

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**SYSLODFLOW – UMA FERRAMENTA DE APOIO A AUTOMAÇÃO
DA PUBLICAÇÃO E MANUTENÇÃO DE LINKED DATA
UTILIZANDO O LODFLOW**

JEAN CARLOS DE MORAIS

JHONATAN CARLOS DE MORAIS

FLORIANÓPOLIS – SC

2016 / 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SYSLODFLOW – UMA FERRAMENTA DE APOIO A AUTOMAÇÃO DA
PUBLICAÇÃO E MANUTENÇÃO DE LINKED DATA UTILIZANDO O
LODFLOW

JEAN CARLOS DE MORAIS

JHONATAN CARLOS DE MORAIS

Trabalho de conclusão de curso
apresentado como parte das atividades
para obtenção do grau de Bacharel em
Sistemas de Informação pela
Universidade Federal de Santa Catarina.

Orientador: Prof. Dr. José Leomar
Todesco

Coorientador: Prof. Dr. Sandro
Rautenberg

FLORIANÓPOLIS – SC

2016 / 1

Jean Carlos de Moraes

Jhonatan Carlos de Moraes

**SYSLODFLOW – UMA FERRAMENTA DE APOIO A AUTOMAÇÃO DA
PUBLICAÇÃO E MANUTENÇÃO DE LINKED DATA UTILIZANDO O LODFLOW**

Este trabalho de conclusão de curso foi julgado adequado para obtenção do
Título de Bacharel em Sistemas de Informação, e aprovado em sua forma final pelo
Curso de Bacharelado em Sistemas de Informação.

Florianópolis, julho de 2016.

Prof. Frank Augusto Siqueira, Dr.

Coordenador do Curso

Banca Examinadora:

Prof., Dr. José Leomar Todesco,

Orientador

Universidade Federal de Santa Catarina

Prof., Dr. Fernando Alvaro Ostuni Gauthier,
Universidade Federal de Santa Catarina

Ma. Lidianne Visintin,
Universidade Federal de Santa Catarina

RESUMO

A manutenção de conjuntos de dados *Linked Data* é uma atividade demorada e cara, que envolve muitos recursos. O custo de manutenção pode ser reduzido pela automação do fluxo de trabalho de publicação de dados, a qual proporciona métodos para suportar o ciclo de vida dos conjuntos de dados RDF de uma forma sistemática. *O Linked Data Workflow Management System* (LDWMS), apelidado de LODFlow, fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho *linked data*. O objetivo deste trabalho é o desenvolvimento de uma aplicação de software que apoie o usuário na automação da publicação e manutenção de *linked data*, integrada aos conceitos e estrutura já definida pelo LODFlow.

Palavras-chave: *Linked Data*, RDF, Ontologias, *Web Semântica*.

ABSTRACT

The maintaining linked data datasets is a time consuming and expensive activity, involving many resources. The maintenance cost can be reduced by automation of data publishing workflow, which provides methods to support the life cycle of RDF data sets in a systematic way. The Linked Data Workflow Management System (LDWMS), dubbed LODFlow, provides an environment for planning, execution, reuse and documentation linked data workflows. The objective of this work is to develop a software application that supports the user in the automation of the publication and maintenance of linked data, integrated concepts and structure already defined LODFlow.

Keywords: *Linked Data*, RDF, Ontologies, *Semantic Web*.

LISTA DE FIGURAS

- Figura 1. Exemplos de URIs com seus elementos sintáticos.
- Figura 2. Grafo representando declaração simples no padrão RDF
- Figura 3. Grafo representando declaração RDF com uso de objeto com valor literal
- Figura 4. Grafo ilustrando a ligação entre conjuntos de dados RDF
- Figura 5. Fluxos de trabalho representando padrões de publicação linked data.
- Figura 6. Ciclo de Vida Linked Data suportado pelo LODFlow.
- Figura 7. Visão geral da ontologia LDWPO para descrição e planejamento de fluxos de trabalho do LODFlow.
- Figura 8. Ambiente de manipulação de ontologias do NeOn Toolkit.
- Figura 9. Arquitetura do Virtuoso Universal Server
- Figura 10. Diagrama de Casos de Uso do SysLODFlow
- Figura 11. Esquema representando a arquitetura em três camadas com artefatos utilizados para o desenvolvimento.
- Figura 12. Popularidade do Primefaces perante concorrentes.
- Figura 13. Interação entre as camadas da aplicação.
- Figura 14. Trecho do arquivo pom.xml do Maven para o projeto SysLODFlow
- Figura 15. Workspace do projeto Java utilizando a IDE Eclipse
- Figura 16. Entidades Java relacionadas as classes pertencentes ao modelo contido na LDWPO
- Figura 17. Classe LDWProject
- Figura 18. Classe LDWorkflow
- Figura 19. Propriedade de dados “goal” relacionada a classe LDWProject
- Figura 20. Propriedade de dados “preCondition” relacionada a classe LDWorkflow
- Figura 21. Classes responsáveis pela interação com a ontologia através da OWL API do Jena
- Figura 22. Criação de um modelo e leitura da ontologia para memória.
- Figura 23. Persistência de um modelo para arquivo
- Figura 24. ClassURIEnum: Enum contendo URI’s das classes da ontologia LDWPO
- Figura 25. Uso de classe da ontologia através do Jena
- Figura 26. Criação de uma novo Indivíduo da classe LDWProject
- Figura 27. PropertyURIEnum: Enum contendo URI’s das propriedades da ontologia

- Figura 28. Método genérico para recuperação de valor de datatype property de um indivíduo
- Figura 29. Exemplo de utilização do método getPropertyStringValue para recuperação de valor de datatype property de um indivíduo
- Figura 30. Exemplo de recuperação de uma object property
- Figura 31. Exemplo de inserção de propriedades a indivíduos
- Figura 32. Exemplo de atualização de propriedades a indivíduos
- Figura 33. Visão geral do fluxo de trabalho linked data, contendo o SYSLODFlow como componente
- Figura 34. Página Inicial da Aplicação
- Figura 35. Listagem de LDWProjects
- Figura 36. Formulário de instanciação do LDWProject
- Figura 37. Aba de informações gerais do Workflow
- Figura 38. Aba de informações do Step 02
- Figura 39. Listagem de Execuções de Workflow
- Figura 40. Cadastro de Execuções de Workflow
- Figura 41. Relatório de Execução de Workflow
- Figura 42. Resultado de uma consulta sobre o conjunto de dados RDF através de *endpoint* SPARQL

LISTA DE ABREVIATURAS E SIGLAS

AKSW – *Agile Knowledge Engineering and Semantic Web*

DNS – *Domain Name System*

HTTP – *Hypertext Transfer Protocol*

LDWFM – *Linked Data Workflow Management*

LDWMS – *Linked Data Workflow Management System*

LDWPO – *Linked Data Workflow Project Ontology*

RDF – *Resource Description Framework*

SPARQL – *SPARQL Protocol and RDF Query Language*

SQL – *Structured Query Language*

UML – *Unified Modeling Language*

URI – *Universal Resource Identifier*

XML – *EXtensible Markup Language*

OWL — *Web Ontology Language*

TI — *Tecnologia da Informação*

WMS — *Workflow Management System*

CSV – *Comma-separated Values*

API — *Application Programming Interface*

JSF — *Java Server Faces*

IDE — *Integrated Development Environment*

HTML — *HyperText Markup Language*

CSS – *Cascading Style Sheets*

OO — *Orientação à Objetos (Paradigma de Programação)*

SUMÁRIO

1. INTRODUÇÃO.....	12
1.1. Problema.....	13
1.2. Motivação e Justificativa.....	14
1.3. Objetivos	14
1.3.1. Objetivo Geral	14
1.3.2. Objetivos Específicos.....	14
1.4. Organização do texto	15
2. LINKED DATA	15
2.1.Princípios.....	17
2.2.Recursos	18
2.3.URIs.....	18
2.4.RDF	20
2.5.Fluxos de Trabalho Linked Data	24
2.6.Linked Data Workflow Management.....	25
2.7.LODFlow	27
2.7.1. Componentes	28
2.7.1.1.Linked Data Workflow Knowledge Model e Knowledge Base	29
2.7.1.2.Linked Data Workflow Maintenance Component	31
2.7.1.3.Linked Data Workflow Execution Engine.....	32
2.7.1.4.Linked Data Workflow Report Component.....	32
2.7.2. Benefícios	32
2.7.3. Ferramentas.....	33
2.7.3.1.Sparqlify	33
2.7.3.2.Virtuoso.....	34
2.7.3.3.LIMES	34
2.7.4. LODFlow em uso – Caso de uso Qualis Brasil	35
3. PROJETO E DESENVOLVIMENTO DO SOFTWARE.....	35
3.1.Visão Geral.....	35
3.2.Definição do Escopo.....	37
3.2.1. Especificação de Requisitos de Software.....	37
3.2.1.1.Requisitos Funcionais	38
3.2.1.2.Requisitos não Funcionais	39

3.2.2. Restrições	40
3.2.3. Limitações	40
3.3. Projeto da Aplicação	40
3.3.1. Modelagem UML	41
3.4. Definição da arquitetura	42
3.5. Desenvolvimento	46
3.5.1. Entidades	49
3.5.2. Trabalhando com Jena OWL API	52
3.5.2.1. Aspectos gerais	52
3.5.2.2. Carga e escrita de uma ontologia	53
3.5.2.3. Classes	55
3.5.2.4. Indivíduos	56
3.5.2.5. Propriedades	58
4. APLICAÇÃO DO SYSLODFLOW	61
4.1. Visão geral do workflow resultante	62
4.2. Interface gráfica do sistema	63
4.3. Artefatos gerados pelo workflow	69
4.4. Análise dos requisitos alcançados	71
5. CONCLUSÃO E TRABALHOS FUTUROS	72
5.1. Conclusão	72
5.2. Trabalhos Futuros	73
REFERÊNCIAS	74
ANEXO 1 – Exemplo de arquivo de mapeamento utilizado pelo Sparqlify	78
ANEXO 2 – Exemplo de arquivo de mapeamento utilizado para criar os links com o DBPedia, utilizado pelo LIMES	80
APÊNDICE A – Artigo	82

1. INTRODUÇÃO

A *Web Semântica* foi projetada para expandir a *web* que conhecemos atualmente, possibilitando que a imensa quantidade de dados disponíveis possa ser compreendida não só por pessoas, mas também por máquinas [4]. Segundo Berners-Lee (2001), ela não é uma *web* separada, mas uma extensão da atual, em que a informação é fornecida através de um significado bem definido, permitindo que computadores e pessoas trabalhem em cooperação. A disponibilização de recursos informacionais melhor estruturados e representados, de modo a construir uma rede de informações conectadas, é possível através de ferramentas tecnológicas tais como agentes de software, XML, RDF, ontologias, padrões ou formatos e metadados. A definição de como construir esta rede informações ligadas surgiu com os princípios de *Linked Data*, introduzidos por Berners-Lee em 2006.

Linked Data é um conjunto de melhores práticas para publicação e conexão de dados estruturados na *web*, permitindo estabelecer *links* entre itens de diferentes fontes de dados para formar um único espaço de dados global (HEATH; BIZER, 2011). Este conjunto de melhores práticas são sumarizados em quatro princípios: i) Use URI's (*Uniform Resource Identifier*) para nomear as coisas; ii) Use URI's HTTP para que as pessoas possam procurar o desejado; iii) Quando alguém olha para um URI, forneça informações úteis, usando os padrões (RDF, SPARQL); iv) Incluir links para outros URI's, para que eles possam descobrir explorar mais as coisas. Estes novos princípios abriram uma vasta gama de oportunidades, permitindo a publicação de dados acerca dos mais diversos temas e o desenvolvimento novas aplicações para interagirem com estes dados, de modo a construir uma “*Web de Dados*”.

Junto com estes novos horizontes, propiciados pelos dados ligados, surgem as dificuldades da manutenção destes dados publicados. A manutenção de conjuntos de dados *Linked Data* é complicada e envolve uma série de atividades bastante onerosas. Em contrapartida, o custo destas atividades de manutenção pode ser reduzido através da automatização do fluxo de trabalho de publicação de dados. Isso proporciona métodos para suportar o ciclo de vida destes conjuntos de dados de uma forma sistemática.

A produção de dados ligados é custosa e pode ser reduzida por um sistema de gerenciamento de fluxo de trabalho, que descreve planos para apoiar sistematicamente o

ciclo de vida de conjuntos de dados RDF. O *Linked Data Workflow Management System* (LDWMS) [18], apelidado de LODFlow, fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho *Linked Data*. A abordagem LODFlow baseia-se num modelo ontológico abrangente, o *Linked Data Workflow Project Ontology* (LDWPO), para descrever os fluxos de trabalho e um mecanismo de apoio sistemático para execução de fluxos de trabalho, relatando e manipulando exceções.

Este trabalho propõe, através do entendimento do modelo de fluxo de trabalho definido pelo LODFlow e reaproveitamento de seus componentes, desenvolver uma aplicação de software para apoio a automação e manutenção da publicação de conjuntos de dados *linked data*, com o objetivo fornecer um ambiente de interação mais intuitivo e integrado ao usuário.

1.1. Problema

Conforme já mencionado, o LODFlow fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho *Linked Data* através da interação entre seus componentes. Cada um destes componentes possui papel bem definido para suportar o ciclo de vida de *datasets linked data* baseado no modelo ontológico LDWPO.

Estes componentes que formam ambiente provido pelo LODFlow em sua atual versão — 1.0 — possuem uma série de requisitos de escopo para seu funcionamento, como criação manual de arquivos de configuração para ferramentas de apoio, população da base de conhecimento — ontologia LDWPO — através de software de terceiros, definição manual de caminhos de arquivos de entrada e saída, necessidade de manipulação de scripts de execução, entre outras, o que torna o processo de configuração do ambiente para execução do *workflow* difícil e oneroso para seu utilizador, podendo resultar em uma baixa aderência no uso da plataforma e baixa produtividade no trabalho com a mesma.

1.2. Motivação e Justificativa

O LODFlow surgiu com objetivo de definir um modelo estruturado para a publicação e manutenção de conjuntos de dados *linked data* baseado em fluxos de trabalho, aliado a um arcabouço tecnológico que suporte sua execução. A partir de estudos sobre a estrutura do sistema novas necessidades e melhorias foram levantadas com o intuito de pontuar os aspectos chave para uma melhoria significativa em sua estrutura de funcionamento. Estas melhorias são fundamentais para tornar o sistema ainda mais robusto, com seu fluxo de execução mais automatizado e com menor necessidade de intervenção humana.

Sendo assim, este trabalho detalha o funcionamento atual do LODFlow, enumera limitações e aspectos a serem aprimorados, através da proposição de alternativas de desenvolvimento a serem trabalhadas. Por fim, uma das alternativas de desenvolvimento é adotada e seu desenvolvimento realizado de modo a resolver limitações e necessidades pontuais.

1.3. Objetivos

1.3.1. Objetivo Geral

Desenvolver uma aplicação de software para apoio a automação da publicação e manutenção de *linked data*, através da reutilização dos componentes do LODFlow.

1.3.2. Objetivos específicos

- Propor, por meio de uma aplicação de software, uma nova forma de interação com o processo de publicação e manutenção de *linked data* suportado pelo LODFlow, fazendo com que este processo seja mais intuitivo e ainda mais automatizado;
- Oferecer uma interface gráfica customizada para que a automatização dos fluxos de trabalho consiga ser realizada com um menor nível de complexidade;

- Pesquisar metodologias de manipulação de dados *Linked Data*.
- Construir uma aplicação de software que suporte à integração de outras ferramentas e modelos de fluxo de trabalho

1.4. Organização do Texto

O presente documento está dividido em 5 capítulos. No capítulo inicial apresenta-se o projeto, expondo o problema vislumbrado, motivações e justificativa de sua relevância, além dos objetivos, gerais e específicos, a serem alcançados.

No segundo capítulo está contida a fundamentação teórica, onde faz-se um apanhado geral sobre *linked data*, LODFlow, tecnologias e conceitos correlacionados a estes. Busca-se apresentar sobre qual contexto este trabalho está inserido.

O terceiro capítulo aborda a etapa de projeto e desenvolvimento do objeto deste trabalho, o SysLODFlow. Definição de escopo do software, limitações, requisitos, tecnologias e metodologia de desenvolvimento utilizadas são tratadas neste capítulo.

O quarto capítulo apresenta de que forma este trabalho pode ser aplicado e os resultados obtidos a partir de seu desenvolvimento, ou seja, demonstra o resultado final do esforço de desenvolvimento do software.

Por fim, o quinto capítulo contém as conclusões e trabalhos futuros.

2. LINKED DATA

Linked Data é um conceito introduzido por Tim Berners-Lee no ano de 2006 [3] que se refere a uma nova forma de publicação de dados na *web*. Este conceito surgiu dentro de um outro mais abrangente, definido como um modelo de extensão da *web*, também proposto por Berners-Lee, no início dos anos 2000, a *Web Semântica* [2].

Quando a *World Wide Web* foi proposta, no início dos anos 1990, ela foi imaginada não apenas como um meio para a comunicação humana, mas também entre máquinas [7]. E foi com o objetivo de resolver esta segunda questão — comunicação entre máquinas, até então não resolvidas — que a *Web Semântica* foi proposta. Ela surgiu como uma proposta de transformar a *web* clássica, formada basicamente por documentos, em padrões estruturados ou semiestruturados, interligados através de *hyperlinks*, sem

nenhum significado associado, em uma “*Web* de Dados” [4]. Tudo isto motivado pela limitação em que computadores e sistemas computacionais possuem em interpretar os conteúdos contidos em páginas *web* tradicionais, apenas seres humanos são capazes de interpretar o conteúdo das páginas, atribuindo significado as mesmas. Além disso, pode-se citar outros problemas relativos a limitação da *web* clássica, como incapacidade de motores de busca proverem todos os dados para um determinado termo pesquisado, consultas sem alto grau de complexidade e dados isolados.

O conceito de *Web* Semântica define fazer para os dados o que o HTML fez para os sistemas de informação textuais: assegurar flexibilidade suficiente de modo a ser capaz de representar todos os bancos de dados e regras lógicas, para conectá-los adicionando grande valor aos mesmos [7]. Segundo Berners-Lee (2001) a *Web* Semântica não é uma *web* separada, mas uma extensão da atual (*web* clássica ou *web* de documentos), em que a informação é fornecida através de um significado bem definido, permitindo computadores e pessoas trabalharem em cooperação.

Mesmo com os enormes benefícios trazidos pela *web* em sua concepção, responsáveis pela sua disseminação até os dias atuais, os dados não tiveram este mesmo tratamento. Páginas HTML não são suficientemente expressivas para permitir a descrição de entidades individuais em um documento particular de modo a ser conectado por *links* a entidades relacionadas [9]. Com o amadurecimento dessas ideias e a necessidades de colocá-las em prática, surgiram algumas propostas para a adição de semântica no conteúdo da *web*, com grande parte delas baseadas em marcações nos conteúdos de páginas HTML, sendo os Microformatos a iniciativa de maior aderência no meio. Apresentado no ano de 2004 como uma forma de agregar significado aos dados publicados na Internet [11], os microformatos podem ser definidos como um simples conjunto de padrões de formatos de dados abertos, em constante desenvolvimento, que busca atender adequadamente *blog*’s estruturados e a publicação de microconteúdo na *web* em geral [6]. São baseados em marcações feitas em páginas HTML de modo a descrever pessoas, organizações, eventos, locais, etc [12].

Porém, como Berners-Lee afirma, a *Web* Semântica não é apenas sobre a colocação de dados na *web*, é também sobre ligá-los [3], permitindo assim a exploração de pessoas e máquinas a *Web* de Dados, através das ligações estabelecidas entre dados relacionados. Portanto, com o objetivo de transformar a *web* em um espaço global de

informações [8], Berners-Lee em 2006 propõe uma nova abordagem que trata da publicação de dados na *web* de modo a materializar a *Web Semântica*, chamada de *linked data*.

2.1. Princípios

Introduzido por Berners-Lee no ano de 2006, em uma de suas notas técnicas de arquitetura da *web* [3], o termo *linked data* refere-se a um conjunto de melhores práticas para publicação e conexão de dados estruturados na *web* [9]. O pressuposto básico por trás de *linked data* é de que o valor e a utilidade dos dados aumentam proporcionalmente ao número de ligações que eles estabelecem com outros dados [10].

Estas melhores práticas são delineadas por um conjunto de princípios propostos por Berners-Lee (2006) em sua nota de proposição de *linked data* [3], princípios estes que colaboram para que os dados publicados se tornem um único espaço de dados global. Os princípios são:

1. Use URIs como nomes para as coisas;
2. Use HTTP URIs para que as pessoas possam procurar esses nomes;
3. Quando alguém procurar uma URI, fornecer informações úteis, usando os padrões (RDF, SPARQL);
4. Incluir *links* para outras URIs, de modo a permitir que se descubram mais coisas.

Estes princípios, conhecidos como “princípios de *linked data*” fornecem uma receita básica para publicação e conexão de dados utilizando a infraestrutura da *web*, sendo aderente a sua arquitetura e padrões [9].

Em um paralelo entre a arquitetura da *web* clássica com a arquitetura *linked data*, a segunda tem como base duas tecnologias mais do que consolidadas, que são a base da *web* clássica: *Universal Resource Identifiers* (URIs), utilizadas como mecanismo global de identificação de recursos; e *HyperText Transfer Protocol* (HTTP), utilizado como mecanismo global de acesso a recursos. Enquanto na *web* clássica utiliza-se *HyperText Markup Language* (HTML) como formato amplamente adotado para a publicação de

conteúdo, no modelo *linked data* o *Resource Description Framework* (RDF) é o padrão utilizado para publicação na *Web* de Dados.

2.2. Recursos

Para publicar dados na *web*, é necessário primeiramente identificar os itens de interesse no domínio a ser trabalhado. Estes itens são elementos que as propriedades e relacionamentos de nosso interesse irão descrever [10]. Documentos, conceitos e coisas do mundo real e abstratos são exemplos disto. De acordo com a terminologia da arquitetura *web*, todos os itens de interesse são chamados de recursos. Um dos objetivos da *web*, desde a sua criação, tem sido a de construir uma comunidade global em que qualquer uma das partes compartilhar informações com qualquer outra. Tanto a arquitetura *web* (clássica), como *linked data* — que tem como base o arcabouço tecnológico da *web* clássica — faz uso de um sistema de identificação mundial único: URI.

2.3. URIs

Um *Uniform Resource Identifier* (URI) é um conjunto compacto de caracteres que identifica um recurso, seja ele físico ou abstrato [14]. Os URIs foram implantados com sucesso desde a criação da *web* e são utilizados até hoje, sendo o mecanismo adotado pelos princípios de *linked data* — citado no princípio de número um — para identificação de recursos na *Web* de Dados. Há benefícios substanciais para participar da rede existente de URIs, incluindo a ligação, marcação, cache e indexação de dados pelos motores de busca [13].

Para exemplificar a importância no uso de URIs podemos utilizar como exemplo a cidade de “São Paulo”. Ela pode ser identificada por alguns como simplesmente seu nome real, que é “São Paulo”, porém existe a possibilidade de ser chamada de “Terra da Garoa”, “Sampa”, “Capital Paulista” ou até mesmo por algum código oficial utilizado para referência de municípios brasileiros. O nome utilizado para o recurso é de extrema importância por questões de inteligibilidade, porém um potencial problema fica evidente: ambiguidade de nomes. “São Paulo” pode ser utilizado para identificar diversos

recursos, como a cidade de São Paulo (capital paulista), São Paulo F.C. (clube de futebol), São Paulo (santo da igreja católica) entre diversas possibilidades, causando a inexistência de precisão na referência de recursos. Com a utilização de URIs evita-se este tipo de problema dado sua natureza de unicidade na identificação de recursos.

No contexto de *linked data* é restrita a utilização de URIs HTTP, deixando de lado outros esquemas URI, fato este mencionado no segundo princípio de *linked data*. URIs HTTP fornecem uma maneira simples de criar nomes exclusivos globalmente sem necessitar de uma gestão centralizada, além de não trabalhar apenas como mecanismo de nomeação, mas também como meio de acesso a informações sobre recursos através da *web*. A preferência pela adoção do esquema HTTP sobre os demais é discutida em detalhes no projeto “TAG W3C” [15].

De forma mais simplificada, podemos representar a sintaxe de um URI genérico através de uma sequência hierárquica, composta por esquema, autoridade, endereço, consulta e fragmento, exemplificada na sintaxe a seguir [14]:

```
<esquema>: [//<autoridade>] [/<endereço>] [?<consulta>] [#<fragmento>]
```

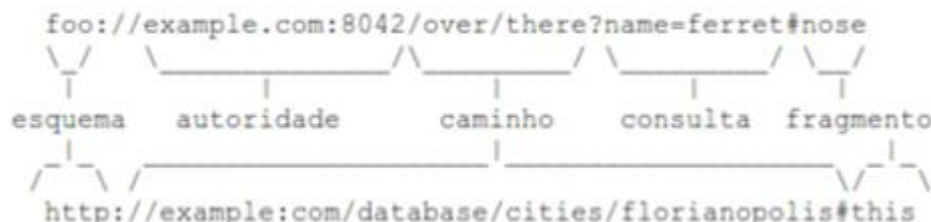
O nome de esquema refere-se a uma especificação para a atribuição de identificadores dentro deste esquema. Incluem endereços *Internet Protocol* (IP), nomes *Domain Name Systems* (DNS), possuindo regras de formação textuais bem definidas. Seu objetivo principal é apontar que tipo de recurso ou aplicação pertence o referido recurso, como por exemplo “mailto” para endereços de email, “http” para páginas da *web*, “ftp” para arquivos diversos, entre outros.

Autoridade faz referência ao “dono” da informação, ou a hierarquia do qual o recurso está delegado. Geralmente indicam o endereço IP ou nome de domínio DNS de quem está fornecendo o recurso, sendo possível também a inclusão opcional de nome de usuário e endereço de serviço (portas TCP/UDP). Como exemplos de autoridade: “ufsc.br”, “testeservidor.inf.ufsc.br:443” e “jhonatan@inf.ufsc.br”.

O endereço refere-se ao local onde se encontra o recurso, dentro da autoridade. No exemplo “inf.ufsc.br/jhonatan”, “jhonatan” seria o endereço, que se encontra na autoridade “inf.ufsc.br”. Consulta e fragmento servem como caminhos locais para uma

parte do recurso, de modo a obter-se apenas uma parte determinada do todo. Na figura 1 dois exemplos de URIs com indicações dos elementos sintáticos que o formam:

Figura 1. Exemplos de URIs com seus elementos sintáticos.



Fonte: BIANCO (2011), adaptado de RFC 3986 [14].

2.4. RDF

De modo a permitir o processamento e intercâmbio de dados pelas aplicações na *web* é necessária a adoção de um formato padrão de representação de conteúdo. Como já mencionado anteriormente na seção 2.1., o padrão de conteúdo em *linked data* é o *Resource Description Framework* (RDF), assim como o HTML é o padrão da arquitetura *web* clássica. Sendo assim, este é o formato do qual devem ser publicados dados na *Web* de Dados, de acordo com o terceiro princípio de *linked data*. Um fato importante a ser destacado é o de que o RDF trata apenas de como estruturar os dados e não de como apresentá-los, diferentemente do HTML. Existem diversos formatos de serialização de dados RDF, como XML, N-Triples, RDFa, entre outros, ou seja, formas de materializar em arquivos dados na estrutura proposta por RDF.

O RDF constitui-se em uma arquitetura genérica de metadados que permite representar informações sobre recursos na *World Wide Web*, tais como título, autor e data de atualização de uma página *web*, por exemplo [17]. O modelo RDF codifica os dados no formato de triplas, compostas por sujeito, predicado e objeto [9]. O sujeito é um recurso identificado por um URI. O objeto pode também pode ser um recurso identificado por um URI, relacionado ao sujeito, ou mesmo um valor literal, como um número ou texto qualquer. O predicado é identificado por um URI e estabelece o tipo de relação existente entre sujeito e objeto contidos na tripla (dado em modelo RDF) e podem vir de vocabulários ou conjuntos de URIs que representam informações sobre um determinado domínio.

Segundo Bizer et al (2007), existem dois tipos principais de triplas RDF que podem ser distinguidos: triplas literais e ligações RDF [10]:

- Triplas Literais têm como objeto sempre um literal, pode ser um texto, um número, uma data, etc. Geralmente utilizados para descrever propriedades de recursos, como por exemplo, nome ou data de nascimento de uma pessoa.
- Ligações RDF representam ligações tipificadas entre dois recursos. Uma ligação RDF se dá pela formação de uma tripla “sujeito-predicado-objeto”, onde sujeito e objeto são recursos interligados por um predicado que representa a relação entre eles. Utilizado para descrição, por exemplo, de relações entre pessoas, organizações, etc.

As ligações RDF são a base da *Web* de Dados, pois elas permitem que através de um recurso referenciado por um URI seja possível alcançar diversos outros recursos ligados a este, e assim sucessivamente. Com isso, torna-se possível a navegação na *Web* de Dados, seja através de navegadores *linked data* ou por aplicações automatizadas de motores de busca [10].

Para exemplificar a representação através do modelo RDF, pode-se utilizar a informação “Brasil tem como capital Brasília”. Em formato de tripla, a mesma informação forma a seguinte declaração RDF:

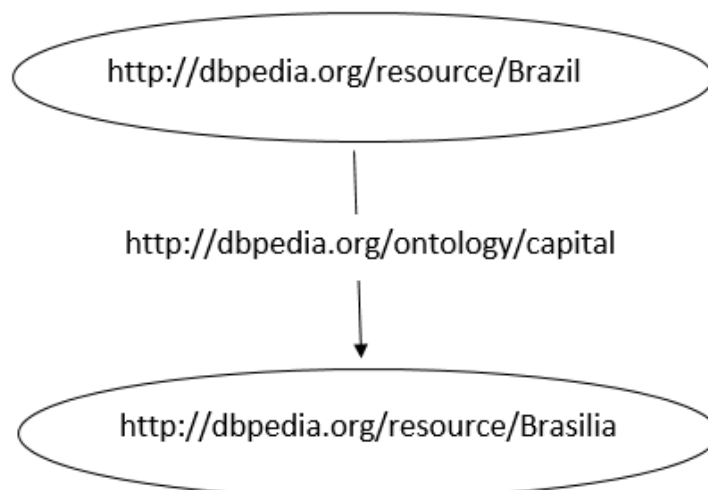
- Sujeito: Brasil
- Predicado: capital
- Objeto: Brasília

Desta forma é possível ter a representação de uma informação real, representada em linguagem natural, que também pode ser representada no padrão RDF, adequado para ser processado de forma automatizada por máquinas. Como citado anteriormente, recursos que compõem as triplas podem ser representados por URIs HTTP. Desta forma, a mesma representação de tripla, exemplificada acima, pode ser descrita através das URIs dos recursos que a compõem:

- Sujeito: <http://dbpedia.org/resource/Brazil>
- Predicado: <http://dbpedia.org/ontology/capital>
- Objeto: <http://dbpedia.org/resource/Brasilia>

Existe a possibilidade de representar triplas RDF através da representação por grafos. Neste tipo de representação o sujeito e o objeto são os nós do grafo e o predicado é representado pelos arcos, interligando sujeito a objeto. A declaração acima pode ser representada como um grafo conforme a Figura 2:

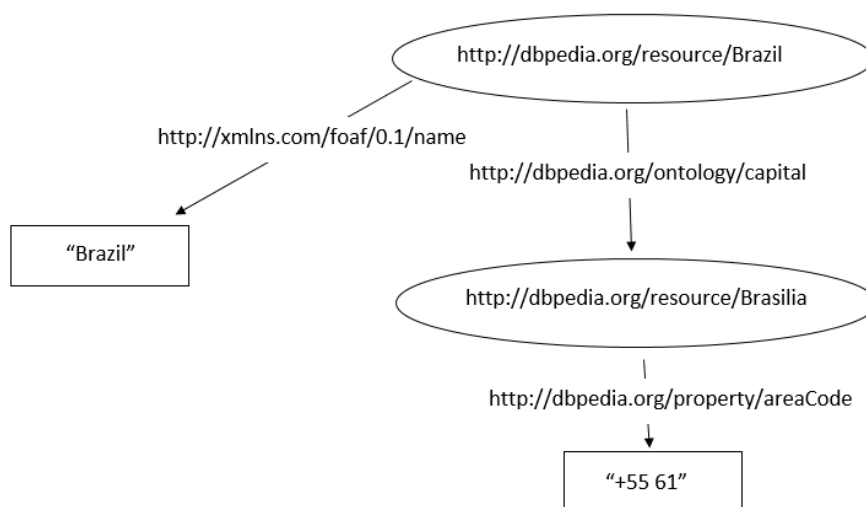
Figura 2. Grafo representando declaração simples no padrão RDF



Fonte: Autor.

Ilustrando a utilização de literais como objeto, é possível acrescentar novos nodos ao grafo, descrevendo informações a respeito do sujeito utilizado. Os objetos descritos por valores literais serão representados através de retângulos, conforme Figura 3:

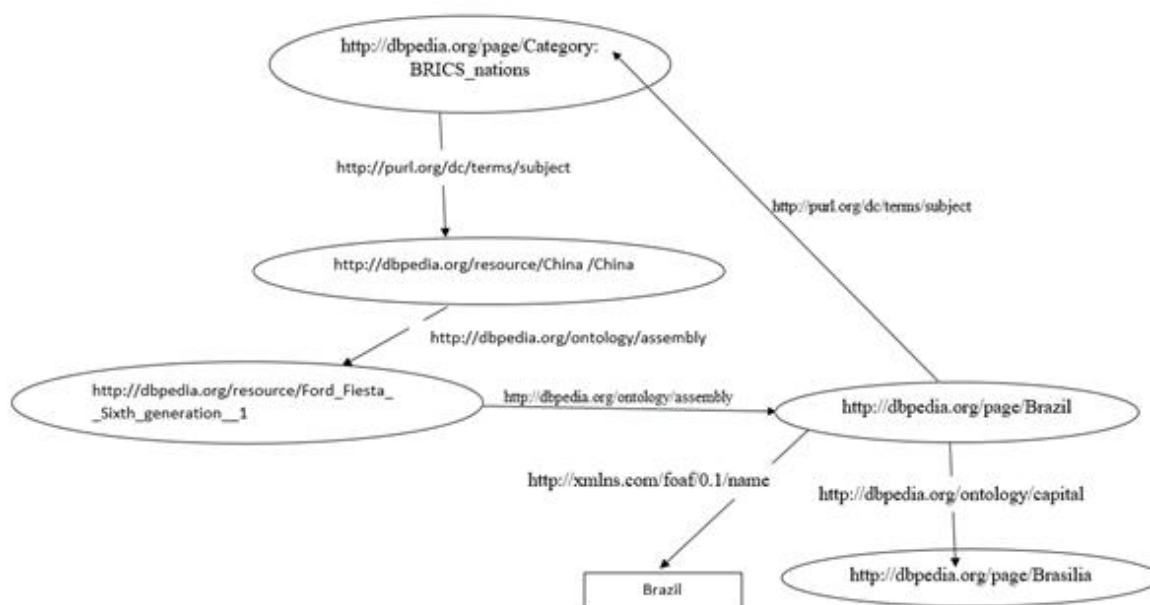
Figura 3. Grafo representando declaração RDF com uso de objeto com valor literal



Fonte: Autor.

De modo a ilustrar as ligações RDF entre diferentes conjuntos de dados, mecanismo que permite a navegação na *Web* de Dados, utilizado por pessoas e máquinas, a Figura 4 traz um novo grafo, interligando estes recursos a outros distintos.

Figura 4. Grafo ilustrando a ligação entre conjuntos de dados RDF



Fonte: Autor.

Simulando a uma navegação, a partir do primeiro grafo (Figura 3) temos 2 triplas, sendo uma delas literal para nome e outra uma ligação RDF para o recurso Brasília na DBPedia¹. Após o recebimento da URI do recurso Brazil da DBPedia, o navegador começa a seguir as ligações contidas neste recurso, os identificadores contidos na descrição do mesmo. Desta forma, retorna um novo grafo, com o conteúdo de um novo recurso e se este possuir ligações RDF permitirá a navegação sucessiva, gerando novos grafos e assim por diante. Neste caso, a partir da união de dois grafos, formando um novo grafo mais abrangente, ganha-se novas ligações permitindo a obtenção de mais informações a respeito dos recursos envolvidos.

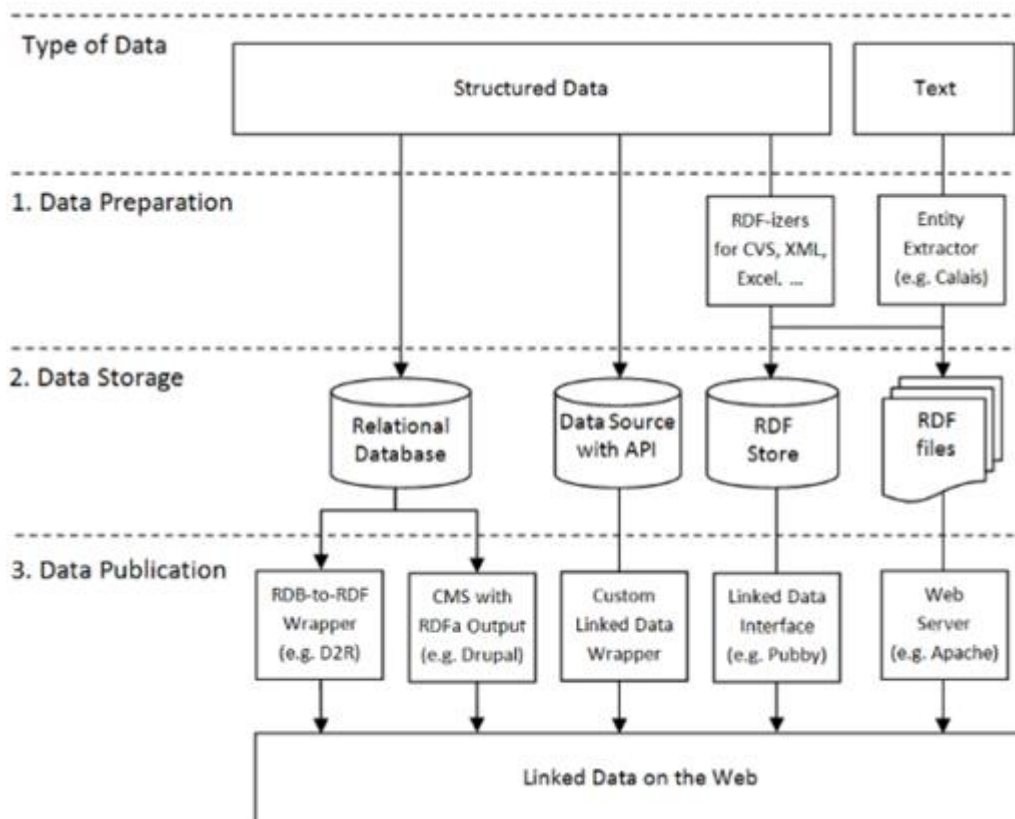
2.5. Fluxos de Trabalho Linked Data

Como citado anteriormente, a publicação de *linked data* requer a adoção de princípios para que possa se materializar de fato. Obedecendo a estes princípios, diversas abordagens possíveis podem ser utilizadas, de acordo com a finalidade, origem dos dados, ferramentas disponíveis, entre outros fatores que possam vir a influenciar esta escolha.

Cada uma destas abordagens possui um conjunto de ferramentas associadas e de passos a serem seguidos dentro de um determinado fluxo de trabalho. Mesmo havendo uma grande quantidade de tipos de fontes de dados que podem se interligar a *Web de Dados*, poucos padrões de publicação atendem a todos os casos [5]. A Figura 5 mostra um diagrama proposto por Bizer e Heath [8] contendo os padrões de publicação mais comuns, organizados em forma de fluxos de trabalho.

¹ <http://pt.dbpedia.org/>

Figura 5. Fluxos de trabalho representando padrões de publicação linked data.



Fonte: BIZER & HEATH (2011).

O livro de Bizer e Heath [5] traz detalhadamente informações sobre os diversos padrões, suas tecnologias e ferramentas associadas. Já o trabalho proposto por Bianco [16] aborda com profundidade o processo de publicação de dados oriundos de bases de dados relacionais através de um estudo de caso, passando por todos os passos da publicação e por fim efetuando comparações entre tecnologias e ferramentas utilizadas no processo. Como objeto de estudo deste trabalho, focaremos na utilização do *Linked Data Workflow Management System (LDWMS)*, também cunhado de LODFlow, proposto por Rautenberg *et al.* [18] que fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho para publicação de dados *linked data*.

2.6. Linked Data Workflow Management

O conceito de *Linked Data Workflow Management (LDWFM)* fornece uma base formal para a representação de processos de gerenciamento de *linked data*. O conceito central do LDWFM é o fluxo de trabalho, originado no domínio da gestão de negócios e

definido pela *Workflow Management Coalition* (WfMC)² como “a automação de um processo de negócio, no todo ou parte, durante a qual documentos, informações ou tarefas são passadas de um participante para outro por uma ação, de acordo com um conjunto de regras processuais” [25]. Neste contexto, a Gestão de Fluxo de Trabalho (do inglês “*Workflow Management*”) é uma tecnologia de apoio a reengenharia de processos de negócios e de informação [26]. Segundo Georgakopoulos et al. (1995), a Gestão de Fluxo de trabalho envolve basicamente: i. definição do fluxo de trabalho, isto é, a descrição dos aspectos relevantes para controle e coordenação da execução das tarefas e; ii. infraestrutura de tecnologia, que fornece mecanismos para o (re)desenho e (re)implementação de fluxos de trabalho eficientes. Para facilitar a gestão do fluxo de trabalho, a infraestrutura tecnológica deve preencher os seguintes requisitos:

- Ser orientado a componente, permitindo a integração e interoperabilidade entre sistemas com baixo acoplamento e componentes de serviço;
- Suportar a implementação de um fluxo de trabalho, facilitando recursos de proveniência e reprodutibilidade;
- Garantir a correção e confiabilidade de aplicações na presença de concorrência e falhas;
- Suportar a evolução, substituição e adição de etapas do fluxo de trabalho em um processo de reengenharia.

Segundo Rautenberg et al. (2015), em um ambiente *Linked Data*, o Gerenciamento de Fluxo de Trabalho deve cobrir a especificação, execução, registro e controle dos processos de produção e manutenção de conjuntos de dados *linked data*. Neste contexto são definidos requisitos para um LDWFM como:

Requisito 1. *Linked Data Workflow Planning*: a capacidade para descrever uma estratégia de produção para conjuntos de dados *Linked Data*.

Requisito 2. *Linked Data Workflow Execution*: a capacidade para automatizar fluxos de trabalho, que envolve um ambiente controlado para a execução de um plano.

² <http://www.wfmc.org/>

Requisito 3. *Linked Data Workflow Reusability*: a capacidade para reutilizar o todo ou uma parte para fluxos de trabalho existentes.

Requisito 4. *Linked Data Workflow Documentation*: a capacidade de representar planos e execuções de fluxos de trabalho *Linked Data* em formatos legíveis.

Requisito 5. *Linked Data Workflow Repeatbility*: a capacidade para reproduzir o mesmo resultado ao longo do tempo.

2.7. LODFlow

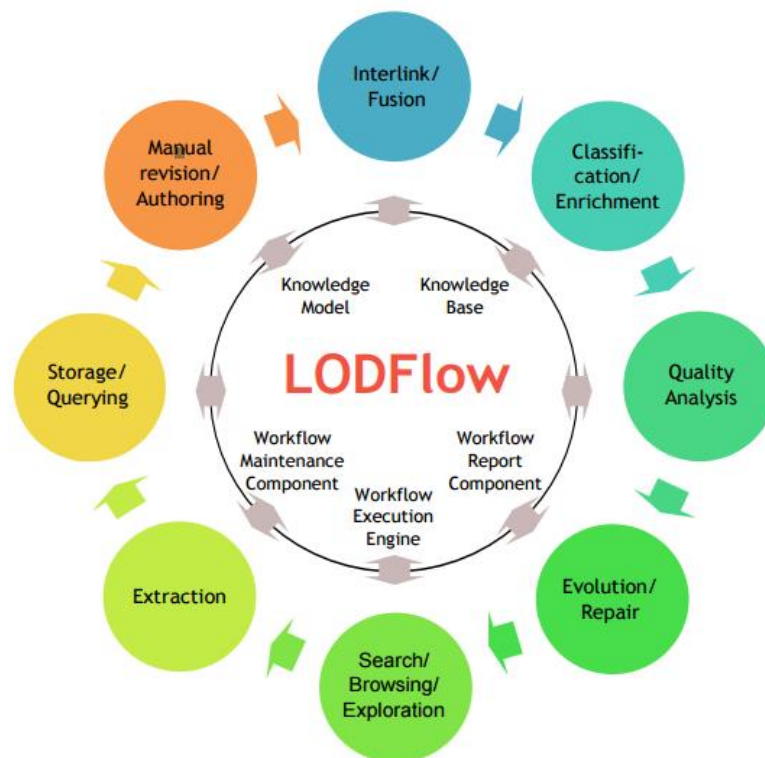
A produção de dados ligados é custosa e pode ser reduzida por um sistema de gerenciamento de fluxo de trabalho, que descreve planos para apoiar sistematicamente o ciclo de vida de conjuntos de dados RDF. O processo de criação e manutenção de conjuntos de dados *Linked Data* é tradicionalmente feito através de fluxos de trabalho manuais ou através de *scripts* proprietários que não são portáteis e apresentam certa dificuldade de manutenção. A extração de dados de bases de conhecimento *Linked Data* torna-se uma tarefa custosa, por conta de diversos passos que devem ser realizados para se obter um conjunto de dados, como carga de dados brutos, extração e análise. Feito isso os dados são disponibilizados em *endpoints* e ligados a outros conjuntos de dados [18].

O *Linked Data Workflow Management System* (LDWMS), apelidado de LODFlow, pode ser entendido como uma extensão do *Workflow Management System* (WMS), oriundo da gestão de negócios. O WMS “define, cria e gerencia a execução e fluxos de trabalho através do uso de software, rodando em um ou mais *engines* de fluxos de trabalho, capaz de interpretar a definição do processo, interagir com os participantes e, se necessário, recorrer ao uso de ferramentas de TI e aplicações” [25]. Entendendo esta definição para o contexto *Linked Data*, e considerando os requisitos descritos na seção 2.6., o LODFlow pode ser definido como um sistema que consiste em um conjunto de componentes, apresentados na seção que segue.

2.7.1. Componentes

1. *Linked Data Workflow Knowledge Model*: define um vocabulário comum para modelagem, análise, execução e documentação de fluxos de trabalho Linked Data.
2. *Knowledge Base*: registra os dados de acordo com o *Linked Data Workflow Knowledge Model*.
3. *Linked Data Workflow Maintenance Component*: usado para registrar os planos *Linked Data* e Execuções de *Linked Data* em bases de conhecimento.
4. *Linked Data Workflow Execution Engine*: usado para recuperar um fluxo de trabalho planejado de uma base de conhecimento e a execução de um plano a fim de produzir ou manter os conjuntos de dados.
5. *Linked Data Workflow Report Component*: usado para geração de relatórios para fluxos de trabalho *Linked Data*.

Figura 6. Ciclo de Vida Linked Data suportado pelo LODFlow.



Fonte: RAUTENBERG *et al.* (2015).

2.7.1.1. *Linked Data Workflow Knowledge Model e Knowledge Base*

O *Linked Data Workflow Knowledge Model* é armazenado em uma base de conhecimento, aderindo ao *Linked Data Workflow Project Ontology (LDWPO)*³, uma ontologia que padroniza os conceitos de método, plano e execução para tornar operacional a produção de conjuntos de dados ligados. Proposto por Rautenberg *et al.* (2015), o objetivo do LDWPO é resolver alguns problemas de proveniência e reprodutibilidade em um *Linked Data Workflow Project (LDWProject)*⁴ [1]. Modelando de forma simples uma parte do conhecimento sobre projetos, métodos, planos, execuções, artefatos e pessoas, permite a descrição da produção de conjuntos de dados RDF de forma sistemática, além de apoiar a manutenção de conjuntos de dados RDF ao longo do tempo.

O conceito principal do LDWPO é o LDWProject, conceito que representa todos os aspectos envolvidos para criação e manutenção RDFDatasets⁵. Além da adição de anotação e propriedades de metadados, como para criador, nome, etc, um LDWProject está associado a um ou mais LDWorkflows⁶. Um LDWorkflow contém um plano necessário para produzir RDFDatasets, encapsulando uma sequência de LDWSteps⁷, que são uma unidade atômica e reutilizável de um LDWorkflow. Um LDWStep descreve o processamento de um conjunto de dados de entrada, utilizando uma Tool⁸ de uma determinada versão em conjunto com uma ToolConfiguration⁹, com o objetivo de produzir um Dataset¹⁰ como saída. Um LDWStep pode ser utilizado para duas finalidades:

- i. Reutilizado por diferentes LDWorkflows dentro de LDWProjects existentes;
- ii. Executado automaticamente em um ambiente controlado através da solicitação do usuário.

³ <http://aksw.org/Projects/LDWPO.html>

⁴ Classe que representa um projeto linked data na ontologia LDWPO

⁵ Classe que representa um conjunto de dados RDF na ontologia LDWPO

⁶ Classe que representa um um fluxo de trabalho para a publicação de dados ligados na ontologia LDWPO

⁷ Classe que representa um passo de um fluxo de trabalho para a publicação de dados ligados na ontologia LDWPO

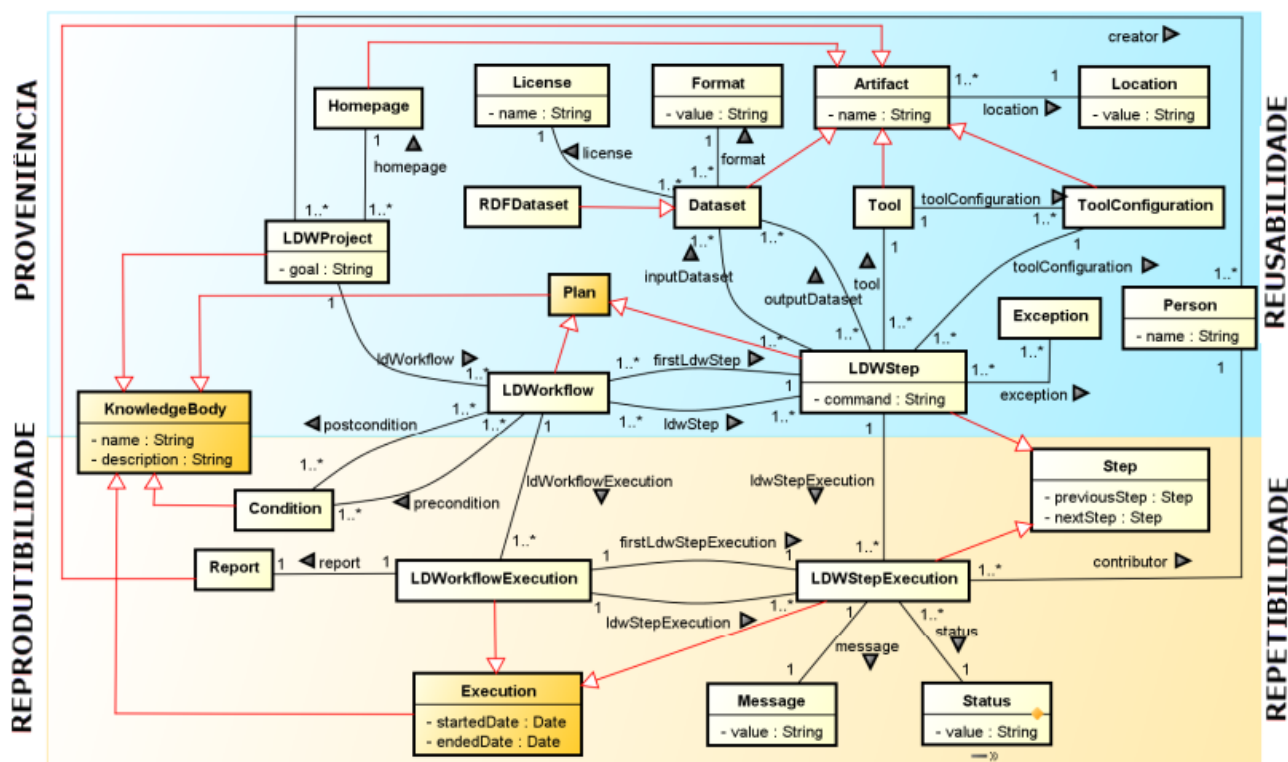
⁸ Classe que representa uma ferramenta tecnológica na ontologia LDWPO

⁹ Classe que representa uma configuração de uma ferramenta tecnológica na ontologia LDWPO

¹⁰ Classe que representa um conjunto de dados na ontologia LDWPO

Um LDWorkflow após definido, pode ser reutilizado como um plano já estabelecido para execuções a qualquer momento. No LDWPO o conceito que representa uma execução de um LDWorkflow é o LDWorkflowExecution¹¹. O LDWorkflowExecution encapsula uma sequência de passos, representados por LDWStepExecutions¹², que nada mais são que a sequência de LDWSteps de um LDWorkflow específico. Durante uma execução de um LDWorkflowExecution, os LDWStepExecutions vinculados a este fluxo de trabalho em execução podem gerar mensagens, relatórios, *status* de execução, entre outros.

Figura 7. Visão geral da ontologia LDWPO para descrição e planejamento de fluxos de trabalho do LODFlow.



Fonte: RAUTENBERG *et al.* (2015).

¹¹ Classe que representa uma execução de um fluxo de trabalho para a publicação de dados ligados na ontologia LDWPO

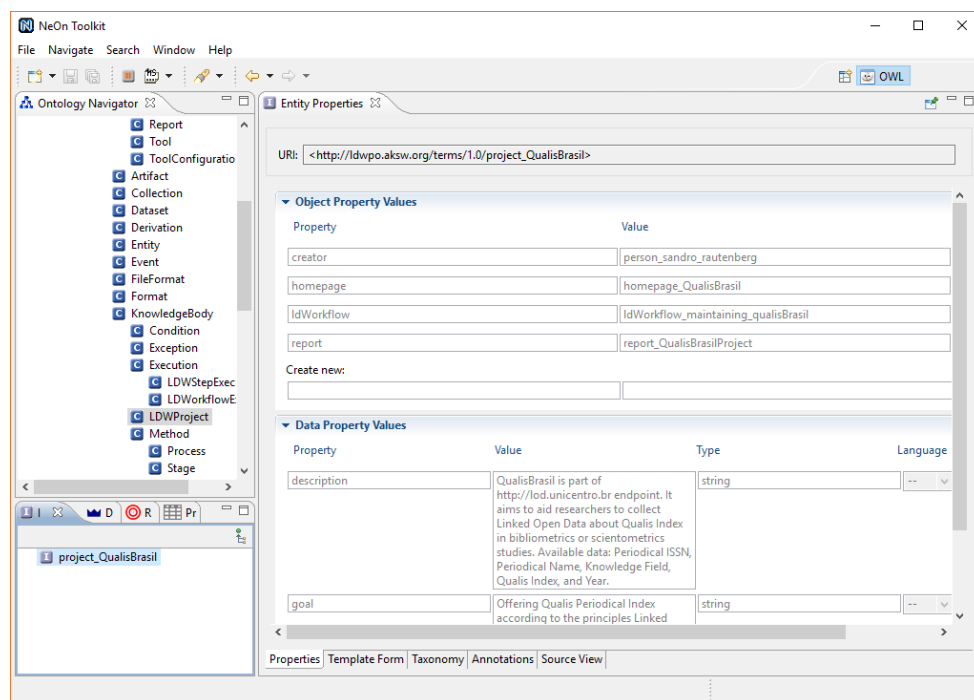
¹² Classe que representa um passo da execução de um fluxo de trabalho para a publicação de dados ligados na ontologia LDWPO

A parte superior da figura 7 ilustra as classes e conceitos envolvidos e suas relações, responsáveis definição de um plano necessário para a publicação de conjuntos de dados RDF, representando o cumprimento dos requisitos de proveniência e reuso do processo. Já a parte inferior mostra ilustra as classes e conceitos envolvidos na execução e suas relações, que permitem o cumprimento dos requisitos de reprodutibilidade e repetibilidade, tendo como figurar central o LDWorkflowExecution, que contém informações acerca de uma execução específica de um LDWorkflow em um certo ponto do tempo.

2.7.1.2. Linked Data Workflow Maintenance Component

Como *Linked Data Workflow Maintenance Component* foi adotado a ferramenta *NeOn Toolkit*¹³. Ela é um ambiente para engenharia de ontologias multiplataforma e *open source* que provê um suporte abrangente para o ciclo de vida de definição de ontologias [27].

Figura 8. Ambiente de manipulação de ontologias do NeOn Toolkit.



Fonte: Autor.

¹³ <http://neon-toolkit.org/>

O papel do NeOn é facilitar a interação do usuário com a ontologia, de modo a permitir o registro das instâncias das classes LDWProjects, LDWorkflows, LDWSteps, LDWorkflowExecutions, LDWStepExecution na base de conhecimento, tratada na seção anterior, a ontologia LDWPO.

2.7.1.3. *Linked Data Workflow Execution Engine*

O *Linked Data Workflow Execution Engine* é o principal componente para a produção de conjuntos de dados *linked data*. Ele é responsável por recuperar o LDWorkflow da base de conhecimento, interpretando os LDWorkflows de acordo com o estabelecido no *Knowledge Model* e gerenciando o andamento da produção de conjuntos de dados *linked data*. A ferramenta desenvolvida para tal finalidade é apelidada de *LODFlow Engine*¹⁴, e ela é capaz de interagir com outros componentes, interpretar recursos da ontologia LDWPO e realizar a invocação de outras ferramentas (representadas pela classe *Tool* na LDWPO).

2.7.1.4. *Linked Data Workflow Report Component*

O *Linked Data Workflow Report Component* é responsável pela geração de relatórios técnicos para LDWProjects, LDWorkflows e LDWorkflowExecutions. A ferramenta que materializa a funcionalidade proposta pelo componente é a *LODFlow Report Tool*¹⁵, que em sua atual versão está apta a gerar relatórios para um LDWProject completo.

2.7.2. Benefícios

Os benefícios do LODFlow incluem [18]:

1. Explicitação: A descrição declarativa de fluxos de trabalho de produção e manutenção de *linked data* faz com que atividades e processos sejam

¹⁴ <https://github.com/AKSW/LODFlow/tree/master/tools/LODFlowEngine>

¹⁵ <https://github.com/AKSW/LODFlow/tree/master/tools/LODFlowReport>

explícitos e de fácil entendimento para engenheiros de *linked data* em um alto nível de abstração.

2. Reutilização: A baixa granularidade na definição das atividades e passos do fluxo de trabalho facilita o reuso do mesmo.
3. Repetibilidade: Fluxos de trabalho podem ser facilmente executados repetidas vezes, tornando-os repetíveis, testáveis e confiáveis.
4. Eficiência: A execução automática de fluxos de trabalho reduz muito a intervenção humana manual requerida, melhorando assim a eficiência.
5. Facilidade de uso: O alto nível da descrição declarativa de fluxos de trabalho e geração de relatórios de execução abrangentes simplifica a definição, execução e análise dos fluxos de trabalho de produção e manutenção de *linked data*.

2.7.3. Ferramentas

Atualmente o LODFlow conta com algumas ferramentas de suporte, sendo fundamental tê-las no ambiente de execução para que o sistema seja executado com sucesso. Estas ferramentas são apresentadas nas subseções a seguir.

2.7.3.1. Sparqlify

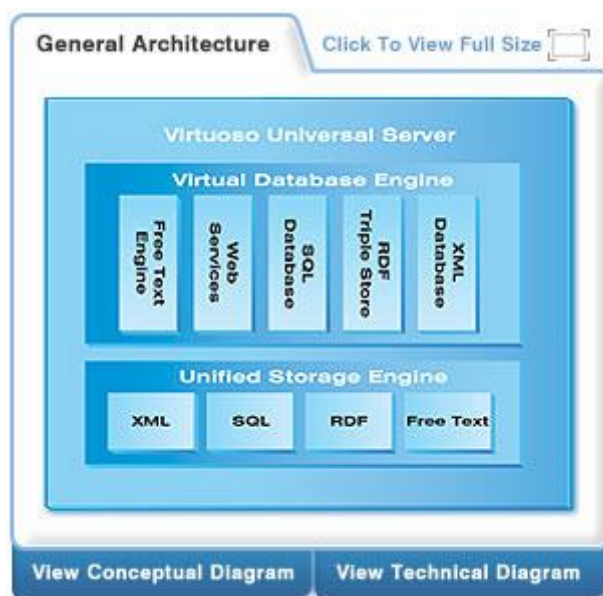
O *Sparqlify* é um tradutor escalável para traduções SPARQL-SQL. Possui características como:

- Suporte à *Sparqlification Mapping Language* (SML), uma linguagem para mapeamento RDB-RDF, projetada para ser expressiva, legível e fácil de aprender. Também é possível realizar conversões do tipo SPARQL-SQL e mapeamento de dados CSV;
- Escalabilidade;
- Suporte a CSV;
- Suporte para tipos de dados espaciais e predicados;
- Suporte ao conjunto de consultas da linguagem SPARQL 1.0, incluindo *subqueries*.

2.7.3.2. Virtuoso

O *Virtuoso Universal Server* é uma plataforma que oferece diversos recursos na área de acesso, gerenciamento e integração de dados. Possui uma estrutura híbrida, que tem como proposta suportar diversos modelos de dados existentes, entre eles estão SQL, XML, RDF e texto livre. Além disso, também possui servidor de aplicações *web*, servidor *Linked Data* baseado em RDF, servidor de arquivos e de *web* de documentos, gerenciador de conteúdos como JSON, TURTLE, HTML, TEXT, RDF, entre outros recursos.

Figura 9: Arquitetura do Virtuoso Universal Server



Fonte: OPENLINK SOFTWARE

2.7.3.3. LIMES

O LIMES é um *framework* voltado para a descoberta de *links* na Web de Dados. Ele implementa abordagens eficientes para descoberta de *links* em larga escala, além de ser facilmente configurável através de sua interface *web*. Também está disponível para *download*, podendo ser executado localmente em um servidor.

2.7.4. LODFlow em uso - Caso de uso Qualis Brasil

O primeiro caso de uso¹⁶ utilizando o LODFlow foi realizado com o conjunto de dados derivados do Índice Qualis¹⁷, uma base de dados de avaliações de pesquisas bibliométricas e cienciométricas no Brasil, descrevendo de forma compreensiva o suporte que o LODFlow fornece a produção e manutenção de um conjunto de dados RDF 5 estrelas¹⁸. O caso de uso abordado mostra como é mantido o conjunto de dados RDF do Qualis de forma automatizada, também como a maioria dos problemas urgentes e desafiadores da gestão *Linked Data* podem ser enfrentados com o uso do LODFlow: a automatização de fluxos de trabalho, atualmente complicada, exigindo muitos recursos. Além disso, aborda os benefícios de explicitação, reutilização, repetibilidade, eficiência, e facilidade de uso com a aplicação do LODFlow.

3. PROJETO E DESENVOLVIMENTO DA APLICAÇÃO

Este capítulo apresenta os aspectos referentes a etapa de projeto e desenvolvimento da aplicação objeto deste trabalho, o SYSLODFlow.

3.1. Visão Geral

Toda orquestração de ferramentas necessárias para realizar o processo de publicação e manutenção de *Linked Data* exige o conhecimento e domínio de vários softwares que auxiliam as diferentes etapas necessárias para obtenção final de um *dataset*, fazendo com que a complexidade aumente consideravelmente. Dada a complexidades de se publicar dados no padrão *Linked Data*, uma ferramenta que auxilie o processo de publicação e manutenção de dados ligados, através do planejamento e execução de fluxos de trabalhos, contendo a definição dos passos e ferramentas necessárias para tal, torna-se uma alternativa que sistematiza todas estas tarefas, viabilizando e facilitando o processo como um todo.

¹⁶ Mais detalhes sobre o caso de uso do LODFlow aplicado aos dados do Qualis Brasil podem ser obtidos em http://svn.aksw.org/papers/2015/SEMANTICS_LDWPO/public.pdf

¹⁷ <https://qualis.capes.gov.br/>

¹⁸ <http://5stardata.info/en/>

O LODFlow, introduz conhecimentos e componentes de software capazes de suportar todo o ciclo de vida de conjuntos de dados no padrão *linked data*, permitindo a produção de novos conjuntos de dados ou manutenção dos mesmos. Cada um dos componentes do LODFlow tem seu papel bem definido dentro deste ciclo de vida – conforme demonstrado na seção 2.7 – construindo um arcabouço tecnológico que suporte o pleno funcionamento do sistema.

Compreendendo toda a sistemática proposta pelo LODFlow, através do estudo dos processos envolvidos, observação do funcionamento, entendimento da relação entre os componentes e manipulação dos artefatos tecnológicos envolvidos pudemos identificar alguns pontos a serem trabalhados para a melhoria no funcionamento e operação deste sistema:

1. Automatização (via código da aplicação) dos controles de execução dos passos do workflow (LDWSteps), realizada através de chamadas externas realizadas a scripts shell;
2. Assistente de configuração das ferramentas de terceiros incluídas no fluxo de trabalho;
3. Viabilizar a utilização de outros tipos de arquivos de entrada para o fluxo além do CSV (*Comma-separated values*);
4. Enriquecimento dos relatórios gerados pelo componente de geração de relatórios (LODFlow Report);
5. Ferramenta gráfica customizada para registro e manipulação das instâncias das classes envolvidas no LODFlow em sua base de conhecimento, a ontologia LDWPO;

Diante do exposto acima a decisão foi por abordar o aspecto citado no item de número cinco. A aplicação desenvolvida, cunhada “**SysLODFlow**” terá como finalidade o registro das instâncias das classes envolvidas no fluxo de trabalho de trabalho para publicação de conjuntos de dados *linked data*, orquestrado pelo LODFlow. O componente do LODFlow que trata da interação entre usuário e base de conhecimento é o *Linked Data Workflow Maintenance Component*, e conforme mencionado na seção 2.7.1.2. a ferramenta adotada e que atualmente permite a interação entre usuário e base de conhecimento (LDWPO) é a ferramenta *NeOn Toolkit*, um kit de ferramentas para a manipulação de

ontologias. O objetivo com o *SysLODFlow* é criar um ambiente customizado para registros de Projetos, seus Workflows, Passos e Execuções, de modo a permitir uma melhor experiência ao usuário, facilitando o entendimento da sistemática envolvida, encapsulando funcionalidades intrínsecas as ferramentas de manipulação de ontologias.

3.2. Definição do Escopo

A definição do escopo é um fator determinante para se alcançar um resultado satisfatório e de acordo com o planejado. Quanto mais avança a tecnologia e mais complexos e inter-relacionados tornam-se os sistemas de informação, mais esse quesito transforma-se em fator essencial para o sucesso, e mais impacto sua deficiência provoca na execução dos projetos de software [28]. Esta seção trata de elucidar a abrangência do software *SysLODFlow*, explicitando os aspectos que serão cobertos pelo desenvolvimento desta aplicação bem como suas restrições e limitações, foram definidos.

3.2.1. Especificação de Requisitos de Software

Após definição da alternativa de desenvolvimento a ser trabalhada — descrita na seção 3.1 — iniciou-se a etapa de mapeamento das características necessárias para que esta aplicação suprisse de forma satisfatória o seu papel. Buscou-se inicialmente por características e funcionalidades fornecidas pela ferramenta *NeOn Toolkit*, que desempenha o papel de componente de manutenção do LODFlow, do qual a aplicação objeto deste trabalho busca substituir. Adicionalmente, após reuniões de orientação e discussão entre os membros da equipe de desenvolvimento, percebeu-se novas necessidades e ajustes a serem realizados. Desta forma, após o refinamento de todas estas informações foram definidos um conjunto de requisitos funcionais e não funcionais para guiar o desenvolvimento da aplicação de *software*.

3.2.1.1. Requisitos Funcionais

RF01. Criar LDWProject — O software deve permitir a criação de um projeto de publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWProject na ontologia LDWPO.

RF02. Criar LDWorkflow — O software deve permitir a criação de um plano de execução de um fluxo de trabalho para publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWorkflow na ontologia LDWPO.

RF03. Criar LDWSteps — O software deve permitir a criação de um conjunto de passos de um fluxo de trabalho para publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWStep na ontologia LDWPO.

RF04. Criar LDWStep — O software deve permitir a criação de um conjunto de passos de um fluxo de trabalho para publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWStep na ontologia LDWPO.

RF05. Criar LDWorkflowExecution — O software deve permitir a criação de uma execução de um determinado fluxo de trabalho (LDWorkflow) para publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWorkflowExecution na ontologia LDWPO.

RF06. Criar LDWStepExecution — O software deve permitir a criação de um conjunto de passos de execução de um determinado fluxo de trabalho (LDWorkflow) para publicação de conjunto de dados *linked data*, baseado no *Linked Data Workflow Knowledge Model*, representado pela classe LDWStep na ontologia LDWPO.

RF07. Executar um Workflow — O software deve permitir ao usuário a execução de um Workflow armazenado na base de conhecimento (LDWPO) através de sua interface.

RF08. Editar LDWProject — O software deve permitir a edição dos dados de um LDWProject recuperado da base de conhecimento (LDWPO) bem como suas propriedades vinculadas.

RF09. Editar LDWorkflow — O software deve permitir a edição dos dados de um LDWorkflow recuperado da base de conhecimento (LDWPO) bem como suas propriedades vinculadas.

RF10. Editar LDWStep — O software deve permitir a edição dos dados de um LDWStep recuperado da base de conhecimento (LDWPO) bem como suas propriedades vinculadas.

RF11. Editar LDWorkflowExecution — O software deve permitir a edição dos dados de um LDWorkflowExecution recuperado da base de conhecimento (LDWPO) bem como suas propriedades vinculadas.

3.2.1.2. Requisitos Não Funcionais

RNF01. O software deve fornecer uma interface gráfica acessível através de um navegador *web* atual, de modo a permitir uma interação fácil e intuitiva ao usuário.

RNF02. O software deve ser desenvolvido na linguagem de programação Java, no *backend* sendo utilizado a tecnologia Java EE e gerenciador de dependências *Maven*¹⁹.

RNF03. O software deve utilizar o *framework* “*Apache Jena*”²⁰ para manipulação da ontologia LDWPO.

RNF04. O software deve prover integração com o componente *Linked Data Workflow Engine*, representado pela ferramenta *LODFlow Engine*, de modo a permitir o disparo da execução dos fluxos de trabalho *linked data* a partir de sua própria interface.

RNF05. O software deve ter seu código aberto e disponível através de repositório no *Github*, permitindo a colaboração com o projeto e evolução do mesmo.

¹⁹ <https://maven.apache.org/>

²⁰ <https://jena.apache.org/>

3.2.2. Restrições

RST 01. O servidor onde será executada aplicação deve rodar uma distribuição Linux baseada em Debian (preferencialmente Debian ou Ubuntu Server), possuir Java JDK versão 7 e todas as ferramentas envolvidas no fluxo de trabalho definido pelo LODFlow — MySQL Server²¹, Virtuoso DB²², Sparqlify²³, LIMES²⁴ — instaladas e corretamente configuradas.

RST 02. O software deverá ser executado na mesma máquina em que o LODFlow estiver hospedado.

RST 03. É necessário que os componentes LODFlow *Engine* e LODFlow *Report* estejam devidamente configurados e operacionais.

3.2.3. Limitações

LMT 01. O software não prevê a criação e edição de arquivos de configuração das ferramentas terceiras envolvidas através de sua interface gráfica.

LMT 02. Cada arquivo da ontologia LDWPO só conterá uma instância de LDWProject e este uma instância de LDWorkflow. A cada nova criação de Projeto será criada uma nova instância da ontologia.

3.3. Projeto da Aplicação

Este processo de desenvolvimento tem como proposta a construção de um ambiente gráfico customizado para instanciação de projetos segundo modelo definido pelo LODFlow, através do desenvolvimento de uma aplicação *web* que suporte as funcionalidades esperadas para o integrar o *Linked Data Workflow Maintenance Component*.

²¹ <https://dev.mysql.com/downloads/mysql/>

²² https://www.w3.org/2001/sw/wiki/OpenLink_Virtuoso

²³ <https://github.com/AKSW/Sparqlify>

²⁴ <http://aksw.org/Projects/LIMES.html>

Segundo Mendes et al. (2004) uma aplicação de software *web* é uma aplicação de software convencional que depende da infraestrutura *web* para a sua execução. Esta dependência pode ser parcial, visto que uma aplicação onde apenas um ou alguns módulos da aplicação utilizam a infraestrutura *web* para sua execução. A escolha pelo desenvolvimento de uma aplicação *web* deu-se por diversos motivos, sendo os principais:

1. Facilidade no acesso através dos computadores atuais visto sua dependência apenas de um navegador *web* para ser acessada;
2. Compatibilidade garantida independente do sistema operacional;
3. Atualização de versões facilitada;
4. Maior facilidade de integração com os demais componentes e ferramentas utilizadas pelo LODFlow;

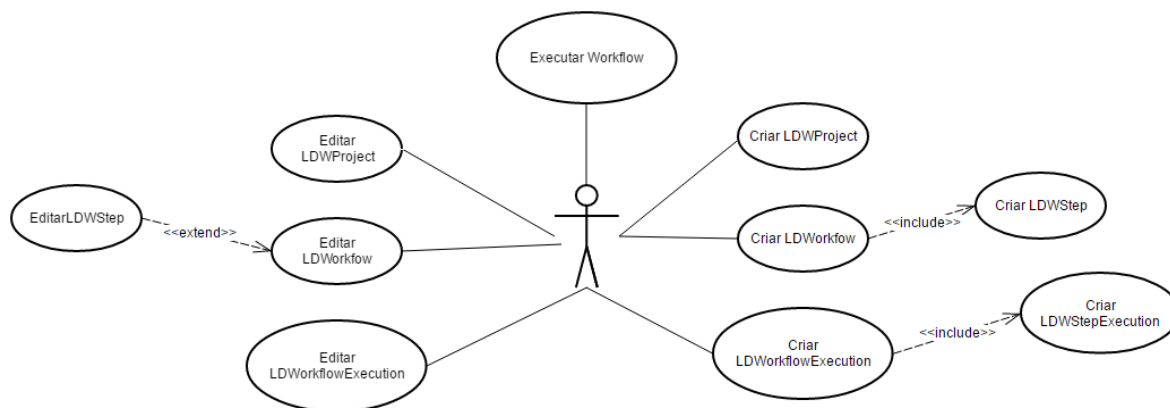
A ferramenta de terceiros, utilizada atualmente pelo LODFlow para manipulação de sua base de conhecimento é um ambiente tradicional de manipulação de ontologias, onde necessita que o usuário tenha conhecimento prévio desta modelo de dados para entender como lidar com a manipulação das informações a serem inseridas e trabalhadas. A LDWPO contém uma diversidade de conceitos acerca do processo de publicação de dados *linked data* e o objetivo principal é encapsular todos estes conceitos — que podem dificultar o entendimento com relação a instanciação dos indivíduos necessários para que o fluxo de trabalho seja materializado — trazendo nesta nova aplicação uma interface mais amigável, customizada para o LODFlow e inteligível para interação com a LDWPO, necessária para que o fluxo de trabalho para publicação de *linked data* seja corretamente definido e possa ser executado com êxito.

3.3.1. Modelagem UML

Esta seção apresenta alguns dos treze diagramas que compõem a linguagem UML (*Unified Modelling Language*) e que auxiliam na definição do projeto da aplicação bem como elucidam algumas questões acerca do funcionamento da mesma. Utilizaremos o diagrama de casos de uso — que modela a dinâmica do sistema no mais alto nível de

abstração propiciado por UML, relacionando as funcionalidades do sistema e os elementos externos que interagem com ele (E SILVA, 2007, p. 93).

Figura 10: Diagrama de Casos de Uso do SysLODFlow

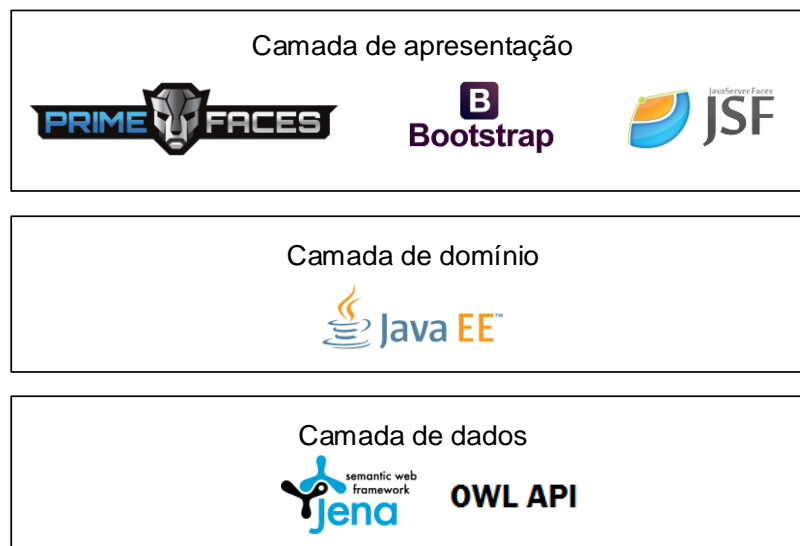


Fonte: Autor.

3.4. Definição da arquitetura

Definido o tipo de aplicação a ser desenvolvida, a arquitetura mais usual que nos vem em mente quando falamos de aplicações *web* é a arquitetura em três camadas. Podemos segmentar as camadas, segundo nomenclatura definida por Brown et al. (2003), em apresentação, domínio e fonte de dados. A camada de apresentação trata da interação entre usuário e *software*. Já a camada de domínio, também conhecida como camada de lógica de negócio define as regras de negócio da aplicação (inteligência do sistema). Por fim a camada de dados, que trata da manipulação dos dados que estão ou serão persistidos em um repositório, tradicionalmente bancos de dados ou arquivos.

Figura 11: Esquema representando a arquitetura em três camadas com artefatos utilizados para o desenvolvimento.



Fonte: Autor, adaptado de Brown et al. (2003).

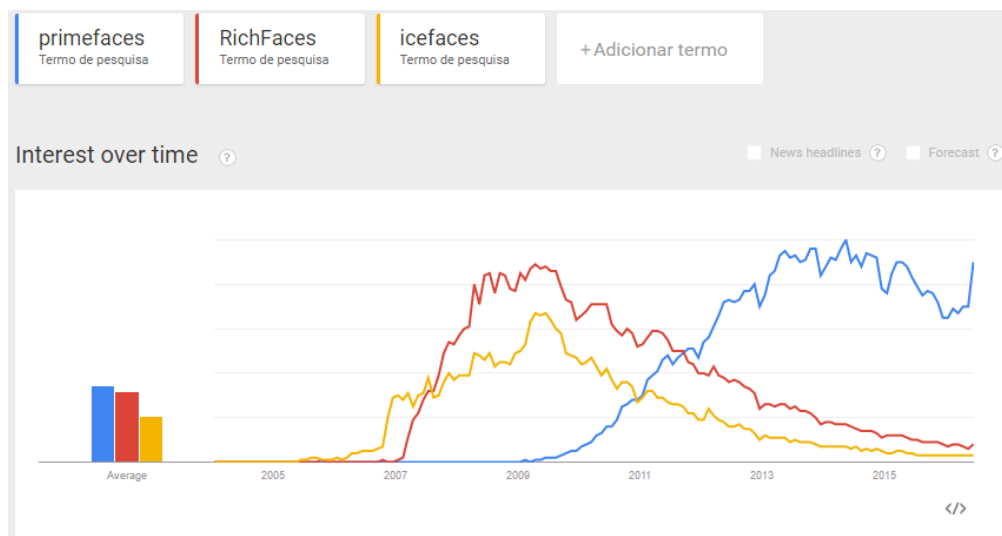
A figura 11 ilustra a segmentação da aplicação de acordo com o modelo proposto pela arquitetura em três camadas e os artefatos tecnológicos utilizados para materialização das mesmas. A tecnologia Java, uma das linguagens de programação mais utilizadas atualmente [35], está presente em todos os segmentos de aplicação.

Para camada de apresentação a opção deu-se pelo *framework Primefaces*²⁵. O *Primefaces* é um *framework* para desenvolvimento de projetos *web* baseados na tecnologia *JavaServer Faces*²⁶ (JSF), que permite a implementação ágil de aplicações, sejam elas simples ou sofisticadas. Na prática o *Primefaces* fornece uma biblioteca vasta de componentes de interface gráfica para utilização em aplicações *web* baseadas em JSF, tendo como fator chave para sua utilização a melhora da produtividade do time de desenvolvimento e da experiência do usuário.

²⁵ <http://primefaces.org/>

²⁶ <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

Figura 12: Popularidade do Primefaces perante concorrentes.



Fonte: *Google Trends* (2016).

Os motivos que levaram a escolha deste *framework* para desenvolvimento da interface da aplicação foram a experiência prévia da equipe e principalmente o crescimento perante aos demais concorrentes, comprovado através do gráfico de popularidade do *Google Trends*²⁷ mostrado na figura 12. Além dos componentes fornecidos pelo *Primefaces*, componentes do JSF “puro” também foram utilizados para construção da interface. Integrado ao *Primefaces* e JSF, o *Bootstrap*²⁸ — um dos mais populares *frameworks* HTML, CSS, e JavaScript para desenvolvimento de projetos responsivo e focado para dispositivos móveis na *web* — foi adotado para incorporar uma maior gama de customização dos componentes de interface.

Para desenvolvimento do *core* da aplicação, ou seja, a camada de domínio, onde contém a lógica da aplicação, a tecnologia Java *Enterprise Edition*²⁹ (Java EE) foi adotada. A camada de dados, que contém as classes responsáveis pela manipulação dos dados persistidos é composta pelo *framework Apache Jena*. O *Jena* é um *framework* Java, livre e de código aberto para desenvolvimento de aplicações *Linked Data* e para *Web Semântica* [36]. A escolha por este *framework* deu-se devido sua boa documentação e principalmente pelo alto grau de adesão deste pela comunidade que desenvolve aplicações

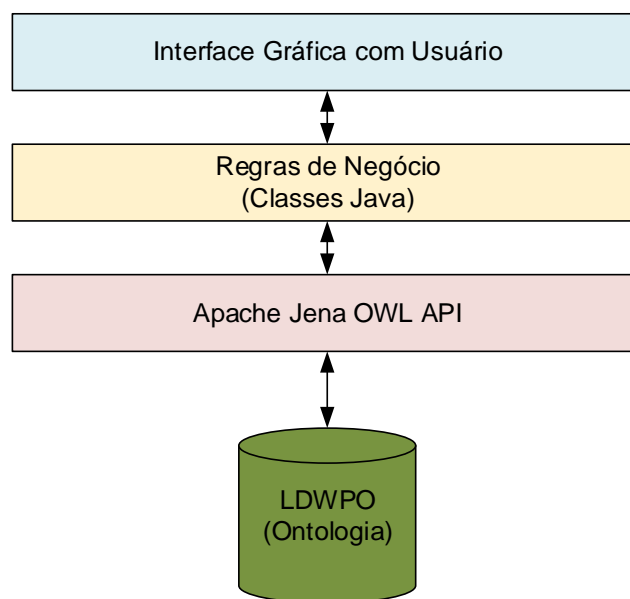
²⁷ <https://www.google.com/trends>

²⁸ <http://getbootstrap.com.br/>

²⁹ <http://www.oracle.com/technetwork/java/javaee/overview/index.html>

no segmento de *Web Semântica*, fatores importantes para uma rápida aprendizagem por parte da equipe.

Figura 13: Interação entre as camadas da aplicação.



Fonte: Autor.

Dentre os diversos recursos e API's fornecidos pelo *Jena*, a OWL API³⁰ fornece os recursos necessários para a manipulação de ontologias OWL, realizando a abstração do modelo ontológico para um conjunto de classes muito abrangente, que no caso deste trabalho, permite a interação com a base de conhecimento a ser trabalhada, a ontologia LDWPO. A figura 13 demonstra um esquema simplificado de interação entre as camadas da aplicação, com ênfase na interação entre a camada de dados (representada pelo seu componente OWL API do *Jena*) e a base de conhecimento que compõe o LODFlow, a ontologia LDWPO, visto que esta mesma ontologia será utilizada pela aplicação *SysLODFlow*.

³⁰ <https://jena.apache.org/documentation/ontology/>

3.5. Desenvolvimento

Conforme mencionado ao longo deste capítulo, a aplicação foi arquitetada em três camadas com o intuito de torná-la o mais modular possível, facilitando futuras manutenções e seguindo boas práticas de desenvolvimento adquiridas ao longo da vivência acadêmica. Utilizando-se de *frameworks* terceiros para viabilizar o desenvolvimento optou-se por utilizar o gerenciador de dependências *Maven* para gestão das dependências de módulos, componentes externos, bibliotecas, etc. O *Maven* trata de realizar o download das dependências (bibliotecas Java, por exemplo) de um ou mais repositórios centralizados e mantém em uma área de cache local, permitindo de igual forma a atualização deste cache local com artefatos de software produzidos por projetos locais. O *Maven* possui um arquivo XML (pom.xml) responsável por descrever o projeto construído, suas dependências, aspectos relativos a compilação de código e empacotamento.

Figura 14: Trecho do arquivo pom.xml do Maven para o projeto SysLODFlow

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>lodflow</groupId>
    <artifactId>syslodflow</artifactId>
    <version>1.0.0</version>
    <packaging>war</packaging>
    <name>syslodflow</name>
</project>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/xsd/maven-4.0.0.xsd" e>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <repositories>
        <repository>
            <id>prime-repo</id>
            <name>PrimeFaces Maven Repository</name>
            <url>http://repository.primefaces.org</url>
            <layout>default</layout>
        </repository>
    </repositories>

    <dependencies>
        <!-- PRIMEFACES -->
        <dependency>
            <groupId>org.primefaces</groupId>
            <artifactId>primefaces</artifactId>
            <version>5.0</version>
```

```

</dependency>

<dependency>
  <groupId>org.primefaces.themes</groupId>
  <artifactId>all-themes</artifactId>
  <version>1.0.10</version>
</dependency>
<!-- JENA -->
<dependency>
  <groupId>org.apache.jena</groupId>
  <artifactId>jena-arq</artifactId>
  <version>2.13.0</version>
</dependency>

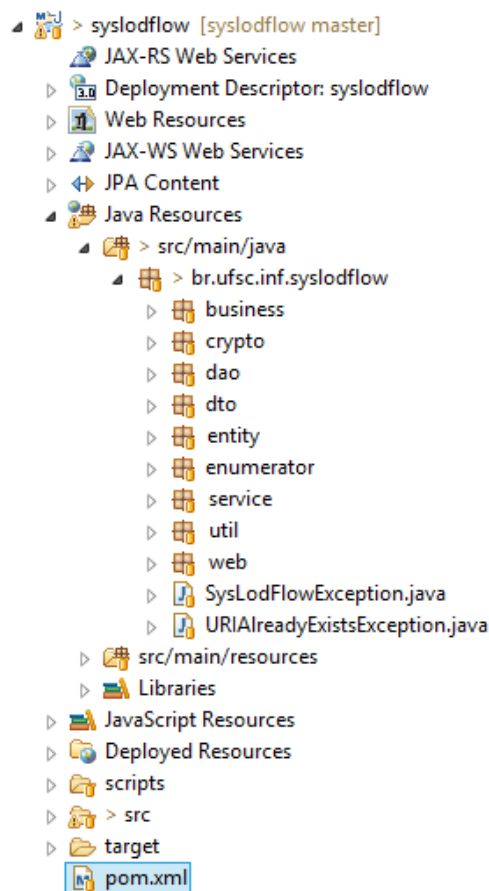
```

Fonte: Autor.

Previamente abordados na etapa de definição da arquitetura do *software*, os artefatos tecnológicos utilizados, como *Primefaces*, *JSF*, *Apache Jena* foram adicionados gerenciador *Maven* — conforme figura 14 — permitindo suas utilizações no código a ser escrito.

O projeto Java foi dividido em uma estrutura de pacotes onde cada um deles contém uma ou mais classes com comportamento comum, ou seja, que proveem funcionalidades similares para a composição da aplicação.

Figura 15: Workspace do projeto Java utilizando a IDE Eclipse



Fonte: Autor.

A figura 15 mostra a estrutura do projeto Java no ambiente de desenvolvimento da IDE Eclipse³¹. Abaixo uma descrição do papel de cada um dos conjuntos de classes contidos nos pacotes definidos:

1. *br.inf.ufsc.br.syslodflow.business*: Classes responsáveis por definir as regras de negócio de autenticação de usuários no sistema.
2. *br.inf.ufsc.br.syslodflow.crypto*: Contém classe utilizada para criptografia das senhas de usuário.
3. *br.inf.ufsc.br.syslodflow.dao*: Classes responsáveis pelo acesso aos objetos armazenados em banco de dados, neste escopo aplica-se apenas aos usuários para autenticação no sistema.

³¹ <http://www.eclipse.org/>

4. *br.inf.ufsc.br.syslodflow.dto*: Abstrações de algumas entidades, com um número reduzido de atributos para fins de exibição em tela.
5. *br.inf.ufsc.br.syslodflow.entity*: Conjunto de classes que representam no modelo de orientação à objeto as classes contidas na ontologia LDWPO que serão utilizadas para manipulação desta. Ou seja, foi realizado um mapeamento de tais classes da ontologia para classes Java de modo a permitir a inserção e recuperação de dados na ontologia a partir desta aplicação.
6. *br.inf.ufsc.br.syslodflow.enumerator*: Conjunto de enumerados contendo constantes a serem utilizadas pelo programa, como as URI's das classes e propriedades da ontologia LDWPO, facilitando o desenvolvimento e melhorando a consistência do código.
7. *br.inf.ufsc.br.syslodflow.service*: Contém o conjunto de classes responsável pela interação direta com a ontologia LDWPO. As operações de inserção, atualização e recuperação de dados da ontologia são realizadas por estas classes, que se utilizam das classes fornecidas pela OWL API do *Jena* para realizar estas funcionalidades.
8. *br.inf.ufsc.br.syslodflow.util.util*: Classes utilitárias para realização de operações internas do software.
9. *br.inf.ufsc.br.syslodflow.web*: Contém as classes responsáveis por fazer a integração entre interface gráfica com usuário e regras de negócio da aplicação, através da utilização de *ManagedBeans*³²

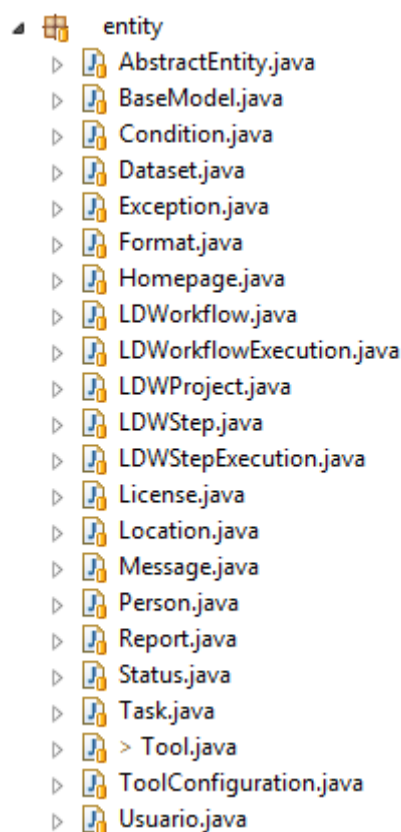
3.5.1. Entidades

Para criação de um modelo orientado à objeto (OO) das classes que compõem o modelo de conhecimento proposto pelo *Linked Data Workflow Knowledge Model* — armazenado na ontologia LDWPO — foi realizado um mapeamento destas classes e extraído seus atributos vinculados para posteriormente então criar uma abstração das mesmas em Java, procurando manter fidedignidade do modelo OO Java em relação ao representado pela ontologia. O processo de mapeamento foi realizado a partir de estudos

³² <http://docs.oracle.com/javaee/7/api/javax/faces/bean/ManagedBean.html>

realizados sobre o conteúdo do arquivo OWL que representa a ontologia, como também do trabalho de Rautenberg *et al.* (2015) que apresenta a *Linked Data Workflow Project Ontology* (LDWPO). A figura 16 exibe as classes criadas, contidas no pacote *br.inf.ufsc.br.syslodflow.entity*.

Figura 16: Entidades Java relacionadas as classes pertencentes ao modelo contido na LDWPO



Fonte: Autor.

Tomando como exemplo, para demonstrar como foi realizado o mapeamento das classes e atributos vinculados, abaixo as classes *LDWProject* (figura 17) e *LDWorkflow* (figura 18), que representam as classes de mesmo nome na LDWPO.

Figura 17: Classe LDWProject

```

1 package br.ufsc.inf.syslodflow.entity;
2
3 public class LDWProject extends BaseModel {
4
5     private Person creator;
6     private Homepage homePage;
7     private LDWorkflow ldWorkFlow;
8     private Report report;
9     private String goal;
10    private String description;
11    private String name;
12    private String fileName;
13    private String uri;
14

```

Fonte: Autor.

Figura 18: Classe LDWorkflow

```

1 package br.ufsc.inf.syslodflow.entity;
2
3 import java.util.List;
4
5 public class LDWorkflow extends BaseModel {
6
7     private String description;
8     private String name;
9     private Condition preCondition;
10    private Condition postCondition;
11    private LDWStep firstLdwStep;
12    private List<LDWorkflowExecution> ldWorkFlowExecutions;
13    private List<LDWStep> ldwSteps;
14    private String uri;
15

```

Fonte: Autor.

Na definição da ontologia é possível observar, por exemplo, a relação entre a propriedade de dados (*data property*) *goal* e a classe *LDWProject* e a propriedade de objeto (*object property*) *preCondition* com a classe *LDWorkflow*. Isso se dá através de uma propriedade de domínio, que vincula um sujeito que utiliza esta propriedade com este atributo de domínio associado como um tipo de coisa especificado por este domínio, conforme visto nas figuras 19 e 20.

Figura 19: Propriedade de dados “goal” relacionada a classe LDWProject

```
<!-- http://ldwpo.aks.w.org/terms/1.0/goal -->

<owl:DatatypeProperty rdf:about="http://ldwpo.aks.w.org/terms/1.0/goal">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:label xml:lang="en">goal</rdfs:label>
  <rdfs:comment xml:lang="en">Data property that expresses the set of goals
of an LDWProject. As restriction, goal is a functional property and
accepts only xsd:String values.</rdfs:comment>
  <rdfs:domain rdf:resource="http://ldwpo.aks.w.org/terms/1.0/LDWProject"/>
  <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
```

Fonte: Autor, retirado de Rautenberg *et al.* (2015).

Figura 20: Propriedade de dados “preCondition” relacionada a classe LDWorkflow

```
<!-- http://ldwpo.aks.w.org/terms/1.0/precondition -->

<owl:ObjectProperty rdf:about="http://ldwpo.aks.w.org/terms/1.0/precondition">
  <rdfs:label rdf:datatype="&xsd:string">precondition</rdfs:label>
  <rdfs:comment xml:lang="en">Object property that points to the Condition to be presented
before an LDWorkflow is executed by an LDWorkflowExecution.
As restrictions, one LDWorkflow has one or more Conditions as precondition and one
Condition can be related as recondition to one or more LDWorkflows.
</rdfs:comment>
  <rdfs:range rdf:resource="http://ldwpo.aks.w.org/terms/1.0/Condition"/>
  <rdfs:domain rdf:resource="http://ldwpo.aks.w.org/terms/1.0/LDWorkflow"/>
</owl:ObjectProperty>
```

Fonte: Autor, retirado de Rautenberg *et al.* (2015).

3.5.2. Trabalhando com Jena OWL API

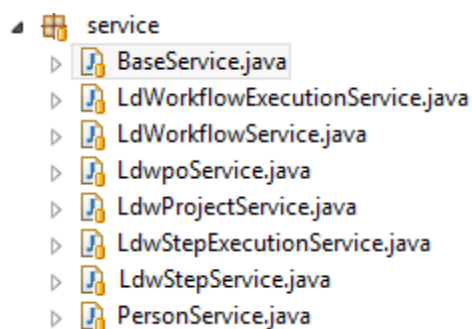
O *framework Jena* fornece um conjunto de artefatos que permitem a construção de aplicações para *Web Semântica* e *Linked Data*. Para os fins de desenvolvimento desta aplicação a OWL API contida no *Jena* foi utilizada por conter toda a lógica envolvida necessária para manipulação de modelos ontológicos. A utilização desta API se faz necessária pelo fato de toda a persistência de dados envolvida no LODFlow estar centralizada na ontologia LDWPO.

3.5.2.1. Aspectos gerais

Esta ontologia fica armazenada no servidor, em um arquivo de extensão “.owl” e é carregada para a memória a cada operação requisitada através da OWL API. O conjunto

de classes implementadas responsáveis por esta interação foram abrigadas no pacote *br.inf.ufsc.br.syslodflow.service*, conforme mostra a figura 21.

Figura 21: Classes responsáveis pela interação com a ontologia através da OWL API do Jena



Fonte: Autor.

Os elementos básicos de uma ontologia OWL são as classes, os indivíduos (também chamadas de instâncias) das classes e os relacionamentos (as propriedades) entre estes indivíduos [40]. Nas próximas subseções apresentaremos de que forma estes elementos foram trabalhados a partir dos recursos fornecidos pelo *Jena*.

3.5.2.2. Carga e escrita de uma ontologia

A classe *LdwpoService.java* é a responsável na aplicação pela manipulação dos dados trabalhados na ontologia e para isto é necessário carregar a ontologia para a memória. O *Jena* permite esta operação à partir da instanciação de um *OntModel*, que através do método *createOntologyModel* da classe *ModelFactory* cria um modelo que irá armazenar um arquivo contendo a ontologia contido no diretório definido como parâmetro do método *read*, conforme ilustrado no trecho de código da figura 22. O modelo criado é do tipo *OWL_MEM*, que segundo a documentação da API [37] representa a linguagem *OWL Full*³³ que não conta com motor de inferência.

³³ <https://www.w3.org/TR/owl-ref/>

Figura 22: Criação de um modelo e leitura da ontologia para memória.

```
public OntModel doLoadModel(Path path) {

    OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
    model.read(path.toUri().toString(), "");
    return model;
}
```

Fonte: Autor.

Todas as operações de inserção na ontologia são previamente salvas no modelo instanciado (*OntModel*) para posteriormente, ao voltar o controle da execução para a classe *LdwpoService.java* sejam persistidas na ontologia. O método *doSaveModel* (vide figura 23) foi implementado para ao receber o fluxo de execução, contendo o modelo utilizado no contexto atual e o nome do arquivo que será escrito em disco, seja realizada a persistência das informações. O método *write* da classe *OntModel* do *Jena* permite o salvamento dessas informações, onde é necessário definir o caminho onde o arquivo será salvo bem como a sintaxe a ser utilizada.

Figura 23: Persistência de um modelo para arquivo

```
public void doSaveModel(OntModel model, String fileName) {
    FacesContext fc = FacesContext.getCurrentInstance();
    String filePath = fc.getExternalContext().getInitParameter("filePath").toString();

    try {
        File file = new File(filePath + fileName);
        model.write(new FileOutputStream(file), LangModelEnum.RDFXMLABBREV.getType());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Fonte: Autor.

Uma observação importante a ser feita com relação ao controle da execução para que não ocorra perda da referência é que todo método que irá manipular diretamente o modelo ontológico — ou seja, alterar informações contidas no mesmo — retorne de sua execução o modelo alterado, para que os métodos de níveis acima possam realizar suas operações sem perda de referência.

3.5.2.3. Classes

As classes de uma ontologia classes proveem um mecanismo de abstração para agrupar recursos com características similares, ou seja, uma classe define um grupo de indivíduos que compartilham algumas propriedades [40]. Num contexto de orientação a objetos a Classe tem o mesmo objetivo, descrever um conceito, um conjunto de características que definem algo, no caso de orientação à objetos um objeto, já no contexto de uma ontologia um indivíduo. A ontologia LDWPO define um conjunto de classes para descrever do domínio de fluxo de trabalho *linked data* e sendo assim é necessário a realização de um mapeamento destas classes para possibilitar a execução de operações sobre as mesmas.

A OWL API do *Jena* fornece uma série de artefatos para trabalhar com classes de um modelo ontológico. Para este trabalho utilizamos principalmente a *Interface*³⁴ *OntClass*, que assim como outras interfaces da API pertence à uma hierarquia de interfaces, fornecendo uma cadeia de métodos extensa para manipulação de recursos. Como trabalhamos com uma ontologia já definida, a LDWPO, as operações realizadas sobre classes foram principalmente de recuperação das mesmas para utilização na instanciação de seus indivíduos e operações internas. Como adoção de boas práticas foi realizada a criação de um *Enum*³⁵ (vide figura 24) para mapeamento das URI's das classes utilizadas, úteis para realização de operações a partir dos métodos fornecidos pelo *Jena*.

³⁴ <https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

³⁵ H
<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

Figura 24: *ClassURIEnum*: Enum contendo URI's das classes da ontologia LDWPO

```

3 public enum ClassURIEnum {
4     LDWPROJECT("http://ldwpo.aksw.org/terms/1.0/LDWProject"),
5     LDWORKFLOW("http://ldwpo.aksw.org/terms/1.0/LDWorkflow"),
6     LDWORKFLOWEXECUTION("http://ldwpo.aksw.org/terms/1.0/LDWorkflowExecution"),
7     LDWSTEP("http://ldwpo.aksw.org/terms/1.0/LDWStep"),
8     CONDITION("http://ldwpo.aksw.org/terms/1.0/Condition"),
9     PERSON("http://ldwpo.aksw.org/terms/1.0/Person"),
10    HOMEPAGE("http://ldwpo.aksw.org/terms/1.0/Homepage"),
11    LOCATION("http://ldwpo.aksw.org/terms/1.0/Location"),
12    TOOL("http://ldwpo.aksw.org/terms/1.0/Tool"),
13    TOOLCONFIGURATION("http://ldwpo.aksw.org/terms/1.0/ToolConfiguration"),
14    REPORT("http://ldwpo.aksw.org/terms/1.0/Report"),
15    FORMAT("http://ldwpo.aksw.org/terms/1.0/Format"),
16    DATASET("http://ldwpo.aksw.org/terms/1.0/Dataset"),
17    LICENSE("http://ldwpo.aksw.org/terms/1.0/License"),
18    MESSAGE("http://ldwpo.aksw.org/terms/1.0/Message");
19 }

```

Fonte: Autor.

Um trecho de código que exemplifica uma das operações realizadas sobre uma classe da ontologia pode ser observado na figura 25. Neste trecho é realizada uma busca no modelo pela classe *LDWStep*, através da passagem do atributo URI armazenado no Enum mostrado na figura 24. De posse desta informação, através da execução do método *getOntClass* do modelo obtém-se uma instância do tipo *OntClass* representando a classe *LDWStep* definida na LDWPO.

Figura 25: *Uso de classe da ontologia através do Jena*

```

273 public OntModel insertLdwStep(OntModel model, LDWStep step) {
274
275     Individual ldwstep = model.getOntClass(ClassURIEnum.LDWSTEP.getUri())
276         .createIndividual(step.getUri());

```

Fonte: Autor.

3.5.2.4. Indivíduos

Os indivíduos são objetos básicos de uma ontologia, são a materialização de um conceito contido definido por uma classe, fazendo analogia a com uma linguagem orientada a objetos um indivíduo seria um objeto (instância) de uma classe. A API OWL do *Jena* possui diversos artefatos para trabalhar com este tipo de elemento. Para fins de desenvolvimento deste trabalho utilizamos a interface *Individual* que faz parte de uma

hierarquia de interfaces para manipulação de recursos oriundos da ontologia. A partir desta *Interface* podemos instanciar novos indivíduos no modelo ontológico, adicionar e remover propriedades e seus valores e recuperar informações acerca dos mesmos. Na figura 26 é possível observar uma série de operações realizadas sobre um indivíduo da classe LDWProject criado através dos métodos fornecidos pelo *Jena* através da interface *Individual*.

Figura 26: Criação de uma novo Indivíduo da classe LDWProject

```

150 private OntModel insertLdwProject(OntModel model, LDWProject project) {
151
152     Individual ldwProject = model.getOntClass(
153         ClassURIEnum.LDWPROJECT.getUri()).createIndividual(
154         project.getUri());
155     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.NAME.getUri()),
156         project.getName());
157     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.GOAL.getUri()),
158         project.getGoal());
159     ldwProject.addLiteral(
160         model.getProperty(PropertyURIEnum.DESCRPTION.getUri()),
161         project.getDescription());
162     ldwProject.addProperty(
163         model.getProperty(PropertyURIEnum.CREATOR.getUri()),
164         model.getIndividual(project.getCreator().getUri()));
165     model = writeHomepage(model, project.getHomePage());
166     ldwProject.addProperty(
167         model.getProperty(PropertyURIEnum.HOMEPAGE.getUri()),
168         model.getIndividual(project.getHomePage().getUri()));
169     model = writeReport(model, project.getReport());
170     ldwProject.addProperty(
171         model.getProperty(PropertyURIEnum.REPORT.getUri()),
172         model.getIndividual(project.getReport().getUri()));
173     return model;
174 }
175

```

Fonte: Autor.

Na linha 152, 153 e 154 da figura 26 observa-se a instrução para criação de um novo indivíduo na ontologia. A mesma lógica é utilizada em todos os pontos do programa quando da necessidade de se instanciar novos elementos. Através do método *getOntClass* do modelo a ser manipulado naquele contexto é possível obter a classe desejada, recuperada através da URI da classe informada no parâmetro, para então esta classe recuperada executar o seu método *createIndividual*, que cria um novo indivíduo desta classe e adiciona-o ao modelo. Por fim este novo indivíduo criado é armazenado na variável local *ldwProject* para operações subsequentes a serem realizadas.

3.5.2.5. Propriedades

As propriedades, que são relações binárias, podem ser usadas para estabelecer relacionamentos entre indivíduos ou entre indivíduos e valores de dados [40]. Estes relacionamentos permitem afirmar fatos gerais sobre os membros das classes e podem também especificar fatos sobre indivíduos [41]. McGuinness, Smith e Welty (2004) descrevem a distinção contida na OWL entre duas categorias de propriedades:

- *Datatype properties* (propriedades de dados tipados): relação entre indivíduos e valores de dados.
- *Object Properties* (propriedades de objeto): relação entre indivíduos.

Para utilização de propriedades definidas na LDWPO foi utilizada a *Interface Property*, que contém um conjunto de métodos abrangente — principalmente pela herança das *SuperInterfaces* contidas em sua hierarquia — para manipulação de propriedades, sejam elas *datatype properties* ou *object properties*. Assim como realizado para as classes, foi criado um *Enum* (vide figura 27, apenas trecho do código) para conter as URI's das propriedades definidas na LDWPO, seguindo boas práticas e facilitando a escrita de código.

Figura 27: *PropertyURIEnum*: Enum contendo URI's das propriedades da ontologia LDWPO

```

3 public enum PropertyURIEnum {
4     NAME("http://ldwpo.aksw.org/terms/1.0/name"),
5     DESCRIPTION("http://ldwpo.aksw.org/terms/1.0/description"),
6     GOAL("http://ldwpo.aksw.org/terms/1.0/goal"),
7     HOMEPAGE("http://ldwpo.aksw.org/terms/1.0/homepage"),
8     LOCATION("http://ldwpo.aksw.org/terms/1.0/location"),
9     VALUE("http://ldwpo.aksw.org/terms/1.0/value"),
10    CREATOR("http://ldwpo.aksw.org/terms/1.0/creator"),
11    CONTRIBUTOR("http://ldwpo.aksw.org/terms/1.0/contributor"),
12    REPORT("http://ldwpo.aksw.org/terms/1.0/report"),
13    PRECONDITION("http://ldwpo.aksw.org/terms/1.0/precondition"),
14    POSTCONDITION("http://ldwpo.aksw.org/terms/1.0/postcondition"),
15    LDWORKFLOW("http://ldwpo.aksw.org/terms/1.0/ldWorkflow"),
16    LDWSTEP("http://ldwpo.aksw.org/terms/1.0/ldwStep"),
17    LDWSTEPEXECUTION("http://ldwpo.aksw.org/terms/1.0/ldwStepExecution"),
18    FIRSTLDWSTEP("http://ldwpo.aksw.org/terms/1.0/firstLdwStep"),
19    FIRSTLDWSTEPEXECUTION("http://ldwpo.aksw.org/terms/1.0/firstLdwStepExecution"),
20    LDWORKFLOWEXECUTION("http://ldwpo.aksw.org/terms/1.0/ldWorkflowExecution"),

```

Fonte: Autor.

Na figura 28 apresentamos um trecho de código do método *getPropertyStringValue*, implementado com a finalidade de ser um método genérico para recuperação de valores literais de *datatype properties* de um indivíduo qualquer. Uma aplicação deste método pode ser vista na figura 29, para obter o valor da propriedade *description* de um indivíduo da classe *LDWorkflow*.

Figura 28: Método genérico para recuperação de valor de *datatype property* de um indivíduo

```

46 public static String getPropertyStringValue(Individual individual,
47     OntModel model, String uriProperty) {
48     if (individual != null) {
49         Property property = model.getProperty(uriProperty);
50         String value = individual.getPropertyValue(property).asLiteral()
51             .getString();
52         return value;
53     }
54     return "";
55 }
56

```

Fonte: Autor.

Figura 29: Exemplo de utilização do método *getPropertyStringValue* para recuperação de valor de *datatype property* de um indivíduo

```

44 String ldwWorkflowDescription = getPropertyStringValue(ontLdWorkflow,
45     model, PropertyURIEnum.DESCRPTION.getUri());

```

Fonte: Autor.

Para recuperação de um valor de *object property* de um indivíduo, ou seja, um outro indivíduo, utilizamos uma instrução como a contida na figura 30, que na prática recupera um indivíduo da classe *Person* associado a um indivíduo da classe *LDWProject* através da propriedade de objeto *creator*.

Figura 30: Exemplo de recuperação de uma object property

```

36 Individual creator = model.getIndividual(ontProject
37     .getPropertyResourceValue(
38         model.getProperty(PropertyURIEnum.CREATOR.getUri()))
39     .getURI());

```

Fonte: Autor.

Operações de inserção de propriedades são fundamentais para a definição correta e completa de indivíduos. Estas operações estão, assim como as demais relacionadas a ontologia, estão implementadas nas classes que interagem diretamente o *Jena*. Os métodos *addLiteral* e *addProperty* são frequentemente utilizados para inserção respectivamente de valores literais para *datatype properties* e indivíduos para *object properties*. A figura 31 ilustra este comportamento através do método implementado *insertLdwProject* cria um indivíduo da classe *LDWProject* e adiciona suas propriedades com seus respectivos valores. Já a figura 32 apresenta o método *editHomepage* que atualiza os valores das propriedades de um indivíduo da classe *Homepage*, a partir de novos dados definidos na interface. Para fins de atualização foi realizada a remoção das propriedades vinculadas aquele indivíduo, através do método *removeAll* passando como parâmetro a propriedade a ser removida, e inserindo-a novamente através dos respectivos métodos já descritos anteriormente.

Figura 31: Exemplo de inserção de propriedades a indivíduos

```

153 private OntModel insertLdwProject(OntModel model, LDWProject project) {
154
155     Individual ldwProject = model.getOntClass(
156         ClassURIEnum.LDWPROJECT.getUri()).createIndividual(
157         project.getUri());
158     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.NAME.getUri()),
159         project.getName());
160     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.GOAL.getUri()),
161         project.getGoal());
162     ldwProject.addLiteral(
163         model.getProperty(PropertyURIEnum.DESCRPTION.getUri()),
164         project.getDescription());
165     ldwProject.addProperty(
166         model.getProperty(PropertyURIEnum.CREATOR.getUri()),
167         model.getIndividual(project.getCreator().getUri()));
168     model = writeHomepage(model, project.getHomePage());
169     ldwProject.addProperty(
170         model.getProperty(PropertyURIEnum.HOMEPAGE.getUri()),
171         model.getIndividual(project.getHomePage().getUri()));
172     model = writeReport(model, project.getReport());
173     ldwProject.addProperty(
174         model.getProperty(PropertyURIEnum.REPORT.getUri()),
175         model.getIndividual(project.getReport().getUri()));
176     return model;
177 }
178

```

Fonte: Autor.

Figura 32: Exemplo de atualização de propriedades a indivíduos

```

242 private OntModel editHomepage(OntModel model, Homepage h) {
243
244     Individual homepage = model.getIndividual(h.getUri());
245     homepage.removeAll(model.getProperty(PropertyURIEnum.NAME.getUri()));
246     homepage.addLiteral(model.getProperty(PropertyURIEnum.NAME.getUri()), h.getName());
247     Individual location = model.getIndividual(h.getLocation().getUri());
248     location.removeAll(model.getProperty(PropertyURIEnum.VALUE.getUri()));
249     location.addLiteral(model.getProperty(PropertyURIEnum.VALUE.getUri()), h.getLocation().getValue());
250     return model;
251 }

```

Fonte: Autor.

4. APLICAÇÃO DO SYSLODFLOW

Esta seção tem por objetivo expor os resultados obtidos através dos esforços de desenvolvimento do aplicativo de software *SysLODFlow*, os quais serão expostas suas principais funcionalidades no decorrer deste capítulo. Imagens de captura de tela serão utilizadas para melhor entendimento do exposto.

O conjunto de dados utilizado para validação da aplicação foi o do Qualis Brasil, mencionado na seção 2.7.4.

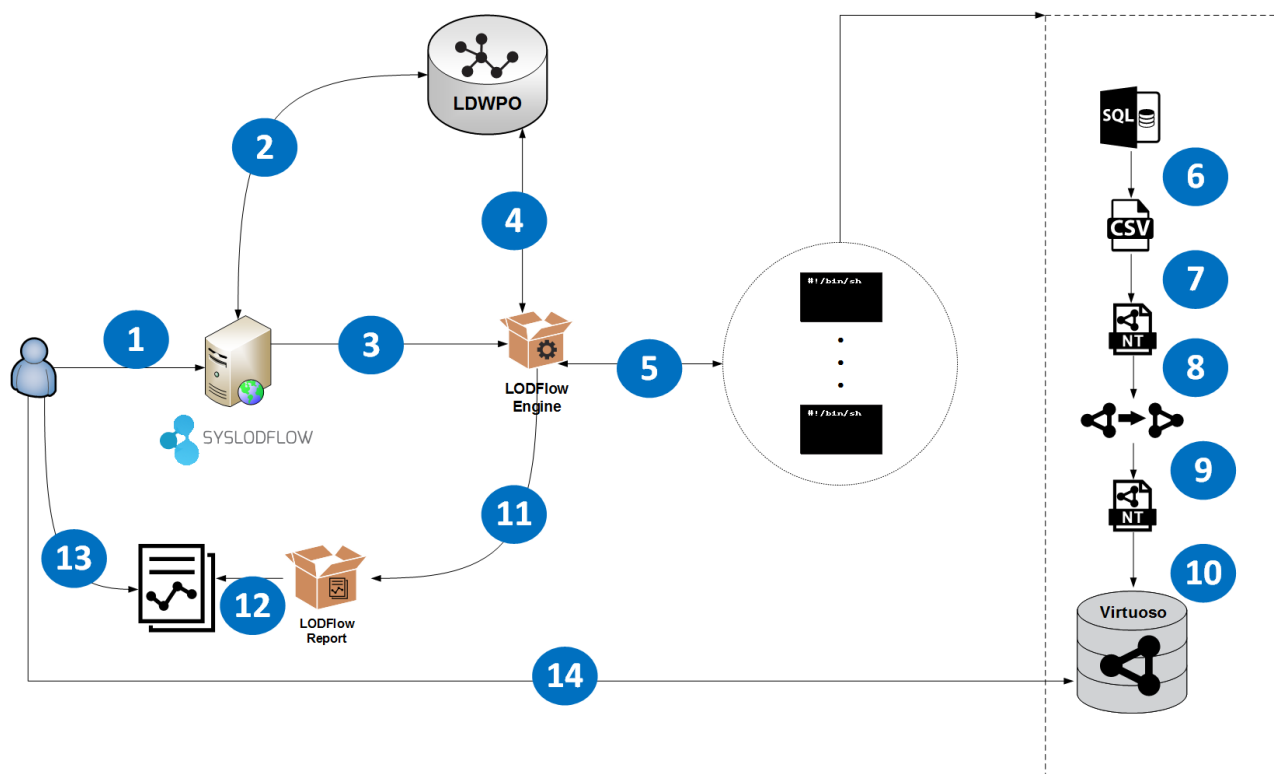
4.1. Visão geral do *workflow* resultante

O *workflow* suportado pela aplicação é baseado no modelo definido pela *LDWPO*, apresentado logo abaixo:

1. Usuário acessa SYSLODFLOW através de seu navegador, realiza as operações devidas e mandar executar o fluxo.
2. SYSLODFLOW armazena dados na ontologia.
3. SYSLODFLOW chama a execução do LODFlow *Engine* para execução do fluxo informando o fluxo a ser executado.
4. LODFlow *Engine* recupera da ontologia os dados relativos ao *workflow* a ser executado.
5. LODFlow *Engine* realiza chamadas via script para execução de ferramentas de apoio, configuradas nos passos.
6. Dados são extraídos de um banco de dados relacional (MySQL) e convertidos para CSV.
7. Sparqlify realiza o mapeamento dos dados contidos no CSV e converte para triplas em formato NT.
8. Arquivos de triplas é armazenado no banco de dados Virtuoso.
9. Triplas são recuperadas do virtuoso e através do LIMES ligadas ao DBPedia.
10. Triplas já ligadas ao DBPedia são armazenadas novamente no triple store Virtuoso.
11. LODFlow *Engine* aciona LODFlow *Report* para geração de relatório de execução do *workflow*.
12. LODFlow *Report* gera relatório em HTML contendo informações acerca da execução.
13. Usuário pode realizar o acesso ao relatório em HTML para verificar status da execução.

14. Usuário pode realizar consultas sobre o banco de dados de triplas Virtuoso através de seu endpoint SPARQL.

Figura 33: Visão geral do fluxo de trabalho linked data, contendo o SYSLODFlow como componente



Fonte: Autor.

4.2. Interface Gráfica do Sistema

Ao selecionar no menu principal da aplicação o item *LDWProject*, o usuário visualizará uma listagem de projetos cadastrados, carregados através do seu arquivo de ontologia específico. Através da tabela de listagem, é possível efetuar o *download* da ontologia referente ao projeto selecionado, assim como abrir a edição/instanciação de um *LDWorkflow*. No botão “Editar” o usuário será direcionado para um formulário de cadastro com informações gerais do projeto como: nome, descrição, objetivo, criador, homepage (vide figura 35). O botão “Excluir” realiza a exclusão física do arquivo de ontologia referente ao projeto selecionado.

Figura 34: Página Inicial da Aplicação.



Fonte: Autor.

Figura 35: Listagem de LDWProjects

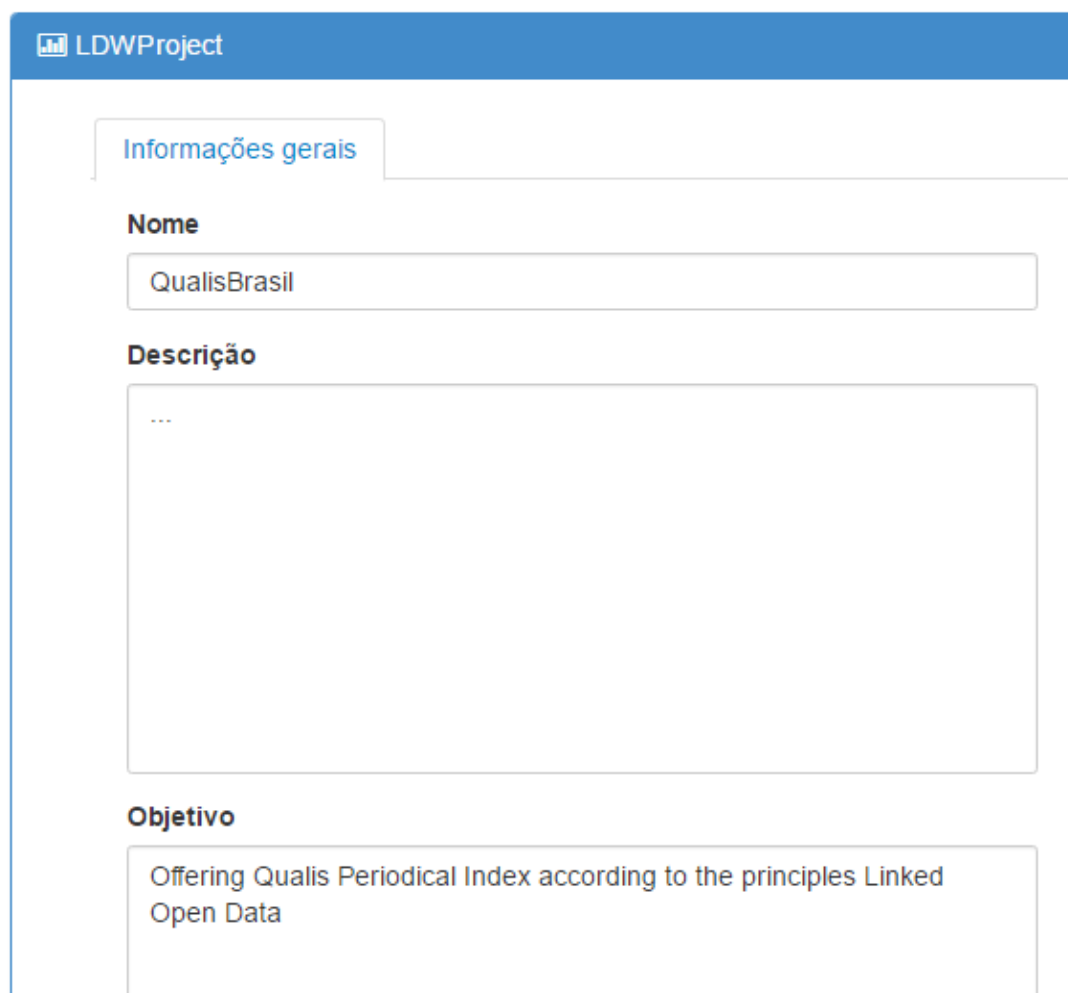
LDWProject

LDWProject				
<div>Novo</div> <div>Mostrar 10 registros por página</div> <div>Pesquisar</div>				
Nome	Criador	OWL	LDWorkflow	Ações
QualisBrasil	Ivan Ermilov	Download	Abrir	Editar Excluir
Exibir de 1 a 1 de 1 registro(s)				Anterior 1 Proximo

Fonte: Autor.

Figura 36: Formulário de instanciação do LDWProject

Instanciar LDWProject



LDWProject

Informações gerais

Nome

QualisBrasil

Descrição

...

Objetivo

Offering Qualis Periodical Index according to the principles Linked Open Data

Fonte: Autor.

A opção de instanciação/edição de um *LDWorkflow*, se dá através do botão “Abrir” na coluna destinada da listagem de projetos. Nesta tela temos subdivisões em abas, que facilita o entendimento do sequenciamento dos passos, assim como o uma melhor visualização dos mesmos. Esta divisão se dá em seis abas: Planejamento do *Workflow*, *Step 01*, *Step 02*, *Step 03*, *Step 04* e *Step 05*. Uma delas contém informações gerais pertinentes ao *workflow*, e as demais para cada um dos *steps*.

Figura 37: Aba de informações gerais do Workflow

LDWorkflow

Planejamento do Workflow

Planejamento do Workflow

LDWStep 01

LDWStep 02

LDWStep 03

Nome

Maintain QualisBrasil

Descrição

Workflow applied to create linked dataset of Qualis Periodicals Index (all years), in an automatized way.

Pré-condições

1. Availability of the data (sql dump or what is there)
2. Running system with installed Ubuntu and all

Fonte: Autor.

As abas referentes aos *steps* possuem a parametrização das informações necessárias para a execução do *workflow* da maneira proposta pelo modelo descrito da *LWPO*, citado no início desta seção. O *step 02* por exemplo, é responsável por mapear um arquivo *CSV* para *N-Triples*.

Figura 38: Aba de informações do Step 02.



The screenshot shows a web interface titled "Planejamento do Workflow". At the top, there are three tabs: "Planejamento do Workflow", "LDWStep 01", and "LDWStep 02". The "LDWStep 02" tab is selected. Below the tabs, the interface is divided into sections:

- Nome:** A text input field containing "PlanningStep - Extracting QualisBrasil applying SPARQLIFY to".
- Descrição:** A text input field containing "What should be done to Extracting QualisBrasil applying SPARQLIFY described here".
- Ferramenta:** A dropdown menu with "Sparqlify" selected.
- Arquivo de mapeamento de dados:** A section with a button "Escolher arquivo" and the text "Nenhum arquivo selecionado". Below this, a note states: "Utilize arquivos no formato .sml. Tamanho máximo: 1MB."

Fonte: Autor.

Através do menu “Execuções” é acessada a tela para visualizações de execuções do *Workflow*. Ao selecionar o projeto, são carregadas informações do *Workflow*, assim como a listagem de execuções do mesmo. O botão “editar” leva o usuário à tela de edição de um *LDWorkflowExecution*. Já o botão novo direciona para a tela de instanciação de um *LDWorkflowExecution*. Os arquivos de mapeamento de dados, que necessitam de upload através da interface do sistema (*steps 02 e 04*), são requisitos das ferramentas utilizadas no *backend* da aplicação. O Sparqlify necessita de um arquivo no formato SML para fazer a relação entre as propriedades do *dataset* CSV para o formato *N-Triples*. Já o LIMES precisa de um mapeamento das propriedades necessárias para efetuar a ligação

do *dataset* local com o DBPedia, este no formato XML. Os exemplos de arquivos utilizados no caso de uso do Qualis Brasil estão expostos nos anexos deste trabalho.

Figura 39: Listagem de Execuções de Workflow.

Execuções

LDWProject

QualisBrasil

LDWorkflow: Informações Gerais

Nome: Maintain QualisBrasil
URI: http://ldwpo.aks.org/terms/1.0/ldWorkflow_maintaining_qualisBrasil
Descrição: Workflow applied to create linked dataset of Qualis Periodicals Index (all years), in an automatized way.

LDWorkflowExecutions Armazenados

Novo

Nome	Relatório	Ações
Maintaining QualisBrasil2011		Editar Excluir
Maintaining QualisBrasil2007		Editar Excluir
Maintaining QualisBrasil2014		Editar Excluir

Fonte: Autor.

Ao selecionar a edição um *LDWorkflowExecution* já existente, o usuário é direcionado para a tela de edição do mesmo, onde é possível efetuar o cadastro/edição das informações gerais, assim como as execuções de *steps*. Cada execução de *step* descrita nesta etapa é baseada na configuração definida no momento do planejamento do *Workflow*. O botão “salvar” persiste os dados na ontologia, já o botão “executar” dispara o *LODFlowEngine*, enviando para execução o *LDWorkflowExecution* definido na tela.

Figura 40: Cadastro de Execuções de Workflow.

Fonte: Autor.

4.3. Artefatos Gerados pelo Workflow

Após a execução do *LDWorkflowExecution* desejado com sucesso, dois artefatos finais são gerados: Um relatório contendo os resultados da execução do *Workflow* e um conjunto de dados RDF.

O relatório de resultados da execução do *workflow* é disponibilizado para download através da interface gráfica, como mostra a figura 39. É um relatório no formato HTML que torna possível observar o detalhamento dos passos executados, como o tempo de execução, status, quantitativo de triplas geradas, entre outros. Para cada um dos passos executados são fornecidas informações que servem de subsídio para determinar o sucesso ou não da operação. As saídas oriundas de ferramentas de apoio são registradas como *output* dos passos correspondentes de modo a facilitar a depuração da execução. Este relatório é provido pelo próprio *LODFlowEngine* e um exemplo pode ser observado na figura 41.

Figura 41: Relatório de Execução de Workflow.

```

WORKFLOW - IdWorkflowExecution_maintaining_qualisBrasil_2014
=====
First step: http://ldwpo.aks.w.org/terms/1.0/ldwStepExecution_02_qualisBrasil2014
Executed workflow: IdWorkflowExecution_maintaining_qualisBrasil_2014
Executed step: IdwStepExecution_02_qualisBrasil2014
Planning step: IdwStep_02_qualisBrasil_converting_data_with_sparqlify Command: Qualis/commands/applyingSparqlify.sh
output of the command: an error occurs: 2016-07-03 17:07:53.892 INFO org.aks.w.sparqlify.csv:CsvMapperCliMain: Errors: 0, Warnings: 0 2016-07-03 17:07:53.965 DEBUG
org.aks.w.sparqlify.csv:CsvMapperCliMain: Detected column names: [Evaluation, Journal, issnJournal, nameJournal, KnowledgeField, idKnowledgeField, nameKnowledgeField, YearEvaluation, yearIndex
Qualis, qualisIndex] Variable #Unbound Triples generated: 13973360 Potential triples omitted: 0 Triples total: 13973360
difference time: 92690.0

started date: Sun Jul 03 17:07:52 BRT 2016

ended date: Sun Jul 03 17:09:24 BRT 2016

Executed workflow: IdWorkflowExecution_maintaining_qualisBrasil_2014
Executed step: IdwStepExecution_03_qualisBrasil2014
Planning step: IdwStep_03_qualisBrasil_storing_converted_data Command: Qualis/commands/savingIntoVirtuoso.sh Qualis/temp/QualisBrasil.nt http://lod.unicentro.br/QualisBrasil/
output of the command: Connected to OpenLink Virtuoso Driver: 06.01.3127 OpenLink Virtuoso ODBC Driver OpenLink Interactive SQL (Virtuoso), version 0.9849b. Type HELP: for help and EXIT: to
exit. SQL> Done. -- 12 msec. SQL> Size = 2249632992 File is large. Performing split on file Qualis/temp/QualisBrasil.nt creating load statement Connected to OpenLink Virtuoso Driver: 06.01.3127

```

Fonte: Autor.

Ao fim de uma execução bem-sucedida de um *Workflow* espera-se como resultado um conjunto de dados RDF. Este conjunto de dados é armazenado no servidor de triplas *Virtuoso*, que provê um *endpoint* SPARQL, permitindo assim consultas sobre o conteúdo das triplas armazenadas utilizando a linguagem SPARQL. No exemplo da figura 41, o resultado de uma consulta simples com o objetivo de retornar o nome de cada “*Journal*” contido do conjunto de dados do Qualis Brasil, utilizado no experimento.

Figura 41: Resultado de uma consulta sobre o conjunto de dados RDF através de endpoint SPARQL

journal	name
http://lod.unicentro.br/QualisBrasil/Journal_1414-915X	(Syn)Thesis (Rio de Janeiro)
http://lod.unicentro.br/QualisBrasil/Journal_1981-030X	19&20 (Rio de Janeiro)
http://lod.unicentro.br/QualisBrasil/Journal_1678-6416	7 Faces (Itabora)
http://lod.unicentro.br/QualisBrasil/Journal_1516-1528	@rquivos da Fundação Otorrinolaringologia
http://lod.unicentro.br/QualisBrasil/Journal_1677-7530	@rquivos de Otorrinolaringologia (Cessou em 2005. Cont. 1809-4872 @rquivos Internacionais de Otorrinolaringologia)
http://lod.unicentro.br/QualisBrasil/Journal_1809-4872	@rquivos Internacionais de Otorrinolaringologia
http://lod.unicentro.br/QualisBrasil/Journal_1809-4856	@rquivos internacionais de otorrinolaringologia (Online)
http://lod.unicentro.br/QualisBrasil/Journal_0104-7922	A Água em Revista
http://lod.unicentro.br/QualisBrasil/Journal_1677-0579	A Construção em Goiás
http://lod.unicentro.br/QualisBrasil/Journal_0010-6631	A Construção São Paulo

Fonte: Autor.

4.4. Análise dos Requisitos Alcançados

Dentre os requisitos especificados na seção 3.2.1, todos os requisitos foram implementados, porém com algumas ressalvas a serem feitas no requisito denominado RF07. As dificuldades na execução do Workflow foram inúmeras, pois com a automatização dos scripts, toda a parte de criação de diretórios de trabalho do ambiente da aplicação, assim como parametrização dos scripts com os caminhos de diretórios, tornaram-se tarefas custosas e que demandaram muitos testes. O conhecimento envolvido acerca de diversas ferramentas de terceiros também foi um fator agravante nas dificuldades encontradas durante a implementação deste requisito.

Os demais requisitos demandaram um esforço para o domínio da biblioteca *Apache Jena*, mais especificamente da sua API de manipulação de Ontologias (OWL API), onde a criação de serviços de auxílio a inserção e manipulação da LDWPO levaram grande parte dos esforços de implementação destes requisitos.

5. CONCLUSÃO E TRABALHOS FUTUROS

Após a realização deste trabalho e avaliação dos resultados obtidos é possível concluir que os objetivos foram alcançados, ainda que com ressalvas, visto a não conclusão com êxito da totalidade das funcionalidades previstas na etapa de projeto, devido não mensuração real da complexidade envolvida em algumas das funcionalidades, o que demandou um tempo não planejado para execução das atividades de desenvolvimento. Trabalhos futuros foram vislumbrados durante a execução deste trabalho devido a delimitação do escopo definida, bem como de novas ideias que emergiram durante o processo. Nas subseções à seguir são realizadas as conclusões finais deste trabalho e sugestão de novos trabalhos a serem desenvolvidos.

5.1. Conclusão

O processo de publicação e manutenção de conjuntos de dados *linked data* envolve diversos fatores para que seja materializado de fato. Desde a proposta inicial de Berners-Lee (2006) até os dias de hoje muitas coisas evoluíram: linguagens com maior expressividade, *middlewares* para suportar operações inerentes ao processo, ferramentas de apoio diversas, formatos de serialização mais expressivos e eficientes. Isso leva a uma diversidade de alternativas e ideias de processo a serem utilizados para levar um conjunto de dado 1 estrela a um conjunto de dados *linked data* (5 estrelas)³⁶. Tradicionalmente realizado a partir de uma definição de fluxo de trabalho muito particular, inerente ao usuário quem está projetando, estes processos são realizados a partir de scripts e ferramentas proprietárias, não portáteis, dificultando a aplicabilidade para contextos mais abrangentes e a manutenção.

O LODFlow (Rautenberg *et al.* 2015) propõe uma série de componentes para orquestração de fluxos de trabalho para a publicação e manutenção de conjuntos de dados *linked data*. Através da definição de um modelo bem definido e ferramentas de apoio para suportar tecnologicamente o processo o LODFlow introduz um fluxo de trabalho *linked data* de fato, suportando todo o ciclo de vida de um conjunto de dados RDF.

³⁶ <http://5stardata.info/pt-BR/>

Neste contexto surgem diversas alternativas oriundas de escopos não abordados na versão 1 (atual) do LODFlow. A aplicação cunhada de *SYSLODFlow*, proposta neste trabalho, tem como objetivo principal facilitar a interação do usuário com o LODFlow, criando um ambiente customizado e dedicado para definição — através de uma aplicação *web* — de um fluxo de trabalho *linked data*. Além de permitir a instanciação de novos *workflows*, baseado no modelo originalmente proposto pelo caso de uso descrito no documento de proposta do LODFlow [18], a ideia é de suportar o desenvolvimento de novos modelos de fluxo de trabalho, além do atualmente suportado, e integração de novas ferramentas ou melhoria na usabilidade da interação com o ambiente atual.

5.2. Trabalhos Futuros

A partir de todo o trabalho de pesquisa e desenvolvimento realizado pode-se notar o quanto é possível explorar e evoluir com base na aplicação desenvolvida assim como na gama de alternativas oriundas a partir do entendimento deste novo modelo proposto pelo LODFlow.

Sintetizando todas as ideias emergidas e debatidas a partir do desenvolvimento deste trabalho pudemos sugerir os seguintes trabalhos a serem desenvolvidos futuramente:

- Interface para edição ou automatização da geração de arquivos de configuração de mapeamento para serem utilizados pela ferramenta de apoio Sparqlify;
- Interface para edição ou automatização da geração de arquivos de configuração utilizados pela ferramenta de apoio LIMES;
- Desenvolvimento de aplicação para suporte a arquivos de entrada em outros formatos além do CSV;
- Melhoria e expansão do *SYSLODFlow* no que tange aspectos não levados em conta a partir da definição do escopo inicial;
- Criação de novos modelos de *workflows* a serem integrados com o LODFlow, integrando novas ferramentas ao fluxo.

REFERÊNCIAS

- [1] RAUTENBERG, S. *et al.* “*Linked Data Workflow Project Ontology*”. *Ontology Development Process Technical Report, Document Version 0.1.* (2015). Disponível em: <https://github.com/AKSW/ldwpo/blob/master/misc/technicalReport/LDWPO_technicalReport.pdf>. Acesso em: 7 jul. 2015.
- [2] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. “*The Semantic Web*” by, *Scientific American*. (2001). Disponível em <<http://www.disciplineoforganizing.org/wp-content/uploads/2013/01/SemanticWeb.pdf>>. Acesso em: 7 jul. 2015.
- [3] BERNERS-LEE, T. *Linked Data - Design Issues*. (2006). Disponível em <<http://www.w3.org/DesignIssues/LinkedData.html>>. Acesso em: 7 jul. 2015.
- [4] BATISTA, M. G. R; LÓSCIO, B. F. *OpenSBBD: Usando Linked Data para Publicação de Dados Abertos sobre o SBBD*. (2013). Disponível em <<http://www.lbd.dcc.ufmg.br/colecoes/sbbd/2013/0010.pdf>>. Acesso em: 7 jul. 2015.
- [5] HEATH, T; BIZER, C. *Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool, 1st edition*. Disponível em <<https://www.uni-koblenz-landau.de/campus-koblenz/fb4/west/teaching/ws1213/seminar-web-science/linked-data.pdf>>. Acesso em: 7 jul. 2015.
- [6] ALLSOPP, John. *What are Microformats?*. Apress, 2007. p. 6-9. Disponível em: <<http://ir.nmu.org.ua/bitstream/handle/123456789/140133/35c10501466b022d9efe2b64a5ceca95.pdf?sequence=1>>. Acesso em 03 nov. 2015.
- [7] BERNERS-LEE, T. *et al.* *Semantic Web Development – Technical Proposal*. (2000). Disponível em: <<http://www.w3.org/2000/01/sw/DevelopmentProposal>>. Acesso em 03 nov. 2015.
- [8] BIZER, C. *Evolving the Web into a Global Data Space*. In: BNCOD. 2011. p. 1.
- [9] BIZER, C.; HEATH, T.; BERNERS-LEE, T. *Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts*, p. 205-227, 2009. Disponível em: <http://www.researchgate.net/profile/Christian_Bizer/publication/225070216_Linked_Data_-_The_Story_So_Far/links/02e7e51d283241260c000000.pdf>. Acesso em 03 nov. 2015.

- [10] BIZER, C. et al. *How to publish linked data on the web*. 2007. Disponível em: <<http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/>>. Acesso em 03 nov 2015.
- [11] JUNIOR, V. F., ANDERLE, D. F., GONÇALVES, A. L., GAUTHIER, F. O., & SELL, D. Aplicações Semânticas Baseadas em Microformatos. In: ONTOBRAS-MOST. 2012. p. 194-199. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.8828&rep=rep1&type=pdf#page=194>>. Acesso em 03 nov. 2015.
- [12] Microformats Wiki, “Microformats”. Disponível em: <http://microformats.org/wiki/Main_Page>. Acesso em 03 nov. 2015.
- [13] W3C. *Architecture of the World Wide Web, Volume One*. 2004. Disponível em: <<http://www.w3.org/TR/webarch/>>. Acesso em 04 nov. 2015.
- [14] BERNERS-LEE, T., FIELDING, R. & MASINTER, L. "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, Janeiro, 2005. Disponível em: <<http://www.rfc-editor.org/info/rfc3986>>. Acesso em 04 nov. 2015.
- [15] W3C. *The W3C Technical Architecture Group (TAG)*. Disponível em: <<http://www.w3.org/2001/tag/>>. Acesso em 04 nov. 2015.
- [16] BIANCO, R. Disponibilização de dados abertos utilizando *linked data*: uma avaliação teórico-prática. 2011. Monografia (Bacharelado em Sistemas de Informação), UFSC (Universidade Federal de Santa Catarina), Florianópolis, Brasil.
- [17] DE LIMA, J. C.; DE CARVALHO, C. L. *Resource description framework (rdf). Technical report*, Universidade Federal de Goiás, 2005. Disponível em: <http://www.portal.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_003-05.pdf>. Acesso em 07 nov. 2015.
- [18] RAUTENBERG, Sandro et al. LODFlow: a workflow management system for linked data processing. In: Proceedings of the 11th International Conference on Semantic Systems. ACM, 2015. p. 137-144.
- [19] LIMES. *Link Discovery framework for MEtric Spaces*. Disponível em: <<http://aksw.org/Projects/LIMES.html>>. Acesso em 07 nov. 2015.
- [20] *Sparqlification mapping language*. Disponível em: <<http://sparqlify.org/wiki/SML>>. Acesso em 07 nov. 2015.

- [21] “*Sparqlify*” In. Github. Disponível em: <<https://github.com/AKSW/Sparqlify>>. Acesso em 07 nov. 2015.
- [22] W3C, “*VirtuosoUniversalServer*”. Disponível em: <<http://www.w3.org/wiki/VirtuosoUniversalServer>>. Acesso em 07 nov. 2015.
- [23] OpenLink Software. *Virtuoso Universal Server*. Disponível em: <<http://virtuoso.openlinksw.com/>>. Acesso em 07 nov. 2015.
- [24] Debian: Package virtuoso-opensource (6.1.6+dfsg2-2). Disponível em: <<https://packages.debian.org/pt-br/sid/virtuoso-opensource>>. Acesso em 07 nov. 2015.
- [25] WfMC: *WfMC: Terminology and Glossary*. Online PDF, 1999. Disponível em: <http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf>. Acesso em: 03 nov. 2015.
- [26] GEORGAKOPOULOS, Diimitrios; HORNICK, Mark; SHETH, Amit. *An overview of workflow management: From process modeling to workflow automation infrastructure*. *Distributed and parallel Databases*, v. 3, n. 2, p. 119-153, 1995.
- [27] ERDMANN, Michael; WATERFELD, Walter. *Overview of the neon toolkit*. In: *Ontology Engineering in a Networked World*. Springer Berlin Heidelberg, 2012. p. 281-301.
- [28] DEBASTIANI, C. A. Definindo escopo em projetos de software. São Paulo: Novatec, 2015.
- [29] MENDES, Emilia; MOSLEY, Nile; COUNSELL, Steve. *Investigating Web size metrics for early Web cost estimation*. *Journal of Systems and Software*, v. 77, n. 2, p. 157-172, 2005.
- Disponível em: <https://www.researchgate.net/publication/223190350_Investigating_Web_size_metrics_for_early_Web_cost_estimation>. Acesso em: 01 jun. 2016.
- [30] FOWLER, Martin. Padrões de arquitetura de aplicações corporativas. Bookman, 2009.
- [31] BROWN, Kyle et al. *Enterprise Java Programming with IBM Websphere*. Addison-Wesley Professional, 2003.
- [32] PrimeFaces. *Ultimate UI Framework for Java EE*. Disponível em: <<http://primefaces.org/>>. Acesso em: 02 jun. 2016.

- [33] Google *Trends*. Disponível em: <<https://www.google.com/trends/>>. Acesso em: 02 jun. 2016.
- [34] Bootstrap (página em Português). Disponível em: <<http://getbootstrap.com.br/>>. Acesso em: 02 jun. 2016.
- [35] Coding Dojo, Blog. *The 9 Most In-Demand Programming Languages of 2016*. Disponível em: <<http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2016/>>. Acesso em: 02 jun. 2016.
- [36] Apache Jena. Disponível em: <<https://jena.apache.org/>>. Acesso em: 02 jun. 2016.
- [37] Apache Jena. *Ontology API*. Disponível em: <<https://jena.apache.org/documentation/ontology/>>. Acesso em: 02 jun. 2016.
- [38] E SILVA, Ricardo Pereira. **UML 2: modelagem orientada a objetos**. Visual Books, 2007.
- [39] Apache Jena. *Javadoc*. Disponível em: <<https://jena.apache.org/documentation/javadoc/jena/>>. Acesso em: 05 jun. 2016.
- [40] DE LIMA, Júnio César; DE CARVALHO, Cedric L. **Ontologias-owl (web ontology language)**. Technical report, Universidade Federal de Goiás, 2005. Disponível em: <http://www.inf.ufg.br/sites/default/files/uploads/relatorios-tecnicos/RT-INF_004-05.pdf>. Acesso em 05 jun. 2016.
- [41] WELTY, Chris; MCGUINNESS, Deborah L.; SMITH, Michael K. Owl web ontology language guide. W3C recommendation, W3C (2004) Disponível em: <<http://www.w3.org/TR/2004/REC-owl-guide-20040210>>. Acesso em 05 jun. 2016.

ANEXO 1 – Exemplo de arquivo de mapeamento utilizado pelo Sparqlify

```

PREFIX fn: <http://aksw.org/sparqlify/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX qualis: <http://lod.unicentro.br/QualisBrasil/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

CREATE VIEW Template DefaultView As Construct {
    ?Evaluation rdf:type      rdf:Class      ;
                qualis:hasJournal    ?Journal    ;
                qualis:hasKnowledgeField ?KnowledgeField ;
                qualis:hasYear      ?YearEvaluation ;
                qualis:hasQualis    ?Qualis      .

    ?Journal rdf:type      rdf:Class      ;
              dc:identifier ?issnJournal ;
              dc:title     ?nameJournal .

    ?KnowledgeField rdf:type      rdf:Class      ;
                    dc:identifier ?idKnowledgeField ;
                    dc:title     ?nameKnowledgeField .

    ?YearEvaluation rdf:type      rdf:Class      ;
                    rdf:value ?yearIndex .

    ?Qualis rdf:type      rdf:Class      ;
            rdf:value ?qualisIndex .
}
WITH
#"Evaluation",
#"Journal_2238-0272_KnowledgeField_31_YearEvaluation_2013_Qualis_C",
?Evaluation = uri(concat("http://lod.unicentro.br/QualisBrasil/",fn:urlEncode(?1)))

#"Journal",
#"Journal_2238-0272",
?Journal = uri(concat("http://lod.unicentro.br/QualisBrasil/",fn:urlEncode(?2)))

#"issnJournal",
#"2238-0272",
?issnJournal = plainLiteral(?3)

#"nameJournal",
#"#10.art",
?nameJournal = plainLiteral(?4)

#"KnowledgeField",
#"KnowledgeField_31",
?KnowledgeField = uri(concat("http://lod.unicentro.br/QualisBrasil/",fn:urlEncode(?5)))

#"idKnowledgeField",
#31,
?idKnowledgeField = plainLiteral(?6)

```

```
#"nameKnowledgeField",  
#"CIÊNCIAS SOCIAIS APLICADAS I",  
?nameKnowledgeField = plainLiteral(?7)  
  
#"yearEvaluation",  
#"YearEvaluation_2013",  
?YearEvaluation = uri(concat("http://lod.unicentro.br/QualisBrasil/",fn:urEncode(?8)))  
  
#"yearIndex",  
#2013,  
?yearIndex = plainLiteral(?9)  
  
#"Qualis",  
#"Qualis_C",  
?Qualis = uri(concat("http://lod.unicentro.br/QualisBrasil/",fn:urEncode(?10)))  
  
#"qualisIndex"  
#"C"  
?qualisIndex = plainLiteral(?11)
```

ANEXO 2 – Exemplo de arquivo de mapeamento utilizado para criar os links com o DBPedia, utilizado pelo LIMES

```

<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2010 rel. 3 sp1 (http://www.altova.com)-->
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/1999/02/22-rdf-syntax-ns#</NAMESPACE>
    <LABEL>rdf</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/ontology/</NAMESPACE>
    <LABEL>dbpedia</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://dbpedia.org/property/</NAMESPACE>
    <LABEL>dbpedia-p</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://purl.org/dc/elements/1.1/</NAMESPACE>
    <LABEL>dc</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://lod.unicentro.br/QualisBrasil/</NAMESPACE>
    <LABEL>qualis</LABEL>
  </PREFIX>
  <SOURCE>
    <ID>dbpedia</ID>
    <ENDPOINT>http://dbpedia.org/sparql</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?x rdf:type dbpedia:PeriodicalLiterature</RESTRICTION>
    <PROPERTY>dbpedia:issn AS nolang->lowercase</PROPERTY>
  </SOURCE>
  <TARGET>
    <ID>localqualis</ID>
    <ENDPOINT>http://localhost:8890/sparql/QualisBrasil/</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>1000</PAGESIZE>
    <RESTRICTION>?y rdf:type rdf:Class</RESTRICTION>
    <PROPERTY>dc:identifier AS nolang->lowercase</PROPERTY>
  </TARGET>
  <METRIC>levenshtein(x.dbpedia:issn,y.dc:identifier)</METRIC>
  <ACCEPTANCE>
    <THRESHOLD>0.9</THRESHOLD>
    <FILE>real/Graphs/QualisBrasil/nt/dbpedia_qualis_links.nt</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>
</LIMES>

```



```
</ACCEPTANCE>
<REVIEW>
  <THRESHOLD>0.5</THRESHOLD>
  <FILE>reports/dbpedia_qualis_reviewme.nt</FILE>
  <RELATION>owl:sameAs</RELATION>
</REVIEW>
<EXECUTION>Simple</EXECUTION>
<OUTPUT>NT</OUTPUT>
</LIMES>
```

APÊNDICE A – Artigo

SysLODFlow: Uma Ferramenta de Apoio a Automação da Publicação e Manutenção de Linked Data Utilizando o LODFlow

Jean Carlos de Moraes¹, Jhonatan Carlos de Moraes¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

{jean,jhonatan}@inf.ufsc.br

Abstract. *The maintaining linked data datasets is a time consuming and expensive activity, involving many resources. The maintenance cost can be reduced by automation of data publishing workflow, which provides methods to support the life cycle of RDF data sets in a systematic way. The Linked Data Workflow Management System (LDWMS), dubbed LODFlow, provides an environment for planning, execution, reuse and documentation linked data workflows. The objective of this work is to develop a software application that supports the user in the automation of the publication and maintenance of linked data, integrated concepts and structure already defined LODFlow.*

Resumo. *A manutenção de conjuntos de dados Linked Data é uma atividade demorada e cara, que envolve muitos recursos. O custo de manutenção pode ser reduzido pela automação do fluxo de trabalho de publicação de dados, a qual proporciona métodos para suportar o ciclo de vida dos conjuntos de dados RDF de uma forma sistemática. O Linked Data Workflow Management System (LDWMS), apelidado de LODFlow, fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho linked data. O objetivo deste trabalho é o desenvolvimento de uma aplicação de software que apoie o usuário na automação da publicação e manutenção de linked data, integrada aos conceitos e estrutura já definida pelo LODFlow.*

1. Introdução

A Web Semântica foi projetada para expandir a web que conhecemos atualmente, possibilitando que a imensa quantidade de dados disponíveis possa ser compreendida não só por pessoas, mas também por máquinas [1]. Segundo Berners-Lee (2001), ela não é uma web separada, mas uma extensão da atual, em que a informação é fornecida através de um significado bem definido, permitindo que computadores e pessoas trabalhem em cooperação. A disponibilização de recursos informacionais melhor estruturados e representados, de modo a construir uma rede de informações conectadas, é possível através de ferramentas tecnológicas tais como agentes de software, XML, RDF, ontologias, padrões ou formatos e metadados. A definição de como construir esta rede

informações ligadas surgiu com os princípios de *Linked Data*, introduzidos por Berners-Lee em 2006.

Linked Data é um conjunto de melhores práticas para publicação e conexão de dados estruturados na web, permitindo estabelecer links entre itens de diferentes fontes de dados para formar um único espaço de dados global (HEATH; BIZER, 2011). Este conjunto de melhores práticas são sumarizados em quatro princípios: i) Use URI's (Uniform Resource Identifier) para nomear as coisas; ii) Use URI's HTTP para que as pessoas possam procurar o desejado; iii) Quando alguém olha para um URI, forneça informações úteis, usando os padrões (RDF, SPARQL); iv) Incluir links para outros URI's, para que eles possam descobrir explorar mais as coisas. Estes novos princípios abriram uma vasta gama de oportunidades, permitindo a publicação de dados acerca dos mais diversos temas e o desenvolvimento novas aplicações para interagirem com estes dados, de modo a construir uma “Web de Dados”.

Junto com estes novos horizontes, propiciados pelos dados ligados, surgem as dificuldades da manutenção destes dados publicados. A manutenção de conjuntos de dados *Linked Data* é complicada e envolve uma série de atividades bastante onerosas. Em contrapartida, o custo destas atividades de manutenção pode ser reduzido através da automatização do fluxo de trabalho de publicação de dados. Isso proporciona métodos para suportar o ciclo de vida destes conjuntos de dados de uma forma sistemática.

A produção de dados ligados é custosa e pode ser reduzida por um sistema de gerenciamento de fluxo de trabalho, que descreve planos para apoiar sistematicamente o ciclo de vida de conjuntos de dados RDF. O *Linked Data Workflow Management System* (LDWMS) [2], apelidado de LODFlow, fornece um ambiente para planejamento, execução, reutilização e documentação de fluxos de trabalho *Linked Data*. A abordagem LODFlow baseia-se num modelo ontológico abrangente, o *Linked Data Workflow Project Ontology* (LDWPO), para descrever os fluxos de trabalho e um mecanismo de apoio sistemático para execução de fluxos de trabalho, relatando e manipulando exceções.

Este trabalho propõe, através do entendimento do modelo de fluxo de trabalho definido pelo LODFlow e reaproveitamento de seus componentes, desenvolver uma aplicação de software para apoio a automação e manutenção da publicação de conjuntos de dados linked data, com o objetivo fornecer um ambiente de interação mais intuitivo e integrado ao usuário.

2. Linked Data

Linked Data é um conceito introduzido por Tim Berners-Lee no ano de 2006 [4] que se refere a uma nova forma de publicação de dados na web. Este conceito surgiu dentro de um outro mais abrangente, definido como um modelo de extensão da web, também proposto por Berners-Lee, no início dos anos 2000, a Web Semântica [3].

A Web Semântica não é apenas sobre a colocação de dados na web, é também sobre ligá-los [4], permitindo assim a exploração de pessoas e máquinas a Web de Dados, através das ligações estabelecidas entre dados relacionados. Portanto, com o objetivo de transformar a web em um espaço global de informações [5], Berners-Lee em 2006 propõe uma nova abordagem que trata da publicação de dados na web de modo a materializar a Web Semântica, chamada de *Linked Data*.

2.1. Princípios

Introduzido por Berners-Lee no ano de 2006, em uma de suas notas técnicas de arquitetura da web [4], o termo *linked data* refere-se a um conjunto de melhores práticas para publicação e conexão de dados estruturados na web [6]. O pressuposto básico por trás de *linked data* é de que o valor e a utilidade dos dados aumentam proporcionalmente ao número de ligações que eles estabelecem com outros dados [10].

Estas melhores práticas são delineadas por um conjunto de princípios propostos por Berners-Lee (2006) em sua nota de proposição de *linked data* [4], princípios estes que colaboram para que os dados publicados se tornem um único espaço de dados global. Os princípios são:

1. Use URIs como nomes para as coisas;
2. Use HTTP URIs para que as pessoas possam procurar esses nomes;
3. Quando alguém procurar uma URI, fornecer informações úteis, usando os padrões (RDF, SPARQL);
4. Incluir links para outras URIs, de modo a permitir que se descubram mais coisas.

Estes princípios, conhecidos como “princípios de *linked data*” fornecem uma receita básica para publicação e conexão de dados utilizando a infraestrutura da web, sendo aderente a sua arquitetura e padrões [6].

Em um paralelo entre a arquitetura da web clássica com a arquitetura *linked data*, a segunda tem como base duas tecnologias mais do que consolidadas, que são a base da web clássica: *Universal Resource Identifiers* (URIs), utilizadas como mecanismo global de identificação de recursos; e *HyperText Transfer Protocol* (HTTP), utilizado como mecanismo global de acesso a recursos. Enquanto na web clássica utiliza-se *HyperText Markup Language* (HTML) como formato amplamente adotado para a publicação de conteúdo, no modelo *linked data* o *Resource Description Framework* (RDF) é o padrão utilizado para publicação na Web de Dados.

3. Linked Data Workflow Management System

A produção de dados ligados é custosa e pode ser reduzida por um sistema de gerenciamento de fluxo de trabalho, que descreve planos para apoiar sistematicamente o ciclo de vida de conjuntos de dados RDF. O processo de criação e manutenção de conjuntos de dados *Linked Data* é tradicionalmente feito através de fluxos de trabalho manuais ou através de *scripts* proprietários que não são portáteis e apresentam certa dificuldade de manutenção. A extração de dados de bases de conhecimento *Linked Data* torna-se uma tarefa custosa, por conta de diversos passos que devem ser realizados para se obter um conjunto de dados, como carga de dados brutos, extração e análise. Feito isso os dados são disponibilizados em *endpoints* e ligados a outros conjuntos de dados [2].

O *Linked Data Workflow Management System* (LDWMS), apelidado de LODFlow, pode ser entendido como uma extensão do *Workflow Management System* (WMS), oriundo da gestão de negócios. O WMS “define, cria e gerencia a execução e fluxos de trabalho através do uso de software, rodando em um ou mais *engines* de fluxos de trabalho, capaz de interpretar a definição do processo, interagir com os participantes e, se necessário, recorrer ao uso de ferramentas de TI e aplicações” [7]. Entendendo esta definição para o contexto *Linked Data*, o

LODFlow pode ser definido como um sistema que consiste no seguinte conjunto de components:

6. *Linked Data Workflow Knowledge Model*: define um vocabulário comum para modelagem, análise, execução e documentação de fluxos de trabalho Linked Data.
7. *Knowledge Base*: registra os dados de acordo com o *Linked Data Workflow Knowledge Model*.
8. *Linked Data Workflow Maintenance Component*: usado para registrar os planos Linked Data e Execuções de Linked Data em bases de conhecimento.
9. *Linked Data Workflow Execution Engine*: usado para recuperar um fluxo de trabalho planejado de uma base de conhecimento e a execução de um plano a fim de produzir ou manter os conjuntos de dados.
10. *Linked Data Workflow Report Component*: usado para geração de relatórios para fluxos de trabalho Linked Data.

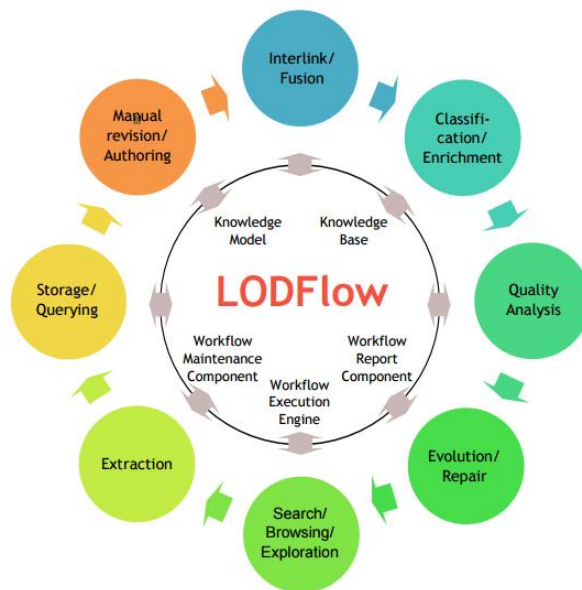


Figura 1. Ciclo de Vida *Linked Data* suportado pelo LODFlow (RAUTENBERG *et al.*, 2015).

4. Projeto e Desenvolvimento da Aplicação

Este capítulo apresenta os aspectos referentes a etapa de projeto e desenvolvimento da aplicação objeto deste trabalho, o SYSLODFlow.

4.1. Visão Geral

Toda orquestração de ferramentas necessárias para realizar o processo de publicação e manutenção de *Linked Data* exige o conhecimento e domínio de vários softwares que auxiliam as diferentes etapas necessárias para obtenção final de um *dataset*, fazendo com que a complexidade aumente consideravelmente. Dada a complexidades de se publicar dados no

padrão *Linked Data*, uma ferramenta que auxilie o processo de publicação e manutenção de dados ligados, através do planejamento e execução de fluxos de trabalhos, contendo a definição dos passos e ferramentas necessárias para tal, torna-se uma alternativa que sistematiza todas estas tarefas, viabilizando e facilitando o processo como um todo.

O LODFlow, introduz conhecimentos e componentes de software capazes de suportar todo o ciclo de vida de conjuntos de dados no padrão *linked data*, permitindo a produção de novos conjuntos de dados ou manutenção dos mesmos. Compreendendo toda a sistemática proposta pelo LODFlow, através do estudo dos processos envolvidos, observação do funcionamento, entendimento da relação entre os componentes e manipulação dos artefatos tecnológicos envolvidos foi possível identificar pontos de melhoria e evolução deste sistema. Baseado nestas observações a decisão foi por abordar o aspecto citado no item de número cinco.

A aplicação desenvolvida, cunhada “**SysLODFlow**” tem como finalidade o registro das instâncias das classes envolvidas no fluxo de trabalho de trabalho para publicação de conjuntos de dados *linked data*, orquestrado pelo LODFlow. O componente do LODFlow que trata da interação entre usuário e base de conhecimento é o *Linked Data Workflow Maintance Component*, e a ferramenta adotada e que atualmente permite a interação entre usuário e base de conhecimento (LDWPO) é a ferramenta *NeOn Toolkit*, um kit de ferramentas para a manipulação de ontologias. O objetivo com o **SysLODFlow** é criar um ambiente customizado para registros de Projetos, seus *Workflows*, Passos e Execuções, de modo a permitir uma melhor experiência ao usuário, facilitando o entendimento da sistemática envolvida, encapsulando funcionalidades intrínsecas as ferramentas de manipulação de ontologias. O diagrama de casos de uso da figura 3 representa em alto nível de abstração o conjunto de funcionalidades e fluxos de execução possíveis da aplicação.

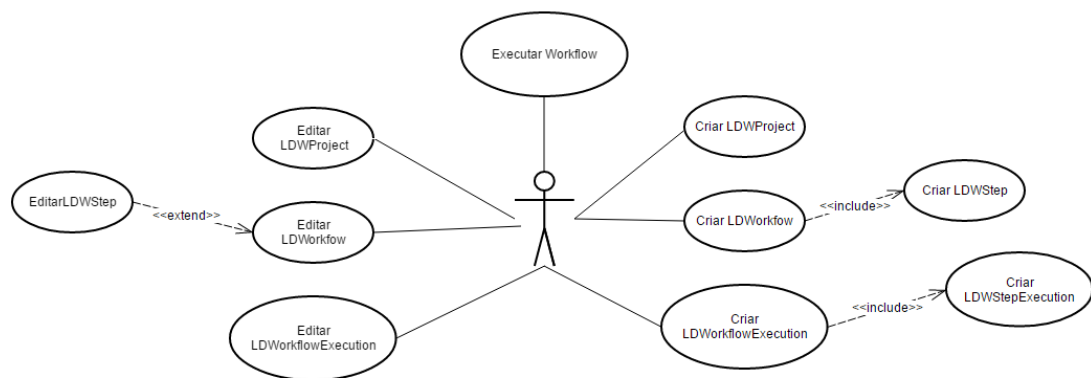


Figura 2: Diagrama de Casos de Uso do SysLODFlow.

4.2. Definição da Arquitetura

Definido o tipo de aplicação a ser desenvolvida, a arquitetura mais usual quando falamos de aplicações *web* é a arquitetura em três camadas. Podemos segmentar as camadas, segundo nomenclatura definida por Brown et al. (2003), em apresentação, domínio e fonte de dados.

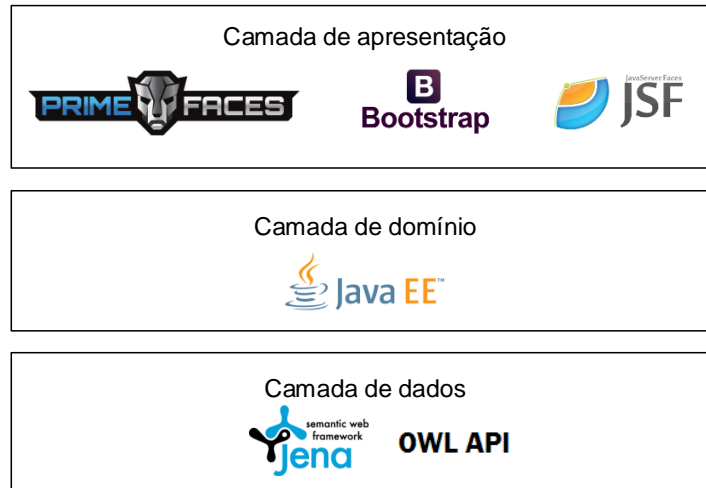


Figura 3: Esquema representando a arquitetura em três camadas com artefatos utilizados para o desenvolvimento.

Para camada de apresentação a opção deu-se pelo *framework Primefaces*³⁷. O *Primefaces* é um *framework* para desenvolvimento de projetos *web* baseados na tecnologia *JavaServer Faces*³⁸ (JSF), que permite a implementação ágil de aplicações, sejam elas simples ou sofisticadas. Para desenvolvimento do *core* da aplicação, ou seja, a camada de domínio, onde contém a lógica da aplicação, a tecnologia *Java Enterprise Edition*³⁹ (Java EE) foi adotada. A camada de dados, que contém as classes responsáveis pela manipulação dos dados persistidos é composta pelo *framework Apache Jena*. O *Jena* é um *framework* Java, livre e de código aberto para desenvolvimento de aplicações *Linked Data* e para *Web Semântica* [8]. Dentre os diversos recursos e API's fornecidos pelo *Jena*, a *OWL API*⁴⁰ fornece os recursos necessários para a manipulação de ontologias OWL, realizando a abstração do modelo ontológico para um conjunto de classes muito abrangente, que no caso deste trabalho, permite a interação com a base de conhecimento a ser trabalhada, a ontologia LDWPO.

4.3. Desenvolvimento

Conforme mencionado ao longo deste capítulo, a aplicação foi arquitetada em três camadas com o intuito de torná-la o mais modular possível, facilitando futuras manutenções e seguindo boas práticas de desenvolvimento adquiridas ao longo da vivência acadêmica. Utilizando-se de *frameworks* terceiros para viabilizar o desenvolvimento optou-se por utilizar o gerenciador de dependências *Maven* para gestão das dependências de módulos, componentes externos, bibliotecas, etc. O projeto Java foi dividido em uma estrutura de pacotes onde cada um deles contém uma ou mais classes com comportamento comum, ou seja, que proveem funcionalidades similares para a composição da aplicação. São eles:

1. *br.inf.ufsc.br.syslodflow.business*: Classes responsáveis por definir as regras de negócio de autenticação de usuários no sistema.

³⁷ <http://primefaces.org/>

³⁸ <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>

³⁹ <http://www.oracle.com/technetwork/java/javaee/overview/index.html>

⁴⁰ <https://jena.apache.org/documentation/ontology/>

2. *br.inf.ufsc.br.syslodflow.crypto*: Contém classe utilizada para criptografia das senhas de usuário.
3. *br.inf.ufsc.br.syslodflow.dao*: Classes responsáveis pelo acesso aos objetos armazenados em banco de dados, neste escopo aplica-se apenas aos usuários para autenticação no sistema.
4. *br.inf.ufsc.br.syslodflow.dto*: Abstrações de algumas entidades, com um número reduzido de atributos para fins de exibição em tela.
5. *br.inf.ufsc.br.syslodflow.entity*: Conjunto de classes que representam no modelo de orientação à objeto as classes contidas na ontologia LDWPO que serão utilizadas para manipulação desta. Ou seja, foi realizado um mapeamento de tais classes da ontologia para classes Java de modo a permitir a inserção e recuperação de dados na ontologia a partir desta aplicação.
6. *br.inf.ufsc.br.syslodflow.enumerator*: Conjunto de enumerados contendo constantes a serem utilizadas pelo programa, como as URI's das classes e propriedades da ontologia LDWPO, facilitando o desenvolvimento e melhorando a consistência do código.
7. *br.inf.ufsc.br.syslodflow.service*: Contém o conjunto de classes responsável pela interação direta com a ontologia LDWPO. As operações de inserção, atualização e recuperação de dados da ontologia são realizadas por estas classes, que se utilizam das classes fornecidas pela OWL API do *Jena* para realizar estas funcionalidades.
8. *br.inf.ufsc.br.syslodflow.util.util*: Classes utilitárias para realização de operações internas do software.
9. *br.inf.ufsc.br.syslodflow.web*: Contém as classes responsáveis por fazer a integração entre interface gráfica com usuário e regras de negócio da aplicação, através da utilização de *ManagedBeans*⁴¹

4.3.1. Entidades

Para criação de um modelo orientado à objeto (OO) das classes que compõem o modelo de conhecimento proposto pelo *Linked Data Workflow Knowledge Model* — armazenado na ontologia LDWPO — foi realizado um mapeamento destas classes e extraído seus atributos vinculados para posteriormente então criar uma abstração das mesmas em Java, procurando manter fidedignidade do modelo OO Java em relação ao representado pela ontologia. O processo de mapeamento foi realizado a partir de estudos realizados sobre o conteúdo do arquivo OWL que representa a ontologia, como também do trabalho de Rautenberg *et al.* (2015) que apresenta a *Linked Data Workflow Project Ontology* (LDWPO). A figura 4 exhibe as classes criadas, contidas no pacote *br.inf.ufsc.br.syslodflow.entity*.

⁴¹ <http://docs.oracle.com/javaee/7/api/javafx/faces/bean/ManagedBean.html>

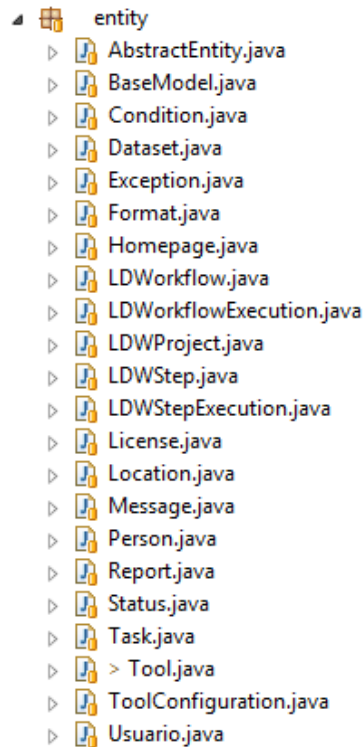


Figura 4: Entidades Java relacionadas as classes pertencentes ao modelo contido na LDWPO.

Tomando como exemplo, para demonstrar de que forma foi realizado o mapeamento das classes e atributos vinculados, na figura 5 é exibida a classe *LDWProject* que representa a classes de mesmo nome na LDWPO.

```

1 package br.ufsc.inf.syslodflow.entity;
2
3 public class LDWProject extends BaseModel {
4
5     private Person creator;
6     private Homepage homePage;
7     private LDWorkflow ldWorkFlow;
8     private Report report;
9     private String goal;
10    private String description;
11    private String name;
12    private String fileName;
13    private String uri;
14

```

Figura 5: Classe LDWProject.

Na definição da ontologia é possível observar, por exemplo, a relação entre a propriedade de dados (*data property*) *goal* e a classe *LDWProject*. Isso se dá através de uma propriedade de domínio, que vincula um sujeito que utiliza esta propriedade com este atributo de domínio associado como um tipo de coisa especificado por este domínio, conforme visto na figura 6.

```

<!-- http://ldwpo.aks.w.org/terms/1.0/goal -->

<owl:DatatypeProperty rdf:about="http://ldwpo.aks.w.org/terms/1.0/goal">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:label xml:lang="en">goal</rdfs:label>
  <rdfs:comment xml:lang="en">Data property that expresses the set of goals
of an LDWProject. As restriction, goal is a functional property and
accepts only xsd:String values.</rdfs:comment>
  <rdfs:domain rdf:resource="http://ldwpo.aks.w.org/terms/1.0/LDWProject"/>
  <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>

```

Figura 6: Propriedade de dados “goal” relacionada a classe LDWProject.

4.3.2. Trabalhando com Jena OWL API

O *framework Jena* fornece um conjunto de artefatos que permitem a construção de aplicações para *Web Semântica* e *Linked Data*. Para os fins de desenvolvimento desta aplicação a OWL API contida no *Jena* foi utilizada por conter toda a lógica envolvida necessária para manipulação de modelos ontológicos. A utilização desta API se faz necessária pelo fato de toda a persistência de dados envolvida no LODFlow estar centralizada na ontologia LDWPO.

4.3.2.1. Aspectos gerais

Esta ontologia fica armazenada no servidor, em um arquivo de extensão “.owl” e é carregada para a memória a cada operação requisitada através da OWL API. O conjunto de classes implementadas responsáveis por esta interação foram abrigadas no pacote `br.inf.ufsc.br.syslodflow.service`, conforme mostra a figura 7.

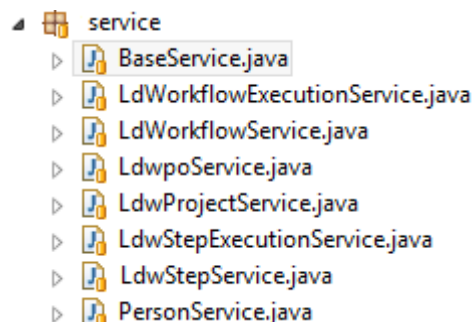


Figura 7: Classes responsáveis pela interação com a ontologia através da OWL API do Jena

Os elementos básicos de uma ontologia OWL são as classes, os indivíduos (também chamadas de instâncias) das classes e os relacionamentos (as propriedades) entre estes indivíduos [40]. Nas próximas subseções apresentaremos de que forma estes elementos foram trabalhados a partir dos recursos fornecidos pelo *Jena*.

4.3.2.2. Carga e Escrita de uma Ontologia

A classe *LdwpoService.java* é a responsável na aplicação pela manipulação dos dados trabalhados na ontologia e para isto é necessário carregar a ontologia para a memória. O *Jena* permite esta operação a partir da instanciação de um *OntModel*, que através do método *createOntologyModel* da classe *ModelFactory* cria um modelo que irá armazenar um arquivo

contendo a ontologia contido no diretório definido como parâmetro do método *read*, conforme ilustrado no trecho de código da figura 8. O modelo criado é do tipo *OWL_MEM*, que segundo a documentação da API [9] representa a linguagem *OWL Full*⁴² que não conta com motor de inferência.

```
public OntModel doLoadModel(Path path) {

    OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
    model.read(path.toUri().toString(), "");
    return model;
}
```

Figura 8: Criação de um modelo e leitura da ontologia para memória.

Todas as operações de inserção na ontologia são previamente salvas no modelo instanciado (*OntModel*) para posteriormente, ao voltar o controle da execução para a classe *LdwpoService.java* sejam persistidas na ontologia. O método *doSaveModel* (vide figura 9) foi implementado para ao receber o fluxo de execução, contendo o modelo utilizado no contexto atual e o nome do arquivo que será escrito em disco, seja realizada a persistência das informações. O método *write* da classe *OntModel* do *Jena* permite o salvamento dessas informações, onde é necessário definir o caminho onde o arquivo será salvo bem como a sintaxe a ser utilizada.

```
public void doSaveModel(OntModel model, String fileName) {
    FacesContext fc = FacesContext.getCurrentInstance();
    String filePath = fc.getExternalContext().getInitParameter("filePath").toString();

    try {
        File file = new File(filePath + fileName);
        model.write(new FileOutputStream(file), LangModelEnum.RDFXMLABBREV.getType());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 9: Persistência de um modelo para arquivo.

4.3.2.3. Classes

As classes de uma ontologia classes proveem um mecanismo de abstração para agrupar recursos com características similares, ou seja, uma classe define um grupo de indivíduos que compartilham algumas propriedades [10]. Num contexto de orientação a objetos a Classe tem o mesmo objetivo, descrever um conceito, um conjunto de características que definem algo, no caso de orientação à objetos um objeto, já no contexto de uma ontologia um indivíduo. A ontologia LDWPO define um conjunto de classes para descrever do domínio de fluxo de trabalho *linked data* e sendo assim é necessário a realização de um mapeamento destas classes para possibilitar a execução de operações sobre as mesmas.

⁴² <https://www.w3.org/TR/owl-ref/>

A OWL API do *Jena* fornece uma série de artefatos para trabalhar com classes de um modelo ontológico. Para este trabalho utilizamos principalmente a *Interface*⁴³ *OntClass*, que assim como outras interfaces da API pertence à uma hierarquia de interfaces, fornecendo uma cadeia de métodos extensa para manipulação de recursos. Como trabalhamos com uma ontologia já definida, a LDWPO, as operações realizadas sobre classes foram principalmente de recuperação das mesmas para utilização na instanciação de seus indivíduos e operações internas.

```
273 public OntModel insertLdwStep(OntModel model, LDWStep step) {
274
275     Individual ldwstep = model.getOntClass(ClassURIEnum.LDWSTEP.getUri())
276         .createIndividual(step.getUri());
```

Figura 10: Uso de classe da ontologia através do Jena.

Um trecho de código que exemplifica uma das operações realizadas sobre uma classe da ontologia pode ser observado na figura 10. Neste trecho é realizada uma busca no modelo pela classe *LDWStep*, através da passagem do atributo URI. De posse desta informação, através da execução do método *getOntClass* do modelo obtém-se uma instância do tipo *OntClass* representando a classe *LDWStep* definida na LDWPO.

4.3.2.4. Indivíduos

Os indivíduos são objetos básicos de uma ontologia, são a materialização de um conceito contido definido por uma classe, fazendo analogia a com uma linguagem orientada a objetos um indivíduo seria um objeto (instância) de uma classe. A API OWL do *Jena* possui diversos artefatos para trabalhar com este tipo de elemento. Para fins de desenvolvimento deste trabalho utilizamos a interface *Individual* que faz parte de uma hierarquia de interfaces para manipulação de recursos oriundos da ontologia. A partir desta *Interface* podemos instanciar novos indivíduos no modelo ontológico, adicionar e remover propriedades e seus valores e recuperar informações acerca dos mesmos. Na figura 26 é possível observar uma série de operações realizadas sobre um indivíduo da classe LDWProject criado através dos métodos fornecidos pelo *Jena* através da interface *Individual*.

⁴³ <https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>

```

150 private OntModel insertLdwProject(OntModel model, LDWProject project) {
151
152     Individual ldwProject = model.getOntClass(
153         ClassURIEnum.LDWPROJECT.getUri()).createIndividual(
154         project.getUri());
155     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.NAME.getUri()),
156         project.getName());
157     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.GOAL.getUri()),
158         project.getGoal());
159     ldwProject.addLiteral(
160         model.getProperty(PropertyURIEnum.DESCRPTION.getUri()),
161         project.getDescription());
162     ldwProject.addProperty(
163         model.getProperty(PropertyURIEnum.CREATOR.getUri()),
164         model.getIndividual(project.getCreator().getUri()));
165     model = writeHomepage(model, project.getHomePage());
166     ldwProject.addProperty(
167         model.getProperty(PropertyURIEnum.HOME PAGE.getUri()),
168         model.getIndividual(project.getHomePage().getUri()));
169     model = writeReport(model, project.getReport());
170     ldwProject.addProperty(
171         model.getProperty(PropertyURIEnum.REPORT.getUri()),
172         model.getIndividual(project.getReport().getUri()));
173     return model;
174 }
175

```

Figura 11: Criação de uma novo Indivíduo da classe LDWProject.

Na linha 152, 153 e 154 da figura 11 observa-se a instrução para criação de um novo indivíduo na ontologia. A mesma lógica é utilizada em todos os pontos do programa quando da necessidade de se instanciar novos elementos. Através do método *getOntClass* do modelo a ser manipulado naquele contexto é possível obter a classe desejada, recuperada através da URI da classe informada no parâmetro, para então esta classe recuperada executar o seu método *createIndividual*, que cria um novo indivíduo desta classe e adiciona-o ao modelo. Por fim este novo indivíduo criado é armazenado na variável local *ldwProject* para operações subsequentes a serem realizadas.

4.3.2.5. Propriedades

As propriedades, que são relações binárias, podem ser usadas para estabelecer relacionamentos entre indivíduos ou entre indivíduos e valores de dados [10]. Para utilização de propriedades definidas na LDWPO foi utilizada a *Interface Property*, que contém um conjunto de métodos abrangente — principalmente pela herança das *SuperInterfaces* contidas em sua hierarquia — para manipulação de propriedades, sejam elas *datatype properties* ou *object properties*.

Na figura 12 apresentamos um trecho de código do método *getPropertyStringValue*, implementado com a finalidade de ser um método genérico para recuperação de valores literais de *datatype properties* de um indivíduo qualquer. Uma aplicação deste método pode ser vista na figura 13, para obter o valor da propriedade *description* de um indivíduo da classe *LDWorkflow*.

```

46 public static String getPropertyStringValue(Individual individual,
47     OntModel model, String uriProperty) {
48     if (individual != null) {
49         Property property = model.getProperty(uriProperty);
50         String value = individual.getPropertyValue(property).asLiteral()
51             .getString();
52         return value;
53     }
54     return "";
55 }
56

```

Figura 12: Método genérico para recuperação de valor de datatype property de um indivíduo.

```

44 String ldwWorkflowDescription = getPropertyStringValue(ontLdWorkflow,
45     model, PropertyURIEnum.DESRIPTION.getUri());

```

Figura 13: Exemplo de utilização do método `getPropertyStringValue` para recuperação de valor de datatype property de um indivíduo.

Para recuperação de um valor de *object property* de um indivíduo, ou seja, um outro indivíduo, utilizamos uma instrução como a contida na figura 14, que na prática recupera um indivíduo da classe *Person* associado a um indivíduo da classe *LDWProject* através da propriedade de objeto *creator*.

```

36 Individual creator = model.getIndividual(ontProject
37     .getPropertyResourceValue(
38         model.getProperty(PropertyURIEnum.CREATOR.getUri()))
39     .getURI());

```

Figura 14: Exemplo de recuperação de uma object property

Operações de inserção de propriedades são fundamentais para a definição correta e completa de indivíduos. Estas operações estão, assim como as demais relacionadas a ontologia, estão implementadas nas classes que interagem diretamente o *Jena*. Os métodos *addLiteral* e *addProperty* são frequentemente utilizados para inserção respectivamente de valores literais para *datatype properties* e indivíduos para *object properties*. A figura 15 ilustra este comportamento através do método implementado *insertLdwProject* cria um indivíduo da classe *LDWProject* e adiciona suas propriedades com seus respectivos valores.

```

153 private OntModel insertLdwProject(OntModel model, LDWProject project) {
154
155     Individual ldwProject = model.getOntClass(
156         ClassURIEnum.LDWPROJECT.getUri()).createIndividual(
157         project.getUri());
158     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.NAME.getUri()),
159         project.getName());
160     ldwProject.addLiteral(model.getProperty(PropertyURIEnum.GOAL.getUri()),
161         project.getGoal());
162     ldwProject.addLiteral(
163         model.getProperty(PropertyURIEnum.DESCRPTION.getUri()),
164         project.getDescription());
165     ldwProject.addProperty(
166         model.getProperty(PropertyURIEnum.CREATOR.getUri()),
167         model.getIndividual(project.getCreator().getUri()));
168     model = writeHomepage(model, project.getHomePage());
169     ldwProject.addProperty(
170         model.getProperty(PropertyURIEnum.HOMEPAGE.getUri()),
171         model.getIndividual(project.getHomePage().getUri()));
172     model = writeReport(model, project.getReport());
173     ldwProject.addProperty(
174         model.getProperty(PropertyURIEnum.REPORT.getUri()),
175         model.getIndividual(project.getReport().getUri()));
176     return model;
177 }
178

```

Figura 15: Exemplo de inserção de propriedades a indivíduos

5. Aplicação do SysLODFlow

Esta seção tem por objetivo expor os resultados obtidos através dos esforços de desenvolvimento do aplicativo de software **SysLODFlow**. O conjunto de dados utilizado para validação da aplicação foi o do Qualis Brasil⁴⁴, tratado em Rautenberg *et al.* (2015).

5.1. Visão Geral do *Workflow* Resultante

O *workflow* suportado pela aplicação é baseado no modelo definido pela *LDWPO*. A figura 16 mostra uma visão geral do *workflow* após inclusão do SysLODFlow como uma ferramenta de apoio.

⁴⁴ <https://qualis.capes.gov.br/>

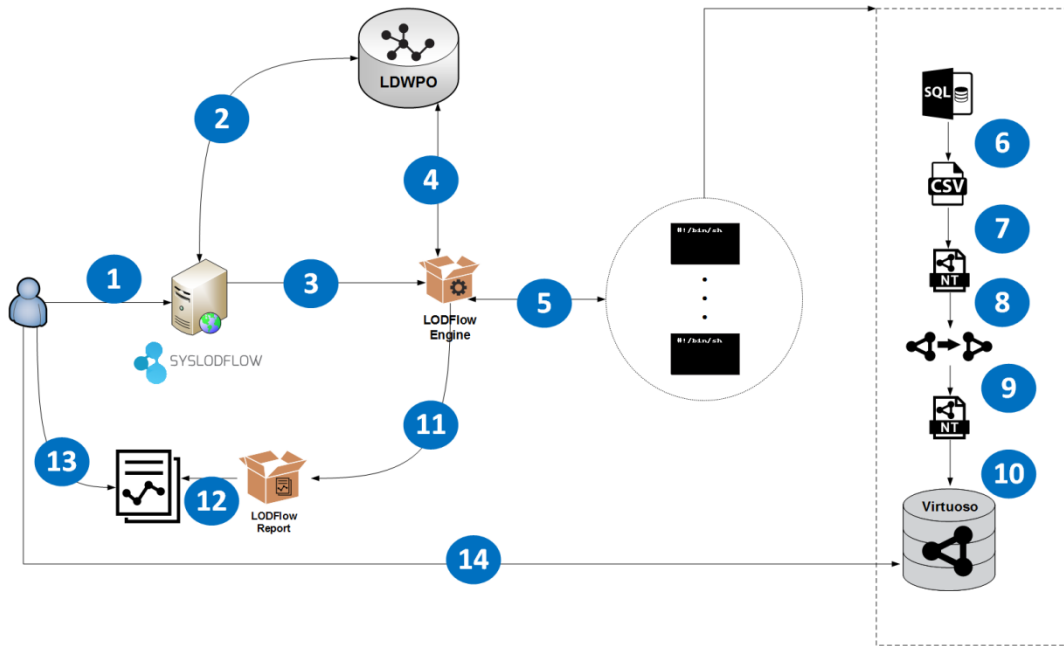


Figura 16: Visão geral do fluxo de trabalho *linked data*, contendo o SYSLODFlow como componente

1. Usuário acessa SYSLODFLOW através de seu navegador, realiza as operações devidas e mandar executar o fluxo.
2. SYSLODFLOW armazena dados na ontologia.
3. SYSLODFLOW chama a execução do LODFlow *Engine* para execução do fluxo informando o fluxo a ser executado.
4. LODFlow *Engine* recupera da ontologia os dados relativos ao *workflow* a ser executado.
5. LODFlow *Engine* realiza chamadas via script para execução de ferramentas de apoio, configuradas nos passos.
6. Dados são extraídos de um banco de dados relacional (MySQL) e convertidos para CSV.
7. Sparqlify realiza o mapeamento dos dados contidos no CSV e converte para triplas em formato NT.
8. Arquivos de triplas é armazenado no banco de dados Virtuoso.
9. Triplas são recuperadas do virtuoso e através do LIMES ligadas ao DBPedia.
10. Triplas já ligadas ao DBPedia são armazenadas novamente no triple store Virtuoso.
11. LODFlow *Engine* aciona LODFlow *Report* para geração de relatório de execução do *workflow*.
12. LODFlow *Report* gera relatório em HTML contendo informações acerca da execução.

13. Usuário pode realizar o acesso ao relatório em HTML para verificar status da execução.
14. Usuário pode realizar consultas sobre o banco de dados de triplas Virtuoso através de seu endpoint SPARQL.

5.2. Interface Gráfica do Sistema

Ao selecionar no menu principal da aplicação o item *LDWProject*, o usuário visualizará uma listagem de projetos cadastrados, carregados através do seu arquivo de ontologia específico. Através da tabela de listagem, é possível efetuar o *download* da ontologia referente ao projeto selecionado, assim como abrir a edição/instanciação de um *LDWorkflow*. No botão “Editar” o usuário será direcionado para um formulário de cadastro com informações gerais do projeto como: nome, descrição, objetivo, criador, homepage (vide figura 18). O botão “Excluir” realiza a exclusão física do arquivo de ontologia referente ao projeto selecionado.



Figura 17: Página Inicial da Aplicação.

LDWProject

LDWProject					
<div> Novo </div> <div> Mostrar 10 registros por página <div>Pesquisar <input type="text"/></div> </div>					
Nome	Criador	OWL	LDWorkflow	Ações	
QualisBrasil	Ivan Ermilov	Download	Abrir	Editar	Excluir
Exibir de 1 a 1 de 1 registro(s)				<div> Anterior 1 Proximo </div>	

Figura 18: Listagem de LDWProjects

A opção de instanciação/edição de um *LDWorkflow*, se dá através do botão “Abrir” na coluna destinada da listagem de projetos. Nesta tela temos subdivisões em abas, que facilita o entendimento do sequenciamento dos passos, assim como o uma melhor visualização dos mesmos. Esta divisão se dá em seis abas: Planejamento do *Workflow*, *Step 01*, *Step 02*, *Step 03*,

Step 04 e *Step 05*. Uma delas contém informações gerais pertinentes ao *workflow*, e as demais para cada um dos *steps*.

LDWorkflow



The screenshot shows the 'LDWorkflow' application interface. At the top, there is a blue header bar with the text 'Planejamento do Workflow'. Below this, there is a navigation bar with four tabs: 'Planejamento do Workflow' (selected), 'LDWStep 01', 'LDWStep 02', and 'LDWStep 03'. The main content area is divided into three sections: 'Nome', 'Descrição', and 'Pré-condições'. Each section has a text input field.

Planejamento do Workflow

Planejamento do Workflow LDWStep 01 LDWStep 02 LDWStep 03

Nome

Maintain QualisBrasil

Descrição

Workflow applied to create linked dataset of Qualis Periodicals Index (all years), in an automatized way.

Pré-condições

1. Availability of the data (sql dump or what is there)
2. Running system with installed Ubuntu and all

Figura 19: Aba de informações gerais do Workflow.

As abas referentes aos *steps* possuem a parametrização das informações necessárias para a execução do *workflow* da maneira proposta pelo modelo descrito da *LWPO*, citado no início desta seção. O *step 02* por exemplo, é responsável por mapear um arquivo *CSV* para *N-Triples*.

The screenshot shows a web interface titled 'Planejamento do Workflow'. At the top, there are three tabs: 'Planejamento do Workflow', 'LDWStep 01', and 'LDWStep 02'. The 'LDWStep 02' tab is active. Below the tabs, the form is divided into several sections:

- Nome:** A text input field containing 'PlanningStep - Extracting QualisBrasil applying SPARQLIFY to'.
- Descrição:** A text input field containing 'What should be done to Extracting QualisBrasil applying SPARQLIFY described here'.
- Ferramenta:** A dropdown menu with 'Sparqlify' selected.
- Arquivo de mapeamento de dados:** A section with a button 'Escolher arquivo' and the text 'Nenhum arquivo selecionado'. Below this, a note states: 'Utilize arquivos no formato .sml. Tamanho máximo: 1MB.'

Figura 20: Aba de informações do Step 02.

Através do menu “Execuções” é acessada a tela para visualizações de execuções do *Workflow*. Ao selecionar o projeto, são carregadas informações do *Workflow*, assim como a listagem de execuções do mesmo. O botão “editar” leva o usuário à tela de edição de um *LDWorkflowExecution*. Já o botão novo direciona para a tela de instanciação de um *LDWorkflowExecution*. Os arquivos de mapeamento de dados, que necessitam de upload através da interface do sistema (*steps 02 e 04*), são requisitos das ferramentas utilizadas no *backend* da aplicação. O Sparqlify necessita de um arquivo no formato SML para fazer a relação entre as propriedades do *dataset* CSV para o formato *N-Triples*. Já o LIMES precisa de um mapeamento das propriedades necessárias para efetuar a ligação do *dataset* local com o DBPedia, este no formato XML. Os exemplos de arquivos utilizados no caso de uso do Qualis Brasil estão expostos nos anexos deste trabalho.

Execuções

LDWProject
QualisBrasil

LDWorkflow: Informações Gerais

Nome: Maintain QualisBrasil
URI: http://ldwpo.aksw.org/terms/1.0/ldWorkflow_maintaining_qualisBrasil
Descrição: Workflow applied to create linked dataset of Qualis Periodicals Index (all years), in an automatized way.

LDWorkflowExecutions Armazenados

[Novo](#)

Nome	Relatório	Ações
Maintaining QualisBrasil2011		Editar Excluir
Maintaining QualisBrasil2007		Editar Excluir
Maintaining QualisBrasil2014		Editar Excluir

Figura 21: Listagem de Execuções de Workflow.

Ao selecionar a edição um *LDWorkflowExecution* já existente, o usuário é direcionado para a tela de edição do mesmo, onde é possível efetuar o cadastro/edição das informações gerais, assim como as execuções de *steps*. Cada execução de *step* descrita nesta etapa é baseada na configuração definida no momento do planejamento do *Workflow*. O botão “salvar” persiste os dados na ontologia, já o botão “executar” dispara o *LODFlowEngine*, enviando para execução o *LDWorkflowExecution* definido na tela.

Execuções

[« Voltar](#)

[Geral](#) [Step Execution 1](#) [Step Execution 2](#) [Step Execution 3](#) [Step Execution 4](#) [Step Execution 5](#)

Nome
Execution step 05 - QualisBrasil 2014

Descrição
3,590,448 triples were sucessfully saved in triple store.

[Salvar](#) [Executar](#)

Figura 22: Cadastro de Execuções de Workflow.

5.3. Artefatos Gerados pelo *Workflow*

Após a execução do *LDWorkflowExecution* desejado com sucesso, dois artefatos finais são gerados: Um relatório contendo os resultados da execução do *Workflow* e um conjunto de dados RDF. O relatório de resultados da execução do *workflow* é disponibilizado para download através da interface gráfica, como mostra a figura 21. É um relatório no formato HTML que torna possível observar o detalhamento dos passos executados, como o tempo de execução, status, quantitativo de triplas geradas, entre outros. Para cada um dos passos executados são fornecidas informações que servem de subsídio para determinar o sucesso ou não da operação. As saídas oriundas de ferramentas de apoio são registradas como *output* dos passos correspondentes de modo a facilitar a depuração da execução. Este relatório é provido pelo próprio *LODFlowEngine* e um exemplo pode ser observado na figura 24.

```

WORKFLOW - ldWorkflowExecution_maintaining_qualisBrasil_2014
=====
First step: http://ldwpo.aksw.org/terms/1.0/ldwStepExecution_02_qualisBrasil2014
Executed workflow: ldWorkflowExecution_maintaining_qualisBrasil_2014
Executed step: ldwStepExecution_02_qualisBrasil2014
Planning step : ldwStep_02_qualisBrasil_converting_data_with_sparqlify Command: Qualis/commands/applyingSparqlify.sh
output of the command: an error occurs: 2016-07-03 17:07:53.892 INFO org.aksw.sparqlify.csv.CsvMapperCliMain: Errors: 0, Warnings: 0 2016-07-03 17:07:53.965 DEBUG
org.aksw.sparqlify.csv.CsvMapperCliMain: Detected column names: [Evaluation, Journal, issnJournal, nameJournal, KnowledgeField, idKnowledgeField, nameKnowledgeField, YearEvaluation, yearIndex
Qualis, qualisIndex] Variable #Unbound Triples generated: 13973360 Potential triples omitted: 0 Triples total: 13973360
difference time: 92690.0

started date: Sun Jul 03 17:07:52 BRT 2016

ended date: Sun Jul 03 17:09:24 BRT 2016

Executed workflow: ldWorkflowExecution_maintaining_qualisBrasil_2014
Executed step: ldwStepExecution_03_qualisBrasil2014
Planning step : ldwStep_03_qualisBrasil_storing_converted_data Command: Qualis/commands/savingIntoVirtuoso.sh Qualis/temp/QualisBrasil.nt http://lod.unicentro.br/QualisBrasil/
output of the command: Connected to OpenLink Virtuoso ODBC Driver OpenLink Interactive SQL (Virtuoso), version 0.9849b. Type HELP: for help and EXIT: to
exit. SQL> Done. -- 12 msec. SQL> Size = 2249632992 File is large. Performing split on file Qualis/temp/QualisBrasil.nt creating load statement Connected to OpenLink Virtuoso Driver: 06.01.3127

```

Figura 23: Relatório de Execução de *Workflow*.

Ao fim de uma execução bem-sucedida de um *Workflow* espera-se como resultado um conjunto de dados RDF. Este conjunto de dados é armazenado no servidor de triplas *Virtuoso*, que provê um *endpoint* SPARQL, permitindo assim consultas sobre o conteúdo das triplas armazenadas utilizando a linguagem SPARQL. No exemplo da figura 24, o resultado de uma consulta simples com o objetivo de retornar o nome de cada “*Journal*” contido do conjunto de dados do Qualis Brasil, utilizado no experimento.

journal	name
http://lod.unicentro.br/QualisBrasil/Journal_1414-915X	(Syn)Thesis (Rio de Janeiro)
http://lod.unicentro.br/QualisBrasil/Journal_1981-030X	19&20 (Rio de Janeiro)
http://lod.unicentro.br/QualisBrasil/Journal_1678-6416	7 Faces (Itabira)
http://lod.unicentro.br/QualisBrasil/Journal_1516-1528	@rquivos da Fundação Otorrinolaringologia
http://lod.unicentro.br/QualisBrasil/Journal_1677-7530	@rquivos de Otorrinolaringologia (Cessou em 2005. Cont. 1809-4872 @rquivos Internacionais de Otorrinolaringologia)
http://lod.unicentro.br/QualisBrasil/Journal_1809-4872	@rquivos Internacionais de Otorrinolaringologia
http://lod.unicentro.br/QualisBrasil/Journal_1809-4856	@rquivos internacionais de otorrinolaringologia (Online)
http://lod.unicentro.br/QualisBrasil/Journal_0104-7922	A Água em Revista
http://lod.unicentro.br/QualisBrasil/Journal_1677-0579	A Construção em Goiás
http://lod.unicentro.br/QualisBrasil/Journal_0010-6631	A Construção São Paulo

Figura 24: Resultado de uma consulta sobre o conjunto de dados RDF através de endpoint SPARQL

6. Conclusão

O processo de publicação e manutenção de conjuntos de dados *linked data* envolve diversos fatores para que seja materializado de fato. Desde a proposta inicial de Berners-Lee (2006) até os dias de hoje muitas coisas evoluíram: linguagens com maior expressividade, *middlewares* para suportar operações inerentes ao processo, ferramentas de apoio diversas, formatos de serialização mais expressivos e eficientes. Isso leva a uma diversidade de alternativas e ideias de processo a serem utilizados para levar um conjunto de dado 1 estrela a um conjunto de dados *linked data* (5 estrelas)⁴⁵. Tradicionalmente realizado a partir de uma definição de fluxo de trabalho muito particular, inerente ao usuário quem está projetando, estes processos são realizados a partir de scripts e ferramentas proprietárias, não portáteis, dificultando a aplicabilidade para contextos mais abrangentes e a manutenção.

O LODFlow (Rautenberg *et al.* 2015) propõe uma série de componentes para orquestração de fluxos de trabalho para a publicação e manutenção de conjuntos de dados *linked data*. Através da definição de um modelo bem definido e ferramentas de apoio para suportar tecnologicamente o processo o LODFlow introduz um fluxo de trabalho *linked data* de fato, suportando todo o ciclo de vida de um conjunto de dados RDF.

Neste contexto surgem diversas alternativas oriundas de escopos não abordados na versão 1 (atual) do LODFlow. A aplicação cunhada de *SYSLODFlow*, proposta neste trabalho, tem como objetivo principal facilitar a interação do usuário com o LODFlow, criando um ambiente customizado e dedicado para definição — através de uma aplicação *web* — de um fluxo de trabalho *linked data*. Além de permitir a instanciação de novos *workflows*, baseado no modelo originalmente proposto pelo caso de uso descrito no documento de proposta do LODFlow [2], a ideia é de suportar o desenvolvimento de novos modelos de fluxo de trabalho, além do atualmente suportado, e integração de novas ferramentas ou melhoria na usabilidade da interação com o ambiente atual.

7. Trabalhos Futuros

A partir de todo o trabalho de pesquisa e desenvolvimento realizado pode-se notar o quanto é possível explorar e evoluir com base na aplicação desenvolvida assim como na gama de alternativas oriundas a partir do entendimento deste novo modelo proposto pelo LODFlow. Sintetizando todas as ideias emergidas e debatidas a partir do desenvolvimento deste trabalho pudemos sugerir os seguintes trabalhos a serem desenvolvidos futuramente:

- Interface para edição ou automatização da geração de arquivos de configuração de mapeamento para serem utilizados pela ferramenta de apoio Sparqlify;
- Interface para edição ou automatização da geração de arquivos de configuração utilizados pela ferramenta de apoio LIMES;
- Desenvolvimento de aplicação para suporte a arquivos de entrada em outros formatos além do CSV;
- Melhoria e expansão do *SYSLODFlow* no que tange aspectos não levados em conta a partir da definição do escopo inicial;
- Criação de novos modelos de *workflows* a serem integrados com o LODFlow, integrando novas ferramentas ao fluxo.

⁴⁵ <http://5stardata.info/pt-BR/>

Referências

- [1] Batista, Mateus Gondim Romão, and Bernadette Farias Lóscio. "OpenSBBD: Usando linked data para publicação de dados abertos sobre o SBBD." *Brazilian Symposium on Databases-SBBD*. 2013.
- [2] Rautenberg, Sandro, et al. "LODFlow: a workflow management system for linked data processing." *Proceedings of the 11th International Conference on Semantic Systems*. ACM, 2015.
- [3] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The semantic web." *Scientific american* 284.5 (2001): 28-37.
- [4] Berners-Lee, Tim. "Linked data-design issues (2006)." URL <http://www.w3.org/DesignIssues/LinkedData.html> (2011).
- [5] Bizer, Christian. "Evolving the Web into a Global Data Space." *BNCOD*. Vol. 7051. 2011.
- [6] Bizer, Christian. "Evolving the Web into a Global Data Space." *BNCOD*. Vol. 7051. 2011.
- [7] WfMC, Glossary. "Terminology and Glossary." *Document No WfMC-TC-1011. Workflow Management Coalition*. Winchester (1999).
- [8] Apache Jena. Disponível em: < <https://jena.apache.org/> >. Acesso em: 02 jun. 2016.
- [9] Apache Jena. *Ontology API*. Disponível em:
< <https://jena.apache.org/documentation/ontology/> >. Acesso em: 02 jun. 2016.
- [10] de Lima, Júnio César, and Cedric L. de Carvalho. *Ontologias-owl (web ontology language)*. Technical report, Universidade Federal de Goiás, 2005.
- [11] Rautenberg, Sandro, et al. "Linked Data Workflow Project Ontology". *Ontology Development Process Technical Report, Document Version 0.1*. (2015).