OntoMongo - Um Método de Acesso a Dados da Plataforma Lattes Baseado em Ontologias e Sistemas NoSQL

Thiago Henrique Dias Araujo

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
Obtenção do Título
de
Mestre em Ciências

Programa: Mestrado em Ciência da Computação
Orientadora: Prof[a]. Dr[a]. Renata Wassermann

São Paulo, agosto de 2017

OntoMongo - Um Método de Acesso a Dados da Plataforma Lattes Baseado em Ontologias e Sistemas NoSQL

Esta é a versão original da dissertação elaborada pelo candidato Thiago Henrique Dias Araujo, tal como submetida à Comissão Julgadora.

Agradecimentos

Agradeço à minha amada esposa Stéfanni Brasil pela confiança, incentivo, carinho, e paciência. Sem ela nada disso teria sido possível.

Minha eterna gratidão à professora Renata Wassermann pela sabedoria, paciência e compreensão, e principalmente pela oportunidade de poder aprender muito com ela.

Agradeço à minha querida amiga Bárbara Tieko pela parceria no projeto e por toda a ajuda durante o Mestrado, e ao Márcio Stábile também pelo incentivo, assim como ao Guilherme Rey por nossas conversas (que também chamamos de sessões de terapia de grupo) que foram muito importantes para nós todos nessa jornada.

À professora Kelly Braghetto pelas valiosas contribuições, ao professor Jesús Mena Chalco pelo apoio durante o desenvolvimento do projeto e pelas ótimas sugestões de melhoria. Também ao professor Alfredo Goldman por me mostrar uma nova forma de enxergar o desenvolvimento de software, e aos demais professores e colegas do IME-USP.

Agradeço à minha família por todo o incentivo e por sempre acreditar em mim.

Finalmente, agradeço ao Data, meu melhor amigo (felino). Mesmo tarde da noite enquanto eu fazia exercícios ou debugava código, ele sempre esteve comigo me fazendo companhia e me mantendo acordado.

Os limites da minha linguagem significam os limites do meu mundo.

Ludwig Wittgenstein

Resumo

ARAUJO, T. H. D. **OntoMongo - Um Método de Acesso a Dados da Plataforma Lattes Baseado em Ontologias e Sistemas NoSQL**. 2017. 60 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Acesso a Dados Baseado em Ontologias é uma forma de acesso a dados que utiliza uma ontologia como camada conceitual, uma base de dados para persistência, e um método de tradução de consultas utilizando um mapeamento que conecta os dados armazenados no banco de dados com o vocabulário da ontologia. A vantagem dessa abordagem é sua capacidade de descrever e representar domínios de maneira coesa, lidar com inconsistências e acessar diferentes tipos de informação de maneira padronizada. Os trabalhos nessa área se concentraram em bases de dados relacionais, mas com o aumento do volume de dados disponível e com o surgimento dos grupos de bancos de dados NoSQL, foi necessário adaptar a técnica para dar suporte aos bancos de dados NoSQL. Entretanto, essas propostas possuem algumas limitações na forma de construção do mapeamento e na flexibilidade da solução. Propomos um novo método de acesso OBDA utilizando uma camada conceitual intermediária extensível e capaz de prover acesso a diferentes tipos de Sistemas de Gerenciamento de Banco de Dados NoSQL, e validamos nossa proposta através da criação de uma aplicação em um domínio real.

Palavras-chave: acesso a dados baseado em ontologias, NoSQL, ontologia, Plataforma Lattes.

Abstract

ARAUJO, T. H. D. **OntoMongo - Ontology-Based Data Access for Plataforma Lattes Using NoSQL**. 2017. 60 f. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2017.

Ontology-based data access (OBDA) is a data access method that connects database entities to the vocabulary of an ontology, using it as a conceptual layer and exploring its ability to describe domains and deal with data incompleteness. This is done through mappings that connect the database entities to the vocabulary of the ontology. OBDA studies were about data stored in relational databases, but the exponential growth of the volume and the velocity of data nowadays and the development of NoSQL databases has brought the need to extend the use of OBDA to NoSQL databases. In this paper, we present a novel approach for OBDA with document-oriented NoSQL databases. Our approach uses an access interface with an intermediate conceptual layer that is extensible and flexible, capable of providing access to different types of database management systems. To validate the approach, we have implemented a prototype for MongoDB, using a real-world application domain as a case study to access data and generate reports for Plataforma Lattes, a scientific collaboration network.

Keywords: ontology-based data access, NoSQL, ontology, Plataforma Lattes.

Sumário

Lista de Abreviaturas

ACID	(*Atomicity, Consistency, Isolation, Durability*)
DSL	(*Domain-Specific Language*)
ILP	(*Inductive Logic Programming*)
OBDA	(*Ontology-Based Data Access*)
OWL	(*Web Ontology Language*)
SGBD	(*Sistema de Gerenciamento de Banco de Dados*)
SPARQL	(*SPARQL Protocol and RDF Query Language*)
SQL	(*Structured Query Language*)
SRL	(*Statistical Relational Learning*)
SVM	(*Support Vector Machine*)
SWRL	(*Semantic Web Rule Language*)

Lista de Figuras

Capítulo 1

Introdução

Nos últimos anos o volume, a variedade e a velocidade na geração de dados provenientes do uso de diversas tecnologias vem crescendo muito e empresas de tecnologia têm cada vez mais aplicado técnicas de mineração de dados com interesse em analisar essas informações para aumentar sua produtividade, entender a necessidade dos seus usuários, aumentar a receita, investigar a eficácia de campanhas de marketing, dentre outros. Isto confirma a necessidade de se criar soluções robustas e flexíveis para gerenciar, analisar e útilizar esses dados a fim de se extrair informações úteis e valiosas.

É crescente a demanda por profissionais dotados de conhecimento do domínio e conhecimento técnico que saibam transformar esses dados em informações relevantes. Podemos citar inúmeras aplicações que extraèm conhecimento desses grandes volumes de dados e o utilizam para tomada de decisão, ou em investigações científicas, como nas análises de comportamento em redes sociais e como se dão as relações entre pessoas.

Como exemplo, podemos citar o trabalho de Ginsberg *et al.* (2009) a respeito de um método de detecção e monitoramento de epidemias do vírus *influenza* através da análise de buscas de milhões de usuários feitas em sites como o Google ao procurarem informações sobre os sintomas da doença. Através da análise da freqüência relativa de alguns tipos de buscas em uma certa região, algo que demonstraram ter grande correlação com o número de visitas ao hospital feita por pacientes portadores dos sintomas da doença, foram capazes de estimar o nível de atividade do vírus em cada região dos Estados Unidos com bastante rapidez.

Um relatório da empresa Mckinsey (Manyika *et al.*, 2011) mostrou que o governo americano poderia potencialmente economizar $300 bilhões de dólares em 10 anos se aplicasse técnicas de análise de dados para reduzir gastos com assistência médica. Na europa, os governos poderiam economizar $100 bilhões de euros aumentando sua eficiência operacional ao utilizar Big Data para reduzir fraudes, erros operacionais e evasão fiscal.

Tem se tornado cada vez mais fácil armazenar e consultar essas informações com o surgimento de novas tecnologias de armazenamento distribuído. Mas, dado o volume e variedade desses dados, que são armazenados em diferentes tipos de formatos, bancos de dados, sistemas de arquivos e locais físicos, isto acaba demandando uma especialização das pessoas que trabalham nessa área e que precisam aprender diversas interfáces de acesso, linguagens de consulta, formatos de arquivo e tipos de bancos de dados para conseguirem fazer suas tarefas (figura 1.1). Portanto, tornou-se cada vez mais importante prover uma interface de acesso simples, intuitiva e de alto nível.

Uma das alternativas para tratar do acesso a esses dados e fazer a análise de forma eficaz é o

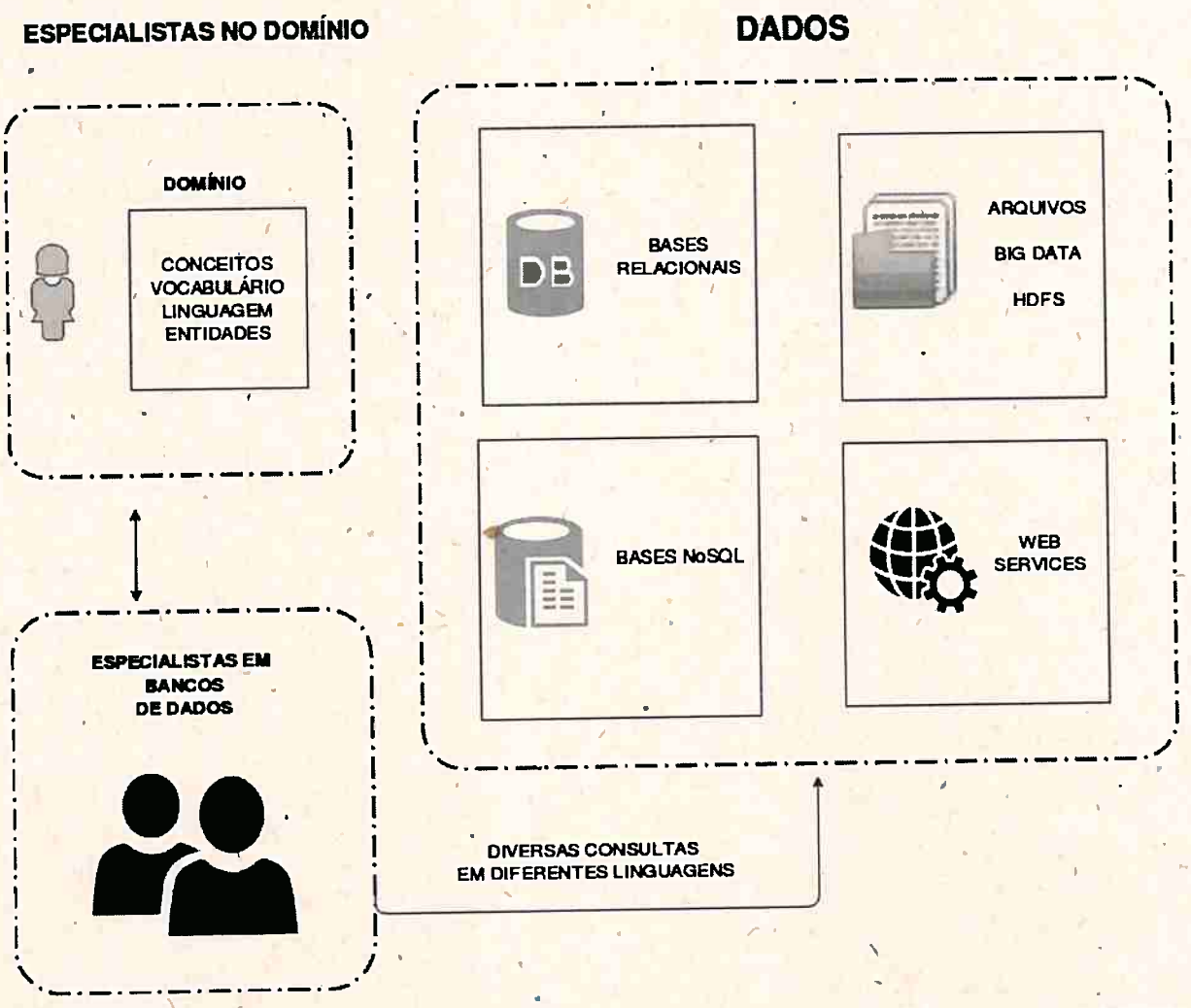


Figura 1.1: *O volume e variedade de dados aumenta, novos sistemas surgem para lidar com o gerenciamento desses dados, cada um com sua linguagem de consultas específica. Com isso a complexidade no acesso também aumenta.*

acesso através de ontologias, conhecida como OBDA (Ontology-Based Data Access), é uma solução de acesso a dados desenvolvida por Calvanese *et al.* (2007) que utiliza um vocabulário consistente e padronizado acerca do domínio, e é um meio mais simples de acesso do que se o especialista do domínio tivesse que fazer consultas manualmente ao banco de dados e analisar a estrutura de tabelas e outras questões relacionadas ao armazenamento dos dados, mas que não são tão úteis para responder às questões de seu interesse.

O OBDA utiliza uma ontologia que descreve determinado domínio, um banco de dados para armazenamento e um mapeamento que conecta o vocabulário da ontologia com os conjuntos de dados persistidos no banco de dados. Também emprega um método de resposta a consultas que utiliza o mapeamento para consultar a base e transformar os dados de maneira eficiente.

Com esse método, as interfaces de acesso aos dados são mais transparentes para o usuário final e também são dotadas de meios mais inteligentes de expansão do conhecimento gerado, pois podem criar novas conexões através do uso das ontologias e dos motores de inferência lógica, permitindo inferir novos fatos e construir novo conhecimento formal. Trabalhar com uma camada conceitual independente da estrutura se apresenta como uma forma mais eficaz de abordar esse problema.

Entretanto, os trabalhos nessa área se concentraram no acesso a bases de dados relacionais e na tradução de consultas SPARQL para SQL. Sistemas tradicionais de análise e gerenciamento de dados se baseiam nesse tipo de SGBD. Entretanto, os bancos de dados relacionais são úteis quando lidamos com dados estruturados e, segundo Chen *et al.* (2014), esse tipo de SGBD só pode ser escalado de forma vertical com a adição de hardware mais caro.

Além, disso, esses tipos de SGBDs também não conseguem dar conta de grandes quantidades heterogêneas de dados, sendo limitados em sua performance e no modo de armazenamento no contexto dos grandes volumes de dados e *Big Data*. Nesse cenário, a quantidade de dados aumenta exponencialmente, a variedade de formatos de dados é grande, e é necessário armazenar e analisar dados semiestruturados ou até mesmo sem estrutura alguma.

A comunidade científica propôs, ainda segundo Chen *et al.* (2014), algumas soluções que atendam aos requisitos de alta performance, baixo custo e elasticidade, além da criação de soluções de gerenciamento e armazenamento distribuído e em larga escala.

Surgiram os grupos de bancos de dados NoSQL como resposta à essa demanda por melhores formas de se trabalhar com grandes volumes de dados, soluções que ofereçam melhor performance, disponibilidade, flexibilidade, e tolerância a falhas (Nayak *et al.*, 2013). Por isso, tornou-se necessário adaptar as técnicas de OBDA para esse novo contexto que é bastante diferente dos SGBDs relacionais, pois não existe um padrão na forma de acesso e consulta a esses tipos de SGBDs.

Neste contexto, o presente trabalho propõe um novo método mais flexível como interface de acesso a bases de dados NoSQL de grande volume usando OBDA.

Buscamos simplificar a tradução de consultas para diferentes tipos de bancos de dados. Para tal, adicionamos uma camada conceitual intermediária (figura 1.2) que representa os dados persistidos no banco de dados e um método extensível de tradução de consultas para o banco de dados de destino.

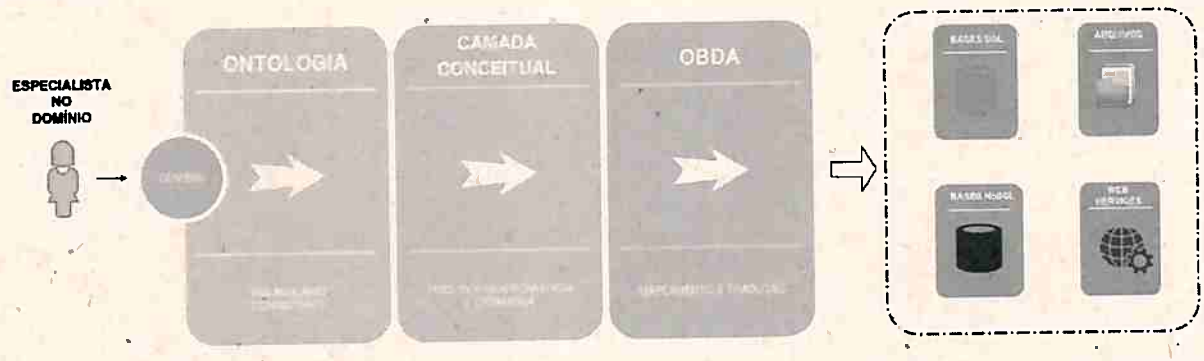


Figura 1.2: *A solução proposta possui uma camada conceitual expansível que facilita o acesso aos dados de forma mais transparente para o usuário especialista no domínio.*

Essa solução foi desenvolvida em conjunto com o projeto de Mestrado da aluna Bárbara Tieko Agena (Agena, 2017) e também gerou uma publicação na Ontobrás 2017 (Agena *et al.*, 2017).

Para verificar a viabilidade dessa proposta, ela foi aplicada ao domínio das redes de colaboração científica e tem como fonte de dados os currículos da Plataforma Lattes, que estão armazenados em um SGBD NoSQL orientado a documentos.

Os principais objetivos deste trabalho são descritos a seguir:

- Criar um método de acesso OBDA a uma base NoSQL com informações a respeito de uma rede

de colaboração científica, contendo informações sobre pesquisadores, publicações, colaborações e instituições.

- Comparar esse método com outros presentes na literatura e verificar sua eficácia e viabilidade
- Gerar alguns relatórios utilizando esse método para demonstrar seu funcionamento

Para desenvolver uma solução que tenha esses objetivos em mente, o trabalho foi dividido nos capítulos descritos a seguir.

No Capítulo 2, apresentamos alguns conceitos teóricos como OBDA, SPARQL, NoSQL, entre outros, que serão explanados de forma sucinta. Isto servirá de apoio para as discussões levantadas nos capítulos posteriores.

Em seguida, no Capítulo 3, exploramos contribuições valiosas de trabalhos relacionados, bem como questões que ainda não foram exploradas por outros pesquisadores. Assim, propomos uma justificativa que mostre a importância de nossa pesquisa.

No Capítulo 4, explicamos o experimento feito ao longo da pesquisa que servirá para implementar e validar as ideias e o desenvolvimento do método proposto, mostrando os testes feitos, fazendo uso de algumas comparações com outros projetos presentes na literatura para oferecer suporte teórico à nossa abordagem e aplicações práticas do método para mostrar sua aplicabilidade ao problema estudado.

Finalmente, no Capítulo 5, delimitamos o escopo do projeto e discutimos algumas conclusões, bem como a proposta de algumas contribuições para trabalhos futuros.

O código do projeto pode ser encontrado em https://github.com/thdaraujo/onto-mongo. Esse repositório contem o método de acesso a bases NoSQL, o mapeamento, e também o módulo de relatórios da Plataforma Lattes.

Capítulo 2

Conceitos

Alguns conceitos teóricos são importantes para o entendimento do presente projeto. Aqui, discutimos os fundamentos mais importantes.

2.1 Ontologia

Diferentemente da noção filosófica de Ontologia como o estudo da natureza do Ser, para a Ciência da Computação uma ontologia é uma estrutura relacional, um modelo de especificação formal explícita de conceitos de um determinado domínio. Esses conceitos podem dizer respeito a entidades de um domínio e suas propriedades, relações, restrições e comportamentos.

Segundo Guarino *et al.* (2009), uma ontologia é um artefato computacional que possui um modelo formal de representação da estrutura de um domínio, contendo as entidades relevantes, bem como as relações que emergem da observação desse mesmo domínio, porém utilizando apenas as que são úteis para algum determinado fim. A modelagem de classes e relações entre entidades é feita através de definições em uma linguagem formal, como por exemplo, em uma lógica de descrição (Baader *et al.*, 2003), de forma compacta e de alto nível de abstração.

Uma ontologia representa um domínio. O escopo e o nível de detalhe de uma ontologia podem ser definidos através das perguntas que elas devem auxiliar a responder, as chamadas *Questões de Competência* (Grüninger e Fox, 1995). No fundo, são informações sobre algum domínio que desejamos que ela nos responda. A ontologia deve também conter a terminologia a respeito dos objetos, conceitos e relações, provendo uma linguagem que será usada para expressar esses conceitos.

Um exemplo de domínio possível é a comunidade científica, com seus diversos pesquisadores e suas relações com os demais. Uma ontologia que representa esse domínio deve ser capaz de capturar as entidades relevantes, organizando as informações em conceitos e relações, para que seja capaz de responder a algumas perguntas, como por exemplo: *Quem é coautor de um artigo A?* Nesse caso, ela pode descrever classes de entidades como *Pesquisador* e *Artigo* e relações binárias entre dois indivíduos, como a relação *colaborou_com* e *publicou_artigo*.

O mais importante é que essa definição formal possa ser entendida por um computador em um formato padronizado, para que a resposta seja encontrada de forma automática. Para isso, uma ontologia pode utilizar a inferência lógica para responder a esse tipo de pergunta, como também descobrir novas relações entre entidades, através do uso de motores de inferência (*reasoners*), que são programas capazes de inferir consequências lógicas a partir de uma base de fatos.

Por tudo isso, podemos dizer que uma ontologia é uma ferramenta bastante poderosa, capaz de

descrever um domínio de forma compacta, permitir consultas complexas e derivar novos conhecimentos de forma automática.

2.2 Ferramentas de Acesso a Dados Baseado em Ontologias

Existem algumas ferramentas interessantes que podem ser utilizadas para se trabalhar com ontologias. Uma delas é o Protégé [1], que é um editor de ontologias de código aberto escrito em Java. O projeto possui uma comunidade bastante ativa, contando com centenas de milhares de usuários, e é utilizado na criação de sistemas inteligentes e na engenharia de ontologias, sendo capaz de armazenar dados em diferentes formatos, como XML, RDF, e até em um banco de dados.

A *Ontology Web Language - OWL* é uma das linguagens utilizadas na Web Semântica e foi definida pela W3C [2], assim como RDF, RDFS e SPARQL, entre outras. Ela é utilizada para representar conhecimento rico e complexo sobre domínios, entidades, grupos de coisas e relações entre entidades de maneira formal. Sua definição é equivalente a um grafo RDF e é feita utilizando-se definições em XML.

A OWL-API é uma biblioteca para Java muito utilizada na construção e manipulação de ontologias OWL, descrita por Horridge e Bechhofer (2011). Ela foi desenvolvida para facilitar a manipulação de ontologias OWL 1.1 para uso em editores e outras ferramentas. Ela se baseia nas especificações da OWL 1.1 e propõe uma experiência de design e usabilidade da aplicações baseadas em OWL.

2.3 Acesso a Dados Baseado em Ontologias (OBDA)

Bases de dados guardam informações diversas sobre um determinado assunto, e as pessoas geralmente precisam de conhecimento técnico e conhecimento sobre o domínio para conseguirem consultar essas bases de dados e fazerem suas análises. Calvanese *et al.* (2007) propuseram uma solução para simplificar o acesso e a consulta a esses dados e facilitar o uso de diferentes origens de dados. Tal proposta é chamada de Acesso a Dados Baseado em Ontologias (*Ontology-Based Data Access*), ou OBDA.

OBDA é uma forma de acesso a dados estruturados através do uso de ontologias. A arquitetura desse tipo de solução funciona através do uso de uma ontologia que descreve um domínio específico, um ou mais bancos de dados do tipo relacional para o armazenamento de dados e um mapeamento que conecta o vocabulário da ontologia com os conjuntos de dados presentes nessas bases de dados.

Ela também provê um método de resposta a consultas que utiliza o mapeamento para acessar e transformar os dados de forma eficiente. As consultas SPARQL oriundas da ontologia são traduzidas para consultas SQL, e o resultado é então transformado em instâncias de objetos na ontologia. Assim, é possível derivar novos fatos que constituam um novo conhecimento formal através da ontologia.

A aplicação de uma camada conceitual independente da estrutura da base de dados é mais eficiente, segundo Botoeva *et al.* (2016a), quando os usuários do sistema são especialistas no domínio mas não possuem nenhum conhecimento acerca da estrutura das bases de dados e nem como a informação é armazenada e organizada.

[1] http://protege.stanford.edu
[2] https://www.w3.org/OWL/

Esse cenário é comum em situações em que o volume e a variedade de dados armazenada é muito grande, o que impossibilita que cada especialista conheça profundamente a estrutura e a forma de armazenamento desses dados.

Nesses cenários em que o volume de dados é imenso, a variedade de formatos é grande e a velocidade de crescimento desse volume também é grande, há a necessidade de se analisar dados semiestruturados de forma rápida e eficiente, e bancos de dados relacionais não são a melhor forma de lidar com acesso e análise de dados nesses cenários, segundo Nayak *et al.* (2013). Por isso, surgiram as alternativas NoSQL, que discutimos na Seção 2.4, e aplicações do NoSQL também para OBDA, dando suporte ao acesso a bases de dados NoSQL através do uso de ontologias.

2.4 NoSQL

O crescimento da internet e dos grandes repositórios de dados e sistemas que são capazes de gerar, capturar e armazenar volumes maciços de dados, dando origem ao chamado *Big Data*, fez surgir uma rica fonte de conhecimento extraída da análise desses dados. Neste tipo de cenário, o volume de dados aumenta de forma expressiva, há uma imensa variedade de formatos de dados e tipos de armazenamento, e uma grande necessidade de se armazenar e analisar esses dados de forma eficiente. Essa análise precisa ser feita com dados semiestruturados ou até mesmo não-estruturados (como textos, imagens e sons). Isso indica que os bancos de dados tradicionais com seu modelo relacional não são a melhor solução de armazenamento e consulta nesses cenários, segundo Nayak *et al.* (2013).

Como resposta a isso, surgiram os sistemas gerenciadores de bancos de dados NoSQL (*Not only SQL*) com o intuito de prover melhor performance, disponibilidade, flexibilidade e escalabilidade.

Existem cinco grupos principais de SGBDs NoSQL: orientados a documentos, orientados a colunas (colunares), orientados a grafos, orientados a objetos, e armazenamento chave-valor.

Os sistemas gerenciadores de bancos de bancos de dados NoSQL orientados a documentos não possuem estrutura ou esquema (*schemaless*), e eles armazenam os dados em documentos tais como XML e JSON e oferecem uma performance muito boa e alta escalabilidade, dando suporte a consultas, processamento e armazenamento distribuídos (Nayak *et al.*, 2013).

O XML (*Extensible Markup Language*) é um formato padronizado de texto que é simples e extensível e é usado para troca de informações estruturadas entre serviços web ou como linguagem de marcação para texto, dados e outros tipos de informação e é bastante utilizado para compartilhamento de informações na internet. A especificação técnica desse padrão é definido pela W3C. É um formato mais antigo que tem dado lugar ao JSON.

O JSON (*javascript object-notation*), ou Notação de Objetos Javascript é também um formato padronizado de texto utilizado para troca de informações e armazenamento de dados para a web. É muito utilizado como forma de armazenamento por sistemas NoSQL orientados a documentos, ou para troca de mensagens entre serviços web, sites, e aplicações web diversas.

Exemplos de documentos[3] XML e JSON podem ser vistos a seguir:

```
1 <!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
2  <glossary><title>example glossary</title>
3    <GlossDiv><title>S</title>
```

[3]Exemplos retirados da documentação do *ECMA-404 The JSON Data Interchange Standard* disponíveis em http://json.org/

```
4    <GlossList>
5     <GlossEntry ID="SGML" SortAs="SGML">
6      <GlossTerm>Standard Generalized Markup Language</GlossTerm>
7      <Acronym>SGML</Acronym>
8      <Abbrev>ISO 8879:1986</Abbrev>
9      <GlossDef>
10      <para>A meta-markup language, used to create markup languages such as
            DocBook.</para>
11      <GlossSeeAlso OtherTerm="GML">
12      <GlossSeeAlso OtherTerm="XML">
13     </GlossDef>
14     <GlossSee OtherTerm="markup">
15    </GlossEntry>
16   </GlossList>
17  </GlossDiv>
18 </glossary>
```

Código 2.1: *Exemplo de documento XML*

O mesmo documento convertido para JSON pode ser visto a seguir:

```
1 {
2     "glossary": {
3         "title": "example glossary",
4         "GlossDiv": {
5             "title": "S",
6             "GlossList": {
7                 "GlossEntry": {
8                     "ID": "SGML",
9                     "SortAs": "SGML",
10                    "GlossTerm": "Standard Generalized Markup Language",
11                    "Acronym": "SGML",
12                    "Abbrev": "ISO 8879:1986",
13                    "GlossDef": {
14                        "para": "A meta-markup language, used to create markup
                              languages such as DocBook.",
15                        "GlossSeeAlso": ["GML", "XML"]
16                    },
17                    "GlossSee": "markup"
18                }
19            }
20        }
21    }
22 }
```

Código 2.2: *Exemplo de documento JSON equivalente*

Os SGBDs orientados a documentos com suporte a documentos XML armazenam documentos desse formato e permitem sua consulta e atualização. São úteis para armazenar dados nesse formato com as seguintes características: dados hierárquicos, esparsos, aninhados, páginas web em XHTML, metadadosm, e dados provenientes da Web Semântica, que comumente são representados através de RDF ou XML. Os mais famosos bancos de dados com suporte a XML são o *IBM DB2*[4], e o

[4]IBM DB2 https://www.ibm.com/analytics/us/en/db2/

Microsoft SQL Server[5].

O MongoDB[6] é um SGBD orientado a documentos muito utilizado no mercado e dá suporte ao armazenamento de documentos JSON. Outro banco de dados desse grupo é o *Amazon DynamoDB*[7], que dá suporte tanto a documentos quanto a armazenamento chave-valor.

SGBDs NoSQL orientados a colunas são similares aos bancos de dados relacionais e armazenam as informações em tabelas. A diferença está no modo de organização desses dados: enquanto os bancos de dados armazenam linhas de forma contígua, os bancos colunares armazenam colunas. Portanto, são otimizados para leitura e escrita de colunas de dados, que é útil para consultas analíticas, oferecendo melhor performance em operações de sumarização e agrupamento por colunas.

Os bancos de dados NoSQL com suporte a armazenamento colunar mais conhecidos são o *Amazon Redshift*[8], o *Apache Kudu*[9], e o *MariaDB*[10] com a extensão *ColumnStore*.

Bancos de dados orientados a grafos representam a informação como vértices e arestas em um grafo, e a ênfase está nas relações entre entidads. São ótimos para representar como uma rede as associações entre conceitos, indivíduos e objetos, algo útil para se fazer análise de redes de conceitos, redes sociais, sistemas de recomendação e análise de fluxo de informações e relações entre entidades. As consultas são expressas como buscas ou travessias em grafos e usam algoritmos de busca muito eficientes, o que melhora bastante sua performance. Entretanto, o suporte a *sharding* e clusterização é limitado. Um dos exemplos mais populares desse grupo de bancos de dados é o *Neo4j*[11].

Bancos de dados orientados a objetos armazenam os dados em representações similares a objetos em uma linguagem orientada a objetos. Também oferecem propriedades similares como encapsulamento, herança e polimorfismo. Esses objetos podem ser armazenados e consultados diretamente, sem a necessidade de serem transformados em linhas de tabelas e relações com outras tabelas. O banco de dados orientado a objetos mais conhecido é o *db4o*.

Bancos de dados de armazenamento chave-valor são similares a *hashtables* ou dicionários e são bons para rápido acesso e leitura de dados. Permitem o armazenamento de dados não-estruturados a partir de uma chave única de identificação que funciona como índice. O dado armazenado possui duas partes: uma chave que representa um identificador, e qualquer tipo de dado ou informação que se deseje salvar, formando um par chave-valor. Esse tipo de sistema é altamente escalável, com ótimo suporte a concorrência e particionamento, mas sacrificando consistência. Também não dá suporte a alguns tipos de operações como agregação e *joins*. É muito utilizado para guardar informações voláteis como sessões de usuários, linha do tempo (*timelines*) em redes sociais, e outros dados que precisam de bastante performance na consulta.

Os bancos de dados NoSQL com suporte a armazenamento chave-valor mais conhecidos são o *Amazon DynamoDB*, o *Apache Kudu*[12], e o *MariaDB*[13] com a extensão *ColumnStore*.

Segundo DeCandia *et al.* (2007), o DynamoDB possui vantagens como particionamento automático e escalabilidade horizontal, alta disponibilidade para escritas e latência média de menos de 10 milissegundos, tolerância a falhas e recuperação automática, replicação e sincronização de répli-

[5]Microsoft SQL Server https://www.microsoft.com/pt-br/sql-server/sql-server-2016/
[6]MongoDB https://www.mongodb.com/
[7]Amazon DynamoDB https://aws.amazon.com/pt/dynamodb/
[8]Amazon Redshift https://aws.amazon.com/pt/redshift/
[9]Apache Kudu https://kudu.apache.org/
[10]MariaDB Columnstore https://mariadb.com/products/technology/columnstore/
[11]Neo4j https://neo4j.com/
[12]Apache Kudu https://kudu.apache.org/
[13]MariaDB Columnstore https://mariadb.com/products/technology/columnstore/

cas em caso de falhas. Uma desvantagem é que pode sacrificar consistência de dados em algumas situações.

Existem vantagens e desvantagens no uso de NoSQL. Segundo Nayak *et al.* (2013), as *vantagens* dos SGBDs NoSQL em relação aos bancos de dados relacionais são:

- Possuem diversos tipos de modelos de dados que podem se adequar melhor ao problema tratado
- Facilmente escaláveis
- Administradores de bancos de dados não são necessários
- Podem ser tolerantes a falhas de hardware
- Mais rápidos, mais eficientes e flexíveis
- Tem evoluído e se desenvolvido de forma muito rápida

Algumas das *desvantagens* discutidas no artigo citado são:

- Pouca maturidade
- Não existe uma linguagem padrão de consultas como o SQL
- Alguns SGBDs NoSQL não atendem aos requisitos de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) propostos por Haerder e Reuter (1983) que garantem que transações sejam atômicas e mantenham a consistência dos dados em caso de falha
- Não possuem uma interface de acesso padrão
- Podem ser de difícil manutenção e gerenciamento

2.4.1 MongoDB

O MongoDB (de *humongous*) é um SGBD NoSQL orientado a documentos de código aberto, que armazena informações utilizando documentos JSON.

Ele provê alta disponibilidade através de replicação dos dados em réplicas primárias e secundárias. Também dá suporte a balanceamento de carga e pode ser escalado horizontalmente através de *sharding*, que são particionamentos horizontais do banco de dados. Cada partição é um *shard*, que fica em servidores diferentes e até locais geográficos diferentes, perimitindo que um mesmo banco de dados possa ficar distribuído por diferentes partes do mundo, algo que diminui latência e melhora a performance de consultas, já que o número de dados salvos em cada servidor é diminuída assim como o tamanho dos índices. A desvantagem disto é que podem surgir problemas de consistência e durabilidade, visto que cada servidor pode eventualmente falhar durante atualizações de dados, mas o MongoDB minimiza esse tipo de problema na replicação e duplicação desses dados.

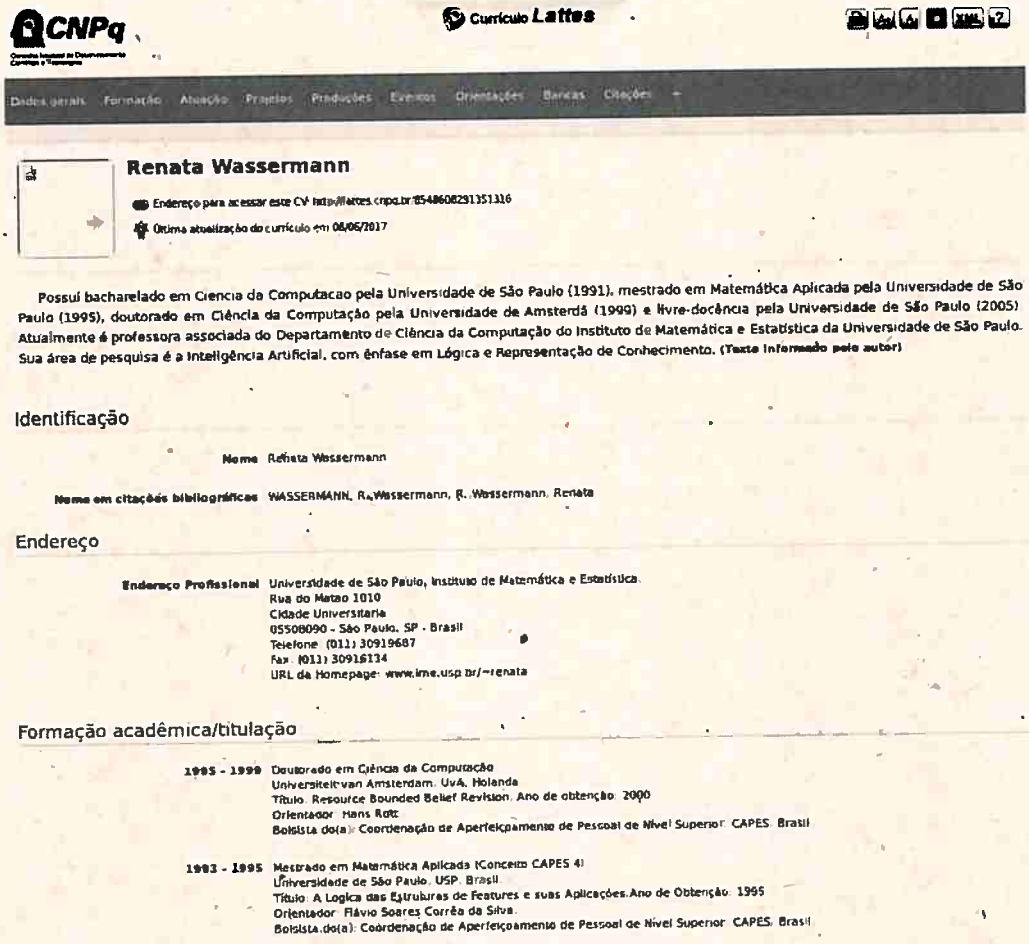
O MongoDB dá suporte a operações de agregação, como agrupamentos similares ao *GROUP BY* do SQL. Também dá suporte ao operador *lookup*, que é uma forma de aglutinar um documento com outros, gerando visualizações parecidas com a ideia de um *JOIN* do SQL. Também é possível executar código javascript direto no servidor de banco de dados. Isto é utilizado para se fazer filtragem e agregação de documentos, e operações como *map/reduce*.

Mais detalhes de consultas e do modo de armazenamento do MongoDB podem ser encontrados na Seção 2.6.2.

2.5 Plataforma Lattes

A Plataforma Lattes[14] é uma plataforma vinculada ao CNPq e a mais importante base integrada de currículos, grupos de pesquisa e instituições de ensino do Brasil, registrando informações valiosas sobre as atividades de pesquisa, publicações, e o perfil de pesquisadores de diversas áreas do saber em todo o país, e é uma base de dados pública e aberta.

Qualquer pesquisador do Brasil pode criar seu currículo Lattes gratuitamente, basta preencher algumas informações pessoais e seu currículo ficará disponível em uma página web aberta.


CNPq

Currículo Lattes

Dados gerais Formação Atuação Projetos Produções Eventos Orientações Bancas Citações +

Renata Wassermann

Endereço para acessar este CV: http://lattes.cnpq.br/8548608291351316

Última atualização do currículo em 08/06/2017

Possui bacharelado em Ciencia da Computacao pela Universidade de São Paulo (1991), mestrado em Matemática Aplicada pela Universidade de São Paulo (1995), doutorado em Ciência da Computação pela Universidade de Amsterdã (1999) e livre-docência pela Universidade de São Paulo (2005). Atualmente é professora associada do Departamento de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. Sua área de pesquisa é a Inteligência Artificial, com ênfase em Lógica e Representação de Conhecimento. (Texto informado pelo autor)

Identificação

Nome: Renata Wassermann

Nome em citações bibliográficas: WASSERMANN, R.;Wassermann, R.;Wassermann, Renata

Endereço

Endereço Profissional: Universidade de São Paulo, Instituto de Matemática e Estatística
Rua do Matao 1010
Cidade Universitaria
05508090 - São Paulo, SP - Brasil
Telefone: (011) 30919687
Fax: (011) 30916134
URL da Homepage: www.ime.usp.br/~renata

Formação acadêmica/titulação

1995 - 1999: Doutorado em Ciência da Computação
Universiteit van Amsterdam, UvA, Holanda
Título: Resource Bounded Belief Revision. Ano de obtenção: 2000
Orientador: Hans Rott
Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, Brasil

1993 - 1995: Mestrado em Matemática Aplicada (Conceito CAPES 4)
Universidade de São Paulo, USP, Brasil
Título: A Lógica das Estruturas de Features e suas Aplicações. Ano de Obtenção: 1995
Orientador: Flávio Soares Corrêa da Silva
Bolsista do(a): Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, Brasil

Figura 2.1: *Página de exibição de um currículo Lattes*

As informações disponibilizadas no currículo são:

- Dados pessoais como nome, endereço, e nome em citações
- Um resumo de sua atividade acadêmica, profissional e seus interesses de pesquisa
- Formação acadêmica, titulação e formação complementar

[14] Plataforma Lattes http://lattes.cnpq.br/

- Atuação profissional
- Participação em projetos de pesquisa
- Área de atuação
- Produção bibliográfica, seja ela técnica, artística ou científica
- Apresentações e participações em eventos científicos
- Orientações em andamento e concluídas

Os dados dos currículos Lattes também são disponibilizados em formato XML e possuem todo um vocabulário padronizado para os elementos relacionados à produção científica dos pesquisadores cadastrados. Um exemplo de currículo em XML pode ser visto em 2.3.

```
1 <?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
2 <CURRICULO-VITAE SISTEMA-ORIGEM-XML="LATTES_OFFLINE" NUMERO-IDENTIFICADOR="
      8548608291351316" DATA-ATUALIZACAO="08062017" HORA-ATUALIZACAO="184437">
3   <DADOS-GERAIS
4     NOME-COMPLETO="Renata Wassermann"
5     NOME-EM-CITACOES-BIBLIOGRAFICAS="WASSERMANN, R.;Wassermann, R.;Wassermann,
          Renata"
6     NACIONALIDADE="B"
7     PAIS-DE-NASCIMENTO="Brasil"
8     UF-NASCIMENTO="SP"
9     CIDADE-NASCIMENTO="Sao Paulo"
10    PERMISSAO-DE-DIVULGACAO="NAO"
11    DATA-FALECIMENTO=""
12    SIGLA-PAIS-NACIONALIDADE="BRA"
13    PAIS-DE-NACIONALIDADE="Brasil"><RESUMO-CV
14    TEXTO-RESUMO-CV-RH="Possui bacharelado em Ciencia da Computacao pela
          Universidade de Sao Paulo (1991), mestrado em Matematica Aplicada pela
          Universidade de Sao Paulo (1995), doutorado ..."
15    TEXTO-RESUMO-CV-RH-EN="Bachelor in Computer Science from Universidade de Sao
          Paulo (1991), Masters in Applied Mathematics from Universidade de Sao
          Paulo (1995), PhD from the ..."/>
16    <ENDERECO FLAG-DE-PREFERENCIA="ENDERECO_INSTITUCIONAL"><ENDERECO-
          PROFISSIONAL
17      CODIGO-INSTITUICAO-EMPRESA="006700000002"
18      NOME-INSTITUICAO-EMPRESA="Universidade de Sao Paulo"
19      CODIGO-ORGAO="006705000000"
20      NOME-ORGAO="Instituto de Matematica e Estatistica"
21      CODIGO-UNIDADE=""
22      NOME-UNIDADE=""
23      LOGRADOURO-COMPLEMENTO="Rua do Matao 1010"
24      PAIS="Brasil"
25      UF="SP"
26      CEP="05508090"
27      CIDADE="Sao Paulo"
28      BAIRRO="Cidade Universitaria"
29      DDD="011"
30      TELEFONE="30919687"
31      RAMAL=""
```

```
32        FAX="30916134"
33        CAIXA-POSTAL=""
34        HOME-PAGE="www.ime.usp.br/~renata"/></ENDERECO>
35      <FORMACAO-ACADEMICA-TITULACAO><GRADUACAO
36        SEQUENCIA-FORMACAO="1"
37        NIVEL="1"
38        TITULO-DO-TRABALHO-DE-CONCLUSAO-DE-CURSO=""
39        ...
```

Código 2.3: *Exemplo de currículo Lattes em XML*

O sistema também disponibiliza relatórios consolidados de produção científica e meios para busca de currículos por área de interesse e palavras-chave. Essas informações são de grande interesse para se fazer análise de dados das redes de colaboração científica.

2.6 A linguagem *Ruby*

A linguagem de programação *Ruby*[15] é uma linguagem interpretada de código aberto. Sua sintaxe foi inspirada nas linguagens Lisp, Perl, Smalltalk, Ada, e Eiffel. É uma linguagem multi-paradigma, que mistura elementos de linguagens funcionais com linguagens imperativas, e todos os seus tipos são objetos e podem ter métodos e variáveis de instância. Inclusive tipos primitivos como números são objetos.

Por ser bastante flexível, qualquer parte da linguagem pode ser alterada, removida ou reescrita. Os operadores são também métodos que podem ser reescritos. Um exemplo disso foi retirado da documentação: podemos adicionar ao tipo *Numeric* um novo método chamado *plus* que pode ser usado para soma. No corpo do método, chamamos o operador +, que é apenas açúcar sintático para uma função que recebe um número como parâmetro e faz a soma.

```
1 class Numeric
2   def plus(x)
3     self.+(x)
4   end
5 end
6
7 y = 5.plus 6
```

A linguagem possui blocos, que são funções anônimas que podem ser invocadas ao serem passadas para um método. Isso permite inversão de controle e injeção de dependência, algo que ajuda a aumentar o desacoplamento entre módulos, classes e elementos de um projeto.

Uma biblioteca *Ruby* de código aberto pode ser empacotada e transformada numa chamada *gem*[16], que é algo parecido com um pacote *.jar* do Java. Isso permite uma fácil distribuição e utilização dessa biblioteca em outros projetos *Ruby*. Uma *gem* muito famosa para a linguagem é chamada de *Ruby on Rails*, que discutimos em 2.6.1.

Por sua flexibilidade e pela facilidade em se fazer metaprogramação, a linguagem se presta muito bem para a criação das chamadas *linguagem de domínio específico*. Segundo Fowler (2010), uma linguagem de domínio específico é uma linguagem com a mesma sintaxe da linguagem de

[15] The Ruby Language https://www.ruby-lang.org/

[16] *RubyGems* é um serviço de hospedagem e distribuição de *gems* https://rubygems.org/

programação em que foi desenvolvida, porém muito mais limitada e cujo propósito é resolver um problema específico em um determinado domínio de maneira mais simples. Sua sintaxe é estilizada para facilitar a definição do problema, e ela pode ser usada para também gerar código a partir das definiçoes criadas, diminuindo assim o trabalho do usuário.

Como exemplo disso, podemos citar a biblioteca *Capybara*[17] que serve para fazer testes de navegação e interface de projetos web. Essa biblioteca simula as interações de um usuário real com o seu sistema, como abrir um navegador, digitar um endereço e clicar em um botão.

Criamos um teste de interação utilizando essa biblioteca que pode ser visto a seguir. O método abre um navegador, visita uma página de busca, digita um texto no campo de busca e clica no botão "Pesquisar".

```
1 describe "o processo de busca", :type => :feature do
2   it "acessa o site e busca por Plataforma Lattes" do
3     visit "http://google.com"
4     fill_in 'Busca', with: 'Plataforma Lattes'
5     click_button 'Pesquisar'
6     expect(page).to have_content 'Plataforma Lattes'
7   end
8 end
```

Por trás desse código há todo o processo de iniciar a aplicação do navegador, mandar uma requisição para abrir o endereço do site, todo o gerenciamento do ciclo de vida da página, assim como os eventos que devem ser feitos na tela, e o método para encontrar os elementos, como campos de texto e botões. Só que tudo isso fica transparente para o desenvolvedor que cria o teste, pois ele não precisa se preocupar com nada disso. Isso é um exemplo de como uma DSL pode ser uma ferramenta bastante útil para tratar de alguns tipos de problemas.

2.6.1 Ruby on Rails

O *Ruby on Rails*[18] é um *framework* para o desenvolvimento de aplicações web que utilizam banco de dados. Sua estrutura segue a ideia de Convenção Sobre Configuração (*convention over configuration*) e não é necessário que o desenvolvedor precise configurar todos os detalhes para ter uma aplicação funcional, pois o funcionamento e a separação dos módulos é feita seguindo convenções bem definidas.

O *Rails* segue um padrão arquitetural de desenvolvimento de software chamado MVC (*Model-View-Controller*), que divide a aplicação em três camadas principais, cada uma com responsabilidades bem específicas, que podem ser vistas no diagrama 2.2.

A camada de visualização ou apresentação (*View*) contém os arquivos (*templates*) que geram telas e visualizações, provendo representações apropriadas dos recursos e dados da aplicação, e isto é feito através da geração de vários formatos de arquivo, como HTML, JSON, XML, imagens, gráficos e outros tipos de arquivos. Geralmente, os *templates* são compostos por código HTML e código *Ruby* interpretado.

A camada de modelo (*Model*) contém um conjunto de classes que répresentam o domínio da aplicação, podendo conter classes que dizem respeito a coisas como Usuário, Produto, Pessoa, Pesquisador e Publicação. Essas classes contém regras de negócios específicas da aplicação, e podem

[17] Repositório da biblioteca *Capybara* https://github.com/teamcapybara/capybara
[18] *Ruby on Rails* http://rubyonrails.org/

ter por trás também métodos de persistência de dados, podendo por exemplo salvar informações em uma ou mais tabelas em um banco de dados. O *Rails* utiliza o *Active Record* para consultar dados de tabelas e transformá-los em conjuntos de objetos do tipo *model*.

A **camada de controladores** (*Controller*) é responsável por intermediar as requisições HTTP feitas para o servidor web e responder de forma adequada, seja gerando uma visualização de alguma lista de dados em HTML, ou um aquivo JSON com dados de um usuário, ou um PDF, por exemplo. Os controladores manipulam as classes de modelo e renderizam templates com dados obtidos do banco de dados, respondendo às requisições HTTP com essas saídas.

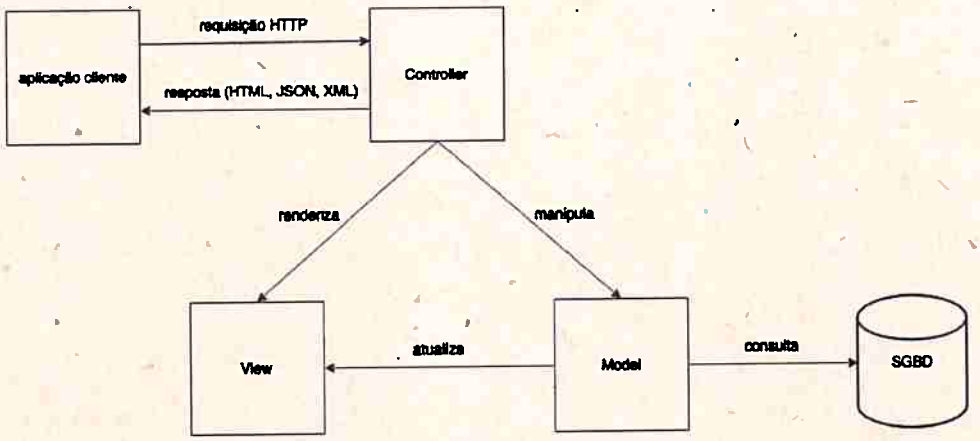


Figura 2.2: *Diagrama do Padrão Arquitetural MVC (model-view-controller)*

Nos últimos anos, o *Ruby on Rails* se tornou uma ferramenta muito utilizada na criação de aplicações web e APIs, que são aplicações web que geralmente aceitam requisições HTTP contendo documentos JSON e que também respondem apenas JSON. São muito utilizadas como *backend* para aplicações móveis para celulares Android e iOS.

2.6.2 Mongoid

O *Mongoid*[19] é uma biblioteca (*gem*) de acesso ao MongoDB para *Ruby* desenvolvida pelos criadores do MongoDB. Trata-se de um ODM (*Object-Document-Mapper*) que transforma documentos JSON em objetos *Ruby* e vice-versa. Possui funções para consulta, inserção de dados, atualização, exclusão, manipulação de coleções e acesso em geral a bases de dados MongoDB.

O *Mongoid*, assim como o MongoDB, trabalha com documentos JSON. Na verdade, trabalha com os chamados documentos BSON (*binary JSON*), que são documentos JSON serializados e codificados em binário que são utilizados como meio de troca de informações para obter uma melhor performance e melhor utilização de espaço.

Uma classe *Ruby* pode utilizar o *Mongoid* ao incluir os métodos de serialização e mapeamento. Como exemplo, considere a classe **Pessoa**:

```
1 class Pessoa
2   include Mongoid::Document
3
4   field :nome, type: String
5   field :idade, type: Integer
6   field :profissao, type: String
```

[19] Documentação do *Mongoid* https://docs.mongodb.com/mongoid/master/

```
7 end
```

O método *include* adiciona todas as funções do módulo *Mongoid::Document* nessa classe, que pode agora ser persistida no banco de dados. Os macros *field* adicionam alguns atributos a essa classe, que serão salvos no banco de dados como campos em um documento JSON.

Podemos criar uma nova *pessoa* e salvar suas informações no MongoDB:

```
1 pessoa = Pessoa.new(nome: 'Carlos Drummond de Andrade',
2                     idade: 84,
3                     profissao: 'poeta')
4 pessoa.save
```

Inicializamos uma nova pessoa com alguns atributos preenchidos, e chamamos o método *save*, que cria um novo documento JSON e armazena esses dados em uma base de dados MongoDB. O formato do documento pode ser visto a seguir:

```
1 {
2   "_id": {
3     "\$oid": "5991a3d48606231a82802cb6"
4   },
5   "idade": 84,
6   "nome": "Carlos Drummond de Andrade",
7   "profissao": "poeta"
8 }
```

Também podemos consultar a coleção através de métodos de consulta e filtragem, como o *where*. Como exemplo, filtramos a coleção *Pessoa* para selecionar apenas poetas com mais de 80 anos de idade:

```
1 Pessoa.where(:profissao => 'poeta')
2       .where(:idade.gt -> 80)
```

Também é possível fazer consultas mais complicadas compostas até por código javascript que é executado no banco de dados para se fazer a filtragem e agregação dos dados. O MongoDB também dá suporte a alguns tipos de consultas com agregação.

Uma operação possível é o chamado *map/reduce*, que consiste de duas funções javascript que são passadas para o banco de dados. A função *map* recebe um conjunto de dados de uma coleção e processa cada item, fazendo o mapeamento dessas informações, a filtragem e ordenação, e gerando novos itens de saída. A função *reduce* recebe os itens mapeados e, para cada item, realiza operações de agregação, como soma, contagem, média e mediana, gerando resultados consolidados.

A função *map/reduce* aplica as operações em paralelo, decompondo cada operação em passos que são aplicados a partições de dados. Após isso, esses resultados são combinados e as operações são sincronizadas para gerar a saída final. Isso permite um ganho de performance para operações que podem ser facilmente paralelizadas e permite a exploração de grandes volumes de dados.

Como exemplo, podemos calcular a idade média das pessoas por profissão:

```
1 map = %Q{
2   function() {
3     emit(this.profissao, { age: this.idade });
4   }
5 }
```

```
6
7 reduce = %Q{
8   function(key, values) {
9     var result = { avg: 0 };
10    var size = values.length;
11    values.forEach(function(value) {
12      result.avg += value.age / size;
13    });
14    return result;
15  }
16 }
17
18 Person.map_reduce(map, reduce).out(inline: 1).to_a
```

Neste caso, a função *map* é aplicada à coleção **Pessoa** e, para cada item dessa coleção, gera um item de chave-valor contendo a profissão como a chave e um *hash* que contém a idade dessa pessoa. Por exemplo: {"*poeta*" : {"*age*" : 84}}.

A função *reduce* recebe esse conjunto de chaves e valores agrupados pela chave (no caso, profissão) e é aplicada em cada um desses itens. Para cada profissão, a função soma as idades dividida pela quantidade de itens, e responde com um conjunto agregado de profissões com o valor médio de idades para cada uma delas. Esse resultado é também um documento JSON.

```
1 [{
2     "_id": "atleta",
3     "value": {
4         "age": 78.0
5     }
6 }, {
7     "_id": "jogador de futebol",
8     "value": {
9         "avg": 62.5
10    }
11 }, {
12    "_id": "poeta",
13    "value": {
14        "avg": 70.6
15    }
16 }, {
17    "_id": "professor",
18    "value": {
19        "avg": 70.5
20    }
21 }]
```

Capítulo 3

Trabalhos Correlatos

Neste capítulo são apresentadas algumas contribuições importantes para as discussões das próximas seções deste projeto de pesquisa. Os trabalhos apresentados a seguir nos ajudarão a visualizar as aplicações dos conceitos pertinentes à nossa proposta, como o problema de acesso a dados baseados em ontologias, bancos de dados NoSQL, análise de redes de colaboração científica, dentre outros.

3.1 Acesso a Bases de Dados baseado em Ontologias (OBDA)

Considerando a quantidade cada vez maior de domínios de conhecimento que surgem a cada dia, o Acesso a Dados baseado em Ontologias pode servir de grande utilidade na análise desses diversos domínios do conhecimento. Alguns trabalhos modelaram ontologias e utilizaram técnicas de acesso para tratar de problemas diversos de forma eficaz, e que discutimos a seguir.

3.1.1 Ontop

O trabalho pioneiro a propor Acesso a Base de Dados baseado em Ontologias de Bagosi *et al.* (2014) é o *Ontop*. Ele oferece suporte a todas as recomendações da W3C e aos bancos de dados mais populares e gratuitos utilizados desde a época.

Calvanese *et al.* (2017) descreve a arquitetura do Ontop, que é um projeto de código aberto que expõe bancos de dados relacionais como grafos RDF virtuais. Isto é feito ligando-se os termos (classes e proriedades) na ontologia para mapeamentos com os dados persistidos em um SGGB relacional.

O grafo RDF virtual pode ser consultado através de SPARQL que é traduzido em consultas SQL que são executadas no banco de dados relacional, e esse processo de tradução é transparente para o usuário final.

Podemos ver na figura 3.1 retirada do artigo a arquitetura geral do Ontop.

A arquitetura do Ontop possui as seguintes camadas:

- Entradas (inputs) que são relacionadas ao domínio, como o mapeamento, as consultas SQL e o banco de dados
- O módulo de tradução, otimização e execução de consultas
- As interfaces de acesso em Java

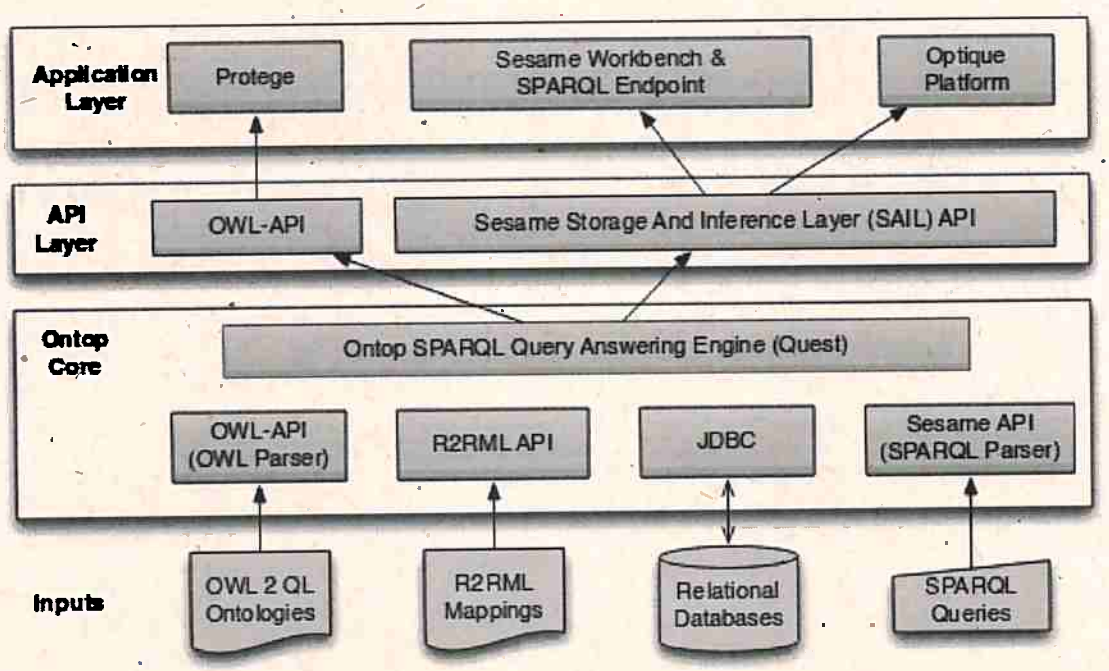


Figura 3.1: *Arquitetura do projeto Ontop*

- Aplicações que permitem que o usuário execute consultas SPARQL

O mapeamento é feito utilizando-se uma sintaxe nativa do Ontop que liga classes da ontologia com fragmentos de consultas SQL.

Uma das principais contribuições que o caracteriza como um trabalho pioneiro é a geração de consultas SQL extremamente eficientes e otimizadas, e essas consultas podem ser feitas por especialistas no domínio sem que precisem conhecer o banco de dados, ou sequer serem especialistas em bancos de dados.

3.1.2 Outros Trabalhos

Uma das primeiras formas de se trabalhar com OBDA com SGBDs relacionais é feita através da tradução de consultas SPARQL originadas de uma ontologia para consultas SQL, que é a linguagem de destino da base de dados a ser acessada. Com isto, é possível extrair informações do banco de dados e exportá-las para RDF.

O trabalho de Prud'hommeaux e Bertails (2008) propõe uma forma de expressar dados relacionais como um grafo RDF e uma álgebra para mapear consultas SPARQL do tipo *SELECT* sobre o grafo RDF em consultas SQL sobre um conjunto de dados relacionais. O grafo RDF é construído com base na estrutura relacional juntamente com identificadores URI.

A álgebra especifica uma função que recebe como entrada um identificador, um esquema relacional e uma consulta SPARQL nesse mesmo grafo, gerando uma consulta relacional que pode ser executada em um banco de dados, produzindo as mesmas soluções que uma consulta SPARQL executada sobre o grafo.

Com essa técnica, um banco de dados relacional pode ser exposto na web semântica e receber consultas SPARQL com a mesma performance de consultas SQL.

3.1.3 Ontop e NoSQL

O trabalho recente de Botoeva *et al.* (2016a) propõe o uso de NoSQL em aplicações OBDA, expandindo a proposta de Bagosi *et al.* (2014) de usar OBDA com bancos de dados relacionais. Essa proposta define uma arquitetura genérica de OBDA que poderia ser aplicada a qualquer tipo de SGBD.

A primeira aplicação desta arquitetura foi feita com o MongoDB, um banco de dados orientado a documentos. Com isso, foi possível mapear uma ontologia com um conjunto de dados salvos em uma coleção do MongoDB e fazer a conversão de uma consulta SPARQL em uma consulta MongoDB. Isto eliminou a necessidade de se extrair esses dados previamente para popular a ontologia antes da consulta.

Em um trabalho relacionado de Botoeva *et al.* (2016b), foi apresentada uma avaliação formal do conjunto de consultas de acesso disponíveis no MongoDB em que se mostrou que era difícil construir uma biblioteca de acesso que fosse totalmente genérica e capaz de acessar qualquer tipo de SGBD NoSQL. Diferentemente dos bancos de dados relacionais que utilizam uma mesma linguagem em comum (SQL), os SGBDs NoSQL não compartilham de muitos padrões de consulta, por isso, é necessário criar um tradutor de consultas específico para cada SGBD NoSQL.

Os dois trabalhos citados (Botoeva *et al.* (2016a) e Botoeva *et al.* (2016b)) são interessantes, porém se concentram muito mais em questões formais como provar algumas propriedades dos métodos sugeridos do que mostrar mais especificamente a implementação prática desses métodos, tampouco dão acesso ao código utilizado ou desenvolvido. Portanto, são discussões relevantes, mas bastante teóricas. Seria de grande auxílio para outros pesquisadores se o código desses projetos fosse aberto. Foi um dos motivos que nos levou a desenvolver nosso próprio método com testes práticos e disponibilizar o código para que outros também possam utilizar e contribuir.

Outra proposta interessante nessa área (Michel *et al.*, 2016) propõe uma ferramenta de mapeamento de consultas SPARQL em consultas a uma base MongoDB para transformá-la em uma fonte de dados pública. A técnica utilizada consiste em gerar uma base RDF virtual, ou seja, os documentos salvos na base foram expostos em formato RDF.

Nesta proposta, foi criado um método de tradução em dois passos: primeiramente, a consulta SPARQL é transformada em uma consulta abstrata usando mapeamentos MongoDB para RDF escritos em uma linguagem intermediária chamada *xR2RML*. Depois, essa consulta intermediária é transformada em uma consulta MongoDB concreta. Como resultado, concluíram que é sempre possível reescrever uma consulta que produza os resultados corretos.

3.2 Plataforma Lattes

Alguns projetos interessantes exploram os dados disponíveis na Plataforma Lattes para fazer análises da produção científica e das redes de colaboração presentes na plataforma. A seguir, discutimos alguns desses trabalhos.

3.2.1 Ontologias e Plataforma Lattes

Um dos primeiros projetos que utiliza Ontologias para tratar dados da Plataforma Lattes foi desenvolvido por Bonifacio (2002), que propõe um modelo de metadados com semântica para o Currículo Lattes baseado em uma ontologia. Outro trabalho desenvolvido por Castano (2008) teve

como motivação a geração de relatórios com informações que apresentem maior consistência semântica, utilizando-se uma ontologia a partir de arquivos no formato HTML. Por se tratar de um dos primeiros trabalhos que fazem essa relação, é importante ressaltar que não se utilizou ainda OBDA para a extração dos dados.

As principais dificuldades encontradas estão relacionadas não só à extração e transformação de uma base de conhecimento com as informações corretas, possibilitando o mapeamento da ontologia com a semântica correta, mas também de se evitar informações duplicadas.

As principais contribuições deste projeto estão relacionadas à criação de instâncias dentro de uma ontologia gerada a partir de scripts automáticos. Além disso, o trabalho citado serve de exemplo da aplicação de ontologias para o tratamento de informações relevantes à comunidade acadêmica.

O nosso projeto se aproxima desse no sentido de oferecer uma possível solução a um dos tópicos apresentados como trabalhos futuros, comparando o uso de SPARQL em bases de conhecimento utilizando consultas em bancos de dados orientados a objetos, que fazem parte do grupo de bancos de dados NoSQL.

3.2.2 scriptLattes

Parte dos problemas citados no trabalho anterior foram resolvidos no projeto *scriptLattes*, proposto por Mena-Chalco e Junior (2009). Trata-se de um sistema capaz de fazer mineração dos dados de currículos presentes na Plataforma Lattes e de gerar vários relatórios acadêmicos, além de disponibilizar informações sobre as publicações dos pesquisadores brasileiros, fazendo desambiguação dos autores e artigos e exportando dados sobre coautoria e outros tipos de colaboração. Entretanto, esse projeto extrai apenas informações sintáticas, não tratando de questões semânticas.

O nosso trabalho utiliza uma base de mais de 4 milhões de currículos extraídos da Plataforma Lattes, gentilmente cedida pelo professor Jesús P. Mena-Chalco, como base de testes para o modelo proposto neste trabalho.

3.2.3 Ontologias e OBDA na Plataforma Lattes

Uma outra forma de automatizar a consulta de informações na Plataforma Lattes envolve o uso de ontologias. Tal abordagem foi desenvolvida no trabalho de Costa e Yamate (2009).

O projeto disponibiliza uma ontologia capaz de responder a diversas questões de competência. O usuário consulta a base usando linguagem natural, e a ontologia consegue encontrar a informação desejada, através de inferência.

A vantagem dessa metodologia é a facilidade na modelagem, nas ferramentas de inferência, e na expressividade da representação. Galego (2013) também desenvolveu uma ontologia para esse domínio como uma extensão do trabalho anterior, e seu interesse foi gerar relatórios e detectar inconsistências.

3.2.4 Análise de Redes de Colaboração através de Grafos Relacionais com Atributos

Um experimento muito interessante e inspirador foi explorado por Cervantes (2014) ao modelar a rede de colaboração científica através de grafos relacionais com atributos para mostrar as relações de coautoria entre pesquisadores. Foi também criado um modelo capaz de prever novas colaborações

(ligações) entre pesquisadores a partir de dados de treinamento extraídas da Plataforma Lattes via *scriptLattes*.

O modelo proposto permite uma série de outras análises relevantes da estrutura dessa rede, como a identificação de pesquisadores mais importantes, ou mais colaborativos, que correspondem aos vértices com mais conexões. Também identifica comunidades dentro de diferentes áreas formadas por componentes conexos desse grafo.

Por sua flexibilidade e alto nível de expressividade, esse modelo em grafo com atributos pode ser facilmente expandido. Entretanto, o modelo de aprendizado utilizado recebe como entrada tabelas de atributos e valores. Por causa disso, é preciso transformar uma representação relacional mais expressiva (um grafo) em uma representação mais simples (uma tabela) para que o algoritmo funcione.

Esse modo de representação se aproxima do modo de representar um domínio por uma ontologia ao expressar as entidades desse domínio como um grafo com atributos. Uma ontologia também pode ser expressa como um grafo de entidades (vértices) e relações (arestas). Essa estrutura da rede pode ser expandida com atributos e restrições, e poderíamos também adicionar informações semânticas nesse modelo de grafos com atributos para enriquecê-lo.

3.3 Justificativa

A partir dos trabalhos já realizados discutidos acima, percebemos ainda haver espaço para contribuições envolvendo ontologias. A principal justificativa para o projeto aqui proposto é testar a hipótese de ser possível criar um modelo mais flexível e extensível de extração e integração entre uma base NoSQL e uma ontologia, através da tradução de consultas SPARQL.

O uso de uma ontologia é justificado por sua expressividade, algo que simplifica a criação de consultas geradoras de novos conhecimentos, e a possibilidade da descoberta de novas características, através do uso de inferência lógica.

Outra vantagem secundária é a uniformização do vocabulário a respeito do domínio, e a fácil reutilização desses conceitos e conhecimentos em outras aplicações de domínios semelhantes, sem que seja necessário que o usuário saiba como esses dados foram armazenados.

Escolhemos aplicar esse modelo à Plataforma Lattes por ser um domínio bem utilizado e por termos disponível uma base de currículos bastante extensa, da ordem de 4 milhões de arquivos.

A partir das contribuições já feitas, nossa pesquisa propõe utilizar OBDA para o acesso a uma base NoSQL de maneira diferente, através de uma forma bastante simples de criação do mapeamento, e também usando uma camada conceitual intermediária que torna o processo extensível, podendo ser reutilizado se for preciso dar suporte a outros tipos de SGBDs NoSQL, e até mesmo a bancos de dados relacionais. Acreditamos que esta seja uma alternativa eficaz e bem prática para o uso de ontologias, NoSQL e OBDA no contexto do *Big Data*.

Capítulo 4

Desenvolvimento

O objetivo geral da pesquisa tem por fim a criação de um método de extração de conhecimento ou mineração de conhecimento de bases de dados com grande volume. Também tem por interesse a criação de métodos de enriquecimento desse conhecimento com informações geradas através de análise de dados e algoritmos de aprendizado de máquina.

4.1 Arquitetura Geral

A arquitetura geral desse método é composta pelas seguintes partes:

1. **Ontologia OWL** para definição formal de conceitos ligados a um certo domínio
2. **Bases de dados NoSQL** com um grande volume de informações semi-estruturadas relacionadas ao mesmo domínio
3. **Acesso a dados NoSQL com OBDA**, com tradução de consultas SPARQL em consultas ao banco de destino, usando mapeamentos
4. **Transformação** dos dados em instâncias na Ontologia
5. **Análise de Dados**, construção e enriquecimento de onhecimento
6. **Relatórios** e gráficos gerados a partir da análise

No presente projeto de pesquisa, focamos no desenvolvimento do método de acesso e extração de dados de bases NoSQL através do uso de Acesso a Dados Baseado em Ontologias (OBDA).

4.2 OBDA com NoSQL

Nesta Seção apresentamos nossa proposta de solução para o problema de acesso a SGBDs NoSQL utilizando OBDA.

4.2.1 Tradução de Consultas

Utilizamos um modelo OBDA como *Interface de Acesso* ao banco de dados. Essa interface de acesso é composta por uma ontologia OWL, um mapeamento e uma camada conceitual intermediária

que permite o acesso aos dados de um SGBD NoSQL. A camada conceitual é a principal diferença do nosso método com os métodos presentes na literatura.

Nossa proposta tem por interesse facilitar a construção do mapeamento ontologia x banco de dados e torná-lo mais flexível e genérico, e permitir a sua reutilização de forma independente do SGBD utilizado para armazenar os dados. Para tornar isso possível, desenvolvemos uma camada conceitual intermediária representada por um conjunto de classes em uma linguagem de programação orientada a objetos (POO) para representar a estrutura das coleções de dados presentes no SGBD. Ela também é dotada de classes de tradução de consultas para SGBDs específicos.

Nossa proposta simplifica a construção do mapeamento entre ontologia e as entidades do Banco de Dados, pois uma representação orientada a objetos fica muito próxima da representação ontológica do domínio. Representação de conhecimento em uma ontologia é definida através do uso de classes, relações entre entidades, uso de hierarquias de conceitos e herança, que são ideias similares aos conceitos por trás das linguagens orientadas a objetos. Além disso, diferentemente de algumas outras propostas que definem o mapeamento como um conjunto de consultas SQL, nosso mapeamento é definido através de uma linguagem de domínio específico que mostra de forma sucinta quais campos ou relações no banco de dados devem ser utilizados para representar tais e tais classes da ontologia e suas relações, algo que torna bem mais clara a intenção por trás do mapeamento, facilitando também a geração automática do mapeamento.

Além disso, a tarefa de acesso aos dados persistidos no banco de dados e sua transformação em instâncias de objetos pode ser feita por bibliotecas ORM (Object-Relational Mapping) ou ODM (Object Document Mapper), que são bastante utilizadas na indústria para fazer o acesso a diferentes bancos de dados utilizando linguagens POO. Essas bibliotecas podem acessar diversos tipos de dados armazenados em diferentes grupos de SGBDs, tanto os relacionais, através da geração de consultas SQL, quanto bancos de dados orientados a documentos, como o MongoDB. E o resultado dessas consultas é transformado em coleções de objetos.

No mapeamento proposto aqui, cada classe ou atributo proveniente da ontologia deve ser associado a sua respectiva classe ou atributo de classe equivalente no modelo orientado a objetos (modelo POO). O modelo, por sua vez, é associado ao esquema do banco de dados através do uso de um ORM ou ODM. Relações presentes na ontologia também podem ser mapeadas para relações entre classes do modelo. Após a configuração do mapeamento, a Interface de Acesso é capaz de receber uma consulta escrita em SPARQL e executar a recuperação dos dados.

A tradução da consulta SPARQL é feita através da transformação da consulta em um grafo descrevendo os dados que deverão ser acessados. Os vértices desse grafo representam as classes e as arestas representam as relações entre as classes. Cada vértice pode conter a lista de atributos pertencentes à classe que serão necessários para se responder à consulta. Essa representação pode ser feita utilizando-se um grafo com atributos similar ao método proposto por Cervantes (2014). Através dela, o método consegue avaliar quais entidades do banco de dados deve consultar e quais atributos utilizar.

A definição de todos os dados que deverão ser extraídos do banco de dados para responder à consulta SPARQL estão codificados no mapeamento e no grafo. Dessa forma, as operações de consulta a dados são feitas através do uso da biblioteca ORM/ODM para o banco de dados escolhido ou através de mecanismos internos de geração de consultas para o banco de dados específicos.

Uma primeira implementação dessa arquitetura foi feita através do projeto *OntoMongo* e foi

aplicado a SGBDs NoSQL. Descrevemos o projeto na seção 4.3.

4.3 OntoMongo

O projeto *OntoMongo*[1] é um protótipo funcional que foi desenvolvido para testar o método proposto de tradução de consultas usando OBDA. Foi desenvolvido em conjunto com o projeto de mestrado da aluna Bárbara Tieko Agena (Agena, 2017).

O método gera triplas RDF como resultado da consulta, que então são usadas para popular uma ontologia OWL. Como ilustrado pela figura 4.1, o projeto possui os seguintes componentes:

1. Ontologia OWL
2. Bases de dados NoSQL
3. Interface de Acesso
4. Mapeamento Ontologia x Coleções na Base de Dados
5. Tradutor SPARQL para NoSQL
6. Exportação para RDF

Descrevemos cada uma das partes a seguir.

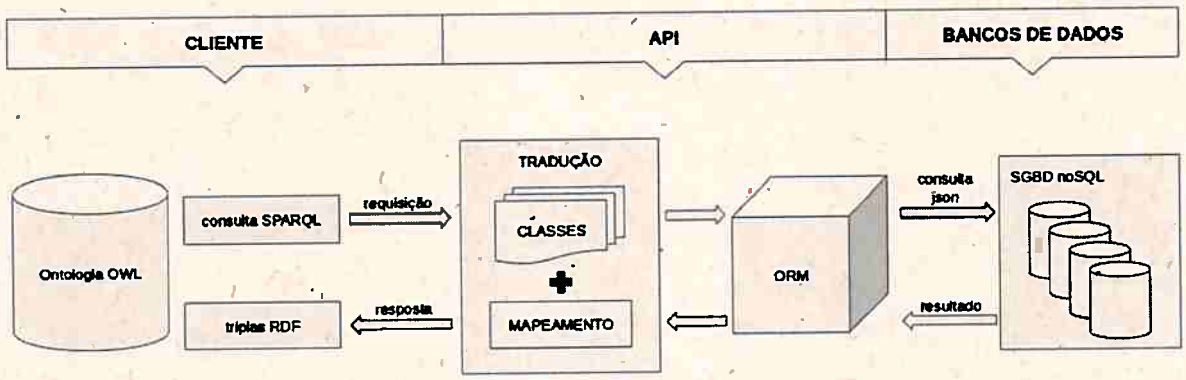


Figura 4.1: *Arquitetura do projeto OntoMongo*

4.3.1 Ontologia OWL

Utilizamos uma ontologia simples (Figura 4.2) que representa conhecimento acerca de Pesquisadores e Publicações registradas na Plataforma Lattes. A ontologia é um cliente da aplicação, e serve como um conjunto de definições formais do domínio que podem ser usadas para responder a questões de competência que podem ser expressas em consultas SPARQL. A ontologia foi desenvolvida em conjunto com Agena (2017).

[1] O projeto é de código aberto: http://github.com/thdaraujo/onto-mongo

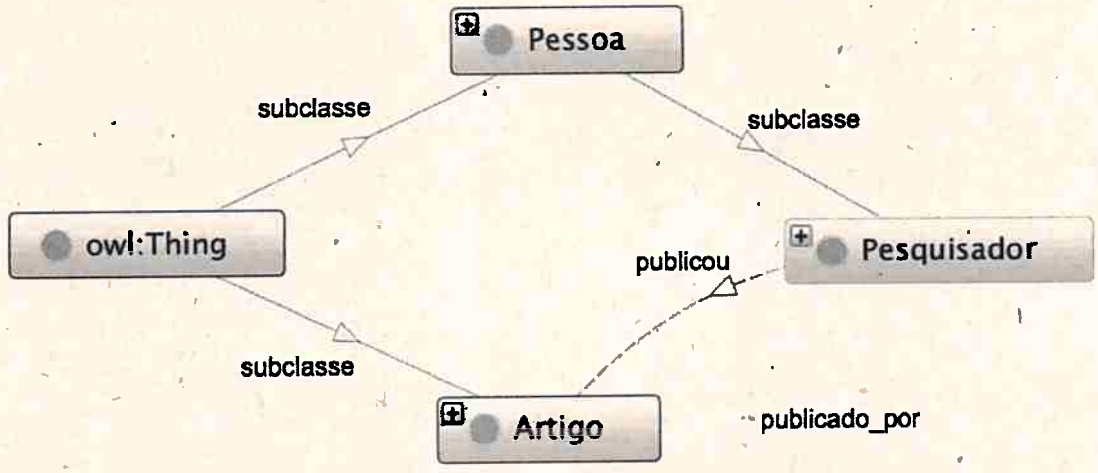


Figura 4.2: *Ontologia basic-lattes*

4.3.2 Base de dados NoSQL

Os dados a respeito dos pesquisadores e publicações científicas foram salvos em uma base de dados NoSQL orientada a documentos (MongoDB). Os dados se referem à produção bibliográfica desses pesquisadores registrados na Plataforma Lattes, e essas informações foram extraídas da plataforma previamente usando a ferramenta *scriptLattes*. Esta ferramenta é capaz de fazer a mineração dos currículos e a exportação de informações detalhadas sobre sua produção científica em formato XML.

Depois disso, fizemos a conversão dos arquivos em documentos JSON pois utilizamos o MongoDB, e selecionamos campos de interesse, como nome, área de conhecimento, e informações sobre as publicações, como coautores, veículo de publicação, local, data, tipo de publicação e título. Esses documentos então foram armazenados em uma coleção numa base de dados NoSQL.

4.3.3 Interface de Acesso

A Interface de Acesso é um módulo criado para transformar mensagens (consultas) SPARQL enviadas para o sistema e responder com um conjunto de triplas RDF com informações extraídas do banco de dados.

A interface foi desenvolvida utilizando a linguagem de programação *Ruby* e também utilizamos o framework web *Ruby on Rails* para montar a aplicação de testes.

A estrutura das coleções armazenadas no banco de dados é representada por classes *Ruby*. Essas classes são chamadas de **classes de modelo**, ou simplesmente de *models*. Vamos nos referir aqui a esse tipo de classe *Ruby* como *model* para maior clareza.

As duas principais *models* do projeto são a *Researcher* e a *Publication*. A primeira representa pesquisadores, e a segunda representa as informações sobre publicações de um dado pesquisador, e uma se relaciona com a outra, pois um pesquisador pode ter uma lista de publicações.

Uma classe de modelo é utilizada para manipular objetos persistidos no banco de dados. Elas

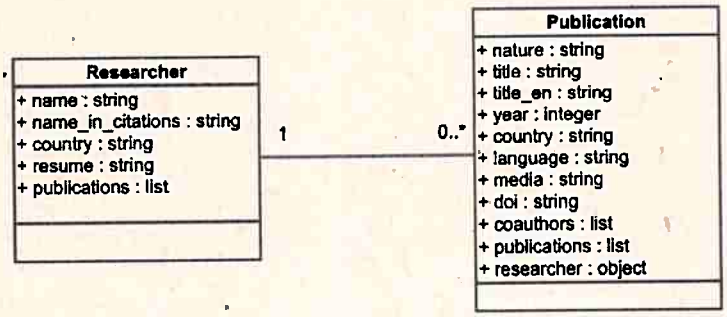


Figura 4.3: *Diagrama de classes de modelo: Researcher e Publication*

interagem com o banco de dados através de métodos de Consulta, Leitura, Atualização e Exclusão[2]. Podemos definir os atributos das instâncias dessas classes que serão persistidos no banco de dados ao anotar essas classes de modelo, adicionando, por exemplo, a anotação *name* e *country* para indicar os atributos para o nome e o país de uma dada entidade. Também podemos definir relações entre classes, indicando por exemplo que uma instância da model *Researcher* pode ter uma lista de suas publicações, que são do tipo *Publication*, salvas na base de dados. Do outro lado, anotamos a classe *Publication* para que esta se relacione (pertença a) um *Researcher*.

Quando falamos do uso de SGBDs NoSQL para persistir esses objetos, as classes de modelo servem como um esquema conceitual para o banco de dados, definindo as características, atributos e relações entre entidades persistidas em cada coleção. Por essa razão, o uso da classe de modelo simplifica a fase de mapeamento com a ontologia.

4.3.4 Mapeamento

O mapeamento é responsável por ligar as classes da Interface de Acesso com a definição da ontologia para que seja possível fazer a consulta aos dados presentes no banco de dados. Uma classe presente na ontologia é mapeada para uma classe de model que represente conceitos similares. E as relações entre classes da ontologia também são mapeadas para relações entre as classes de modelo.

Módulo OntoMap

O módulo **OntoMap** é responsável por implementar os métodos de mapeamento. Para tal, ele utiliza duas classes principais: a classe *Relation* (ver 4.1) que define uma relação entre atributos ou *models* e coleções na base de dados, e a classe *Factory* (ver 4.2) que é responsável por aceitar os métodos de mapeamento e registrar as ligações do mapeamento no módulo *OntoMap* (ver 4.3).

```
1 class Relation
2   attr_reader :class_attribute, :model_attribute, :from_model, :to_model
3
4   def initialize(class_attribute, model_attribute, from_model, to_model)
5     @class_attribute = class_attribute
6     @model_attribute = model_attribute
7     @from_model = from_model
8     @to_model = to_model
9   end
```

[2]Também conhecido como CRUD (*Create, Read, Update, Delete*).

```
10 end
```

Código 4.1: *Classe* ***Relation***

Podemos ver a classe *Relation* com métodos que ligam uma *model* a outra, e atributos a outros atributos.

```
1 class Factory < Object
2   attr_accessor :attr_mapping, :relation_mapping, :onto_class, :model_class
3
4   def initialize(o)
5     @attr_mapping = {}
6     @relation_mapping = {}
7     @onto_class = o
8   end
9
10  def model(m)
11    @model_class = m
12  end
13
14  def maps(attributes)
15    class_attribute = attributes[:from]
16    model_attribute = attributes[:to]
17    relation = attributes[:relation]
18    if relation.nil?
19      @attr_mapping[class_attribute] = model_attribute
20    else
21      from_model = model_class
22      to_model = model_class.reflect_on_association(model_attribute).class_name.
          constantize
23
24      if to_model.nil?
25        raise ArgumentError, "association not found in model", attributes[:to]
26      end
27
28      @relation_mapping[relation] = Relation.new(class_attribute, relation,
          from_model, to_model)
29    end
30  end
31 end
```

Código 4.2: *Classe* ***Factory***

A classe *Factory* é responsável por criar os mapeamentos e instanciar as ligaçoes entre conceitos da ontologia e classes de modelo. O métodò *model* recebe como parâmetro uma *model* que será ligada a uma classe da ontologia.

Já o método *maps* recebe um dicionário de parâmetros. Pode receber o atributo *:from* e *:to*, que mapeia uma propriedade da classe da ontologia com um atributo de classe da *model*. Também pode receber um atributo *:relation*, que mapeia uma relação de classes contidas na ontologia para uma relação entre *models* ou entre atributos da *model*.

A *factory*, então, registra esses mapeamentos no módulo *OntoMap*, que utiliza essas informações para traduzir as consultas SPARQL e transformar os dados de uma representação a outra.

```
1 module OntoMap
2   def self.mapping(onto_class, &block)
3     factory = Factory.new(onto_class)
4     factory.instance_eval(&block)
5     register(factory)
6   end
7
8   def self.register(factory)
9     self.registry[factory.onto_class]         = factory
10    self.class_mapping[factory.onto_class]    = factory.model_class
11    self.model_mapping[factory.model_class]   = factory.onto_class
12  end
13
14  def self.registry
15    @@registry ||= {}
16  end
17
18  def self.class_mapping
19    @@class_mapping ||= {}
20  end
21
22  def self.model_mapping
23    @@model_mapping ||= {}
24  end
25
26  def self.model_for(k)
27    self.class_mapping[k]
28  end
29
30  def self.class_for(k)
31    self.model_mapping[k]
32  end
33
34  def self.attributes_for(onto_class)
35    self.registry[onto_class].attr_mapping
36  end
37
38  def self.relations_for(onto_class)
39    self.registry[onto_class].relation_mapping
40  end
41
42  def self.included(base)
43    base.extend(ClassMethods)
44  end
45
46  module ClassMethods
47    attr_reader :onto_query
48
49    def query(sparql)
50      @onto_query = OntoQuery.new(sparql)
51      ...
52    end
53 end
```

Código 4.3: *Trecho de código do módulo* ***OntoMap***

O módulo *OntoMap* é responsável por receber os mapeamentos da *factory* e registrá-los para uso posterior. Ele foi criado como uma Linguagem de Domínio Específico para mapeamentos. Isso foi feito para facilitar a definição dos mapeamentos, para que pudesse ficar bastante clara a intenção do usuário. E que também fosse fácil fazer manutenção desses mapeamentos.

Podemos ver um exempo de como pode ser feito um mapeamento a seguir.

Criando um Mapeamento

Para se criar um novo mapeamento, devemos usar alguns métodos e parâmetros indicando de uma forma declarativa a equivalência entre a ontologia e as classes de modelo, formando um mapeamento entre a estrutura encontrada na ontologia com a estrutura das coleções presentes no banco de dados, passando pelas classes da Interface de Acesso.

O mapeamento é feito utilizando-se um trecho de código de configuração (*script*) salvo na pasta *initialize* do projeto *Rails* e é executado na inicialização do programa. Neste código é definido o mapeamento para cada classe da ontologia que se desejar mapear, dizendo qual classe de modelo deve representar qual classe da ontologia, e quais atributos da model são equivalentes aos atributos dessa classe da ontologia, e quais relações entre classes devem ser mapeadas para relações presentes nos documentos salvos no banco de dados. Também podemos dizer o caminho do arquivo OWL que contenha a definição da ontologia para uso, se necessário.

Considere o mapeamento da classe *Pesquisador*:

```
1 OntoMap.mapping 'Pesquisador' do
2   model Researcher
3   maps from: 'nome', to: :name
4   maps from: 'pais', to: :country
5   maps from: 'nome_em_citacoes', to: :name_in_citations
6   maps relation: 'publicou', to: :publications
7 end
```

Código 4.4: *Mapeamento da classe* ***Pesquisador*** *para a model* ***Researcher***

Na linha 1, é chamada a função *mapping* do módulo *OntoMap* recebendo como parâmetro a classe **Pesquisador** presente na ontologia, e um bloco de código com algumas definições.[3]

Na linha 2, definimos que a classe de modelo **Researcher** é equivalente à classe **Pesquisador** da ontologia.

Nas linhas de 3 a 5, mapeamos alguns atributos da classe de modelo para atributos da classe na ontologia. A função *maps* mapeia alguns atributos da classe de modelo com atributos da classe na ontologia, e recebe os parâmetros *from*, que se refere a alguma relação ou propriedade que pertence à clase da ontolgia, e o parâmetro *to*, que define qual atributo de uma instância de *Researcher* ou sua relação com outra classe é seu equivalente. Na linha 3, há o mapeamento da propriedade *nome* e o atributo *name*. Na linha 6, a relação *publicou* que existe entre as classes *Researcher* e *Publication* é mapeada para o atributo *publications* da classe de modelo, que é uma relação com outra classe, e

[3] Um bloco (*block*) é uma função anônima similar a um fechamento (*closure*) em outras linguagens.

que representa o conjunto de publicações de um dado pesquisador, podendo ser nenhuma, uma ou mais publicações, dependendo do que estiver salvo na base de dados.

Do outro lado, também podemos mapear a classe *Publicacao* e a model *Publication* de maneira semelhante:

```
1 OntoMap.mapping 'Publicacao' do
2   model Publication
3   maps from: 'natureza', to: :nature
4   maps from: 'titulo', to: :title
5   maps from: 'titulo_em_ingles', to: :title_en
6   maps from: 'ano', to: :year
7   maps from: 'pais', to: :country
8   maps from: 'idioma', to: language
9   maps from: 'veiculo', to: media
10  maps from: 'doi', to: doi
11 end
```

Código 4.5: *Mapeamento da classe publication para a model Publication*

Podemos notar que este também segue o mesmo padrão da definição da classe Pesquisador, contendo os mapeamentos entre atributos da model e informações da classe da ontologia

O mapeamento permite a tradução de consultas SPARQL para consultas ao banco de dados NoSQL sem que haja a necessidade de se fazer um mapeamento direto com a estrutura das coleções presentes no banco de dados. Outros trabalhos encontrados na literatura, quando tratam de acesso a dados via OBDA para SGBDs relacionais, propõem um mapeamento direto entre classes da ontologia e consultas SQL, e essas consultas podem ser bastante complexas. E quem escreve essas consultas necessita conhecer a estura de tabelas do banco de dados.

Nossa proposta utiliza uma camada conceitual intermediária servindo de ligação entre os conceitos da ontologia e as coleções de dados, e essa camada conceitual é descrita em uma linguagem orientada a objetos e utiliza um padrão arquitetural bastante utilizado na indústria de software.

Isto simplifica bastante a construção do mapeamento e traz maior clareza nas definições. A principal vantagem desse método é a possibilidade de reutilização da camada conceitual para outros tipos de SGBDs NoSQL, bastando construir um tradutor para a linguagem de consultas desejada. E mais: com nenhum esforço, podemos também utilizar essa mesma camada conceitual para acessar dados de uma base de dados relacional que aceite consultas SQL tradicionais. Essa abordagem para bancos de dados relacionais e consultas SQL foi explorada por Agena (2017) que utilizou o método aqui proposto e o aplicou a um SGBD relacional.

O mapeamento tem algumas limitações, pois ainda não é possível expressar axiomas de uma ontologia e restrições de termos e de significação impostas no domínio que são suportadas pela ontologia OWL mas ainda não são suportados pelo método aqui proposto.

4.3.5 Tradutor

O tradutor é responsável por transformar a consulta SPARQL de entrada em uma consulta NoSQL para um SGBD específico utilizando o mapeamento. A consulta SPARQL é primeiro transformada em uma expressão *SPARQL Syntax Expression* (S-Expression), através da biblioteca *sxp* library[4], pois isto facilita a tradução ao separar a consulta em suas partes principais.

[4]Disponível em: https://github.com/dryruby/sxp.rb

A partir da expressão, é gerado um grafo representando as triplas RDF, filtros, agrupamentos e variáveis que deverão ser retornadas pela consulta. O gerador do grafo foi desenvolvido por Agena (2017).

O tradutor pode possuir duas estratégias: a primeira traduz a consulta usando métodos de recuperação de dados das classes de modelo (através de um ORM), e cada uma das operações é executada como chamadas de métodos e operações com essas classes. A outra estratégia traduz as operações em uma consulta na própria linguagem usada pelo banco de dados que é então executada no próprio SGBD. No caso do tradutor para MongoDB, ela é transformada em uma consulta em formato JSON que é executada no banco de dados.

A primeira estratégia é bem mais simples e flexível, pois depende apenas de funções de seleção comumente presentes nos ORMs, e a consulta é transformada em chamadas de métodos nas classes de modelo, e aproveitamos métodos de filtro presentes, por exemplo, no Mongoid, que dá suporte a métodos como o *WHERE*, que recebe um conjunto de filtros que devem ser aplicados a uma lista de objetos. Os objetos retornados também podem ser processados e filtrados através da função *map/reduce* encontrada na biblioteca de listas e coleções da própria linguagem *Ruby*.

A segunda estratégia é mais complexa e mais apropriada para consultas complexas em bancos de dados com grande volume, ou com vários tipos de agregações e filtros de coleções internas ou relacionadas. Com essa estratégia, as operações são todas executadas pelo próprio SGBD, o que oferece uma melhor performance. Além disso, alguns SGBDs NoSQL oferecem alta disponibilidade, espelhamento e clusterização, o que melhora ainda mais a performance das consultas em domínios com uma quantidade grande de dados armazenados.

Como cada SGBD NoSQL possui diferentes linguagens de consultas, é necessário construir um tradutor específico que gere a consulta no formato necessário para o SGBD utilizado quando se deseja utilizar a segunda estratégia de tradução direta e execução no próprio banco de dados, porém a primeira estratégia é um pouco mais genérica e flexível, pois não é preciso gerar uma consulta específica para um dado SGBD.

O tradutor só dá suporte a consultas SPARQL cujo predicado das triplas RDF na cláusula *WHERE* são relações ou atributos definidos na ontologia e também possuem filtros de igualdade. Ou seja, filtros como maior-igual (>=) não são ainda suportados.

4.3.6 Exportação para RDF

Os conjuntos de objetos recuperados pelo tradutor na consulta são transformados em triplas RDF levando-se em conta o mapeamento, e essas triplas são retornadas. Também somos capazes de inserir essas triplas RDF na ontologia OWL como um meio de populá-la. Isto permite rodar, por exemplo, um *reasoner* que gera novos fatos a partir dessas informações. A inserção das triplas RDF na ontologia foi melhor explorada por Agena (2017).

Após isso, podemos também analisar e consultar essas informações sem qualquer informação prévia da estrutura das coleções do banco de dados, inclusive pode-se também integrar diferentes bases de dados com esse método de forma transparente para o usuário.

4.4 Estudo de Caso

O método proposto foi testado a partir da construção de uma aplicação que utiliza informações persistidas em uma base NoSQL MongoDB, que é orientada a documentos. Os dados salvos foram de informações a respeito de pesquisadores e suas publicações extraídas da Plataforma Lattes, e utilizamos a ontologia *basic-lattes* para esse domínio.

Os dados foram armazenados em uma base MongoDB. Para responder a essa pergunta, implementamos um tradutor capaz de gerar a consulta necessária, também criamos o mapeamento, e partimos de uma ontologia básica sobre esse domínio (discutida na Seção 4.3.1).

O banco de dados foi populado com currículos Lattes e escolhemos alguns pesquisadores que possuiam publicações no mesmo ano para que o teste fosse mais interessante e fácil de ilustrar.

Um exemplo de coleção armazenada no banco de dados é a seguinte:

```
{
  "_id": { "$oid": '5835effcc048bd0001cdc239' },
  "country": 'Norway',
  "name": 'Kristen Nygaard',
  "name_in_citations": 'Nygaard, K.',
  "publications": [{
    "_id": { "$oid": '5835effcc048bd0001cdc23a' },
    "country": 'Denmark',
    "language": 'EN',
    "media": 'PRINT',
    "title": 'COOL (Comprehensive Object-oriented Learning)',
    "year": 2002
  }, {
    "_id": { "$oid": '5835effcc048bd0001cdc23b' },
    "country": 'United states',
    "language": 'EN',
    "media": 'PRINT',
    "title": 'Class and Subclass Declarations',
    "coauthors": ['Ole-Johan Dahl', 'Kristen Nygaard'],
    "year": 2002
  }, {
    "_id": { "$oid": '5835effcc048bd0001cdc23c' },
    "country": 'France',
    "language": 'EN',
    "media": 'PROCEEDING',
    "title": 'Classification of actions or inheritance also for methods',
    "coauthors": ['Bent Bruun Kristensen',
                  'Ole Lehrmann Madsen',
                  'Birger M_ller-Pedersen',
                  'Kristen Nygaard'],
    "year": 1987
  }],
  "resume": 'Kristen Nygaard was a Norwegian computer scientist...'
}
```

Código 4.6: *Coleção researchers com informações que serão filtradas*

O estudo de caso gerou o artigo (Agena *et al.*, 2017) apresentado na ONTOBRÁS 2017.

A consulta é a seguinte: *quais pesquisadores publicaram duas publicações diferentes no mesmo*

ano? Podemos expressar essa pergunta como uma consulta SPARQL:

```
1 PREFIX : <http://onto-mongo/basic-lattes/#>
2 SELECT
3     ?nomePesquisador ?tituloPublicacao1 ?tituloPublicacao2 ?ano
4 WHERE
5 {
6     ?pesquisador :nome ?nomePesquisador .
7     ?pesquisador :publicou ?publication1 .
8     ?pesquisador :publicou ?publication2 .
9     ?publicacao1 :ano_publicacao ?ano .
10    ?publicacao2 :ano_publicacao ?ano .
11    ?publicacao1 :titulo ?tituloPublicacao1 .
12    ?publicacao2 :titulo ?tituloPublicacao2 .
13    FILTER
14        (?publicacao1 != ?publicacao2)
15 }
```

Código 4.7: *Consulta SPARQL*

A consulta é, então, transformada em uma *SPARQL Syntax Expression* equivalente:

```
1 (project
2   (?nomePesquisador ?tituloPublicacao1 ?tituloPublicacao2 ?ano)
3   (filter
4     (!= ?publicacao1 ?publicacao2)
5     (bgp
6       (triple ?pesquisador :nome ?nomePesquisador)
7       (triple ?pesquisador :publicou ?publicacao1)
8       (triple ?pesquisador :publicou ?publicacao2)
9       (triple ?publicacao1 :ano_publicacao ?ano)
10      (triple ?publicacao2 :ano_publicacao ?ano)
11      (triple ?publicacao1 :titulo ?tituloPublicacao1)
12      (triple ?publicacao2 :titulo ?tituloPublicacao2))))
```

Código 4.8: *Consulta SPARQL transformada em uma S-Expression*

A expressão é dividida em algumas partes para que possam ser traduzidas em uma consulta ao banco de dados, e as partes são: variáveis de saída (*outputs*), filtros (*filters*), o corpo principal da consulta (*body*), e as triplas de atributos (*attributes*) e relações (*relations*). Discutimos os componentes a seguir.

4.4.1 Corpo da consulta

O grafo da figura 4.4 é construído a partir da consulta. Pelo mapeamento, sabemos que o campo *nome* se relaciona com o atributo *name* presente na model *Researcher*. Portanto, ao gerar a consulta ao MongoDB, esse campo deve ser projetado (através do comando *project*) para que apareça como um campo JSON no documento resultante final. Esse comando é semelhante ao comando *SELECT* do SQL.

São gerados dois nós adjacentes ao primeiro para tratar da classe *Publicação*, através da relação *publicou*, que liga pesquisador e publicações. Sabemos que, pelo mapeamento, *publicou* é uma relação entre *researcher* e *publication*. Como também sabemos que *publication* é uma lista, devemos usar a operação de *unwind* em *publicacao*1 e *publicacao*2, para que o banco de dados gere como saída

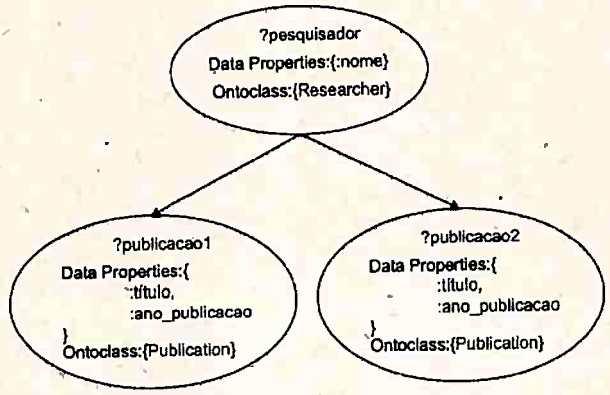


Figura 4.4: *Grafo criado a partir da consulta SPARQL*

intermediária um documento para cada item da lista de publicações. O efeito desse comando é semelhante a um *join* entre *publicacao*1 e *publicacao*2.

Também adicionamos um prefixo para identificar os campos *year* e *title* de cada variável, pois *publication* é um documento que reside dentro de *researcher*, e projetamos esses atributos para cada variável com seu prefixo. Com isso, geramos documentos com todos os dados necessários para a filtragem posterior, que discutimos a seguir.

Filtros

Analisamos os filtros que devem ser aplicados à consulta. Nas linhas 3 e 4 da expresão 4.8, temos alguns filtros que serão aplicados, e devemos levar também em conta a informação das linhas 9 e 10, pois a consulta exige que as publicações possuam o mesmo ano, já que a variável ?*ano* aparece em *publicacao*1 e *publicacao*2. Portanto, o filtro deve garantir que cada par de publicações possua valores diferentes para todos os seus campos exceto ano, que devem ser iguais. Assim, a consulta filtrará e retornará todas as publicações distintas cujo ano de publicação é o mesmo.

Para saber quais campos existem no documento e quais deles devem ser distintos, consultamos o mapeamento.

Seja T o conjunto das triplas no corpo da consulta, e I o conjunto das variáveis que precisam ter o mesmo valor. Neste caso, $I = \{?ano\}$, pois ?*publicacao*1 e ?*publicacao*2 devem ter o mesmo ano. Agora seja F o conjunto de variáveis presentes em um filtro de desigualdade. Neste caso, $F =$ { ?*publicacao*1, ?*publicacao*2 }. Para cada váriavel $x \in F$, filtramos as triplas $t \in T$ cujo sujeito $S(t) = x$ e guardamos o predicado $P(x)$ em um conjunto D. Neste caso, $D =$ { ?*tituloPublicacao*1, ?*tituloPublicacao*2, ?*ano* }. Então, construimos o filtro de igualdade para todas as variáveis em I, e um filtro de desigualdade para todas as variáveis do conjunto $D \setminus I =$ { ?*tituloPublicacao*1, ?*tituloPublicacao*2 }, ou seja, para todas as variáveis em D exceto as que estão em I.

Assim, adicionamos esses filtros na consulta para obter o conjunto de documentos desejados, e projetamos os campos desejados na saída.

4.4.2 Saída

As variáveis de saída estão na linha 2. Geramos a projeção de saída adicionando os prefixos necessários. Com os campos retornados pela consulta, ligamos as variáveis e os atributos presentes no documento usando o mapeamento. Como exemplos, podemos ver que ?*tituloPublicacao*1 obtem

seu valor do campo $\$publicaçao1.title$ que foi projetado na saída.

4.4.3 Consulta Final e Resultados

A consulta final executada no banco de dados é a seguinte:

```
1 { :$project =>
2     { :name => true,
3       'publicacao1' => '$publications',
4       'publicacao2' => '$publications' } },
5   { :$unwind => '$publicacao1' },
6   { :$unwind => '$publicacao2' },
7   { :$project =>
8     { :name => true,
9       :publicacao1 => true,
10      :publicacao2 => true,
11      'filter_1' =>
12      { '$and' =>
13        [{ :$eq => ['$publicacao1.year', '$publicacao2.year'] },
14         { :$ne => ['$publicacao1.title', '$publicacao2.title'] }] } } },
15  { :$match => { 'filter_1' => true } },
16  { :$project =>
17    { 'nomePesquisador' => '$name',
18      'tituloPublicacao1' => '$publicacao1.title',
19      'tituloPublicacao2' => '$publicacao2.title',
20      'ano' => '$publicacao1.year' } }
```

Código 4.9: *Resulting query*

Ao consultar os dados de exemplo, recebemos como retorno um documento JSON contendo uma lista de publicações únicas que foram publicadas no mesmo ano e que possuem o mesmo autor, que responde à questão de competência inicial:

```
1 [{ '_id' => BSON::ObjectId('5835effcc048bd0001cdc239'),
2    'researcherName' => 'Kristen Nygaard',
3    'publicationTitle1' => 'COOL (Comprehensive Object-oriented Learning)',
4    'publicationTitle2' => 'Class and Subclass Declarations',
5    'year' => 2002 }]
```

Código 4.10: *Lista dos documentos JSON que respondem à consulta*

Esse resultado é exportado ao ser transformado em triplas e inserido em um grafo RDF, que é então retornado como resultado. Com esse resultado, podemos adicionar essas informações na ontologia como novo conhecimento. Ao fazer isso, é possível fazer outras consultas mais complexas ou rodar inferência lógica e derivar novas conclusões.

4.5 Relatórios

O segundo estudo de caso foi desenvolvido para avaliar o método proposto através da geração de alguns relatórios de produção científica dos pesquisadores com currículo na Plataforma Lattes.

Essa foi uma forma de avaliar a eficácia e aplicabilidade do método, pois é importante verificar se este é de fácil utilização e pode ser realmente aplicado a alguns outros cenários. Também é

importante avaliar se a construção do mapeamento é algo simples, e se a tradução das consultas é eficaz.

Os dados, assim como no primeiro estudo de caso, foram armazenados em uma base de dados MongoDB.

Os relatórios criados foram os seguintes:

1. Lista de Pesquisadores
2. Lista de Publicações
3. Relatório de Publicações por País
4. Mapa de Publicações
5. Relatório de Publicações por Ano

4.5.1 Lista de Pesquisadores

O primeiro relatório lista todos os pesquisadores com currículo persistido na base de dados. É um relatório simples e que pode ser visto na figura 4.5

A consulta retorna uma lista de todos os documentos presentes na coleção *Researchers*, projetando apenas nome, id (no banco de dados) e a quantidade de publicações desse pesquisador. Essa quantidade é calculada através da contagem de itens em publicações, que é uma lista interna em cada documento JSON de pesquisador.

A consulta SPARQL pode ser vista na lista 4.11).

```
1 PREFIX : <http://onto-mongo/basic-lattes/#>
2 SELECT
3     ?nomePesquisador ?nomeEmCitacoes
4 WHERE
5 {
6     ?pesquisador :nome ?nomePesquisador .
7     ?pesquisador :nome_em_citacoes ?nomeEmCitacoes .
8 }
```

Código 4.11: *Consulta SPARQL para retornar todos os Pesquisadores*

4.5.2 Mapa de Publicações

O relatório de publicações por país também possui um mapa das publicações no mundo. A consulta SPARQL pode ser vista em 4.12).

```
1 PREFIX : <http://onto-mongo/basic-lattes/#>
2 SELECT
3     ?pais (COUNT(?tituloPublicacao) AS ?qtd_publicacoes)
4 WHERE
5 {
6     ?pesquisador :publicou ?publicacao .
7     ?publicacao :titulo ?tituloPublicacao .
8     ?publicacao :pais ?publicacao .
9 } GROUP BY ?pais
```

Código 4.12: *Consulta SPARQL para retornar todas as Publicações por país*

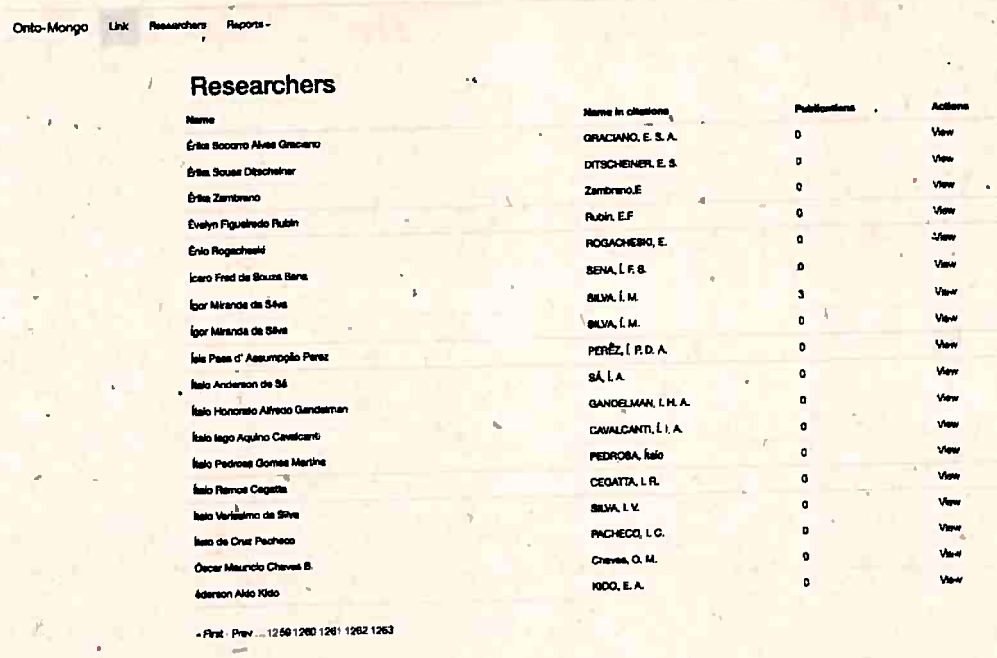


Figura 4.5: *Lista de Pesquisadores*

Como o método ainda não gera uma consulta que executa o comando *GROUP BY* diretamente no MongoDB como um *aggregate*, os dados foram agregados através da função *group_by* do *Ruby*.

Também foi necessário tratar o nome do país e o código, pois muitas publicações estavam com o nome do país em inglês e alguns estavam com nomes abreviados.

```
1 def self.publications_by_country(sparql)
2   Researcher.query(sparql)
3             .map{|i| { country: Researcher.country_to_code(i["country"]) } }
4             .group_by{|i| i[:country]}
5             .map{|k, v| [k, v.size] }
6 end
```

Esses dados são exportados como JSON e alimentam uma biblioteca gráfica javascript que gera um mapa e a intensidade das cores revela o peso por país. Pode ser visto na figura 4.6.

4.5.3 Relatório de Publicações por Ano

O relatório de produtividade por ano agrega as publicações por ano. A consulta SPARQL pode ser vista em 4.13).

```
1 PREFIX : <http://onto-mongo/basic-lattes/#>
2 SELECT
3     ?ano (COUNT(?tituloPublicacao) AS ?qtd_publicacoes)
4 WHERE
5 {
6     ?pesquisador :publicou ?publicacao .
7     ?publicacao :titulo ?tituloPublicacao .
8     ?publicacao :ano ?ano .
9 } GROUP BY ?ano
```

Código 4.13: *Consulta SPARQL para retornar todas as Publicações por ano*

Figura 4.6: *Mapa de Publicações*

Como o método ainda não gera uma consulta que executa o comando *GROUP BY* diretamente no MongoDB como um *aggregate*, os dados também foram agregados na aplicação.

```
1 def self.publications_by_year(sparql)
2   Researcher.query(sparql)
3     .map{|i| {year: i["year"].to_i } }
4     .group_by{|i| i[:year]}.
5     .map{|k, v| [k.to_s, v.size] }
6 end
```

Os dados de produtividade são retornados e é gerafo um gráfico de barras e linhas. Os dados são exportados como JSON e alimentam uma biblioteca gráfica. Pode ser visto na figura 4.7.

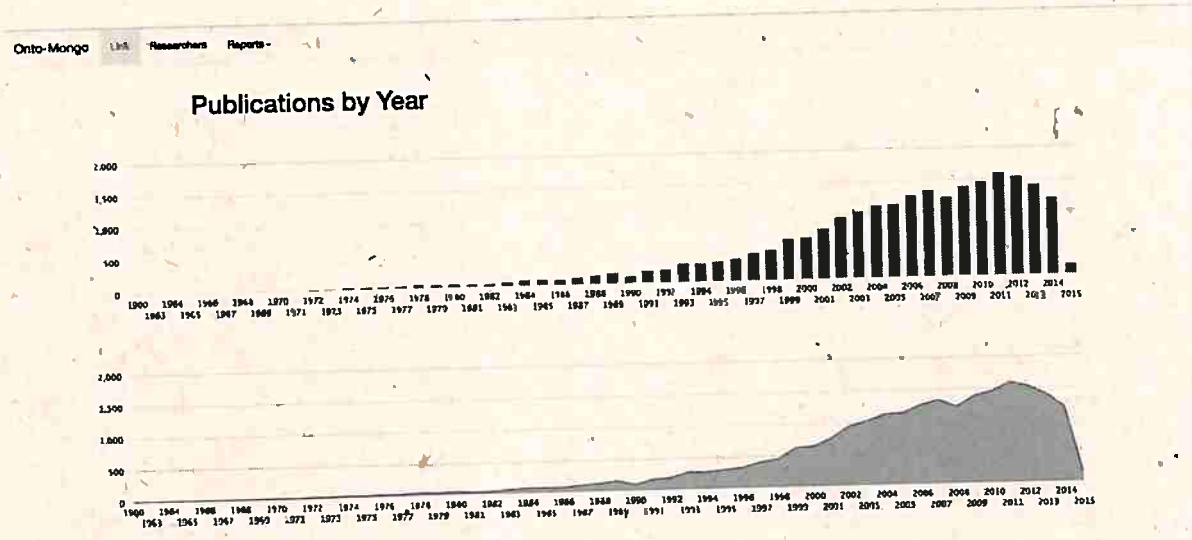


Figura 4.7: *Relatório de Publicações por Ano*

Capítulo 5

Conclusões

Uma ontologia é uma forma mais expressiva de representação de conhecimento, e com ela, podemos gerar várias informações úteis sobre o domínio estudado. O uso dessa ferramenta no estudo das redes de colaboração traz uma melhor utilização do conhecimento e uma melhor exploração do domínio, auxiliando na descoberta de novos significados. Também simplifica a tarefa de exploração desses dados.

O método proposta de acesso a base de dados NoSQL utilizando ontologias oferece diversas vantagens para a comunidade científica. A principal é a forma diferente e mais simples de se criar o mapeamento para as coleções de dados, tornando este processo bem mais simples no desenvolvimento de sistemas OBDA.

Por não ser feito um mapeamento direto com a linguagem de consulta do banco de dados, diferente do que é feito em outras abordagens discutidas no capítulo 3, nossa proposta também diminui o esforço necessário para a construção e manutenção do mapeamento. Não é necessário fazer um mapeamento direto entre classes e consultas, como algumas propostas da literatura que adicionam consultas SQL no mapeamento. E nem seria possível fazer esse mapeamento direto, pois diferente dos bancos de dados relacionais SQL, os SGBDs NoSQL diferem bastante no tipo de linguagem de acesso a dados, portanto a proposta do uso de uma camada intermediária e mais genérica torna a solução muito mais flexível.

Colocar ao lado do mapeamento uma camada de representação conceitual intermediária que opera sobre classes em uma linguagem orientada a objetos dá ao nosso método maior generalidade e flexibilidade no acesso aos dados.

Se for necessário dar suporte a um novo tipo ou grupo de SGBD NoSQL com uma linguagem de acesso diferente, bastaria ser feita a construção de um tradutor para esse nova linguagem. E, dependendo do caso, podemos reutilizar as classes de modelo e o mapeamento, algo que demonstra a flexibilidade da solução.

5.1 Limitações

O mapeamento possui limitações pois não dá suporte ainda a todas as restrições e propriedades que podem ser expressas em uma ontologia OWL, como axiomas e restrições de termos e outras características do domínio que podem ser formalizadas. Como trabalho futuro, desejamos criar uma forma mais poderosa de mapeamento que possa dar conta dessas ideias.

O método proposto possui algumas limitações nos tipos de consultas que ele é capaz de traduzir

do SPARQL. Agrupamentos não são ainda suportados, assim como operações como *COUNT()*, *MIN()*, *MAX()*, *SUM()*, *AVG()* e *GROUP BY*.

O método também não trata consultas a coleções distintas, ou seja, o método só é capaz de acessar uma coleção do banco de dados por vez. É possível acessar documentos internos em outros documentos e fazer operações de *unwind*, mas ainda não é possível, por exemplo, fazer uma operação parecida com um *JOIN* entre coleções. O próprio MongoDB tem limitações quanto a isso, mas outros tipos de bancos de dados NoSQL ou SQL poderiam utilizar esse tipo de operação, então é um objetivo futuro dar algum tipo de suporte a esse tipo de operação quando a consulta é executada diretamente no SGBD.

Outra limitação é a necessidade do usuário saber *Ruby* para poder implementar o mapeamento. Apesar dessa definição ser relativamente simples, exige que o usuário saiba um pouco de linguagens de programação. Nossa intenção é criar uma forma automática de mapeamento que simplifique esse processo e possa ser usada por usuários sem experiência com a linguagem.

5.2 Trabalhos Futuros

Como trabalho futuro, desejamos aplicar nosso método em outros tipos de SGBDs NoSQL, como o Neo4j, que seria bastante interessante para se fazer análises apuradas da rede de colaboração da Plataforma Lattes, e também o Cassandra, para gerar relatórios complexos e visualizações de dados. Para isto, o primeiro passo é tornar o tradutor mais flexível e capaz de dar suporte a operações de mais alto nível, como agregações mais complicadas e *joins*.

Um objetivo prático seria facilitar a criação de tradutores para novos SGBDs criando um gerador de código que especifica os métodos que devem ser definidos ou implementados para a linguagem de consulta e o SGBD desejado, e um usuário intermediário seria capaz de implementar esses métodos para esse SGBD usando fragmentos dessa linguagem de consulta ou outras operações com os objetos retornados (como *map/reduce*).

Outro interesse é facilitar ainda mais a criação do mapeamento, que poderia ser feito de forma automática a partir de um arquivo com uma ontologia OWL e através de um gerador de código. Esse código poderia ser gerado sem que o usuário precisasse conhecer a linguagem *Ruby*, talvez via alguma interface gráfica que mostrasse de um lado as classes e relações presentes na ontologia, e de outro as classes de modelo ou até as coleções de dados presentes no banco de dados e o usuário faria as ligações e configurações necessárias no mapeamento.

Também transformaremos a interface de acesso e o método proposto em uma biblioteca (*gem*) para *Ruby*, para que possa ser utilizada em outros projetos por outros pesquisadores.

Um desses projetos futuros deverá utilizar o método para consulta e extração de informações da Plataforma Lattes para explorar as redes de coautoria. Desejamos expandir trabalhos de análise dessas redes e criar um sistema de recomendação utilizando Predição de Ligações e algoritmos de Aprendizado de Máquina eriquecidos com conhecimento presente em uma ontologia OWL e verificar se esse enriquecimento aumenta a eficácia desse sistema.

Referências Bibliográficas

Agena(2017) Bárbara Agena. OntoMongo - Acesso a Dados Baseado em Ontologias para sistemas NoSQL. Dissertação de Mestrado, Universidade de São Paulo. Citado na pág. 3, 27, 33, 34

Agena *et al.*(2017) Bárbara Agena, Thiago Araujo, Kelly Braghetto e Renata Wassermann. OntoMongo – Ontology-Based Data Access for NoSQL. Em *Anais do Seminário de Pesquisa em Ontologias do Brasil (ONTOBRAS)*. Citado na pág. 3, 35

Baader *et al.*(2003) Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi e Peter Patel-Schneider. The description logic handbook: Theory, implementations and applications. *Cambridge U. Press, Cambridge, England.* Citado na pág. 5

Bagosi *et al.*(2014) Timea Bagosi, Diego Calvanese, Josef Hardi, Sarah Komla-Ebri, Davide Lanti, Martin Rezk, Mariano Rodríguez-Muro, Mindaugas Slusnys e Guohui Xiao. The Ontop Framework for Ontology Based Data Access. Em *8th Chinese Semantic Web and Web Science Conference*, páginas 67–77. Citado na pág. 19, 21

Bonifacio(2002) Ailton Sergio Bonifacio. Ontologias e consulta semântica: Uma aplicaçao ao caso Lattes. Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul. Citado na pág. 21

Botoeva *et al.*(2016a) Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Martin Rezk e Guohui Xiao. OBDA Beyond Relational DBs : A Study for MongoDB. *Proc. of the 29th Int. Workshop on Description Logics (DL 2016)*, 1. Citado na pág. 6, 21

Botoeva *et al.*(2016b) Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Martin Rezk e Guohui Xiao. A Formal Presentation of MongoDB (Extended Version). *CoRR Technical Report abs/1603.09291, arXiv.org e-Print archive, 2016.* URL http://arxiv.org/abs/1603.09291. Citado na pág. 21

Calvanese *et al.*(2007) Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi e Riccardo Rosati. Ontology-based database access. Em *Proc. of the 15th Italian Symposium on Advanced Database Systems(SEBD'07)*, páginas 324–331. ISBN 978-88-902981-0-3. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.2876&rep=rep1&type=pdf. Citado na pág. 2, 6

Calvanese *et al.*(2017) Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro e Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487. Citado na pág. 19

Castano(2008) André Casado Castano. *Populando ontologias através de informações em HTML-O caso do Currículo Lattes.* Tese de Doutorado, Universidade de São Paulo. Citado na pág. 21

Cervantes(2014) Evelyn Perez Cervantes. Análise de Redes de Colaboração Científica: Uma Abordagem Baseada em Grafos Relacionais com Atributos. Dissertação de Mestrado, Universidade de São Paulo. Citado na pág. 22, 26

Chen *et al.*(2014) Min Chen, Shiwen Mao e Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209. Citado na pág. 3

Costa e Yamate(2009) Anauê Costa e Fabio Yamate. Semantic Lattes: uma ferramenta de consulta de informações acadêmicas da base Lattes baseada em ontologias, 2009. Trabalho de Conclusão de Curso, Universidade de São Paulo. Citado na pág. 22

DeCandia *et al.*(2007) Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall e Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. *SIGOPS Oper. Syst. Rev.*, 41 (6):205–220. ISSN 0163-5980. doi: 10.1145/1323293.1294281. URL http://doi.acm.org/10.1145/1323293.1294281. Citado na pág. 9

Fowler(2010) Martin Fowler. *Domain-Specific Languages (Addison-Wesley Signature Series (Fowler))*. Addison-Wesley Professional. ISBN 978-0321712943. Citado na pág. 13

Galego(2013) Eduardo Ferreira Galego. Extração e Consulta de Informações do Currículo Lattes baseada em Ontologias. Dissertação de Mestrado, Universidade de São Paulo. Citado na pág. 22

Ginsberg *et al.*(2009) Jeremy Ginsberg, Matthew H Mohebbi, Rajan S Patel, Lynnette Brammer, Mark S Smolinski e Larry Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014. Citado na pág. 1

Grüninger e Fox(1995) Michael Grüninger e Mark S Fox. Methodology for the design and evaluation of ontologies. Citado na pág. 5

Guarino *et al.*(2009) Nicola Guarino, Daniel Oberle e Steffen Staab. *Handbook on Ontologies*, chapter What Is an Ontology?, páginas 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-92673-3. doi: 10.1007/978-3-540-92673-3_0. URL http://dx.doi.org/10.1007/978-3-540-92673-3_0. Citado na pág. 5

Haerder e Reuter(1983) Theo Haerder e Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317. Citado na pág. 10

Horridge e Bechhofer(2011) Matthew Horridge e Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semant. web*, 2(1):11–21. ISSN 1570-0844. URL http://dl.acm.org/citation.cfm?id=2019470.2019471. Citado na pág. 6

Manyika *et al.*(2011) James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh e Angela H Byers. Big data: The next frontier for innovation, competition, and productivity. Citado na pág. 1

Mena-Chalco e Junior(2009) Jesús Pascual Mena-Chalco e Roberto Marcondes Cesar Junior. scriptLattes: an open-source knowledge extraction system from the Lattes platform. *Journal of the Brazilian Computer Society*, 15(4):31–39. ISSN 0104-6500. doi: 10.1007/BF03194511. URL http://dx.doi.org/10.1007/BF03194511. Citado na pág. 22

Michel *et al.*(2016) Franck Michel, Catherine Faron-Zucker e Johan Montagnat. A mapping-based method to query MongoDB documents with SPARQL. Em *International Conference on Database and Expert Systems Applications*, páginas 52–67. Springer. Citado na pág. 21

Nayak *et al.*(2013) Ameya Nayak, Anil Poriya e Dikshay Poojary. Types of NoSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, 5(4):16–19. Citado na pág. 3, 7, 10

Prud'hommeaux e Bertails(2008) Eric Prud'hommeaux e Alexandre Bertails. A Mapping of SPARQL onto Conventional SQL. páginas 1–9. URL https://www.w3.org/2008/07/MappingRules/SpringerTemplate/rdb2rdf.pdf. Citado na pág. 20