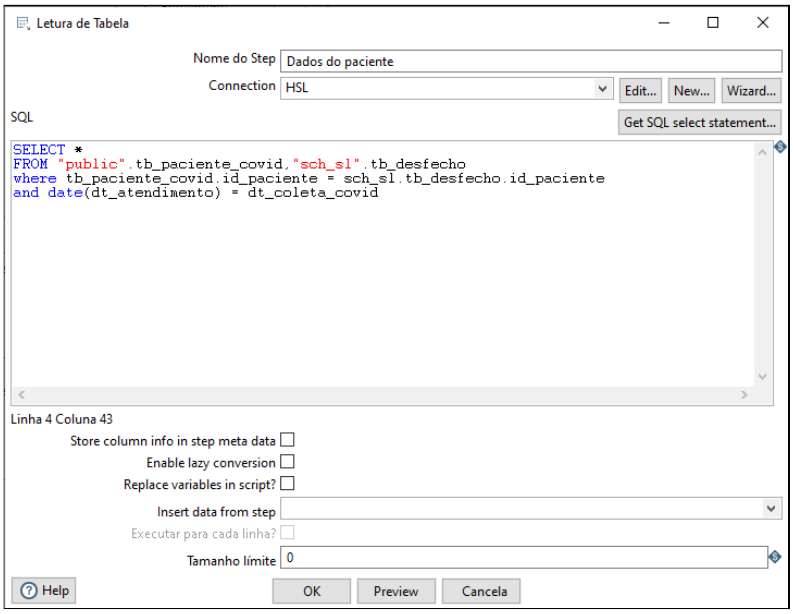


Figura 22 - Consulta para obter os dados do Paciente no momento da internação

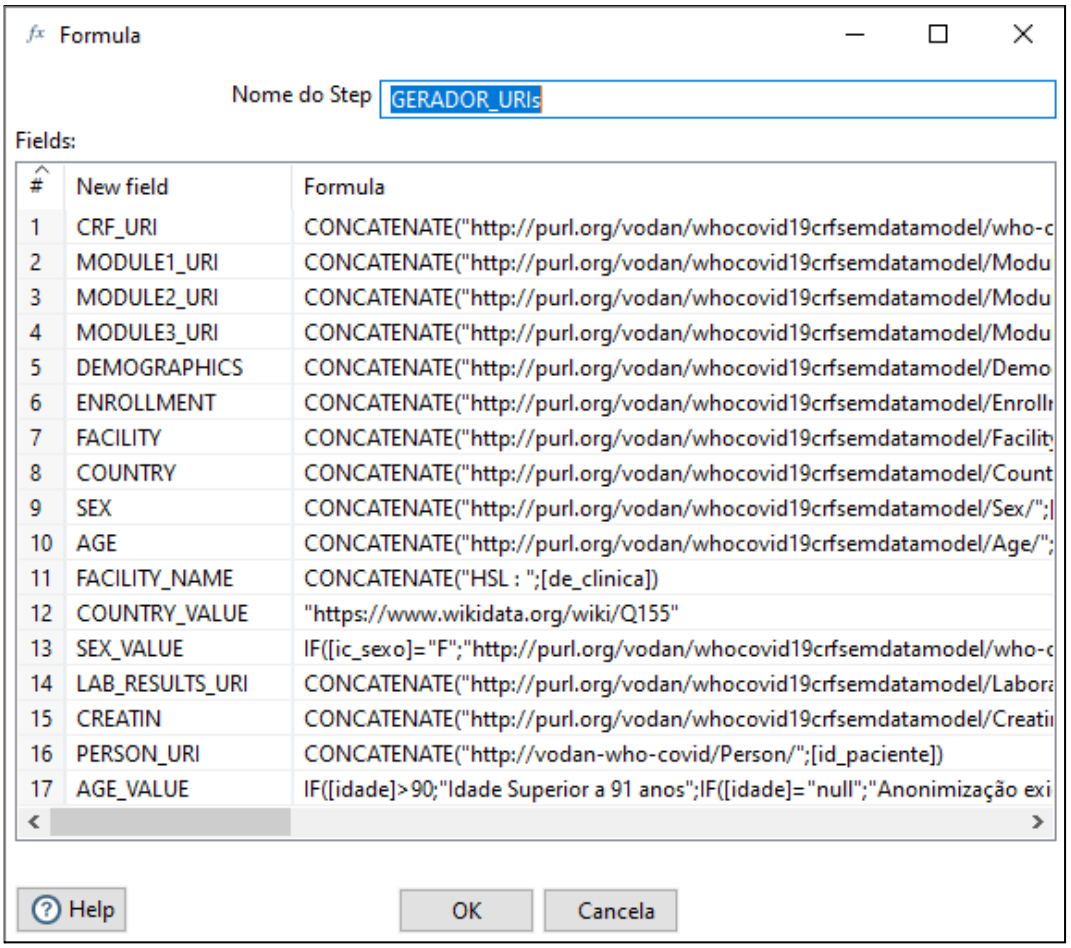


Figura 23 - Gerador de URIs

Observe que, ao realizar a consulta (figura 21), são obtidos os dados dos atendimentos e também é verificada a possibilidade de ser um paciente com recidiva, ou seja, um paciente que já foi internado no passado devido a COVID. Essa verificação acontece através do *JOIN* (junção) da tabela tb_paciente_covid com a tb_desfecho, comparando o identificador único do paciente. É realizado com o propósito de tornar os dados conectados, recuperando, também, os dados de todas as outras possíveis passagens do paciente ao Hospital. Além disso, no Gerador de URIs (figura 22), a grande parte das colunas são derivadas para as URIs correspondentes definidas na ontologia do CRF, com exceção do nome da unidade hospitalar, nome do país em que o hospital está inserido (recuperado pelo *Wikidata*[60] - no caso, Brasil) e os valores de idade, onde um tratamento é exigido quanto a anonimização desses valores (mencionado posteriormente).

Após a criação das URIs, é necessário realizar um tratamento no valor das idades de pacientes, pois como o quantitativo de pessoas acima de 90 anos é muito inferior comparando com o restante, é preciso tornar esses valores de idade anônimos, adequando-se a Lei Geral de Proteção de Dados Pessoais[61] (LGPD). Podemos observar esse tratamento sendo realizado na figura 23, onde duas verificações são realizadas, uma no *step* "idade_maior_90", que valida se paciente tem idade maior que 90, onde o grupo, por ser pequeno, poderia ferir a anonimização, generalizando para maiores de 90. E a outra no *step* "anonym", que verifica se, no conjunto de dados original, a idade já veio com valor anonimizado, referentes a casos requisitados pelo paciente. Após isso, uma função é aplicada para alterar o valor das respostas referentes a idade, de acordo com o definido pela ontologia de referência. Assim definindo uma resposta formatada: idade superior a 91 anos, referente aos casos de mais de 90; anonimização exigida, para os que requisitaram e o preenchimento com o valor da idade para os casos gerais.

[60] https://www.wikidata.org/wiki/Q155
[61] http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm

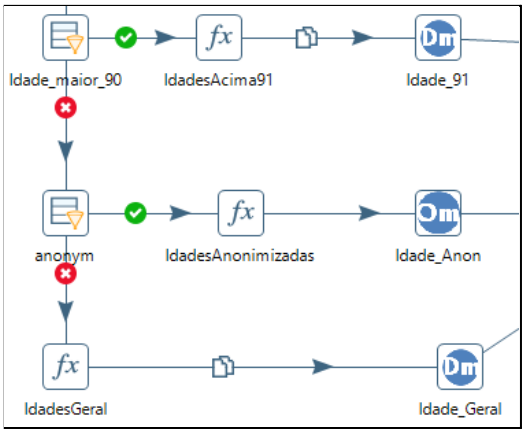


Figura 24 - Tratamento de Idade no Fluxo

Com todas as URIs definidas e as idades tratadas, precisamos anotar todos os dados de acordo com a Ontologia e, finalmente, triplificar os mesmos. Gerando o objeto final, ou seja, todos os dados no modelo FAIR, corretamente triplificados, seguindo a representação correta. Ademais, os conjuntos resultados das triplas são unidos em uma única massa de dados e salvos no *buffer*, isto é, no espaço de memória dedicado do PDI, para serem retornados pelo Módulo de Controle e utilizados para o processo de carga final no *triplestore*.

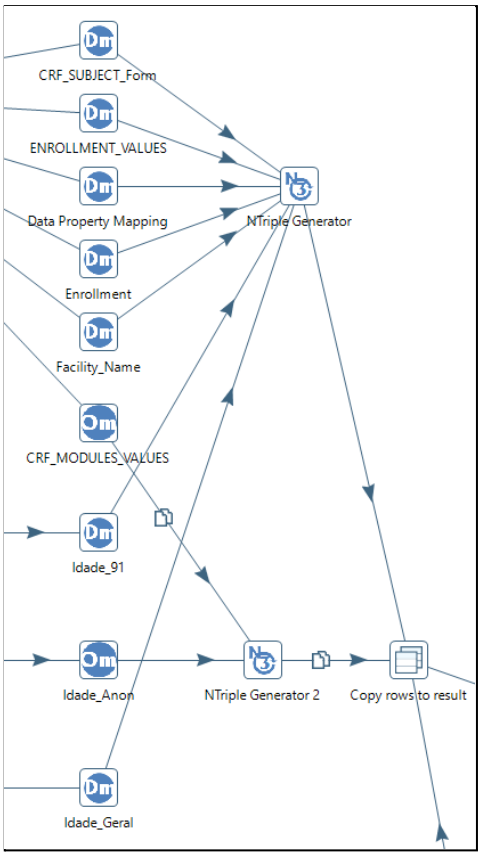


Figura 25 - Anotação e triplificação dos dados

4.3.2 Dados Regionais do Paciente

Além da idade, outro dado importante modelado é quanto a localização geográfica de domicílio do paciente. Tal parte pode ser vista na figura a seguir (figura 25).

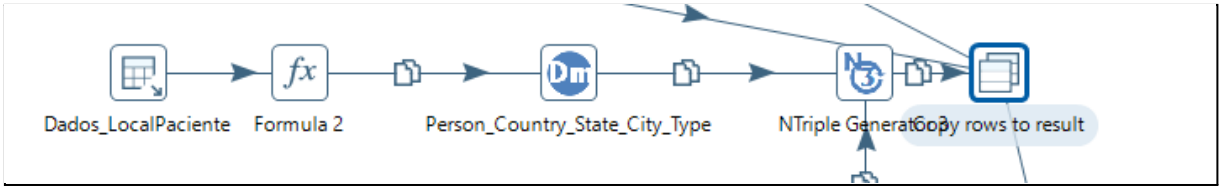


Figura 26 - Tratamento do Local no Fluxo

De acordo com a modelagem feita, um paciente possui dados de país, cidade e estado. Contudo, esses podem variar, havendo países diferentes do Brasil, ou casos de anonimização:

Quadro 7: Atributos regionais do paciente

Atributo	Valor
Country	BR / Estrangeiro
State	CD_UF / Anonimização Exigida
City	Anonimização Exigida

Esses atributos assumem valores dependendo da situação. Os pacientes brasileiros recebem o registro do valor BR, para o atributo *Country*, mas os de nacionalidade diferente são registrados como Estrangeiros. Para os atributos estado e cidade, os pacientes em que não foi exigida a anonimização têm anotado o nome da UF e Cidade, caso contrário é deixado que o dado foi omitido devido à anonimização.

4.3.3 Dados de Exames

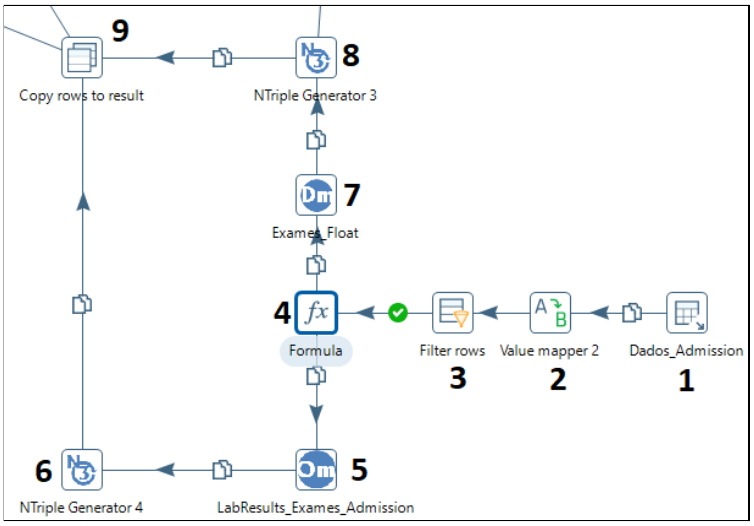


Figura 27 - Tratamento sobre os dados de Exames do Paciente

O último grupo de dados pertinentes à avaliação do paciente em seu momento de internação é, justamente, a coleção referente a suas informações de exames. Seguindo o padrão definido pelo CRF, tem-se uma série de exames, conforme figura 27.

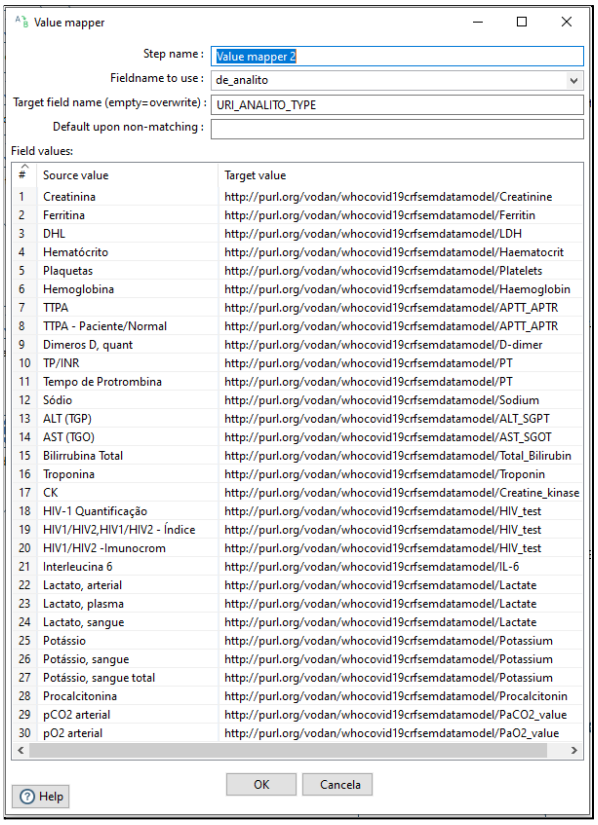


Figura 28 - Lista de Exames aceitos e criação das URIs de Exames

Para recuperar esses dados e utilizá-los no fluxo, uma consulta é realizada na tabela tb_exames_realizados (número 1 da figura 26), cruzando esses com a tb_paciente_covid, para garantir que está se conectando corretamente Exames a Pacientes.

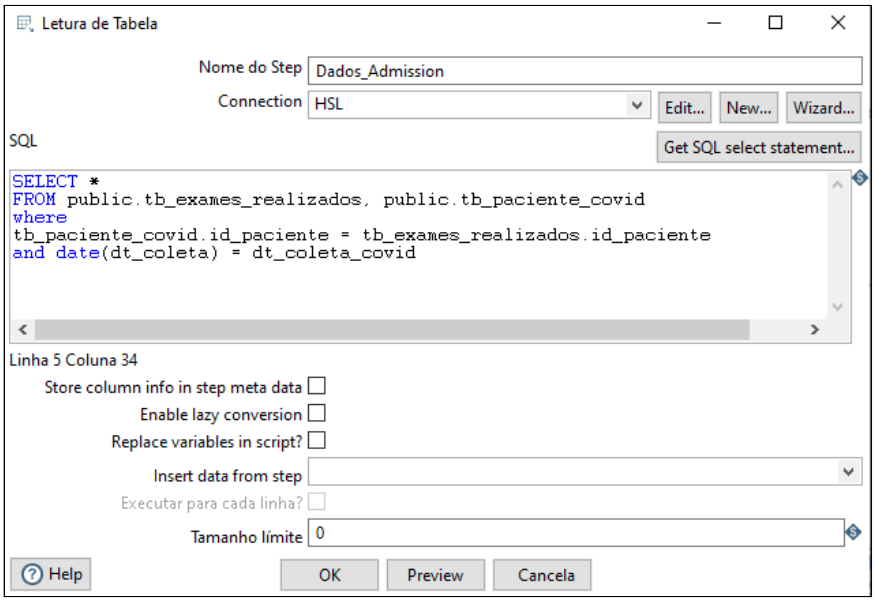


Figura 29 - Consulta para obter os dados de Exames no momento de internação

Com os dados, é realizada a criação das URIs correspondentes a todos os campos da consulta (número 2 da figura 26 e figura 27), assim como a URI que identifica tanto esse conjunto de exames a um atendimento quanto o próprio laboratório (número 4 da figura 26 e figura 29).

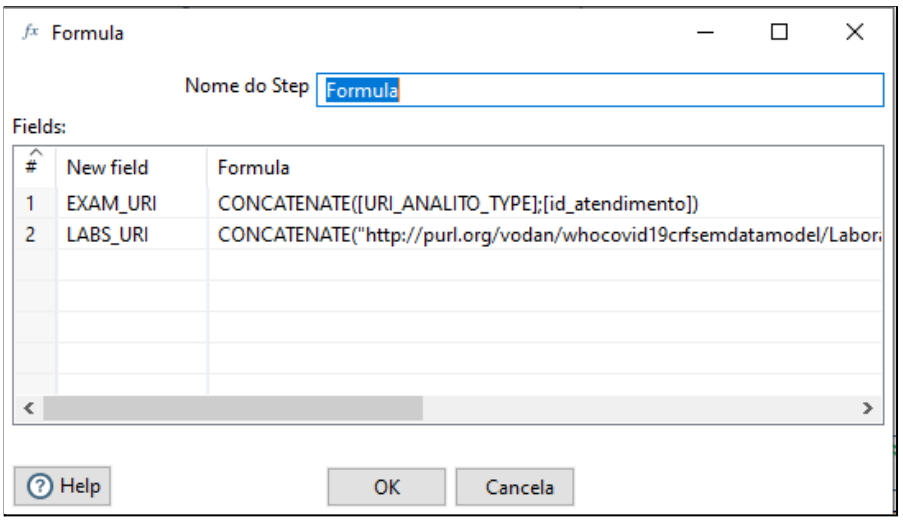


Figura 30 - Criação de URIs sobre Exame e Laboratório

Por fim, todos os dados são anotados (números 5 e 7 da figura 26) e triplificados (números 6 e 8 da figura 26) de acordo com a ontologia definida e salvos no *buffer* do PDI (número 9 da figura 26) para retorno no módulo de controle.

4.4 MÓDULO DE ACOMPANHAMENTO

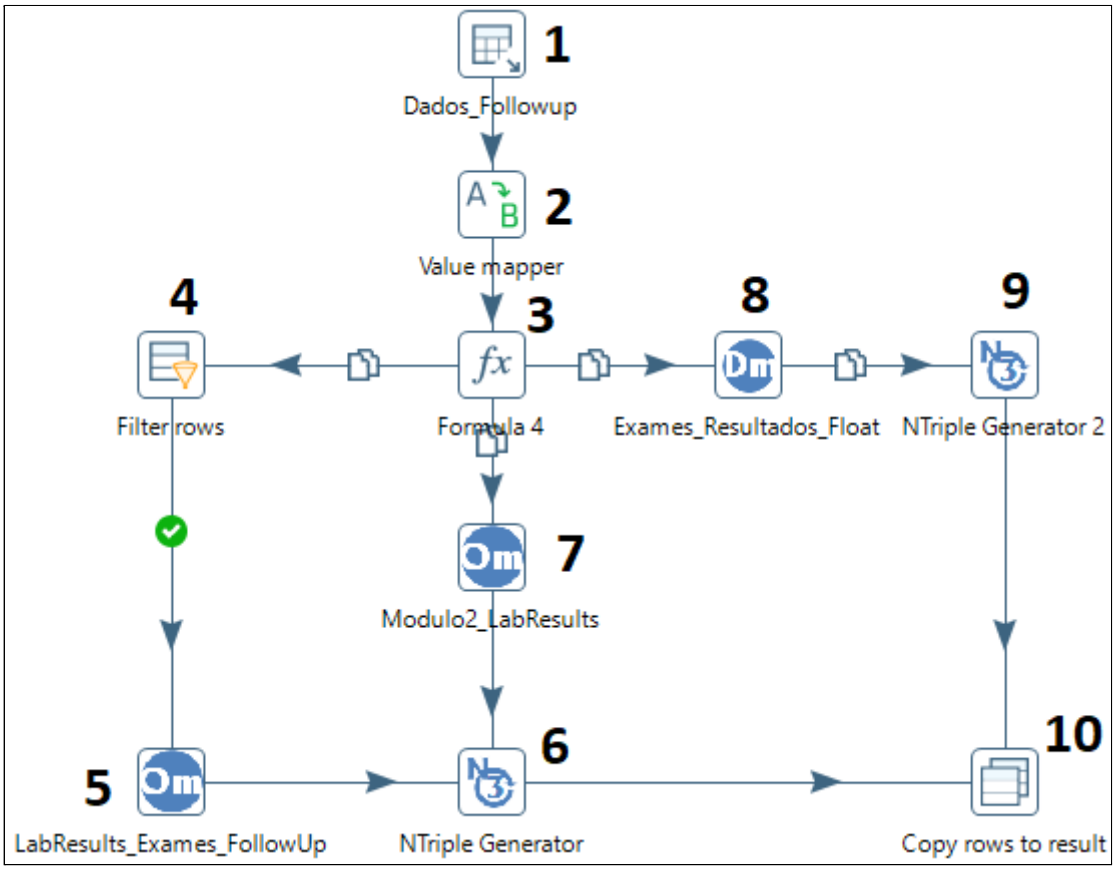


Figura 31 - Fluxo de dados do Módulo de Acompanhamento

Nesta etapa, o objetivo é anotar e triplificar os dados que são gerados ao longo da permanência do paciente no hospital e não os referentes a sua chegada. Isto significa que todo o fluxo de transformação trabalha sobre os dados de acompanhamento por exames do paciente. Dessa forma, aplicando a modelagem da ontologia. A associação se dá por termos que descrevem os exames e os resultados numéricos ou descritos como termo também definido.

A recuperação desse conjunto de dados é através de uma consulta (número 1 da figura 30 e figura 31) nas tabelas tb_desfecho, tb_exames e tb_paciente_covid. Essa consulta é

diferente das outras, sendo a mais complexa, pois deve-se conectar os dados de Exames a Pacientes corretamente, garantindo que o seu retorno é somente de dados que foram gerados para aquela internação durante o seu período de vigência, ou seja, logo após a entrada no hospital e imediatamente anterior ao desfecho. Como retorno, tem-se algo muito semelhante ao da figura 26, estruturalmente falando, já que as colunas continuam as mesmas. A diferença será no processo de geração de URIs e anotações desses dados.

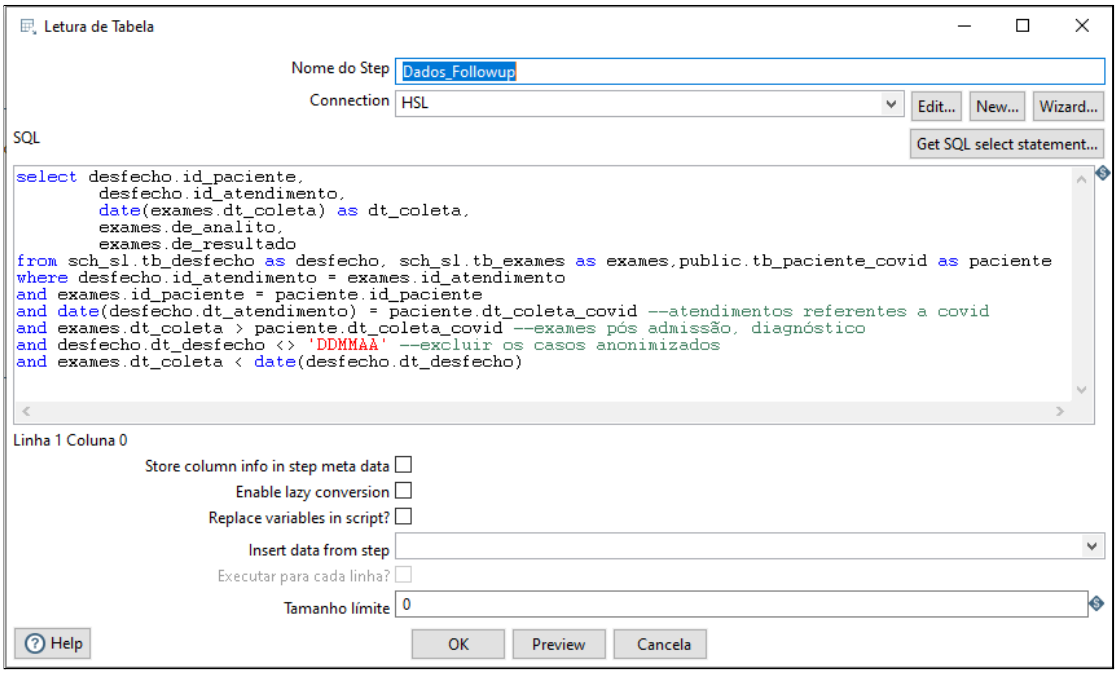


Figura 32 - Consulta para obter dados de Acompanhamento do paciente

Como esses dados são de evolução do paciente, sua anotação, seguindo o padrão estabelecido na ontologia, é diferente, porém, as informações apresentadas são essencialmente as mesmas, um novo mapeamento é necessário. Visto que é preciso, no grupo de dados triplificados ao fim, ter a capacidade de diferenciação de um termo da ontologia para o de uma instância. Para isso, as URIs são diferentes, onde é inserido um caractere "_" ao fim das URIs de cada exame (número 2 da figura 30 e figura 32). Além disso, a URI receberá a identificação do atendimento e data, diferenciando os exames.

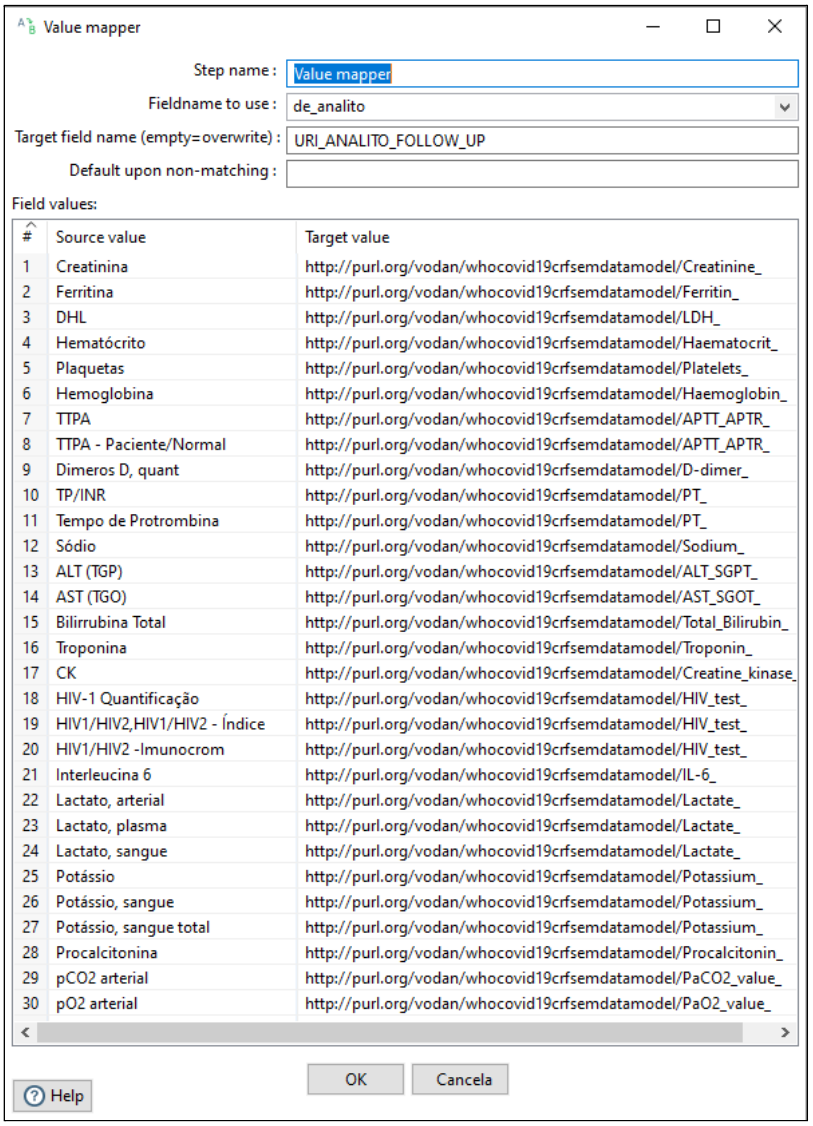


Figura 33 - Criação das URIs de Exames de Acompanhamento

De posse das URIs de cada exame específico, é necessário criar as URIs desse conjunto de exames, do laboratório e do módulo em si, para anotar corretamente todos os dados (número 3 da figura 30 e figura 33). Essas URIs são diferentes, pois possuem o termo *FollowUP* ao fim do agrupamento dos resultados laboratoriais, garantindo a diferenciação de um dado evolutivo a um de admissão.

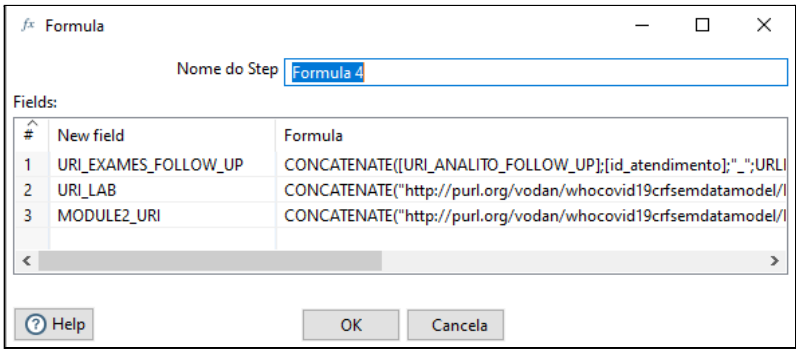


Figura 34 - Criação das URIs do conjunto de exames, laboratório e módulo

Após a criação das URIs, esses dados são anotados (números 5, 7 e 8 da figura 30), triplificados (números 6 e 9 da figura 30) e, ao fim, uma massa é gerada com essas informações e salva no *buffer* do PDI para recuperação no módulo de controle (número 10 da figura 30).

4.5 MÓDULO DE DESFECHO

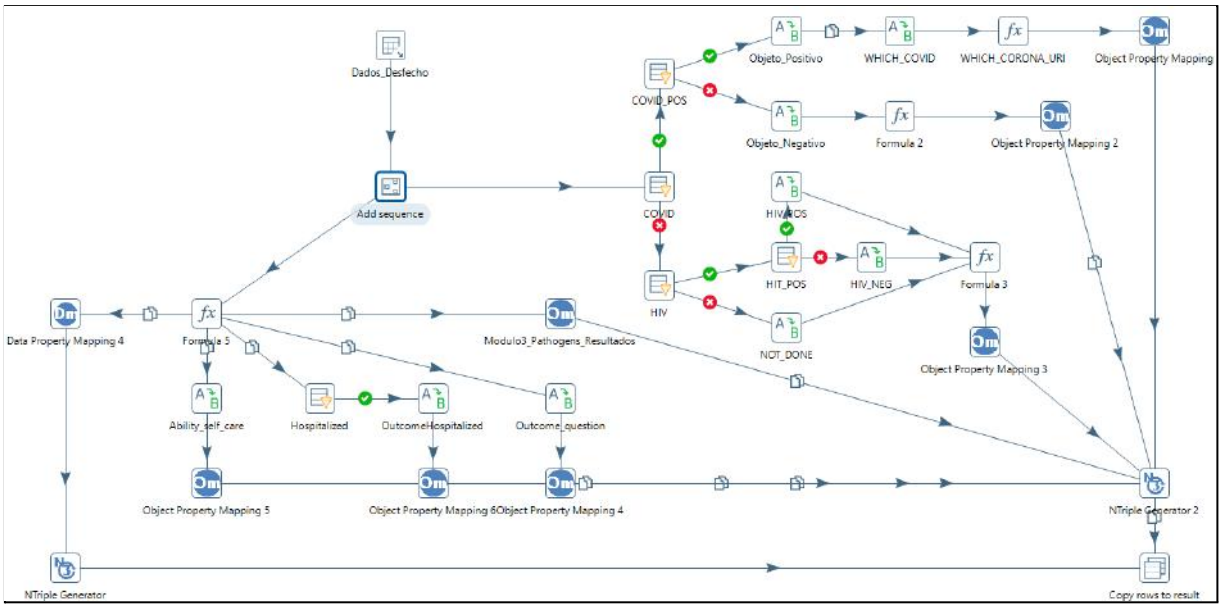


Figura 35 - Fluxo de dados do Módulo de Desfecho

Responsável por anotar e triplificar os dados referentes ao desfecho da internação, ou seja, do encerramento do acompanhamento do paciente. Se um paciente possui desfecho registrado, seus exames de COVID e HIV são processados. As informações do tipo de

desfecho como: melhor, óbito, e etc. são mapeadas com os termos no outcome_question (número 7 da figura 37). Outro dado é sobre a situação do paciente, baseado no desfecho, que é definida com termos do vocabulário controlado, referindo à parte ability_self_care (número 5 da figura 37).

A fim de obter o volume de dados referente ao desfecho de cada paciente para uso no fluxo de transformações, é realizada uma consulta nas tabelas tb_desfecho, tb_exames e tb_paciente_covid. De modo que as informações de desfecho de cada paciente bem como seu acompanhamento, se houve algum, e seus dados (figura 35).

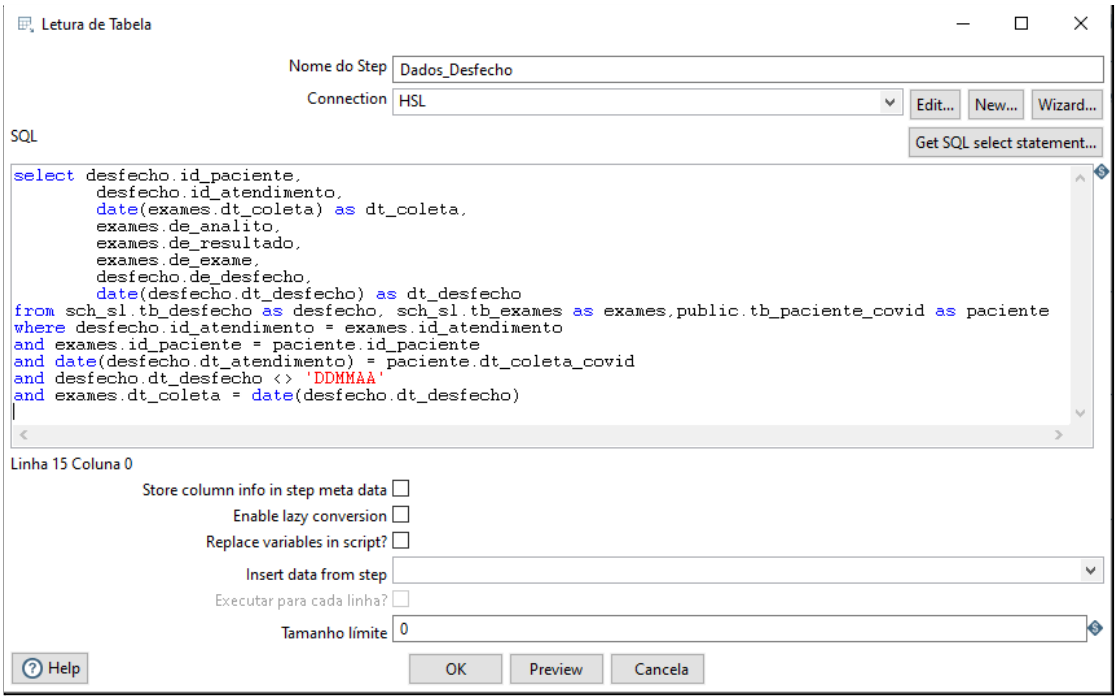


Figura 36 - Consulta para obter os dados de Desfecho dos pacientes

Em seguida, o fluxo é separado em duas partes, uma que vai ficar responsável por anotar e triplificar os resultados de desfechos (figura 37) e a outra por realizar o mesmo processo só que sobre os dados dos exames de COVID e HIV (figura 38).

Observe que, na figura 37, uma parte do fluxo já segue para anotação (número 2 e 3), triplificação (número 4 e 11) e salvamento no *buffer* (número 12), que é sobre a data de desfecho do paciente e resultados de Patógenos. Ainda acerca dessa parte do fluxo, ocorre a anotação e triplificação das informações de qual foi o desfecho (número 7 e 10 da figura 37), a situação do paciente (número 5 e 8 da figura 37) e, por mim, da própria hospitalização (número 6 e 9 da figura 37).

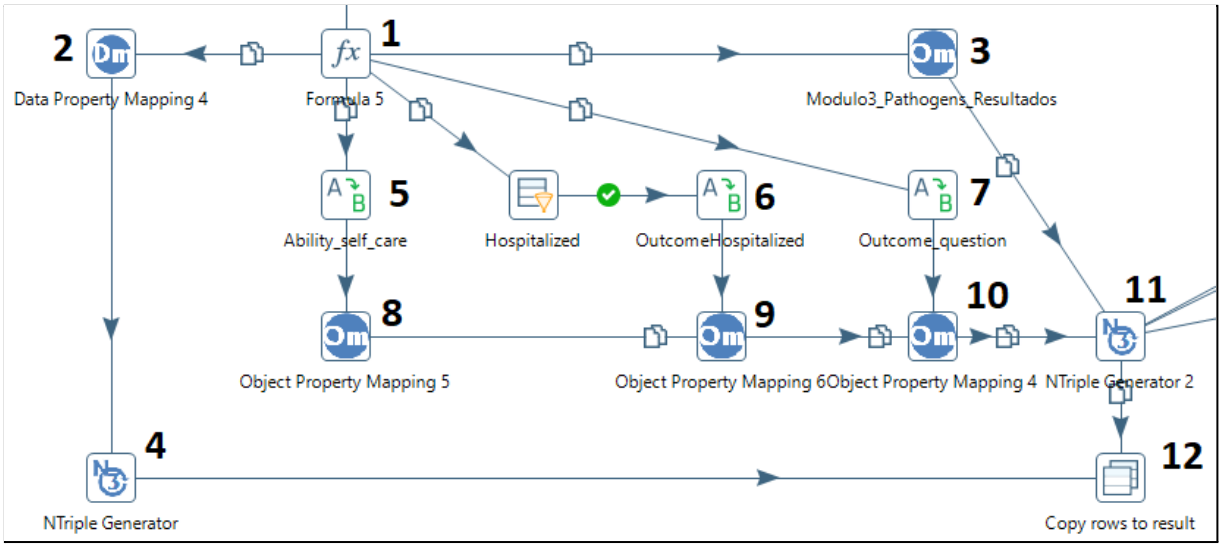


Figura 37 - Parte do Módulo responsável pela anotação e triplificação dos dados de Desfecho

Não obstante, na figura 38, é exibido a segunda parte dessa transformação, onde o foco é sobre a avaliação final da presença ou não de evidência sorológica de infecção causada pelo vírus da COVID-19 ou HIV. Novamente, o foco é em gerar as URIs correspondentes, mapeando os termos segundo a definição do vocabulário definido, anotando e triplificando esses dados.

Note que, uma série de verificações são realizadas logo ao início do processo de transformação (números 1, 2, 9 e 11 da figura 38). Onde, as primeiras (números 1 e 9 da figura 38), são para validar a existência ou não de dados dos exames de COVID-19 ou HIV no prontuário do paciente. Uma observação importante é que, como o objeto maior do estudo foi auxiliar na pandemia de COVID-19, a existência de exames de COVID já torna desnecessário a verificação da existência de exames sobre HIV, pois é considerado uma internação causada diretamente por possível infecção de COVID-19. Posteriormente, detectado a presença de dados sobre um dos exames, é verificado seu resultado, se é positivo ou negativo (números 2 e 11 da figura 38) e as URIs desse resultado são geradas (números 3, 4, 5, 10, 12 e 13 da figura 39), bem como suas respectivas anotações (números 7, 8 e 15 da figura 38) e a triplificação final de todo esse conjunto de dados (número 16 da figura 38).

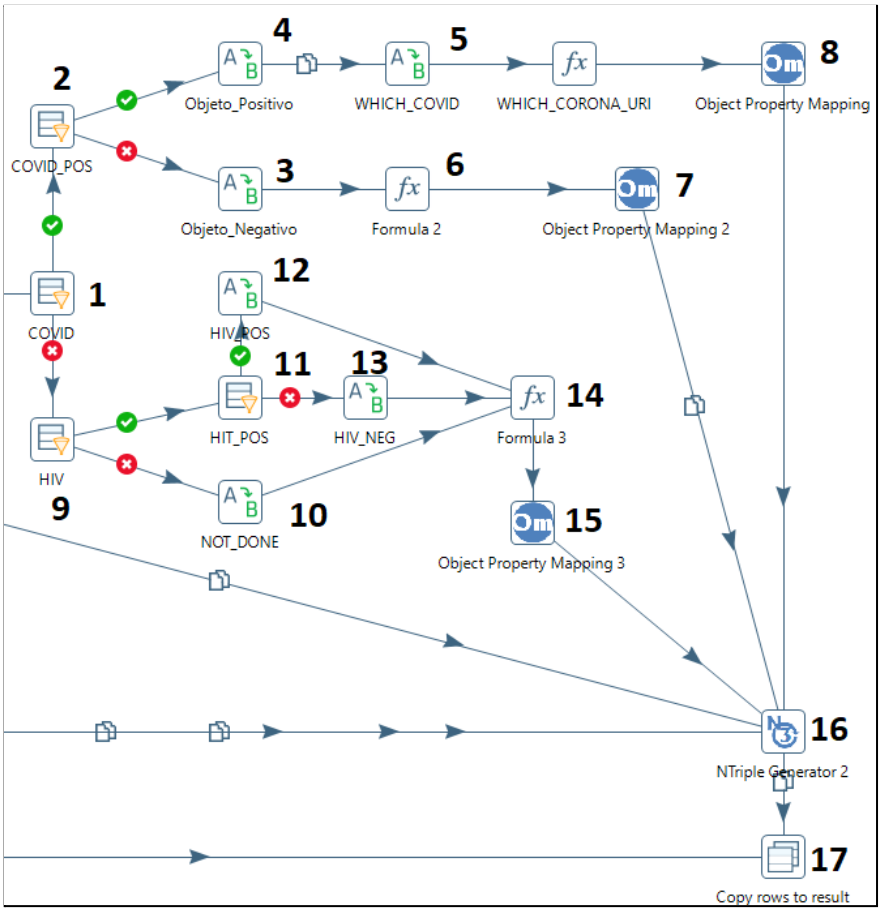


Figura 38 - Parte do Módulo responsável pela anotação e triplificação dos dados de Exames

Ao fim da execução de todos os módulos, o *buffer* do PDI contém todos os dados já no modelo correto para seu uso no VODAN-BR, ou seja, os dados já foram tratados e triplificados, com sua semântica completamente anotada segundo os padrões estipulados pela ontologia do CRF, gerando, assim, a proveniência desse processo. Nesse sentido, o Módulo de Controle realiza a união dos três conjuntos de dados resultantes de cada um dos Módulos de Dados (números 4 e 5 da figura 18), bem como a criação de um arquivo com todas as triplas no formato .n3 (para uso posterior - número 6 da figura 19) e a carga desses dados em um *triplestore*, nesse caso, o GraphBD (número 7 da figura 19).

Na carga, passamos as configurações de URL do *triplestore*, se vamos utilizar um repositório existente ou não, assim como o nome do repositório, o formato do arquivo, seu caminho - nesse caso recuperado do passo anterior - e o nome do grafo que queremos armazenar os dados para exploração no *triplestore*, conforme imagem 39. Definido os parâmetros, o *plugin* estabelece conexão com o *triplestore*, cria o novo repositório (se for definido nos parâmetros), cria o grafo nomeado e realizada a carga, vide figuras 40 e 41.

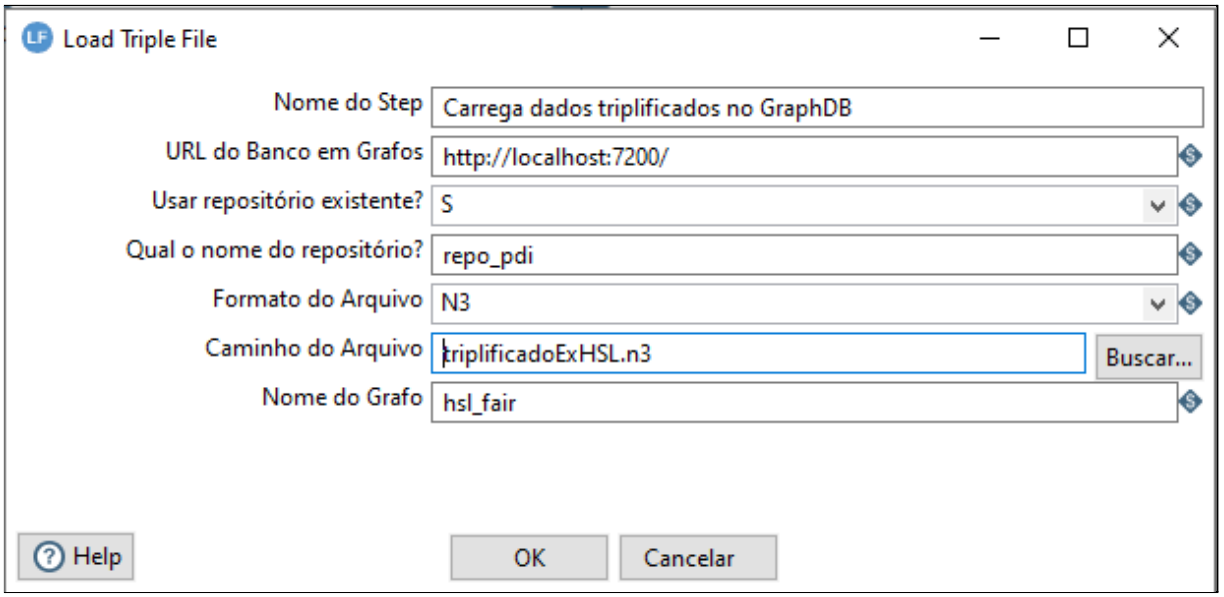


Figura 39 - Configuração do *Load Triple File*

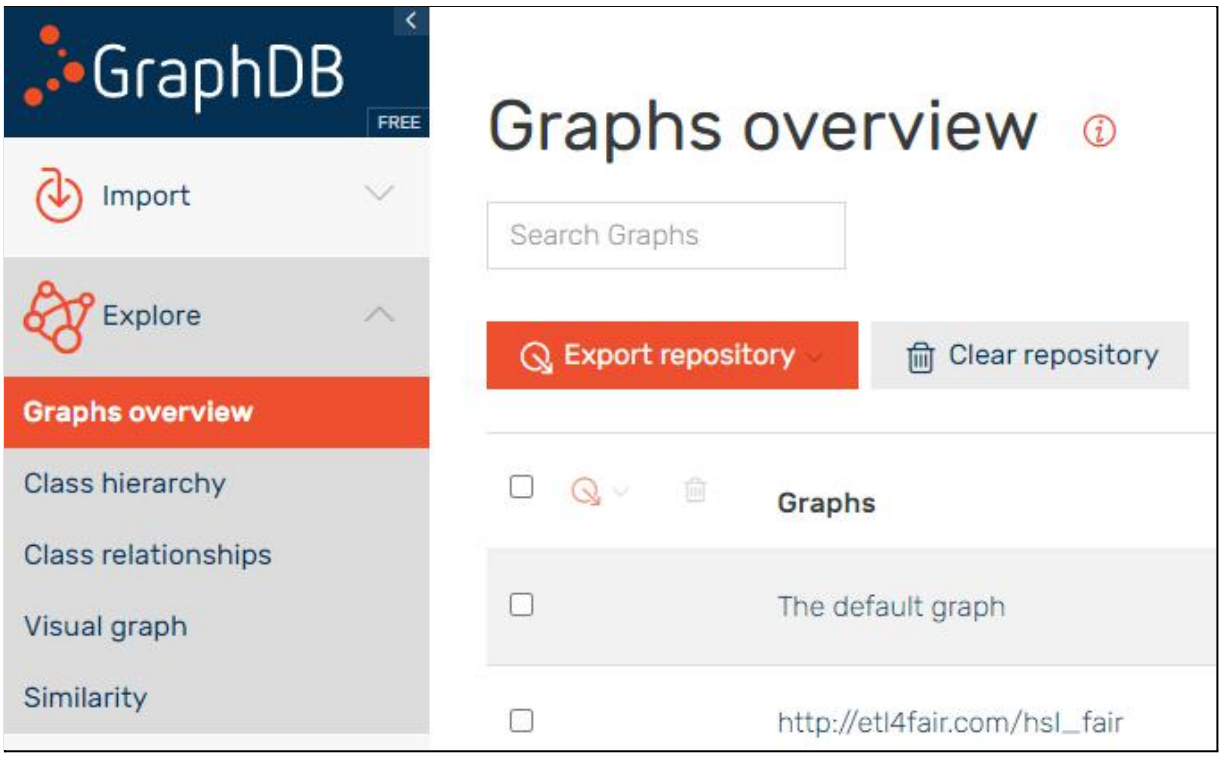


Figura 40 - Grafo criado no GraphDB

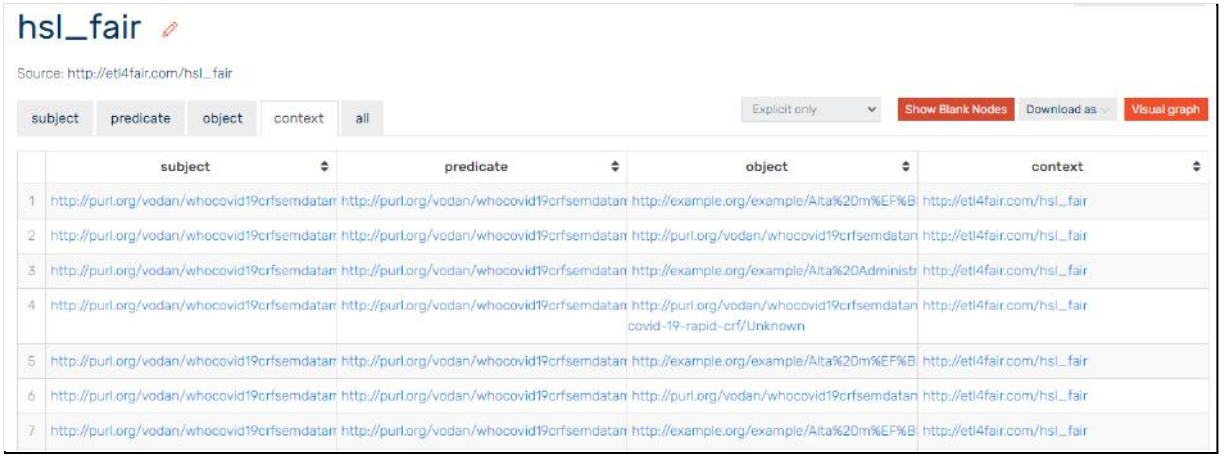


Figura 41 - Visualização dos dados inseridos no GraphDB

Em função disso, é finalizado todo o trabalho de FAIRificação com os dados epidemiológicos de COVID-19 sobre registros e acompanhamento de internações de paciente no HSL, seguindo a estrutura definida no Formulário de Relato de Casos Clínicos Global de COVID-19 e possibilitando o uso desses dados em diversas futuras pesquisas.

4.6 MÓDULO DE METADADOS

Uma vez transformados os dados, esses possuem uma camada de anotação e descrição referente à semântica aplicada. Contudo, há de se destacar a necessidade e existência de mais informações dessa natureza descritiva.

O próprio processo no qual os dados são submetidos acaba gerando um grupo de metadados descritivos. Esses, apresentam uma grande importância no quesito de identificação da forma em que os dados foram gerados, bem como sua origem. Esse ponto traz, de uma forma básica, o conceito de proveniência, agregando dados que possam descrever o caminho desse conjunto.

Ressalta-se que, nos princípios FAIR, a descrição de como são os dados, bem como do próprio *dataset* que os comporta, é essencial para o reuso em novos contextos e aplicações. Esse conjunto de descrições define os metadados. Esses, tal como os dados, devem ser apresentados por meio de uma anotação semântica e em formato de triplas. Para proporcionar gerência e administração desse novo conjunto de descritores de dados, o FAIR Data Point

disponibiliza uma estrutura de anotação básica para os elementos a serem armazenados em seu repositório, contemplando *Catalog*, *Dataset* e *Distribution*.

Assim, visando cumprir essa necessidade, o conjunto de transformações traz mais uma parte, agora, voltada para esse esforço de coleta e adequação dos metadados para serem propriamente submetidos ao FAIR Data Point, com o auxílio do ETL4FAIR.

A fim de realizar os testes, no primeiro momento, é definido um conjunto mínimo de metadados a serem utilizados, com base no processo de ETL. Esses são:

Quadro 8: Atributos do elemento *Catalog*

Termo	Descrição
http://purl.org/dc/terms/description	Descrição
http://purl.org/dc/terms/hasVersion	Versão
http://purl.org/dc/terms/isPartOf	Elemento Superior
http://purl.org/dc/terms/language	Língua
http://purl.org/dc/terms/license	Licença de uso
http://purl.org/dc/terms/publisher	Publicador
http://purl.org/dc/terms/title	Título

Quadro 9: Atributos do elemento *Dataset*

Termo	Descrição
http://purl.org/dc/terms/description	Descrição
http://purl.org/dc/terms/hasVersion	Versão
http://purl.org/dc/terms/isPartOf	Elemento Superior
http://purl.org/dc/terms/language	Língua
http://purl.org/dc/terms/license	Licença de uso
http://purl.org/dc/terms/publisher	Publicador
http://purl.org/dc/terms/title	Título

http://www.w3.org/ns/dcat#theme	Temas relacionados

Quadro 10: Atributos do elemento *Distribution*

Termo	Descrição
http://purl.org/dc/terms/description	Descrição
http://purl.org/dc/terms/hasVersion	Versão
http://purl.org/dc/terms/isPartOf	Elemento Superior
http://purl.org/dc/terms/language	Língua
http://purl.org/dc/terms/license	Licença de uso
http://purl.org/dc/terms/publisher	Publicador
http://purl.org/dc/terms/title	Título
http://www.w3.org/ns/dcat#downloadURL	URL com conteúdo
http://www.w3.org/ns/dcat#mediaType	Tipo do conteúdo

Uma vez feito o levantamento dos dados requisitados, esses são organizados em arquivos CSV, sendo um arquivo com o conteúdo de *Catalog* (Apêndice A), de *Dataset* (Apêndice B) e de *Distribution* (Apêndice C).

De forma similar ao tratamento dos dados, os metadados organizados são adequados ao formato de triplas. Após a transformação, devem ser armazenados de forma apropriada, ou seja, publicados no FAIR Data Point. A fim de alcançar esse objetivo, o último módulo, o Módulo de Metadados, é construído e inserido (número 8 figura 20) no processo da transformação geral. A figura 42 apresenta a estruturação desse Módulo:

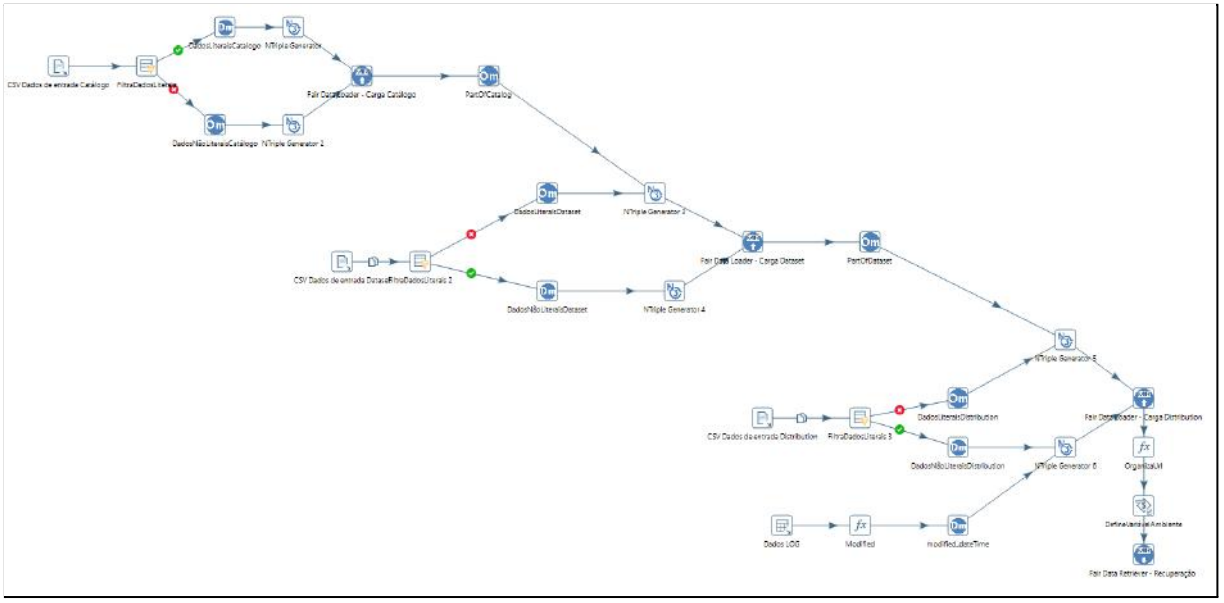


Figura 42 - Processamento total dos metadados

Na primeira parte, na figura 43, são obtidos os metadados preenchidos referentes ao *Catalog*. Esses são separados em dados literais e não literais para devido processamento pelo "DadosLiteraisCatálogo" do tipo *Data Property Mapping*, sendo os outros pelo "DadosNãoLiteraisCatálogo" do tipo *Object Property Mapping*. Em seguida, são serializados em triplas pelo *NTriple Generator* obtendo essas formatadas em NTriple. Por fim, o conteúdo serializado é carregado no FAIR Data Point através do plugin *FAIR Data Loader*, configurado como na figura 47. Destaca-se a possibilidade de o elemento já ser publicado. Como resultado, o *plugin* retorna a tripla que referencia o elemento criado a fim de atrelar subcomponentes em seguida. Essa última condição é realizada através do "PartOfCatalog" do tipo *Object Property Mapping*.

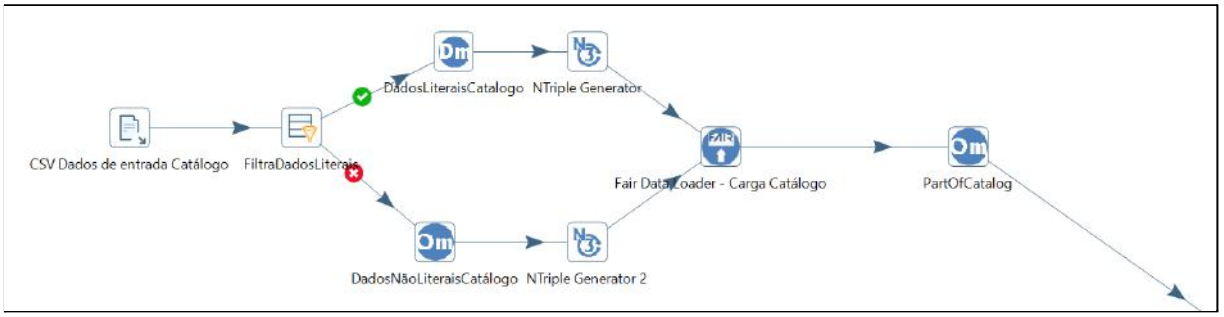


Figura 43 - Processamento dos metadados do *Catalog*

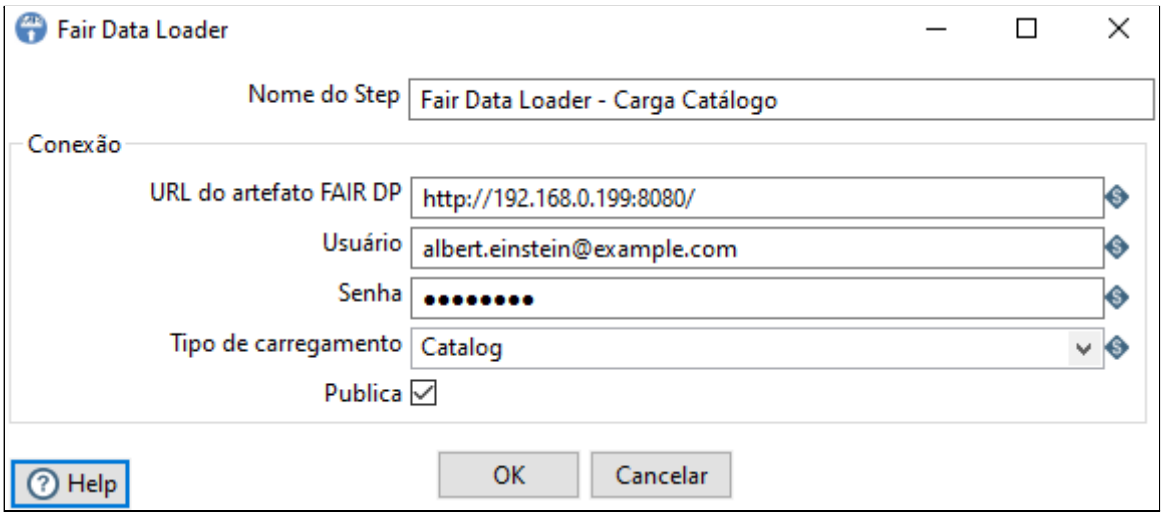


Figura 44 - Configuração da carga e geração do *Catalog*

Em seguida, como pode ser visto na figura 45, os dados referentes ao *Dataset* passam pelo mesmo processo. São obtidos, condicionados em triplas e serializados em *NTriple*. Um ponto de diferença é a tripla do elemento superior, o Catálogo, essa é adicionada às demais condicionadas e todas são carregadas no FAIR Data Point, configurado segundo a figura 46.

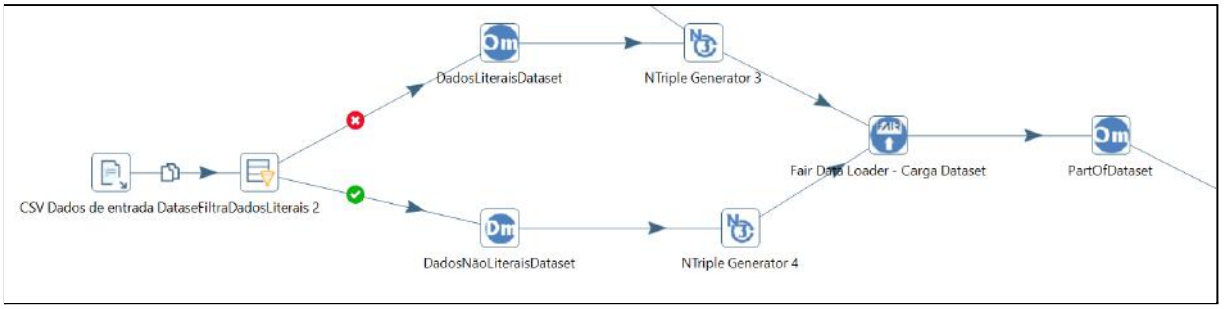


Figura 45 - Processamento dos metadados do *Dataset*

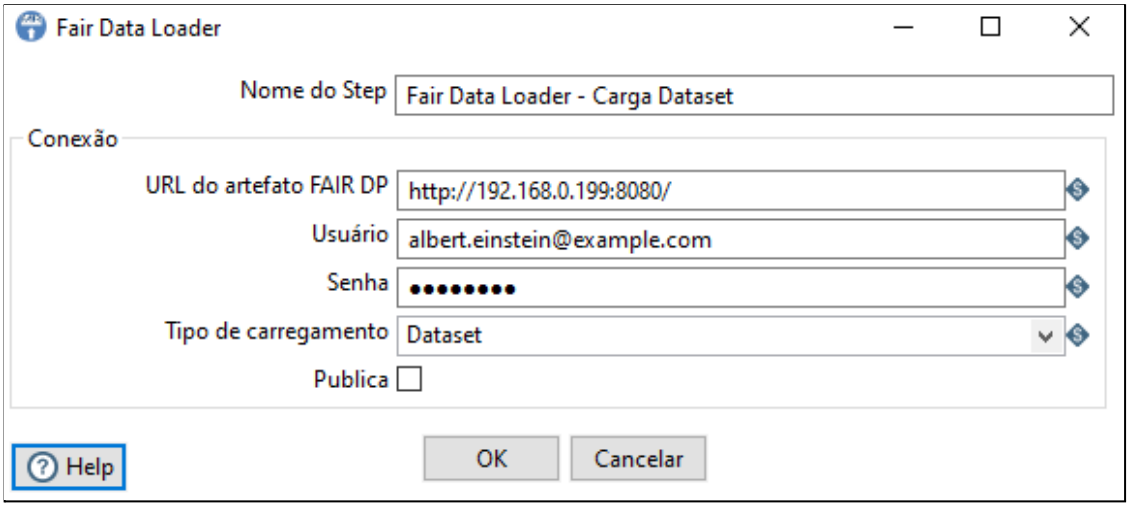


Figura 46 - Configuração da carga e geração do *Dataset*

Por fim, de forma similar, os dados que descrevem o *Distribution*, passam pelo mesmo processo (figura 47 e 48). Um último dado, contudo, é obtido através do sistema do PDI, através do registro de *log* armazenado em um banco de dados local. O dado obtido é da data final da transformação, definindo a propriedade *issued* nos metadados referentes ao conjunto de dados gerados no processo total descrito. Ainda mais, o endereço de referência do banco de triplas onde os dados foram carregados também é adicionado ao conjunto de metadados,a fim de propiciar futuro uso e recuperação do armazenado.

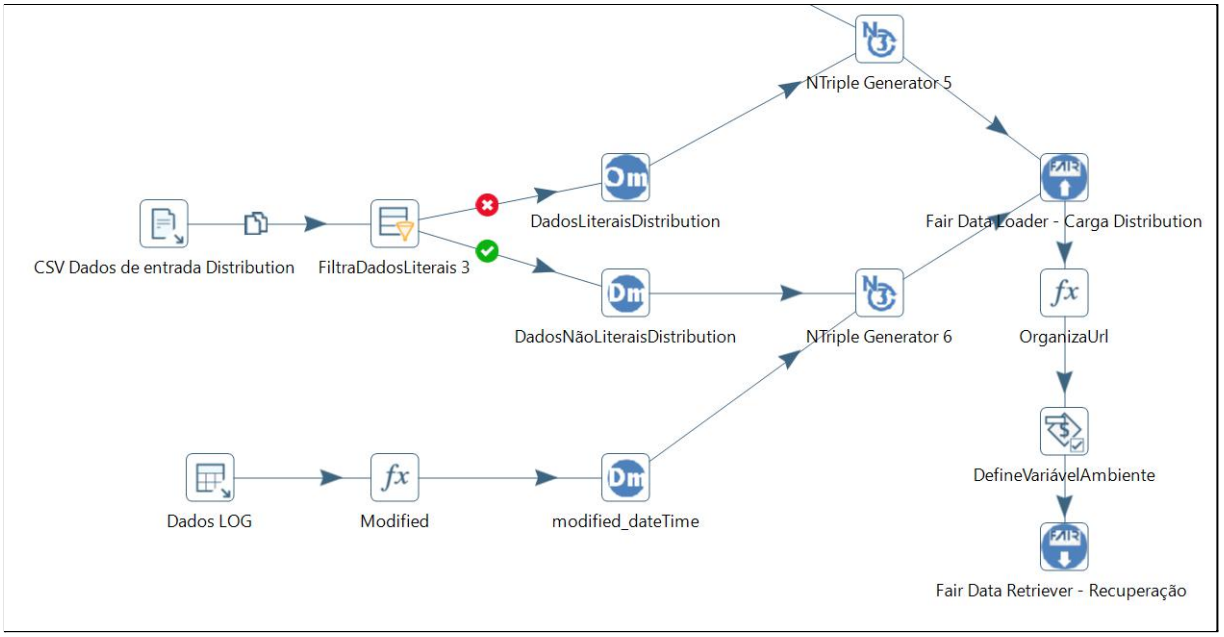


Figura 47 - Processamento dos metadados de *Distribution*

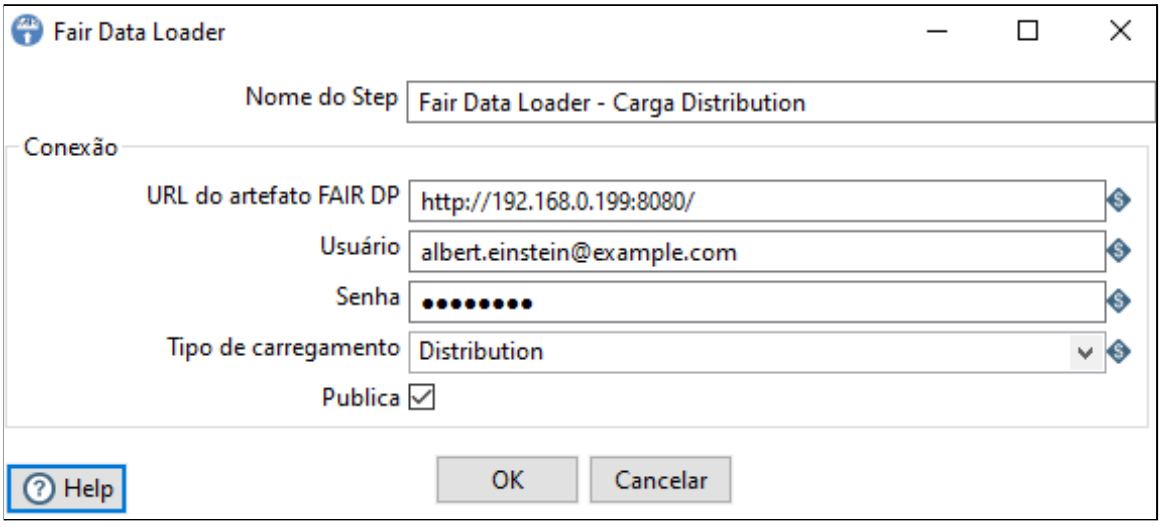


Figura 48 - Configuração da carga e geração de *Distribution*

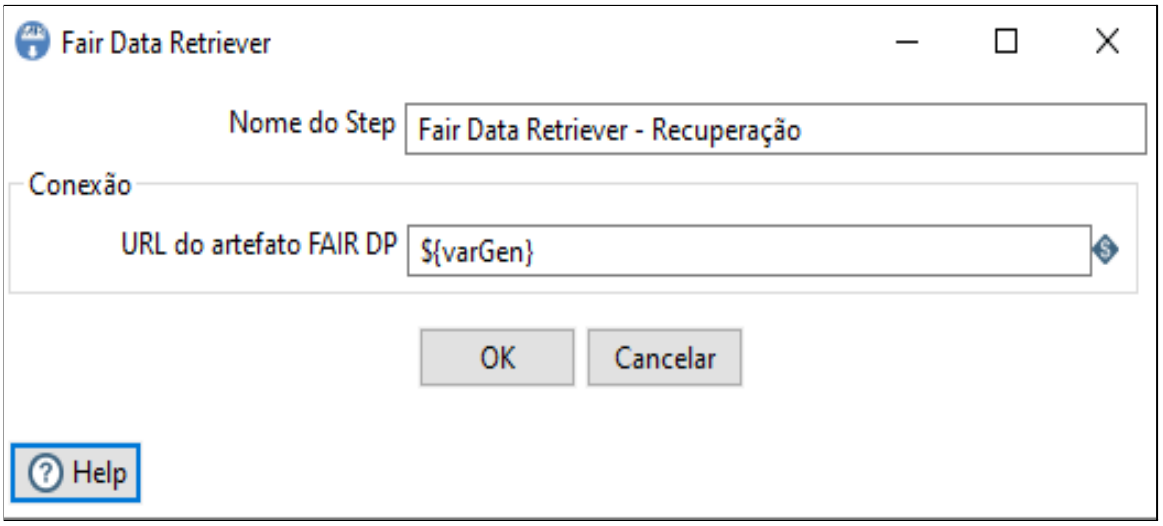
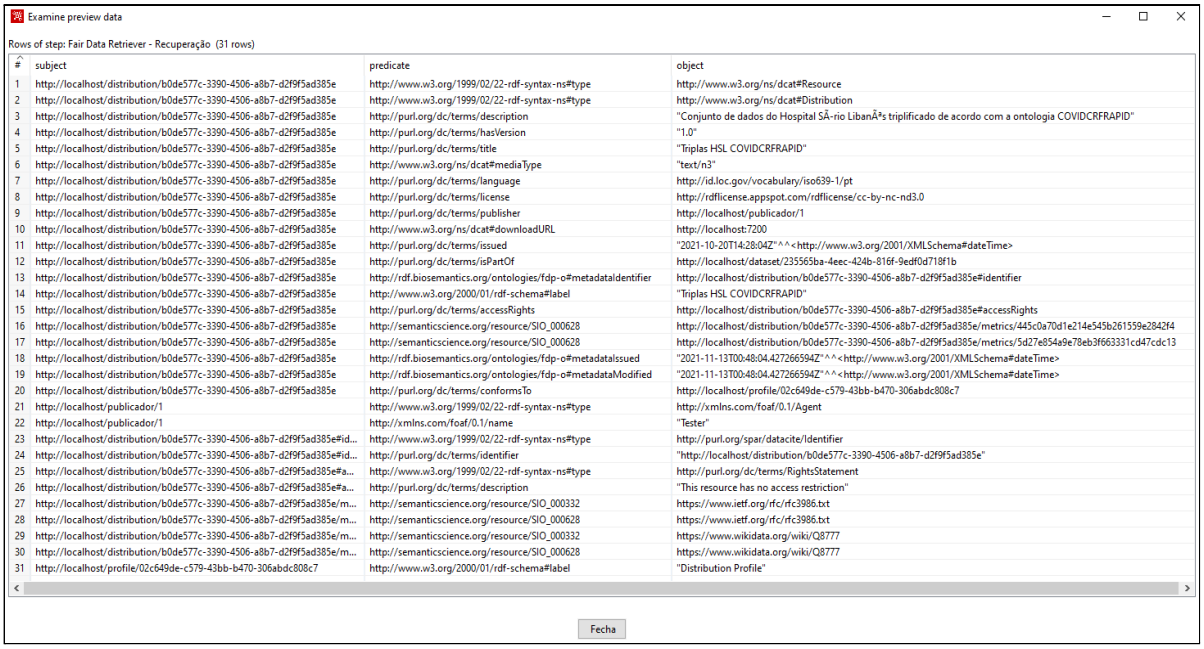


Figura 49 - Configuração da carga e geração de *Distribution*



#	subject	predicate	object
1	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Resource
2	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Distribution
3	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/description	"Conjunto de dados do Hospital SÃ-rio LibanÃªs triplificado de acordo com a ontologia COVIDCRFRAPID"
4	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/hasVersion	"1.0"
5	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/title	"Triplas HSL COVIDCRFRAPID"
6	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://www.w3.org/ns/dcat#mediaType	"text/n3"
7	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/language	http://id.loc.gov/vocabulary/iso639-1/pt
8	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/license	http://rdflicense.appspot.com/rdflicense/cc-by-nc-nd3.0
9	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/publisher	http://localhost/publicador/1
10	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://www.w3.org/ns/dcat#downloadURL	http://localhost:7200
11	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/issued	"2021-10-20T14:28:04Z"^^<http://www.w3.org/2001/XMLSchema#dateTime>
12	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/isPartOf	http://localhost/dataset/235565ba-4eec-424b-816f-9edf0d718f1b
13	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://rdf.biosemantics.org/ontologies/fdp-o#metadataIdentifier	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#identifier
14	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://www.w3.org/2000/01/rdf-schema#label	"Triplas HSL COVIDCRFRAPID"
15	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/accessRights	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#accessRights
16	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://semanticscience.org/resource/SIO_000628	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/metrics/445c0a70d1e214e545b261559e2842f4
17	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://semanticscience.org/resource/SIO_000628	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/metrics/5d27e854a9e78eb3f663331cd47cdc13
18	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://rdf.biosemantics.org/ontologies/fdp-o#metadataIssued	"2021-11-13T00:48:04.427266594Z"^^<http://www.w3.org/2001/XMLSchema#dateTime>
19	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://rdf.biosemantics.org/ontologies/fdp-o#metadataModified	"2021-11-13T00:48:04.427266594Z"^^<http://www.w3.org/2001/XMLSchema#dateTime>
20	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e	http://purl.org/dc/terms/conformsTo	http://localhost/profile/02c649de-c579-43bb-b470-306abdc808c7
21	http://localhost/publicador/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Agent
22	http://localhost/publicador/1	http://xmlns.com/foaf/0.1/name	"Tester"
23	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#id...	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/spar/datacite/Identifier
24	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#id...	http://purl.org/dc/terms/identifier	"http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e"
25	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#a...	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/dc/terms/RightsStatement
26	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e#a...	http://purl.org/dc/terms/description	"This resource has no access restriction"
27	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/m...	http://semanticscience.org/resource/SIO_000332	https://www.ietf.org/rfc/rfc3986.txt
28	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/m...	http://semanticscience.org/resource/SIO_000628	https://www.ietf.org/rfc/rfc3986.txt
29	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/m...	http://semanticscience.org/resource/SIO_000332	https://www.wikidata.org/wiki/Q8777
30	http://localhost/distribution/b0de577c-3390-4506-a8b7-d2f9f5ad385e/m...	http://semanticscience.org/resource/SIO_000628	https://www.wikidata.org/wiki/Q8777
31	http://localhost/profile/02c649de-c579-43bb-b470-306abdc808c7	http://www.w3.org/2000/01/rdf-schema#label	"Distribution Profile"

Figura 50 - *Preview* do FAIR *Data Retriever* recuperando o elemento *Distribution* gerado

Após carga dos metadados no FAIR Data Point, esses estão disponíveis para recuperação e consulta. Uma das formas de se averiguar a carga é pela recuperação, através do *plugin Fair Data Retriever* (figura 49), obtendo os dados gerados do FAIR Data Point, por meio da URL contida na variável de ambiente *varGen*. O preview de extração pode ser visto na figura 50. Outra forma de acesso é pelo *client* do FAIR Data Point, pode ser visualizada a seguir, o repositório (figura 51), o *Catalog*(figura 52), *Dataset* (figura 53) e *Distribution*(figura 54).

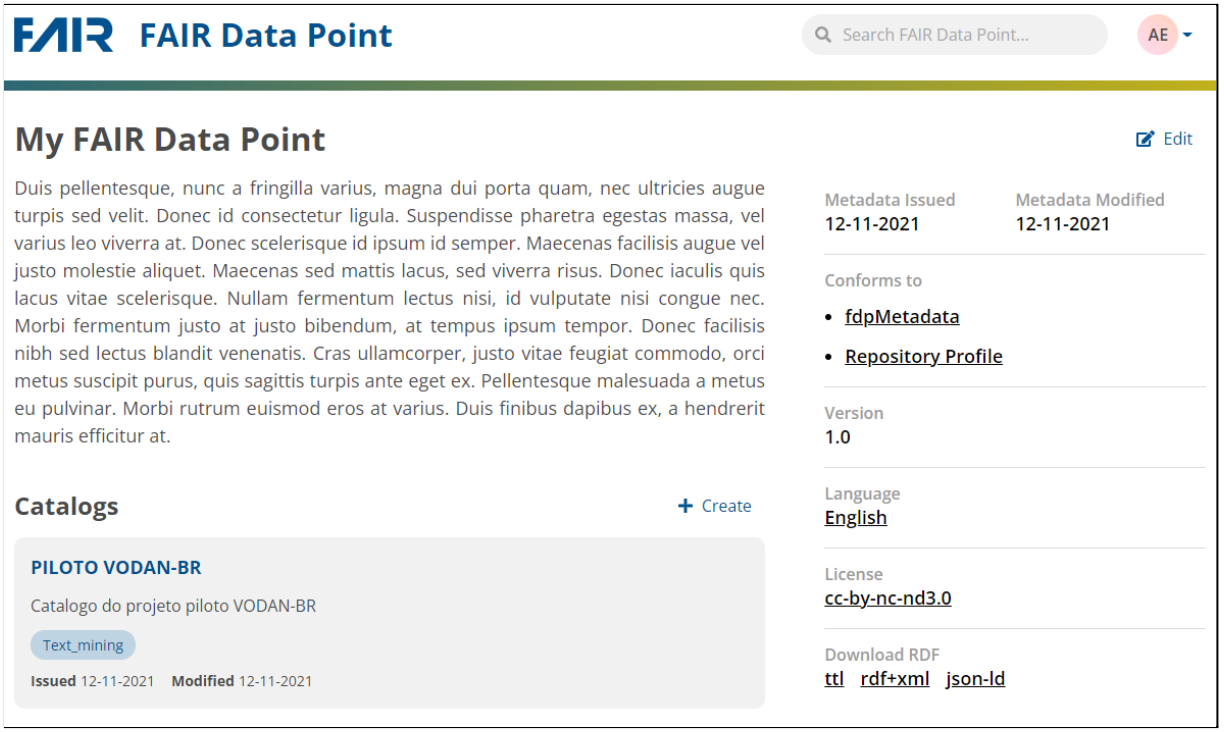


Figura 51 - Visualização dos metadados do repositório FAIR Data Point

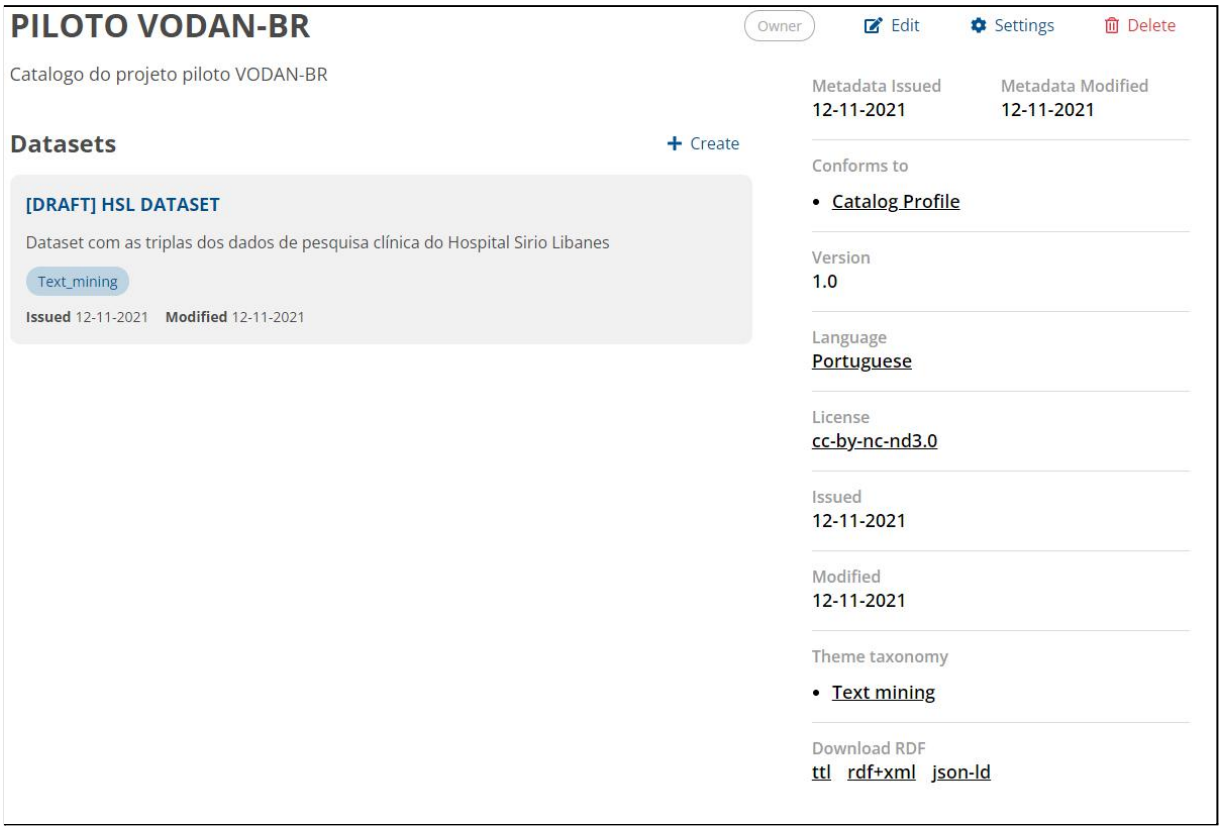


Figura 52 - Visualização dos metadados de *Catalog*

Figura 53 - Visualização dos metadados de *Dataset*

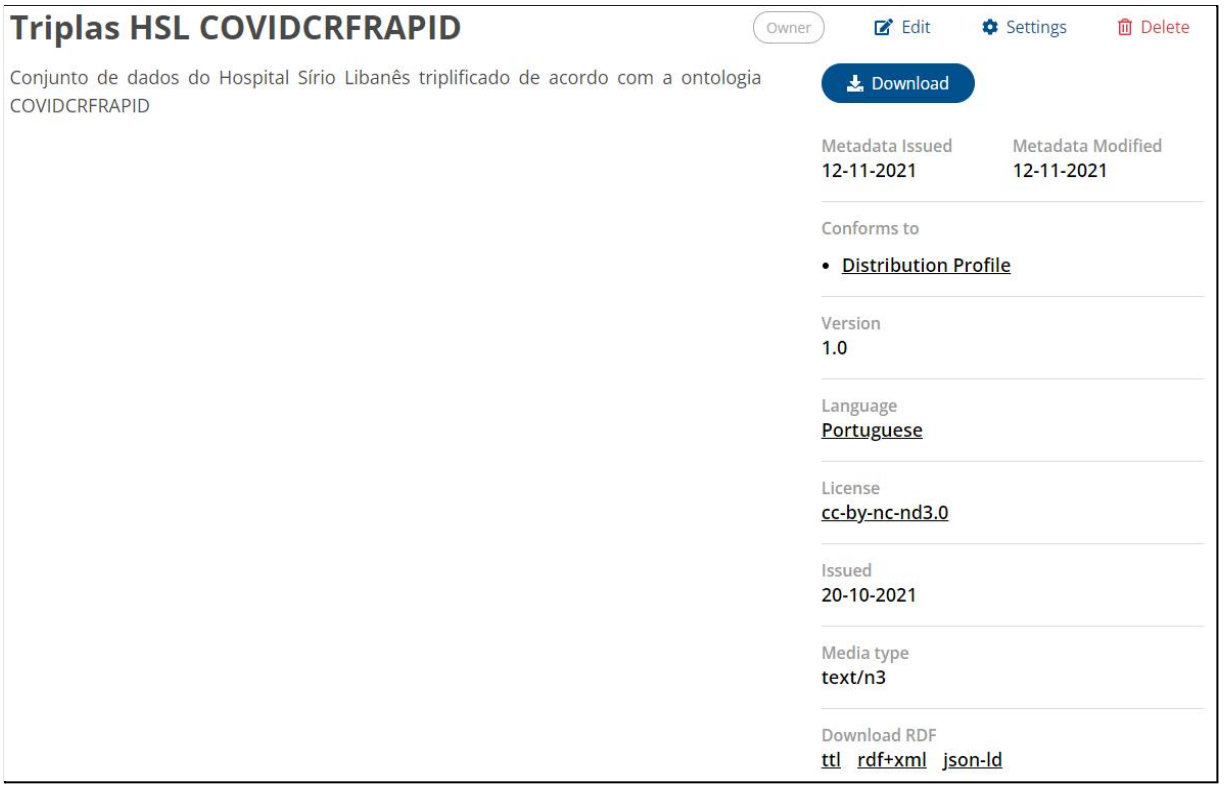


Figura 54 - Visualização dos metadados de *Distribution*

4.7 CONSIDERAÇÕES FINAIS

Ao longo de todo o experimento surgiram novas informações e definições de padrões, o que auxiliou no processo de criação das transformações. Também foi possível entender como os metadados são mapeados dentro dos recursos e FAIR Data Points e o quão completos eles eram. Além disso, muito se descobriu sobre a ontologia definida pelo CRF e processos epidemiológicos seguidos no mundo todo.

Essas experimentações mostram que o ETL4FAIR é uma ferramenta extremamente adequada para trabalhar no ciclo de vida de FAIR, bem como auxiliar na arquitetura definida pelo VODAN-BR e que o uso do PDI auxilia demasiadamente em todo o processo, visto que ele oferece um ecossistema rico de ferramentas e funcionalidades. Fica evidente, portanto, a necessidade de se investir na expansão de tal ferramenta.

A jornada de mil passos começa com o primeiro.

5 CONCLUSÃO

O principal propósito deste trabalho foi o de expor a necessidade de um ferramental robusto e agregado para auxiliar no ciclo de vida de dados FAIR, visto sua importância. Servindo como motivador para a integração inicial do processo de FAIRificação com o ETL4LOD+ e estender suas funcionalidades para incluir a carga de triplas em triplestores, assim como a capacidade de criar catálogos, datasets ou distribuições e carregar metadados em um FAIR Data Point. Além disso, também focou em levantar todos os pontos de melhoria do já existente ETL4LOD+ e implementar alguns, vide todas as atualizações realizadas.

A parte inicial do trabalho foi de estudar a ferramenta proposta pelo ETL4LOD+ e encontrar os ajustes que necessitavam ser realizados, com o objetivo de ter em mãos um ambiente de desenvolvimento mais estável e corretamente atualizado.

Ao fim, após o processo de normalização do ambiente e atualização da documentação e dependências, o trabalho foi focado inteiramente em criar processos que auxiliem o ciclo de vida FAIR, no caso, a carga correta do dado triplificado em um triplestore e a carga e recuperação de metadados de um FAIR Data Point. Criando assim três novos plugins e consolidando a solução como ETL4FAIR.

Em suma, o ETL4FAIR serve como instigador para o auxílio de todas as atividades necessárias ao se trabalhar com um dado FAIR. Oferecendo suporte como um único instrumento consolidado que atua de ponta a ponta neste processo, tornando desnecessário o uso de diversas aplicações que cobrem certos caminhos da jornada de um dado FAIR. Ademais, beneficia os futuros projetos que irão utilizá-los, já que ele é o ponto de partida para todos os futuros esforços e desenvolvimentos possíveis de novas integrações.

5.1 DIFICULDADES ENCONTRADAS

5.1.1 Definição do Escopo

A primeira dificuldade encontrada foi em fechar qual o objetivo do trabalho e o desenvolvimento correto a ser realizado, visto todos os pontos de melhoria encontrados no ETL4LOD+. No começo, o foco do trabalho seria direcionado em resolver os problemas de

integração do ETL4LOD+, corrigir a sua deficiência no suporte a publicação de um dado LOD em seu devido repositório e realizar uma primeira conexão, bem leve, ao FAIR Data Point.

Contudo, visto o agravamento da pandemia e a evolução dos estudos do VODAN-BR, tornou-se muito mais necessário um trabalho em que o foco seria totalmente sobre o processo de tornar FAIR esses dados epidemiológicos, auxiliando as pesquisas e garantindo a correta criação de uma rede federada.

Levando-se em consideração esses aspectos, o propósito deste trabalho deixou de ser o de melhorar a integração do ETL4LOD+ no processo de LOD para o de levantar a necessidade de um conjunto único de ferramentas que deem suporte a FAIR e realizar os desenvolvimentos nesse sentido.

5.1.2 Tecnologia

Definido o que deveria ser desenvolvido, a tecnologia também deveria ser escolhida e, como algum esforço já tinha sido realizado sobre o ETL4LOD+, ele e todo o ecossistema fornecido pelo Pentaho acabou se mostrando a escolha natural para servir como base do projeto.

O maior problema em todo esse processo foi o de descobrir os passos necessários para a implementação dos plugins na já existente estrutura do ETL4LOD+ e sua compatibilidade com o Pentaho, visto que a documentação existente para criação de *plugins* customizados no PDI é bem escassa[62] e os exemplos disponíveis são muito simples e ensinam praticamente nada para o caso de uma implementação mais rebuscada e complexa, tanto na parte gráfica quanto na parte de processamento, que é o caso do ETL4FAIR. E, além da escassez na documentação do PDI, a documentação do ETL4LOD+ era igualmente carente, não tendo nenhum exemplo dos *plugins* já existentes ou descrição de como funcionavam.

Tendo em vista os aspectos observados, a melhor maneira de prosseguir no desenvolvimento foi, primeiro, realizar um extenso processo de engenharia reversa[63] sobre o código existente, com muita leitura e experimentações. Simulando e avançando por meio de tentativa e erro no processo de criação de integração de novos plugins.

---

[62] https://help.hitachivantara.com/Documentation/Pentaho/9.2/Developer_center
[63] https://en.wikipedia.org/wiki/Reverse_engineering

5.1.3 TESTES

Por último, o aspecto mais desafiador foi o de testar os plugins. Primeiramente, para realmente testar qualquer modificação no código dos *plugins*, era necessário construir uma nova versão, compilando todo o pacote de *plugins*, e então rodar o Spoon (interface gráfica do PDI) para validação do que foi feito e batimento de *logs* e possíveis erros, o que ocorria com certa frequência, visto que a falta de documentação orientou boa parte do processo por tentativa e erro. Tornando-se um grande obstáculo no desenvolvimento, não somente pela complexidade mas, principalmente, pela elevada regularidade.

De mesmo modo, além da dificuldade acarretada pela necessidade de uma compilação completa a mínima mudança no código, outro ponto de grande impacto foi o fato de os plugins desenvolvidos precisarem de uma integração com outras ferramentas além do PDI. Pois, o Load Triple File tem o propósito de inserir arquivos triplificados em um banco de dados de triplas, logo, é necessário integração com triplestore, no caso, o GraphDB. E o FAIR Data Retriever e FAIR Data Loader com o objetivo de recuperar e inserir metadados em um FAIR Data Point. Portanto, a toda modificação, era necessário carregar outros ambientes e, em certos casos, a demanda de configurações repetitivas nessas outras ferramentas, até que o processo fosse completamente adequado.

5.2 TRABALHOS FUTUROS

Todos os *plugins* desenvolvidos visam oferecer suporte a obtenção dos dados, tratamento e triplificação, permitindo a anotação semântica e distribuição em um triplestore. Assim como a recuperação, triplificação e exposição dos metadados, gerando a proveniência e permitindo a reprodutibilidade desse conjunto de dados e metadados. Onde cada *plugin* possui o seu propósito e funcionalidade única, de modo que funcionam independentemente um do outro e auxiliam o processo FAIR. Com isso, certas limitações conhecidas podem ser resolvidas em trabalhos futuros.

5.2.1 Conectividade com um administrador de Esquemas de Metadados

Não obstante, é de grande importância a integração do conjunto de ferramentas existente com sistemas administradores de esquemas de metadados, que são capazes de definir e padronizar o processo de criação e uso de templates sobre conjuntos de metadados, a fim de garantir a melhor gerência do processo de dados FAIR e aumentar a reprodutibilidade científica, enriquecendo toda a Ciência Aberta. Por consequência, a integração do ETL4FAIR ao CEDAR, um gerenciador de metadados focado em dados biomédicos e utilizados no VODAN-BR, é um trabalho que precisa ser realizado, possibilitando a recuperação da estrutura dos metadados para manipulação e tratamentos dentro do PDI.

5.2.2 Conectividade com um centralizador de dados de pesquisas

Outro ponto a ser explorado é a capacidade de comunicação do ETL4FAIR com o Dataverse, uma aplicação Web, *open-source*, que procura compartilhar, preservar, citar, explorar e analisar dados e metadados de pesquisas científicas. Facilitando a disponibilização dos mesmos para outros núcleos e permitindo a réplica de trabalhos já desenvolvidos com exímia facilidade. Permitindo que pesquisadores, periódicos, autores de dados e metadados, editores, distribuidores de dados e metadados, assim como instituições afiliadas recebam o devido crédito acadêmico e visibilidade na Web. Aumentando, assim, as chances do estabelecimento de uma rede completamente integrada e intrínseca a suas definições.

5.2.3 Manutenção

Por fim, mas igualmente importante, como o software é o estado inicial de auxílio aos pontos levantados ao longo do trabalho, algumas atualizações serão necessárias com o passar do tempo, seja para corrigir alguns bugs quanto a evoluções tecnológicas de versões em suas dependências, atualizar a documentação, melhorar o desempenho ou para refatorar os *plugins* conforme as mudanças nas definições arquiteturais e de escopo do FAIR ou do VODAN-BR.

Por se tratar de um trabalho extenso, o software é de livre uso e está disponível no GitHub[64], com documentação para ajudar a todos que queiram trabalhar nele.

[64] https://github.com/NickolasGomes/ETL4FAIR

REFERÊNCIAS

Arenas, M., Bertails, A., Prud'hommeaux, E., & Sequeda, J. (2012). **A direct mapping of relational data to RDF**. Disponível em: https://www.w3.org/TR/2012/REC-rdb-directmapping-20120927/. Acesso em 28 jun. 2021.

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., & Ruslan, V.R. (2011). OWLIM: **A family of scalable semantic repositories. Semantic Web**, 2(1), 33–42. Disponível em: http://www.semantic-web-journal.net/sites/default/files/swj97_0.pdf. Acesso em: 05 mai. 2021.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). **Linked data—the story so far**. International Journal on Semantic Web and Information Systems, 5(3), 1–22. Disponível em: https://www.researchgate.net/publication/225070216_Linked_Data_The_Story_so_Far. Acesso em: 07 setem. 2021

Cyganiak, R., Wood, D., & Lanthaler, M. (2014). RDF 1.1 **Concepts and Abstract Syntax. W3C Recommendation**. Disponível em: https://www.w3.org/TR/rdf11-concepts/. Acesso em 08 jul. 2021.

DA SILVA, João Felipe C. **ETL4LOD+: evolução do suporte ao ciclo de publicação de dados conectados**. Universidade Federal do Rio de Janeiro, 2018. Acessado em: set. 14, 2021. Disponível em: http://pantheon.ufrj.br/handle/11422/6346. Acesso em: set. 14, 2021.

Das, S., Sundara, S., & Cyganiak, R. (2012). **R2RML: RDB to RDF Mapping Language. W3C Recommendation**. Disponível em: http://www.w3.org/TR/r2rml/. Acesso em: 19 mai. 2021.

Elmasri, R., & Navathe, S.B. (2003). **Sistemas de Banco de Dados**. Addison Wesley
Erling, O., & Mikhailov, I. (2009). **RDF Support in the Virtuoso DBMS**. In: Pellegrini T., Auer S., Tochtermann K., Schaffert S. (eds) Networked Knowledge—Networked Media. Studies in Computational Intelligence, vol 221. Springer, Berlin, Heidelberg. Disponível em: https://www.researchgate.net/publication/221143443_RDF_support_in_the_virtuoso_DBMS. Acesso em: 05 jun. 2021

Gearon, P., Passant, A., & Polleres, A. (2013). **SPARQL 1.1 update**. W3C Recommendation. Disponível em: https://www.w3.org/TR/sparql11-update/, Acesso em: 07 setem. 2021.

GIL, Yolanda. et al. **Provenance XG Final Report**.
Disponível em: <http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/>. Acesso em: 25 out. 2021.

Harris, S., & Seaborne, A. (2013). **SPARQL 1.1 query language**. W3C Recommendation. Disponível em: http://www.w3.org/TR/sparql11-query/. Acesso em: 06 setem. 2021.

Heese, R., & Znamirowski, M. (2011). **Resource centered RDF data management. In Proceedings of International Workshop on Scalable Semantic Web Knowledge Base Systems**, pp. 138–153. Inmon, B. (1997). The Data Warehouse Budget. DM Review Magazine. Disponível em: https://www.researchgate.net/publication/256536627_Resource_centered_RDF_data_management. Acesso em: 18 jun. 2021.

HENNING, Patricia Corrêa, et al. **GO FAIR e os princípios FAIR: o que representam para a expansão dos dados de pesquisa no âmbito da Ciência Aberta**. *Em Questão*, vol. 25, nº 2, 2, abril de 2019, p. 389–412. *seer.ufrgs.br*., Disponível em: https://doi.org/10.19132/1808-5245252.389-412. Acesso em: 09 setem. 2021.

Inmon, B. (1997). **The Data Warehouse Budget**. DM Review Magazine.

JACOBSEN, Annika. et al. **A Generic Workflow for the Data FAIRification Process**. Data Intelligence 2020; 2 (1-2): 56–65. Disponível em:doi:https://doi.org/10.1162/dint_a_00028. Acesso em: 15 agost. 2021.

J. Legrand , R. F. Grais, P. Y. Boelle , A. J. Valleron, A. Flahault (2006). **Understanding the dynamics of Ebola epidemics**. Cambridge University Press. Disponível em:https://www.cambridge.org/core/journals/epidemiology-and-infection/article/understanding-the-dynamics-of-ebola-epidemics/BAEEC049DE2893FA58DD8F38E19F62D8. Acesso em: 22 out. 2021.

Klímek, J., Škoda, P., & Nečaský, M. (2016). **LinkedPipes ETL: Evolved Linked Data Preparation**. In Proceedings of International Semantic Web Conference, 95–100. Springer International Publishing. Disponível em: https://www.semanticscholar.org/paper/LinkedPipes-ETL%3A-Evolved-Linked-Data-Preparation-Kl%C3%ADmek-%C5%A0koda/7db4759842da87ff893ce9074df7c50ac3333296. Acesso em: 06 jul. 2021.

Knap, T., Škoda, P., Klímek, J., & Nečaský, M. (2015). **UnifiedViews: Towards ETL Tool for Simple yet powerfull RDF Data Management**. In Proceedings of the Dateso 2015 Annual International Workshop on DAtabases, 111 –120. Disponível em: http://ceur-ws.org/Vol-1343/poster14.pdf. Acesso em: 27 jun. 2021.

Kyzirakos, K., Savva, D., Vlachopoulos, I., Vasileiou, A., Karalis, N., Koubrarakis, M., Manegold, S. (2018). **Geotriples: transforming geospatial data into rdf graphs using r2rml and rml mappings**. Journal of Web Semantics, 16–32. Disponível em: https://www.researchgate.net/publication/327713727_GeoTriples_Transforming_Geospatial_Data_into_RDF_Graphs_Using_R2RML_and_RML_Mappings. Acesso em: 02 setem. 2021.

L. Yu, **A Developer's Guide to the Semantic Web**.
DOI 10.1007/978-3-642-15970-1_11, C Springer-Verlag Berlin Heidelberg, 2011. 409 p

OLIVEIRA, N. , etl al. **A Practical Approach of Actions for FAIRification Workflows** . International Journal of Metadata, Semantics and Ontologies, 2021.

Red Hat (2020). **What Is a REST API?**. Disponível em: https://www.redhat.com/en/topics/api/what-is-a-rest-api. Acesso em: 20 setem. 2021.

Roshdy, H.M., Fadel, M.K., & ElYamany, F.H. (2013). **Developing a RDB-RDF Management Framework for Interoperable Web Environments**. Proceedings of the 4th IEEE Eurocon Conference, 307–313. Zagreb: Croatia. Disponível em: https://ieeexplore.ieee.org/document/6625001/. Acesso em: 02 setem. 2021

Schultz, A., Matteini, A., Isele, R., Bizer, C., & Becker, C. (2011). **LDIF—Linked data integration framework**. In Proceedings of the Second Inter-national Workshop on Consuming Linked Data (COLD2011), 1–6. Disponível em: https://www.researchgate.net/publication/294874596_LDIF_-_Linked_Data_Integration_Framework. Acesso em: 28 jul. 2021.

Shvaiko, P., & Euzenat, J. (2013). **Ontology matching: state of the art and future challenges**. IEEE Transactions on Knowledge and Data Engineering, 25(1), 158–176. Disponível em: https://ieeexplore.ieee.org/document/6104044. Acesso em: 27 jun. 2021

Stefanova, S., & Risch, T. (2013). **Scalable reconstruction of RDF-archived relational databases**. In Proceedings of the Fifth Workshop on Semantic Web Information Management, 1–4. Disponível em: http://www.it.uu.se/research/group/udbl/publ/SWIM2013.pdf. Acesso em: 05 setem. 2021.

Volz, J., Bizer, C., Gaedke, M., & Kobilarov, G. (2009). **Silk–A Link Discovery Framework for the Web of Data**. In Proceedings of the WWW2009 Workshop on Linked Data on the Web. Disponível em: http://silkframework.org/; Acesso em: 12 jun. 2021.

Wilkinson M, Dumontier M, Aalbersberg I, Appleton G, Axton M,Baak A, et al. **The FAIR Guiding Principles for scientific data management and stewardship**. Sci Data. 2016. Disponível em: https://www.nature.com/articles/sdata201618. Acesso em: 27 out. 2021.

WILKINSON, Mark D., et al. **Interoperability and FAIRness through a Novel Combination of Web Technologies**. *PeerJ Computer Science*, vol. 3, PeerJ Inc., abril de 2017, p. e110. *peerj.com*., Disponível em: https://doi.org/10.7717/peerj-cs.110. Acesso em:14 setem. 2021

APÊNDICES

APÊNDICE A – DADOS DA CARGA *CATALOG*

SUJEITO	PREDICADO	OBJETO
http://localhost/n	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Catalog
		http://www.w3.org/ns/dcat#Resource
	http://purl.org/dc/terms/description	"Catalogo do projeto piloto VODAN-BR"
	http://purl.org/dc/terms/hasVersion	"1.0"
	http://purl.org/dc/terms/isPartOf	http://localhost
	http://purl.org/dc/terms/language	http://id.loc.gov/vocabulary/iso639-1/pt
	http://purl.org/dc/terms/license	http://rdflicense.appspot.com/rdflicense/cc-by-nc-nd3.0
	http://purl.org/dc/terms/publisher	http://localhost/publicador/1
	http://purl.org/dc/terms/title	"PILOTO VODAN-BR"
http://localhost/publicador/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Agent
	http://xmlns.com/foaf/0.1/name	"Tester"

Fonte: Própria

APÊNDICE B – DADOS DA CARGA *DATASET*

SUJEITO	PREDICADO	OBJETO
http://localhost/n	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Dataset
		http://www.w3.org/ns/dcat#Resource
	http://purl.org/dc/terms/description	"Dataset com as triplas dos dados de pesquisa clínica do Hospital Sirio Libanes"
	http://purl.org/dc/terms/hasVersion	"1.0"
	http://purl.org/dc/terms/language	http://id.loc.gov/vocabulary/iso639-1/pt
	http://purl.org/dc/terms/license	http://rdflicense.appspot.com/rdflicense/cc-by-nc-nd3.0
	http://purl.org/dc/terms/publisher	http://localhost/publicador/1
	http://purl.org/dc/terms/title	"HSL DATASET"
http://localhost/publicador/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Agent
	http://xmlns.com/foaf/0.1/name	"Tester"

Fonte: Própria

APÊNDICE C – DADOS DA CARGA *DISTRIBUTION*

SUJEITO	PREDICADO	OBJETO
http://localhost/n	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/ns/dcat#Distribution
		http://www.w3.org/ns/dcat#Resource
	http://purl.org/dc/terms/description	"Conjunto de dados do Hospital Sírio Libanês triplificado de acordo com a ontologia COVIDCRFRAPID"
	http://purl.org/dc/terms/hasVersion	"1.0"
	http://purl.org/dc/terms/language	http://id.loc.gov/vocabulary/iso639-1/pt
	http://purl.org/dc/terms/license	http://rdflicense.appspot.com/rdflicense/cc-by-nc-nd3.0
	http://purl.org/dc/terms/publisher	http://localhost/publicador/1
	http://purl.org/dc/terms/title	"Triplas HSL COVIDCRFRAPID"
	http://www.w3.org/ns/dcat#downloadURL	http://localhost:7200
	http://www.w3.org/ns/dcat#mediaType	"text/n3"
http://localhost/publicador/1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Agent
	http://xmlns.com/foaf/0.1/name	"Tester"

Fonte: Própria

APÊNDICE D – TRECHO DE CÓDIGO DE CARGA DO *LOAD TRIPLE FILE*

```
public static void uploadFile(RepositoryConnection repoConnection, File file_Path, IRI context, String
inputFileFormat){


    // Base URI
    String baseURI = "http://example.org/example/local";

    switch (inputFileFormat){

      // Arquivos .ttl
      case "TURTLE":
        try {
          repoConnection.add(file_Path, baseURI, RDFFormat.TURTLE, context);

        } catch (IOException e) {

          System.out.println("\\n");
          System.out.println("Formato de arquivo errado, por favor, informe o formato correto.");
          e.printStackTrace();

        }
        break;
      // Arquivos .rdf, .rdfs, .owl, .xml
      case "RDFXML":
        try {

          repoConnection.add(file_Path, baseURI, RDFFormat.RDFXML, context);

        } catch (IOException e) {

          System.out.println("\\n");
          System.out.println("Formato de arquivo errado, por favor, informe o formato correto.");
          e.printStackTrace();

        }
        break;
```

APÊNDICE E – CÓDIGO DE RECUPERAÇÂO DO *FAIR DATA RETRIEVER*

```
private BufferedReader ProcessaGET(String uri){
		BufferedReader bReader = null;
		try {
			HttpClient client = HttpClientBuilder.create().build();

			HttpGet getRequest = new HttpGet(uri);

			// Header do request, definindo a captura
			getRequest.addHeader("accept", "text/n3");//Captura NTriple



			// Executa e obtém resposta
			HttpResponse response = client.execute(getRequest);



			//Obtém a resposta para um leitor
			bReader = new BufferedReader(new
InputStreamReader((response.getEntity().getContent())));

			System.out.println("============Output:============");



		}catch (ClientProtocolException e) {
			e.printStackTrace();

		} catch (IOException e) {
			e.printStackTrace();
		}

		return bReader;
	}
```

APÊNDICE F – CÓDIGO DE CARGA DO *FAIR DATA LOADER*

```
private String gravaDados(String fdpURL, String content, String type, String user, String pass) throws
IOException {//requisita autenticação por meio da função com esse fim e constroi POST para carga de
dados
    HttpClient client3 = HttpClientBuilder.create().build();
    BufferedReader bReader = null;
    String token="";
    String output;
    int responseCode;
    String outputTot="";
    StringReader r;
    Iterator<Statement> iterator;
    Statement statement;
    String generatedElement = "";



    HttpPost postRequest = new HttpPost(fdpURL+"/"+type);//Tipo de POST no FAIR Data Point
catalog, dataset ou distribution
    StringEntity entity = new StringEntity(content);

    token=AuthorizeFAIRDP(fdpURL+"/tokens",user,pass);

    postRequest.setEntity(entity);

    postRequest.addHeader("accept", "text/turtle");
    postRequest.addHeader("Content-Type", "text/n3");
    postRequest.addHeader("Authorization","Bearer "+token);

                logBasic(type);


    // Monta e executa request
    HttpResponse response = client3.execute(postRequest);
    responseCode = response.getStatusLine().getStatusCode();  // verifica resposta
    switch (responseCode) {
      case 201: {
```

```
            bReader = new BufferedReader(new InputStreamReader((response.getEntity().getContent())));
            while ((output = bReader.readLine()) != null) {
               outputTot += output;

            }
                                    logBasic("Dados gravados com sucesso:"+outputTot);
            break;
         }
         case 400: {
                                    bReader = new BufferedReader(new
InputStreamReader((response.getEntity().getContent())));
                                    System.out.println("Bad requestion. Parser failure or missing a FAIR
metadata triple"+outputTot);
                                    logBasic("Resposta 400:Bad request. Falha no parser ou metadados
FAIR faltando"+outputTot);
                                    while ((output = bReader.readLine()) != null) {
               outputTot += output;
               System.out.println(output);
            }

                                    logBasic(outputTot);
                                    break;
         }
         case 401: {
            System.out.println("Unauthorized");
                                    logBasic("Resposta 401: Não autorizado");
                                    break;
         }

      }
```

ANEXOS

ANEXO A – FAIR DATA POINT METADATA LAYER

Ontology	Term name	Datatype	Required/Optional	Description
RDF	rdf:type	IRI	Required	Required to be of type r3d:Repository
DC terms	dct:title	String	Required	Name of the repository with the language tag
	dct:hasVersion	String	Optional	Version of the repository
	dct:description	String	Optional	Description of the repository with the language tag
	dct:publisher	IRI	Required	Organisation(s) responsible for the repository
	dct:language	IRI	Optional	
	dct:license	IRI	Optional	
	dct:conformsTo	IRI	Optional	The specification of the repository metadata schema (for example ShEx)
	dct:rights	IRI	Optional	
	dct:references	IRI	Optional	Reference to documentation (API or otherwise).
	dct:accessRights	IRI	Optional	Description of the access rights, see Access rights rdf model
FDP ontology	fdp:metadataIdentifier	IRI	Required	Identifier of the metadata entry. Define new sub property ‘metadataID’ for dct:identifier
	fdp:metadataIssued	DateTime	Required	Created date of the metadata entry

	fdp:metadataModified	DateTime	Required	Last modified date of the metadata entry
RDF Schema	rdfs:label	String	Optional	Name of the repository with the language tag
RE3Data	r3d:institution	IRI	Optional	
	r3d:startDate	DateTime	Optional	Release date of the repository
	r3d:lastUpdate	DateTime	Optional	Last update timestamp of the repository
	r3d:dataCatalog	IRI	Required	List of catalog metadata URLs
	r3d:country	IRI	Optional	
	r3d:repositoryIdentifier	IRI	Required	Identifier of the repository.

Fonte: FAIR Data Team, 2018

ANEXO B – CATALOG METADATA LAYER

Ontology	Term name	Datatype	Required/Optional	Description
RDF	rdf:type	IRI	Required	Required to be of type dcat:Catalog
DC terms	dct:title	String	Required	Name of the catalog with the language tag
	dct:hasVersion	String	Required	Version of the catalog
	dct:description	String	Optional	Description of the catalog with the language tag
	dct:publisher	IRI	Required	Organisation(s) or Person(s) responsible for the catalog
	dct:language	IRI	Optional	
	dct:license	IRI	Optional	
	dct:issued	DateTime	Optional	Created data of the catalog entry
	dct:conformsTo	IRI	Optional	The specification of the catalog metadata schema (for example ShEx)
	dct:rights	IRI	Optional	
	dct:references	IRI	Optional	Reference to documentation (API or otherwise).
	dct:accessRights	IRI	Optional	Description of the access rights, see Access rights rdf model
	dct:isPartOf	IRI	Required	Relation to the parente metadata
FDP ontology	fdp:metadataIdentifier	IRI	Required	Identifier of the metadata entry. Define new sub property

				‘metadataID’ for dct:identifier
	fdp:metadataIssued	DateTime	Required	Created date of the metadata entry
	fdp:metadataModified	DateTime	Required	Last modified date of the metadata entry
RDF Schema	rdfs:label	String	Optional	Name of the catalog with the language tag
FOAF	foaf:homepage	IRI	Optional	
DCAT	dcat:dataset	IRI	Required	List of dataset URLs
	dcat:themeTaxonomy	IRI	Required	List of taxonomy URLs

Fonte: FAIR Data Team, 2018

ANEXO C – DATASET METADATA LAYER

Ontology	Term name	Datatype	Required/Optional	Description
RDF	rdf:type	IRI	Required	Required to be of type dcat:Dataset
DC terms	dct:title	String	Required	Name of the dataset with the language tag
	dct:publisher	IRI	Required	Organisation(s) or Persons(s) responsible for the dataset
	dct:hasVersion	String	Required	Version of the dataset
	dct:description	String	Optional	Description of the dataset with the language tag
	dct:license	IRI	Optional	
	dct:issued	DateTime	Optional	Created data of the dataset entry
	dct:modified	DateTime	Optional	Last modified date of the dataset entry
	dct:language	IRI	Optional	
	dct:conformsTo	IRI	Optional	The specification of the dataset metadata schema (for example ShEx)
	dct:rights	IRI	Optional	
	dct:accessRights	IRI	Optional	Description of the access rights, see Access rights rdf model
	dct:isPartOf	IRI	Required	Relation to the parente metadata
FDP ontology	fdp:metadataIdentifier	IRI	Required	Identifier of the metadata entry. Define new sub property ‘metadataID’ for dct:identifier

	fdp:metadataIssued	DateTime	Required	Created date of the metadata entry
	fdp:metadataModified	DateTime	Required	Last modified date of the metadata entry
RDF Schema	rdfs:label	String	Optional	Name of the distribution with the language tag
DCAT	dcat:distribution	IRI	Required	List of distribution URLs
	dcat:theme	IRI	Required	List of concepts that describe the dataset
	dcat:contactPoint	IRI	Optional	
	dcat:keyword	String	Optional	Keyword(s) related to the dataset with the language tag
	dcat:landingPage	IRI	Optional	Home page of the dataset

Fonte: FAIR Data Team, 2018

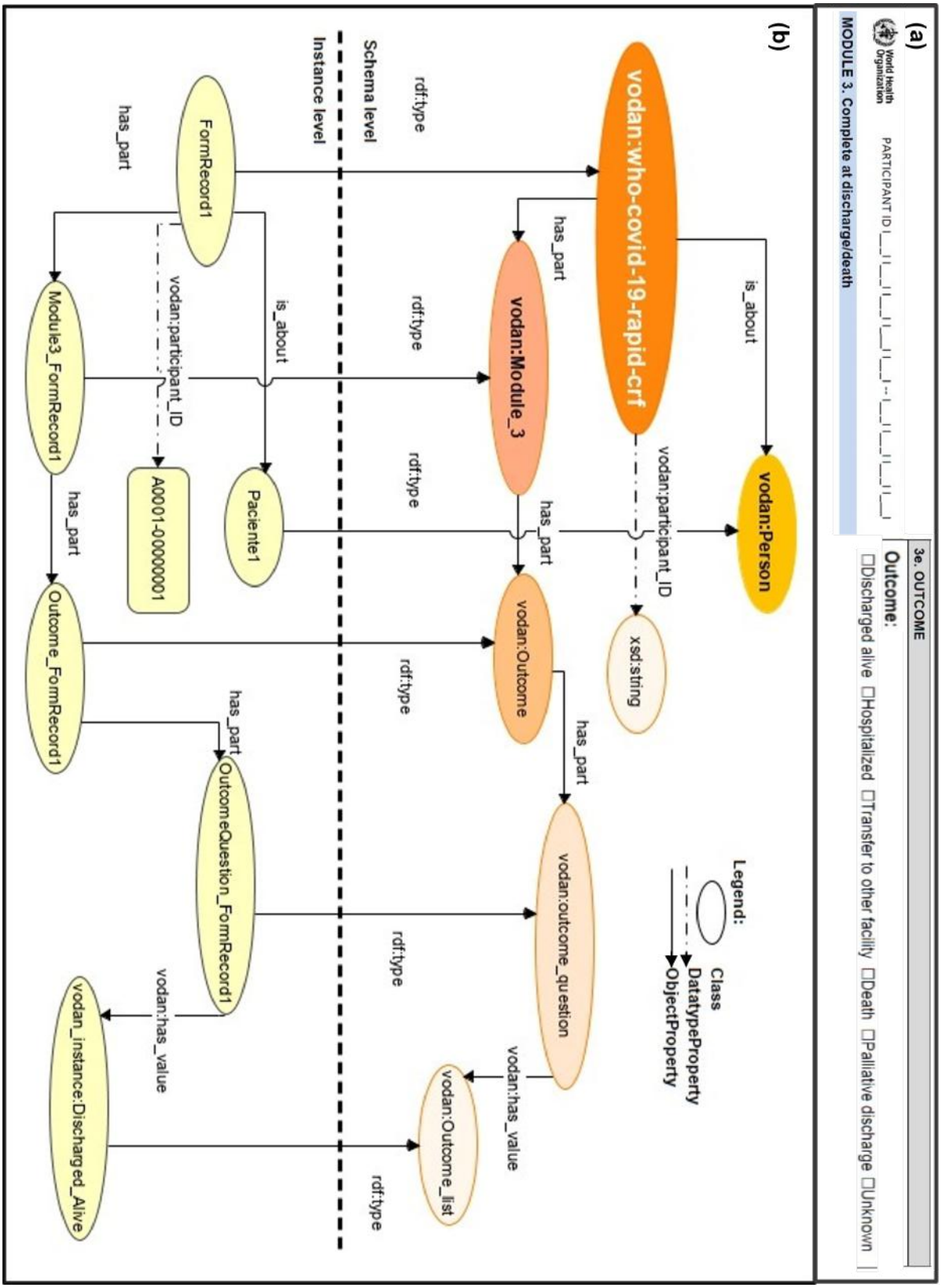
ANEXO D – DISTRIBUTION METADATA LAYER

Ontology	Term name	Datatype	Required/Optional	Description
RDF	rdf:type	IRI	Required	Required to be of type dcat:Distribution
DC terms	dct:title	String	Required	Name of the distribution with the language tag
	dct:hasVersion	String	Required	Version of the distribution
	dct:description	String	Optional	Description of the distribution with the language tag
	dct:license	IRI	Required	
	dct:issued	DateTime	Optional	Created data of the distribution entry
	dct:conformsTo	IRI	Optional	The specification of the distribution metadata schema (for example ShEx)
	dct:rights	IRI	Optional	
	dct:accessRights	IRI	Optional	Description of the access rights, see Access rights rdf model
	dct:isPartOf	IRI	Required	Relation to the parente metadata
FDP ontology	fdp:metadataIdentifier	IRI	Required	Identifier of the metadata entry. Define new sub property ‘metadataID’ for dct:identifier
	fdp:metadataIssued	DateTime	Required	Created date of the metadata entry
	fdp:metadataModified	DateTime	Required	Last modified date of the metadata entry

RDF Schema	rdfs:label	String	Optional	Name of the distribution with the language tag
DCAT	dcat:accessURL	IRI	Required(or dcat:downloadURL)	A landing page, feed, SPARQL endpoint or other type of resource that gives access to the distribution of the dataset
	dcat:downloadURL	IRI	Required(or dacat:accessURL)	A file that contains the distribution of the dataset in a given format
	dcat:mediaType	String	Required	The media type of the distribution
	dcat:format	String	Optional	
	dcat:byteSize	Decimal	Optional	

Fonte: FAIR Data Team, 2018

ANEXO E – RECORTE DA MODELAGEM DO MÒDULO 3



Fonte: OLIVEIRA, Natália et al, MTSR 2021

ANEXO F – CRF

Ontologia:
https://drive.google.com/file/d/1RvTHFD7D-6EXRkSUpMDwkmjfQrAX9LcZ/view?usp=sharing

Formulário:
https://drive.google.com/file/d/12aTWofy9_sDxLGK3SALpz7hy74Q-zaGK/view?usp=sharing