DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIAS
MESTRADO EM ENGENHARIA INFORMÁTICA
E DE TELECOMUNICAÇÕES

UNIVERSIDADE AUTÓNOMA DE LISBOA
"LUÍS DE CAMÕES"

GERAÇÃO SEMI-AUTOMÁTICA DE MAPEAMENTOS DE VOCABULÁRIOS ENTRE DATASETS DA WEB DE DADOS USANDO SPARQL

Dissertação para a obtenção do grau de
Mestre em Engenharia Informática e de Telecomunicações

Autora: Vera Sofia Gomes Pinheiro
Orientadora: Prof. Doutora Valéria Magalhães Pequeno
Número da candidata: 20140863

Julho de 2020
Lisboa

(Página em Branco)

Agradecimentos

Ao longo do desenvolvimento deste trabalho nem sempre as coisas correram como esperado, sendo que agradeço a todas as pessoas que de uma maneira ou outra nunca me deixaram desistir e sempre me motivaram para a continuação desta caminhada. Deste modo, agradeço:

À minha família e ao meu namorado, Bruno Martins, pelo apoio incondicional, pela paciência e pelo carinho que me proporcionaram ao longo deste percurso, permitindo assim o término de mais uma etapa na minha vida.

Aos meus amigos por nunca me terem deixado baixar os braços e por terem sempre uma palavra amiga e motivacional que me ajudou sempre a continuar.

À minha orientadora, a professora Doutora Valéria Pequeno, pela paciência e orientação prestada. Sem a sua orientação, persistência e sugestões não era possível concluir esta etapa.

Aos professores do corpo docente da UAL por me permitirem crescer e pelos ensinamentos que me prestaram.

Resumo

Atualmente, a *web* tem evoluído de um espaço global de documentos interligados (*Web de Documentos*) para um espaço global de dados vinculados (*Web de Dados*), de modo a que tanto os humanos como os agentes computacionais consigam compreender e extrair informações úteis desses dados. No entanto, para que seja possível possuir um dia uma *Web de Dados*, é necessário, em primeiro lugar, dar semântica aos dados. Neste sentido, emergiu uma nova abordagem, designada por *Web Semântica*, cujo principal objetivo é facilitar a interpretação e integração de dados na *web*.

Na *Web Semântica*, utilizamos habitualmente as ontologias para descrever formalmente a semântica dos dados. No entanto, à medida que o número de ontologias vai aumentado, é bastante comum existir heterogeneidade entre elas, já que cada ontologia pode usar vocabulários diferentes para representar dados acerca de uma mesma área de conhecimento. Esta heterogeneidade impossibilita a recuperação de informações por parte dos agentes computacionais sem que haja intervenção humana.

Para fazer face aos problemas relacionados com a heterogeneidade, é muito comum efetuar-se mapeamentos entre as ontologias. Existem diversas linguagens no mercado que permitem traduzir e mapear ontologias, dentro as quais destacamos a linguagem *SPARQL Protocol and RDF Query Language (SPARQL 1.1)*[1] e *R2R*[2].

Neste trabalho decidimos usar o *SPARQL 1.1* como linguagem de mapeamento entre ontologias, pois é um padrão recomendado pelo *World Wide Web Consortium (W3C)* e amplamente utilizado pela comunidade. Como esta linguagem é complexa e necessita que o utilizador tenha experiência na definição e criação de mapeamentos, propomos uma ferramenta, chamada *SPARQL Mapping with Assertions (SMA)*, que visa auxiliar os utilizadores no processo de geração de mapeamentos *SPARQL 1.1* entre ontologias.

A ferramenta *SMA* é constituída por quatro partes:

(1) configuração inicial das ontologias: o utilizador indica quais as ontologias que deseja mapear, assim como a linguagem em que os ficheiros das mesmas estão escritos;

(2) criação das Assertivas de Mapeamento (*AMs*): através da interface gráfica, o utilizador especifica quais os mapeamentos que deseja definir, incluindo possíveis transformações ou filtros que sejam necessários aplicar aos dados;

[1] SPARQL 1.1 https://www.w3.org/TR/sparql11-overview/
[2] R2R http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/spec/

(3) configuração para a geração de mapeamentos: o utilizador introduz o ficheiro com o *Dataset* da ontologia fonte e identifica a linguagem de serialização em que o mesmo está escrito. Além disso, também escolhe qual a linguagem de serialização que deseja aquando da geração de triplos;

(4) geração de triplos através dos mapeamentos *SPARQL 1.1*: a partir dos pontos anteriores, a nossa ferramenta irá retornar um ficheiro com todos os resultados na linguagem de serialização escolhida.

A nossa ferramenta permite ainda exportar todos os mapeamentos criados, quer seja através das linguagens formais (assertivas ou regras de mapeamentos) ou dos mapeamentos *SPARQL 1.1*.

Palavras-chave: Web Semântica; Padrões de Mapeamento; Mapeamentos entre Ontologias; *SPARQL*; Ontologias.

Abstract

Nowadays, the web has evolved from a global space of interconnected documents (*Web of Documents*) to a global space of Linked Data (*Web of Data*), so that humans and computational agents can understand and extract useful information from that data. However, in order to one day have a *Web of Data*, we must first give semantics to the data. In this sense, a new approach has emerged, called *Semantic Web*, whose main objective is to facilitate the interpretation and integration of data on the web.

In the *Semantic Web*, we usually use ontologies to formally describe the semantics of the data. However, as the number of ontologies increases, it is quite common to have heterogeneity between them, since each ontology can use different vocabularies to represent data about the same area of knowledge. This heterogeneity makes it impossible for computational agents to recover information without human intervention.

To resolve problems of data heterogeneity, it is very common to make mapping between ontologies. There are several languages on the market that allow to translate and map ontologies, within which we highlight the *SPARQL Protocol and RDF Query Language (SPARQL 1.1)*[3] and *R2R*[4] language.

In this work, we decided to use *SPARQL 1.1* as a mapping language between ontologies, as it's a standard recommended by *World Wide Web Consortium (W3C)* and widely used by the community. As this language is complex and requires the user to have experience in defining and creating mappings, we propose a tool, called *SPARQL Mapping with Assertions* (*SMA*), which aims to assist users in the process of generating *SPARQL 1.1* mappings between ontologies.

The *SMA* tool consists of four parts:

(1) initial configuration of ontologies: the user indicates which ontologies he wants to map, as well as the serialization language in which his files are written;

(2) creation of Mapping Assertions: through the graphical user interface, the user specifies which mappings he wants to define, including possible transformations or filters that are necessary to apply to data;

[3] SPARQL 1.1 https://www.w3.org/TR/sparql11-overview/
[4] R2R http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/spec/

(3) configuration for generating mapping: the user enters the file with the source ontology dataset and identifies the serialization language in which it is written. In addition, the user also choose which serialization language he wants when generating triples;

(4) triples generation through *SPARQL 1.1* mapping: from the previous points, our tool will return a file with all the results in the chosen serialization language.

Our tool also allows the user to export all created mappings, whether through formal languages (mapping assertions or mapping rules) or *SPARQL 1.1* mappings.

Keywords: *Semantic Web*; Mapping Patterns; Ontology Mapping; SPARQL; Ontologies.

Índice

Lista de Quadros/Gráficos

Lista de Fotografias/Ilustrações

Lista de Siglas e Acrónimos

AM	Assertiva de Mapeamento
AMs	Assertivas de Mapeamento
AMC	Assertiva de Mapeamento de Classe
AMCs	Assertivas de Mapeamento de Classe
AMD	Assertiva de Mapeamento de Tipo de Dados
AMDs	Assertivas de Mapeamento de Tipo de Dados
AMO	Assertiva de Mapeamento de Objeto
AMOs	Assertivas de Mapeamento de Objeto
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CMS	Criação de Mapeamentos SPARQL
CRM	Criação de Regras de Mapeamento
CSS	Cascade Style Sheets
DC	Dublin Core
DOM	Document Object Model
ER	Exportação de Resultados
FOAF	Friend of a Friend
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ISBN	International Standard Book Number
IRI	Internationalized Unicode-encoded
JPA	Java Persistence API
JVM	Java Virtual Machine
LDIF	Linked Data Integration Framework
MVC	Model-View-Controller
R2RML	RDB to RDF Mapping Language
RBA	R2R By Assertions
RDB	Relational Database
RDF	Resource Description Framework
RDF-S	RDF Schema

SMA	SPARQL Mapping with Assertions
SML	Sparqlification Mapping Language
SPA	Single Page Application
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SQL	Structured Query Language
Turtle	Terse RDF Triple Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	eXtensible Markup Language

Glossário

API	Facilita a integração de sistemas que possuem linguagens completamente distintas, de forma ágil e segura [1].
Framework	É um conjunto de bibliotecas ou componentes utilizados para resolver um conjunto de problemas relacionados com uma abordagem genérica, permitindo assim que os desenvolvedores se concentrem em resolver determinado problema e não em reescrever código. Por outras palavras, trata-se de uma estrutura base ou plataforma de desenvolvimento [2].
HTTP	Protocolo da camada de aplicação para a transmissão de documentos hipermédia como, por exemplo, documentos *HTML* [3].
Internet	Infraestrutura de redes que estão globalmente interconectadas e seus canais de comunicação [4].
Java	É uma linguagem de programação orientada a objetos desenvolvida nos anos 90, por uma equipa de programadores pertencentes à empresa *Sun Microsystems*, sendo a linguagem mais importante da plataforma Java pertencente agora à Oracle [5]. Diz-se orientada a objetos pois todo e qualquer valor é um objeto.
JavaScript	É uma linguagem de programação que traz melhorias à linguagem de marcação *HTML* [6]. É designada como uma linguagem interpretada ou linguagem de *script*, que está incorporada num documento *HTML*. Uma linguagem interpretada significa que não necessita de ser compilada como ocorre nas linguagens de programação convencionais, sendo que neste caso o interpretador desta linguagem é o *browser* que o utilizador estiver a usar.
Java Beans	É uma especificação que define uma *API* e expõe quais as regras de configuração e comunicação entre componentes e convenções de programação [7].
MVC	É um padrão de projeto cujo objetivo é utilizar uma solução já definida para separar partes distintas do projeto de modo a reduzir as suas dependências ao máximo [8]. Os benefícios de utilizar este padrão é o isolamento das regras de negócio, da lógica de apresentação. Os três constituintes deste padrão são: **Model**, **View** e **Controller**.

	A **View** é a interface do utilizador, sendo que sempre que haja interação (clicar num botão, por exemplo), esta chama o **Controller**. Este, por sua vez, é considerado o intermediário, pois é o único que recebe diretamente os pedidos da **View** e os encaminha para o **Model** responsável por determinada função. De seguida, o **Model** encaminha a resposta para o **Controller** e este, por sua vez, encaminha a resposta para a **View**.
SPARQL endpoint	É um ponto de acesso via *web* que utiliza o protocolo *HTTP*. Este ponto de acesso é capaz de receber e processar solicitações que usam o protocolo *SPARQL* [9]. *Virtuoso SPARQL Query Editor*[5] da *DBpedia* é um exemplo de um dos *SPARQL endpoints* existentes na *web*.
Scala	É uma linguagem de programação multiparadigmática, pois trata-se de uma linguagem de programação orientada a objetos e de uma linguagem de programação funcional. É descrita como uma linguagem de programação orientada a objetos porque todo e qualquer valor é um objeto e é também uma linguagem de programação funcional porque toda e qualquer função é um valor [10].
Servlet Container	Permite a utilização do Java para gerar dinamicamente a página *web* no lado do servidor, sendo que um servidor *web* que não contenha um *Servlet Container* apenas permite a solicitação de uma página *web* estática [11].
Spring	Fornece um modelo abrangente de programação e configuração para aplicações corporativas modernas baseadas em Java [8]. De acordo com [8], o Spring tem como principal objetivo facilitar o desenvolvimento de aplicações, através da utilização de dois conceitos: **Inversão do Controlo** e **Injeção de Dependências**. Além disso, o Spring possui diversos módulos que podem ser utilizados para satisfazer os requisitos da aplicação, tais como o *Spring Data JPA*.
Web (WWW)	Plataforma de transmissão de informação com recursos multimédia, que funciona através da estrutura física da Internet [4].
W3C	Consórcio fundado em 1994 pelo inventor da *web*, que define padrões e tem como objetivo a criação de uma *web* confiável, onde as pessoas possam assumir a autoria e responsabilidade pelas suas publicações [4].

[5] *Virtuoso SPARQL Query Editor* pode ser encontrado no seguinte link: http://dbpedia.org/sparql

Workflow	Sequência de etapas necessárias para concretizar determinada tarefa. Em português é usualmente designado por fluxo de trabalho.

1 Introdução

Atualmente, a *web* possui um imenso volume de documentos que se expande a cada dia, sendo designada por *Web de Documentos*. Esta é um espaço global de documentos interligados que estão organizados de modo a serem compreendidos apenas pelos humanos, ou seja, sem a participação humana os agentes computacionais não conseguem compreender e extrair informação útil desses dados. Deste modo, emergiu uma nova abordagem designada por *Web de Dados*.

A *Web de Dados* é uma abordagem que pretende criar um espaço global de dados vinculados onde humanos e agentes computacionais conseguem recuperar e compreender informações. Esta abordagem segue a mesma linha que já havia sido definida pelo criador original da *web*, Tim Berners-Lee, quando publicou em 1996 o artigo "*The World Wide Web: Past, Present and Future*" [12]. Nesse artigo, Berners-Lee projetou originalmente a *web* como sendo um espaço interativo de compartilhamento de informações entre humanos e agentes computacionais. Mas, para que um dia possamos vir a ter uma *Web de Dados*, é necessário em primeiro lugar dar semântica aos dados, através da *Web Semântica*.

A *Web Semântica* é uma extensão da *Web de Documentos* cujo principal objetivo é facilitar a interpretação e integração de dados na *web*. Para tal, existem dois conceitos fundamentais: *Linked Data* e Ontologias.

Linked Data, ou dados ligados, consiste em dados semanticamente relacionados, isto é, os dados possuem informação que permitem que sejam relacionados semanticamente entre si, independentemente do tipo, formato e origem dos dados [13].

Uma ontologia descreve formalmente a semântica das fontes de dados com o objetivo de facilitar a descoberta e recuperação de informações acerca de uma determinada área de conhecimento [14]. Uma ontologia especifica o modelo de relacionamento de dados de uma determinada área de conhecimento e é composta por um ou vários vocabulários, sendo que estes definem os conceitos e relacionamentos que são utilizados para descrever a área de conhecimento a que a ontologia se refere.

1.1 Motivação e Caracterização do Problema

Na *Web Semântica*, as ontologias são muito utilizadas para descrever formalmente a semântica dos dados. No entanto, quando fazemos uso de diversas ontologias para representar uma mesma área de conhecimento, dados provenientes dessa área poderão ser publicados usando termos diferentes. Deste modo, passamos a ter heterogeneidade no que toca aos

vocabulários o que, por sua vez, impossibilita cumprir o grande objetivo da *Web de Dados*, isto é, a criação de um espaço global de dados homogéneos que permita a descoberta e a integração de novas fontes de dados. A heterogeneidade dos vocabulários entre ontologias torna-se desafiante para os agentes computacionais o que, por sua vez, os impossibilita de recuperar informações sem que haja intervenção humana. Segundo Schultz et al. [15], os desafios que os agentes computacionais enfrentam são três, sendo estes:

- As ontologias usam diversos vocabulários *Resource Description Framework (RDF)* para representar dados sobre uma mesma área de conhecimento;
- As ontologias utilizam diferentes *Uniform Resource Identifier (URI)* para identificar uma mesma área;
- Dados sobre uma mesma área podem apresentar valores distintos dependendo das ontologias onde os dados foram originalmente definidos.

Uma das motivações para o estudo das ontologias foca-se essencialmente em fornecer dados homogéneos, a partir de fontes de dados heterogéneas. A forma mais comum de mitigar a heterogeneidade e atingir o objetivo anterior é através da criação de mapeamentos entre os vocabulários que compõem as ontologias.

Um mapeamento especifica os relacionamentos entre os termos de diversas ontologias, logo é possível recuperar conteúdos semelhantes de diferentes ontologias usando esses mapeamentos.

1.2 Trabalhos relacionados

Gerar mapeamentos entre duas ontologias requer, por parte do utilizador, conhecimentos acerca da definição de mapeamentos e das ontologias que pretende mapear. Logo, tem sido efetuados diversos esforços nesta área de modo a facilitar o mapeamento entre ontologias. Atualmente, existem diversas ferramentas com esse objetivo e, dentro das diversas possibilidades, destacamos a abordagem *Juma* [16], a ferramenta *SPINMap* [17], o *framework LDIF*[6] e a ferramenta *R2R By Assertions* (*RBA*) [14]. Além das ferramentas, também têm surgido diversas linguagens de mapeamento que permitem cobrir diferentes domínios. Por exemplo, existem linguagens que permitem mapeamentos entre Base de Dados relacionais e grafos *RDF*. Dentro das diversas possibilidades que existem nos diferentes domínios, destacamos a linguagem de mapeamento *R2R*[7] e a linguagem *SPARQL 1.1*[8]. A linguagem *R2R*

[6] LDIF http://ldif.wbsg.de/
[7] R2R http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/spec/
[8] SPARQL 1.1 https://www.w3.org/TR/sparql11-query/

é utilizada tanto no *LDIF* como na *RBA* e a linguagem *SPARQL* é utilizada, não só na nossa ferramenta, como ainda em vários trabalhos e pesquisas (tais como em [16] e [17]).

Além dos mapeamentos entre ontologias, também existem outras soluções para fazer face aos problemas relacionados com a heterogeneidade dos dados, tais como o *alinhamento de ontologias*. O alinhamento de ontologias determina correspondências entre termos de duas ontologias, de modo a gerar vínculos que definem que elemento de uma ontologia é similar ao elemento da outra [18].

1.3 Contribuições

Para fazer face aos problemas enumerados anteriormente, a nossa proposta tem como principal contribuição a extensão dos padrões de mapeamentos definidos por Vinuto em [14], através da criação de *templates* para gerar mapeamentos *SPARQL*, utilizando a cláusula *CONSTRUCT*. De acordo com Vinuto [14], os padrões de mapeamento tem como principal objetivo facilitar a especificação de mapeamentos entre ontologias, sendo indispensáveis quando se trata da representação de problemas de incompatibilidades entre ontologias distintas.

A principal motivação para a troca da linguagem deve-se essencialmente ao facto da linguagem *SPARQL* ser uma recomendação do *World Wide Web Consortium* (*W3C*), enquanto que a linguagem *R2R* não o é.

A segunda contribuição deste trabalho é a criação da ferramenta *SMA* que visa auxiliar e simplificar a tarefa de geração de mapeamentos *SPARQL 1.1* através de Assertivas de Mapeamento (*AMs*).

1.4 Estrutura da Dissertação

Esta dissertação é composta pelos seguintes capítulos:

- **Capítulo 2 – Fundamentação Teórica:** capítulo onde serão apresentados, de forma resumida, os conceitos mais relevantes, de modo a que seja fornecida uma base teórica para o entendimento dos restantes capítulos que compõem esta dissertação;
- **Capítulo 3 – Trabalhos Relacionados:** capítulo onde serão expostas algumas das soluções existentes para fazer face aos problemas relacionados com a heterogeneidade e reutilização de ontologias;
- **Capítulo 4 – Representação e Padrões de Mapeamentos:** neste capítulo apresentamos o formalismo descrito por Vinuto em [14]. Também

apresentaremos aqui o estudo de caso que será utilizado para uma melhor compreensão da nossa proposta e o catálogo de padrões de mapeamentos proposto por Vinuto em [14]. Aquando da explicação dos padrões de mapeamento, iremos apresentar os nossos *templates* para a geração de mapeamentos *SPARQL*;

- **Capítulo 5 – Implementação da Proposta:** capítulo onde irá ser exposta, não só a interface gráfica da nossa proposta para a criação de mapeamentos *SPARQL*, como também todos os intervenientes para que a mesma pudesse ser efetuada com sucesso. Por outras palavras, as tecnologias e ferramentas utilizadas no desenvolvimento da proposta, bem como a arquitetura da nossa ferramenta serão apresentadas neste capítulo;
- **Capítulo 6 – Conclusão:** capítulo onde expomos as considerações finais desta dissertação;
- **Capítulo 7 – Trabalho futuro:** neste capítulo apresentamos possíveis desenvolvimentos tanto a nível teórico como prático que visam o prosseguimento do que já foi desenvolvido por nós.

2 Fundamentação Teórica

Neste capítulo serão expostos os conceitos, as principais tecnologias e padrões da *Web Semântica* que serão essenciais para a compreensão deste trabalho. Na secção 2.1, serão apresentados, não só o conceito da *Web Semântica*, como os de *Web de Documentos* e *Web de Dados*, de modo a compreender as diferenças entre as três abordagens e qual a sua interligação com os restantes conceitos deste trabalho. Nas secções 2.2 e 2.3, serão apresentados os dois conceitos fundamentais da *Web Semântica*: *Linked Data* e *Ontologia*, respetivamente. Além destes dois conceitos, a *Web Semântica* ainda alberga várias tecnologias e padrões que serão abordados na secção 2.4, tais como *URI*, *XML*, *RDF*, *RDF-S*, *OWL* e *SPARQL*. Por último, na secção 2.5 são apresentadas as considerações finais deste capítulo.

2.1 Web Semântica

A *Web de Documentos* ou *web* tradicional, é um espaço global de documentos interligados, enquanto que, a *Web de Dados* é um espaço global de dados vinculados. Na prática, o que isso significa?

Antes de falarmos da *Web de Dados*, é necessário falar em primeiro lugar da *Web Semântica*, pois é a partir desta última que podemos vir a possuir, um dia, uma *Web de Dados*. O conceito de *Web Semântica* surgiu com o criador original da *web*, Tim Berners-Lee. Segundo este, a *Web Semântica* não conecta apenas documentos uns aos outros, mas também reconhece o significado das informações contidas nesses mesmos documentos [19]. Deste modo, o significado das informações é percetível, não só para os humanos, como também para os agentes computacionais.

Segundo Cunha et al. [20], a *Web Semântica* é considerada uma extensão da *Web de Documentos* cujo principal objetivo é facilitar a interpretação e integração de dados na *web*. Com a *Web Semântica*, surge o conceito de *Linked Data* (em português, dados ligados), o qual consiste em dados semanticamente relacionados, ou seja, ao invés de conter uma listagem de documentos que não têm ligação semântica, como acontece na *Web de Documentos*, possui informação que pode ser relacionada semanticamente entre si, independentemente do tipo, formato e origem dos dados. Por sua vez, esses dados ligados formam um espaço global conhecido como *Web de Dados*.

De acordo com Maio [21], tal como a *Web de Documentos* abstraiu o acesso aos documentos das dificuldades relacionadas com as camadas físicas e lógicas, de armazenamento, e de interligação de redes de computadores, a *Web Semântica* pretende resolver as dificuldades

resultantes da interoperabilidade e integração de dados entre aplicações, bem como da capacidade de um computador ser capaz de compreender o significado desses dados de modo a que possa processá-los adequadamente.

Segundo Santos e Carvalho [4], para atingir os propósitos da *Web Semântica*, é necessária a padronização de tecnologias, linguagens e metadados descritivos, de modo a que todos os utilizadores da *web* obedeçam a determinadas regras comuns e compartilhadas acerca de como armazenar dados e descrever a informação armazenada. Esta padronização tem como objetivo possibilitar a utilização da informação por humanos e agentes computacionais de forma automática e sem ambiguidade.

2.2 *Linked Data*

Linked Data é um conjunto de melhores práticas introduzidas por Tim Berners-Lee para a publicação e interligação de dados estruturados na *web* [13]. A utilização dessas práticas permite estabelecer *links* (em português, ligações) entre itens de diferentes fontes de dados, de modo a formar um único espaço de dados global, denominado por *Web de Dados*. De acordo com Berners-Lee [22], para que seja possível vir a ter um dia uma *Web de Dados*, é necessário seguir um conjunto de "regras", a que foi dado o nome de princípios de *Linked Data*:

- Usar *URIs* como nomes para coisas;
- Usar *URIs Hypertext Transfer Protocol (HTTP)* para que as pessoas possam procurar o que desejam;
- Quando alguém procurar uma *URI*, fornecer informação útil através dos padrões *RDF* e *SPARQL*;
- Incluir *links* para outras *URIs*, de modo a permitir a descoberta de mais coisas.

Os dados que são publicados de acordo com os princípios de *Linked Data* tem três características, nomeadamente: **(1)** poderem ser processados por máquinas; **(2)** possuírem um significado claramente definido; e **(3)** poderem estar ligados a outras fontes de dados.

Segundo Pinheiro [23], o principal objetivo de *Linked Data* é permitir o compartilhamento de dados estruturados de uma forma tão fácil como atualmente se compartilham documentos *Hypertext Markup Language (HTML)*.

Em suma, *Linked Data* baseia-se em quatro padrões bem estabelecidos e amplamente utilizados na *Web Semântica*, nomeadamente:

- **Mecanismo de identificação global e único:** *URI*
- **Mecanismo de acesso universal:** *HTTP*

- **Modelo de dados:** *RDF*
- **Linguagem de consulta para acesso aos dados:** *SPARQL*

2.3 Ontologia

Segundo Vinuto [14], ontologia é um elemento da *Web Semântica* que permite descrever e representar diferentes domínios, que podem ser utilizados pelos agentes computacionais. Uma ontologia é um artefacto que especifica o modelo de relacionamento de dados de um domínio específico.

Apesar de existirem diversas definições do conceito de ontologias, o principal objetivo de uma ontologia e os principais elementos de uma ontologia são comuns a todas as definições. De acordo com Morais e Ambrósio [24], o principal objetivo de uma ontologia é permitir o compartilhamento e a reutilização do conhecimento. Em relação aos principais elementos, segundo Vinuto [14] são os seguintes:

- **Classe:** descreve grupos abstratos, conjuntos ou coleções de objetos. Uma classe pode possuir várias subclasses, em que cada uma representa um conceito mais específico da classe principal. Por exemplo, uma classe Animal representa todos os animais existentes, enquanto que a classe Cavalo deriva da classe Animal, ou seja, é uma subclasse de Animal, e apenas representa todos os cavalos existentes;
- **Instância (ou indivíduo):** traduz-se numa representação concreta de uma classe;
- **Propriedade:** pode ser utilizada para estabelecer relacionamentos entre instâncias, ou entre instâncias e valores de dados, o que permite afirmar factos sobre os membros das classes e sobre as instâncias.

De acordo com Rocha [25], é importante que uma ontologia represente ao máximo o conhecimento de um domínio específico. No entanto, é ainda mais importante que uma ontologia descreva o consenso entre uma comunidade acerca desse mesmo domínio e que sirva os propósitos para o qual foi desenvolvida.

2.4 Tecnologias da *Web Semântica*

A *Web Semântica* possui várias tecnologias e cada uma delas é considerada uma camada da sua arquitetura. Normalmente, as várias tecnologias são organizadas em forma de pilha, formando assim a arquitetura da *Web Semântica* que pode ser observada na Figura 1.

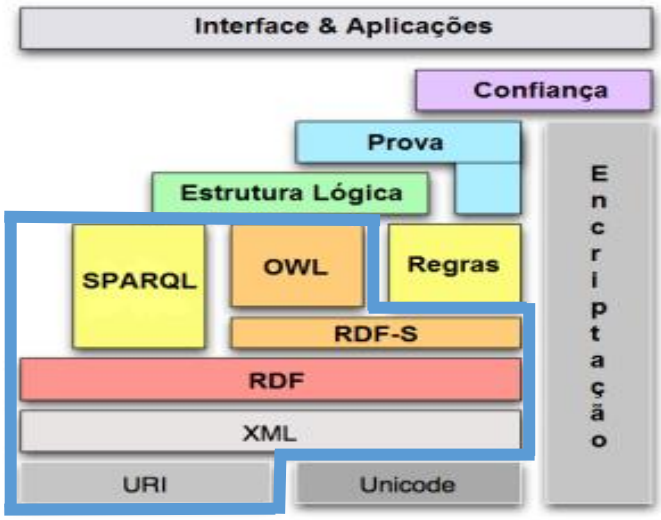

Figura 1 - Arquitetura da Web Semântica [4]

As camadas inferiores da arquitetura são utilizadas como base para as tecnologias das camadas superiores. Por exemplo, a camada "*Unicode*" é a base para a representação de idiomas e símbolos para a camada "*eXtensible Markup Language (XML)*" e demais camadas superiores.

Para o entendimento desta dissertação, apenas são relevantes as tecnologias que estão delimitadas a azul, logo irão ser apresentadas com maior detalhe nas subsecções seguintes.

A camada "*Unicode*" garante a interoperabilidade a todos os níveis, pois fornece um número único para cada caracter, independentemente da plataforma, dispositivo, aplicação ou linguagem [26].

Segundo Vinuto [14], as camadas de regras, estrutura lógica, provas e confiança são utilizadas para se descobrir novos dados a partir dos dados explicitamente definidos. Por último, a camada de encriptação é utilizada para codificar e detetar alterações nas informações.

2.4.1 *URI*

URI é uma cadeia de caracteres compacta que é utilizada para identificar ou denominar um recurso na Internet, sendo que a sua principal finalidade é permitir a interação com as representações de recursos sobre uma rede, tipicamente a *web*, utilizando protocolos específicos [20].

Mas afinal o que é um recurso? Em primeiro lugar, é importante referir que não existe uma definição clara do conceito recurso, no entanto, neste caso podemos definir um recurso como sendo qualquer coisa que possa ser identificada como, por exemplo, imagens, documentos, páginas da *web* [27].

Segundo Mealling e Denenberg [28], o *URI* possui duas especificações: o *Uniform Resource Locator (URL)* e o *Uniform Resource Name (URN)*. O *URL* é o mais utilizado no quotidiano e serve para definir a localização de rede de um determinado recurso [28], sendo composto por um protocolo de acesso e uma localização. Por sua vez, o *URN* serve para identificar um recurso pelo nome num determinado *namespace*, no entanto, ao contrário do *URL*, não define como o mesmo pode ser obtido [29]. Um *namespace* pode ser definido como sendo um espaço abstrato que possui um contexto para os dados que armazena, permitindo que os mesmos sejam exclusivos. Por exemplo, o *namespace International Standard Book Number* (*ISBN)* identifica inequivocamente qualquer livro, sendo definido por *"urn:isbn:"* [29].

De seguida, são listados alguns exemplos de modo a ser mais facilmente percetível as diferenças entre os três conceitos (*URI*, *URL* e *URN*):

- **http://www.example.org/noticias/websemantica.html –** é simultaneamente um *URI* e um *URL*, pois o protocolo de acesso *http* está presente na representação**;**
- **www.example.org/noticias/websemantica.html –** é apenas um *URI*, pois para ser considerado também um *URL* era necessário possuir um protocolo de acesso;
- **maito:contacto@example.org** – é simultaneamente um *URI* e um *URL*, pois possui o protocolo de acesso que é utilizado para envio de e-mails (*mailto*);
- **urn:isbn:978-1420090505 –** é simultaneamente um *URI* e um *URN*, pois identifica inequivocamente o recurso pelo *namespace ISBN*, no entanto, não está especificado como o mesmo pode ser obtido.

Em suma, um *URI* identifica um recurso por nome ou localização, já que todos os *URL* e *URN* são considerados *URI*. No entanto, como se pode observar nos exemplos anteriores, nem todos os *URI* são considerados *URL*.

Existe ainda um conceito mais recente, o *Internationalized Unicode-encoded* (*IRI*), que é uma extensão do *URI* e permite conter caracteres *Unicode*, ao invés de conter apenas caracteres *American Standard Code for Information Interchange* (*ASCII*), como acontecia no caso do *URI*.

No contexto de *Linked Data*, os *URI* são utilizados para identificar objetos e conceitos [23].

2.4.2 *XML*

XML é uma linguagem de marcação recomendada pela *W3C* que evita ambiguidades e dá suporte à troca de informações na *web*, independentemente do formato do ficheiro,

tecnologia ou plataforma. Esta linguagem é classificada como extensível, pois permite que os seus utilizadores definam as suas próprias etiquetas, sendo que a única imposição é que estas estejam bem formatadas de acordo com as regras de sintaxe definidas [4].

Em relação à *Web Semântica*, a linguagem *XML* pertence à sua arquitetura devido ao facto do modelo *RDF* ter sido originalmente serializado em *XML*. Tal aconteceu porque a linguagem *XML* tornou-se um padrão amplamente utilizado para a troca de dados na *web* [14].

2.4.3 *RDF*

RDF é um padrão recomendado pelo *W3C* para descrever recursos [27], ou seja, trata-se de um modelo padrão para a representação de dados na *Web Semântica*. De acordo com Lima e Carvalho [30], o padrão *RDF* tem como principais características, o facto de:

- Ser proposto para situações onde as informações necessitam de ser processadas por aplicações;
- Fornecer uma estrutura comum para expressar informações que podem ser trocadas entre diversas aplicações, sem que haja perda de significado;
- Se basear no princípio de *URI* e na descrição de recursos em termos de propriedades e valores das mesmas.

De acordo com Cunha et al. [20], o modelo *RDF* é baseado no conceito de grafo, é extensível, e possuí um alto nível de expressividade, o que facilita a interligação de dados de diversas fontes.

Segundo Santos e Carvalho [4], tal como uma linguagem natural é adequada para a comunicação entre seres humanos, o *RDF* é adequado para expressar descrições acerca dos recursos, de modo a facilitar o processamento automático pelos agentes computacionais. Em *RDF*, um recurso é exposto como sinónimo de entidade, pois trata-se de um termo genérico para qualquer coisa num determinado domínio [30]. De acordo com Vinuto [14], os recursos em *RDF* são descritos através de declarações e as suas propriedades possuem valores.

As declarações em *RDF* são modeladas através de triplos *RDF*, que são estruturados na forma (*sujeito predicado objeto*), onde:

- **Sujeito:** Identifica o recurso que queremos declarar, em que o valor assumido é sempre uma referência *URI*;
- **Predicado:** Especifica qual o tipo de relacionamento existente entre o sujeito e o objeto, sendo que, como acontece no sujeito, o valor assumido é sempre uma referência *URI*;

- **Objeto:** Representa o recurso que se relaciona com o sujeito, em que o valor assumido pode ser uma referência *URI* ou um valor constante, designado por literal. O literal não pode ser utilizado como sujeito ou predicado.

Por exemplo, a declaração "http://www.example.org/artigos/websemantica.html tendo como autor o valor Maria" pode ser representada pelo seguinte triplo *RDF*:

(*http://www.example.org/artigos/websemantica.html autor Maria*)

em que o **sujeito** é http://www.example.org/artigos/websemantica.html, o **predicado** é autor e o **objeto** é Maria. No exemplo, o predicado autor é uma simplificação para "http://www.example.org/artigos/autor".

Um triplo *RDF* pode ser modelado como um grafo dirigido, em que a propriedade (predicado) é representada por um arco e os recursos (sujeito, objeto) são representados por nós, sendo que a direção da seta é relevante, pois o arco começa sempre no sujeito e aponta sempre para o objeto [4], tal como podemos observar na Figura 2.

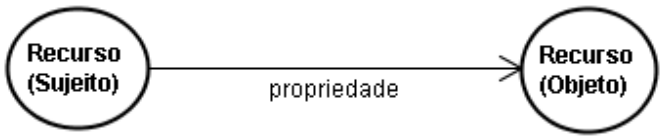

Figura 2 - Representação de um triplo RDF [4]

Um grafo *RDF* é uma coleção de triplos *RDF*. Segundo Lima e Carvalho [30], para diferenciar o tipo de valores assumidos pelos nós, utiliza-se elipses para apresentar nós que tem como valor uma referência *URI* e retângulos para apresentar nós que tem como valor um literal.

Na Figura 3, podemos observar um exemplo simples de um grafo *RDF* que possui duas declarações sobre o mesmo recurso: "http://www.example.org/artigos/websemantica.html tem como autor o valor Maria" e "http://www.example.org/artigos/websemantica.html está apresentado no idioma PT (Português)". Estas declarações dão origem aos seguintes triplos *RDF*:

(*http://www.example.org/artigos/websemantica.html autor Maria*)
(*http://www.example.org/artigos/websemantica.html idioma PT*)

onde "PT" e "Maria" são literais, "http://www.example.org/artigos/websemantica.html" é uma referência *URI* e "autor" e "idioma" são simplificações para os seguintes *URIs*: "http://www.example.org/artigos/autor" e "http://www.example.org/artigos/idioma", respetivamente.

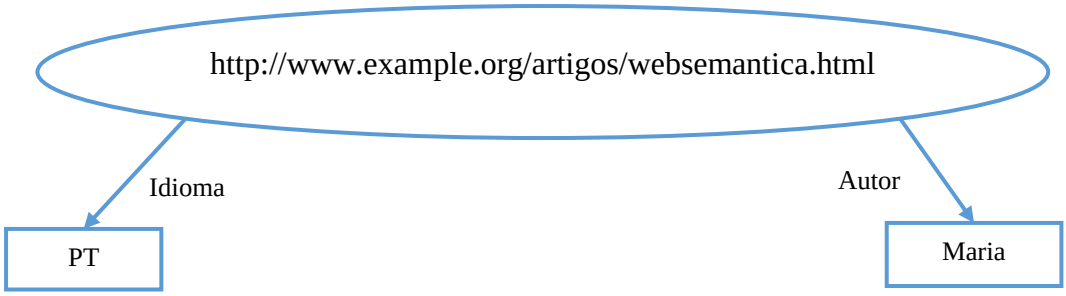

Figura 3 - Simples Grafo RDF

Segundo Maio [21], o *RDF* não determina a forma como os grafos *RDF* devem ser armazenados e transmitidos entre aplicações, logo, dados representados em *RDF* podem ser serializados de diversas formas. Originalmente, o modelo *RDF* foi serializado em *XML*, tendo o nome de *RDF/XML*. Além deste formato, destacam-se ainda o *N-Triples*, o *Notation3* e o *Terse RDF Triple Language* (*Turtle*).

RDF/XML possui uma sintaxe baseada em *XML* e representa as declarações de modo a que possam ser processadas mais facilmente pelos agentes computacionais [14]. No caso do *N-Triples*, as declarações *RDF* são escritas num documento de texto, em que cada declaração representa uma linha do documento. Além disso, este formato possui uma sintaxe muito básica, permitindo assim uma rápida análise. À semelhança do *N-Triples*, no *Notation3* as declarações *RDF* também são escritas num documento de texto. O *Notation3* é uma alternativa compacta e legível à sintaxe *RDF/XML* e, além disso, permite maior expressividade. Por último, segundo Beckett e Berners-Lee [31], o formato *Turtle* utiliza um formato de texto compacto e natural para escrever as declarações *RDF* e fornece níveis de compatibilidade com os formatos *N-Triples* e *Notation3*, pois não só estende o *N-Triples*, como utiliza algumas propriedades do *Notation3*. Além disso, também oferece compatibilidade com a sintaxe do *SPARQL*.

De seguida, são apresentados os triplos *RDF* para o exemplo mostrado anteriormente na Figura 3, nos respetivos formatos: *RDF/XML* (Figura 4), *N-Triples* (Figura 5), *Notation3* e *Turtle* (Figura 6). A sintaxe do *Notation3* e do *Turtle* são muito idênticas, sendo que apenas se notam as diferenças quando as declarações exigem um nível de complexidade superior. Como o nosso exemplo é simples, não existe qualquer diferença entre os formatos e, por isso, apresentámos uma única vez, na Figura 6.

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:ex="http://www.example.org/artigos/">
 <rdf:Description
rdf:about=”http://www.example.org/artigos/websemantica.html">
<ex:autor>Maria</ex:autor>
       <ex:idioma>PT</ex:idioma>
 </rdf:Description>
</rdf:RDF>
```

Figura 4 - Exemplo de declarações RDF no formato RDF/XML

```
<http://www.example.org/artigos/websemantica.html>
<http://www.example.org/artigos/autor> “Maria”
<http://www.example.org/artigos/websemantica.html>
<http://www.example.org/artigos/idioma> “PT”
```

Figura 5 - Exemplo de declarações RDF no formato N-Triples

```
@prefix ex: <http://www.example.org/artigos/>
ex:websemantica.html ex:autor “Maria” ;
ex:idioma “PT” .
```

Figura 6 - Exemplo de declarações RDF no formato Notation3 e Turtle

Nos exemplos das Figuras 4 e 6, foi utilizado o prefixo *ex* para se referir ao *namespace URI* “*http://www.example.org/artigos*”, de modo a facilitar não só a escrita, como também a leitura do documento. Este *namespace URI* refere-se a uma organização fictícia que é utilizada para apresentar exemplos. Além disso, quando se refere a um recurso deste *namespace URI* utiliza-se um *QName* (nome qualificado *XML*). Segundo Lima e Carvalho [30], um *QName* é formado por um prefixo associado a um *namespace URI*, seguido por dois pontos “:” e um nome local. Nas Figuras 4 e 6, são utilizados os seguintes *QName*: *ex:autor, ex:idioma* e *ex:websemantica.html*.

Além do prefixo *ex*, nesta dissertação ainda serão utilizados outros prefixos, tais como os que se encontram descritos na Tabela 1.

Tabela 1 - Prefixos e respetivos namespace URI

Prefixo	*Namespace URI*
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#

2.4.4 *RDF Schema (RDF-S)*

RDF-S é um vocabulário para modelação de dados representados em *RDF*, que estende o modelo *RDF* básico, de modo a permitir restrições semânticas mais complexas no seu vocabulário.

Segundo Lima e Carvalho [30], *RDF-S* é especificado como um conjunto de classes, propriedades e restrições entre os seus relacionamentos. Além disso, de acordo com Rocha [32], *RDF-S* adota um paradigma de modelação similar às linguagens de programação orientadas por objetos, pois assenta nos conceitos de classe e propriedade. No entanto, enquanto que nas linguagens orientadas por objetos, as propriedades apenas existem no contexto de uma classe, no *RDF-S* os dois conceitos são independentes.

Uma classe é uma abstração utilizada para agrupar recursos que possuem características semelhantes e pode ser organizada de forma hierárquica [30], sendo identificada através da notação *rdfs:Class*. Segundo Lima e Carvalho [30], o objetivo da utilização das classes é tornar o modelo *RDF* extensível, através da reutilização e compartilhamento de esquemas. Tal é possível devido à possibilidade de as definições de esquemas existentes poderem ser herdadas. Assim, à semelhança do paradigma orientado a objetos, podemos definir classes como subclasses de outras classes, através da notação *rdfs:subClassOf*.

De acordo com Lima e Carvalho [30], os recursos que pertencem a uma classe são designados por instâncias. Para identificar as instâncias é utilizada uma propriedade com notação *rdf:type*. Por exemplo, o triplo *RDF* (*ex:Pessoa rdf:type rdfs:Class*) indica que *ex:Pessoa* é uma instância de *rdfs:Class*, logo *ex:Pessoa* é uma classe.

Uma propriedade permite expressar relações entre classes e suas instâncias ou superclasses [30]. Segundo Rocha [32], a notação *rdf:Property* representa a classe de todas as propriedades, logo também é uma instância de *rdfs:Class*. Além desta notação, existem três outras notações utilizadas na noção de propriedade, sendo estas: *rdfs:subPropertyOf*, *rdfs:domain* e *rdfs:range*.

A notação *rdfs:subPropertyOf* é usada para especificar hierarquia de propriedades [33]. Segundo Vinuto [14], as notações *rdfs:domain* e *rdfs:range* são utilizadas para especificar restrições de tipos sobre os sujeitos e/ou objetos, sendo que *rdfs:domain* significa domínio e *rdfs:range* significa contradomínio. De acordo com Lima e Carvalho [30], a diferença entre as duas restrições é que a *rdfs:domain* é utilizada para especificar a classe na qual determinada propriedade pode ser aplicada, enquanto que a *rdfs:range* é utilizada para limitar os valores que podem ser aplicados a uma dada propriedade.

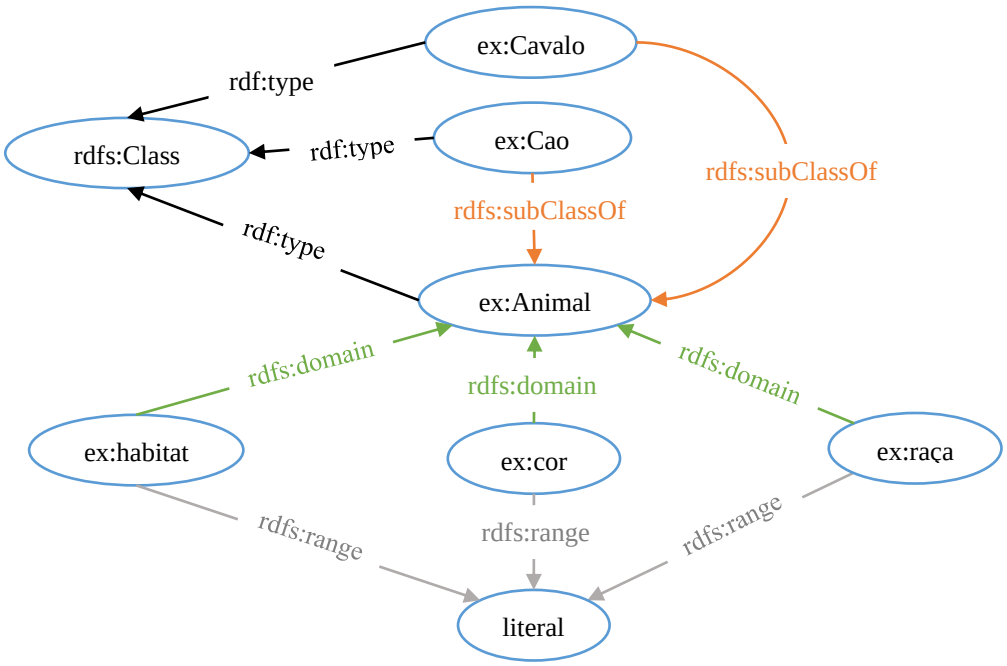

Figura 7 - Representação gráfica de um grafo utilizando os conceitos de RDF e RDF-S

No exemplo da Figura 7, *ex:Animal*, *ex:Cao* e *ex:Cavalo* são classes fictícias, em que *ex:Cao* e *ex:Cavalo* são subclasses de *ex:Animal*. *ex:cor*, *ex:raça* e *ex:habitat* são propriedades cujo domínio é *ex:Animal* e contradomínio é um literal, ou seja, estas três propriedades podem ser utilizadas na classe *ex:Animal* e apenas aceitam valores constantes.

2.4.5 *OWL*

OWL é uma linguagem, recomendada pelo *W3C*, para a definição e a instanciação de ontologias *web*. De acordo com Santos e Carvalho [4], o objetivo principal da *OWL* é permitir a representação eficiente de ontologias.

Segundo Vinuto [14], a *OWL* estende *RDF* e *RDF-S*, e possui um vocabulário mais abrangente de modo a fornecer meios para o tratamento de relacionamentos e restrições mais complexas entre objetos.

Na *OWL* utilizamos a notação *owl:Class* para definir uma classe, no entanto, esta poderia ser definida com recurso ao *RDF-S* através da sua notação *rdfs:Class*. A grande vantagem na utilização da *OWL* em detrimento do *RDF-S* é que conseguimos aumentar o nosso leque em termos de expressividade, sendo que a *OWL* fornece diversas notações para esse efeito. Vejamos um exemplo, se quisermos referir que duas classes são equivalentes ou disjuntas, apenas o podemos efetuar na *OWL* através das notações *owl:equivalentClass* e *owl:disjointWith*, respetivamente. A notação *owl:disjointWith* permite identificar que uma instância de classe não pode ser uma instância de outra classe [34], enquanto que *owl:equivalentClass* identifica que duas classes estão relacionadas, no entanto, não possuem o mesmo significado.

Na Figura 8, podemos observar que o conceito de Animal está relacionado com o conceito de Mamífero, no entanto, estes dois conceitos não são iguais.

```
<owl:Class rdf:about="#Animal">
  <equivalentClass rdf:resource="#Mamifero"/>
</owl:Class>
```

Figura 8 - Exemplo da utilização da notação owl:Class

Ainda na parte relativa às classes, a *OWL* fornece seis tipos especiais de descrições de classes, sendo que de todos eles destacamos as **restrições de propriedade**. De acordo com Bechhofer et al. [34], uma restrição de propriedade descreve uma classe composta por todos os elementos que satisfazem a restrição, sendo que a notação utilizada para definir uma restrição é *owl:Restriction*. Na *OWL* existem dois tipos de restrição: *restrições de valor* e *de cardinalidade*, em que a primeira coloca restrições no intervalo da propriedade e a segunda coloca restrições em relação ao número de valores que uma propriedade pode assumir. Dentro das restrições de valor temos três notações: *owl:someValuesFrom*, *owl:hasValue* e *owl:allValuesFrom*, enquanto que nas restrições de cardinalidade possuímos: *owl:cardinality*, *owl:maxCardinality* e *owl:minCardinality*. Segundo Bechhofer et al. [34], as notações enumeradas anteriormente são utilizadas para:

- ***owl:someValuesFrom*:** descrever uma classe em que todos os elementos possuem pelo menos um valor da propriedade em questão ou um valor de dados no intervalo de dados especificado;
- ***owl:hasValue:*** descrever uma classe em que todos os elementos possuem pelo menos um valor semanticamente equivalente ao valor em questão;

- ***owl:allValuesFrom:*** descrever uma classe em que todos os elementos possuem todos os valores de propriedade em questão ou todos os valores de dados encontram-se dentro do intervalo de dados especificado;
- ***owl:cardinality:*** especificar uma classe que possui exatamente N valores semanticamente distintos;
- ***owl:maxCardinality:*** especificar uma classe que possui, no máximo N valores semanticamente distintos;
- ***owl:minCardinality:*** especificar uma classe que possui no mínimo N valores semanticamente distintos.

Para além da definição e descrição de classes, a *OWL* também fornece duas notações para se definir propriedades, sendo estas: *owl:ObjectProperty* e *owl:DatatypeProperty*. A diferença entre elas é que enquanto que a *owl:ObjectProperty* é utilizada para relacionar um recurso com outro que seja identificado por *URI*, a *owl:DatatypeProperty* é utilizada para relacionar um recurso com um literal ou com o tipo de dados do esquema *XML* [34]. Na definição de propriedade, também são fornecidas diversas notações pela *OWL* de modo a permitir mais expressividade, tais como: *owl:equivalentProperty*. A notação *owl:equivalentProperty* é utilizada para indicar que duas propriedades têm a mesma extensão de propriedade, ou seja, que são equivalentes. Tal como acontece na *owl:equivalentClass*, esta notação apenas refere que duas propriedades estão relacionadas e não que são iguais. Caso se queira indicar que duas classes ou duas propriedades são iguais é necessário utilizar a notação *owl:sameAs*.

Por último, a *OWL* também permite a utilização de notações do *RDF-S*, tais como: *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain* e *rdfs:range*.

Na Figura 9, é apresentado um exemplo utilizando os conceitos da *OWL*. O grafo apresentado é uma extensão do exemplo apresentado anteriormente na Figura 7. Podemos observar que, ao invés da utilização da *rdfs:Class*, passa-se a utilizar a *owl:Class*, para permitir adicionar maior expressividade na definição de classes e assim poder, por exemplo, indicar que as classes *ex:Cao* e *ex:Cavalo* são disjuntas. Em relação às propriedades, como sabemos, a *OWL* fornece duas notações, sendo que neste exemplo foi utilizada a *owl:DatatypeProperty*, pois trata-se de literais.

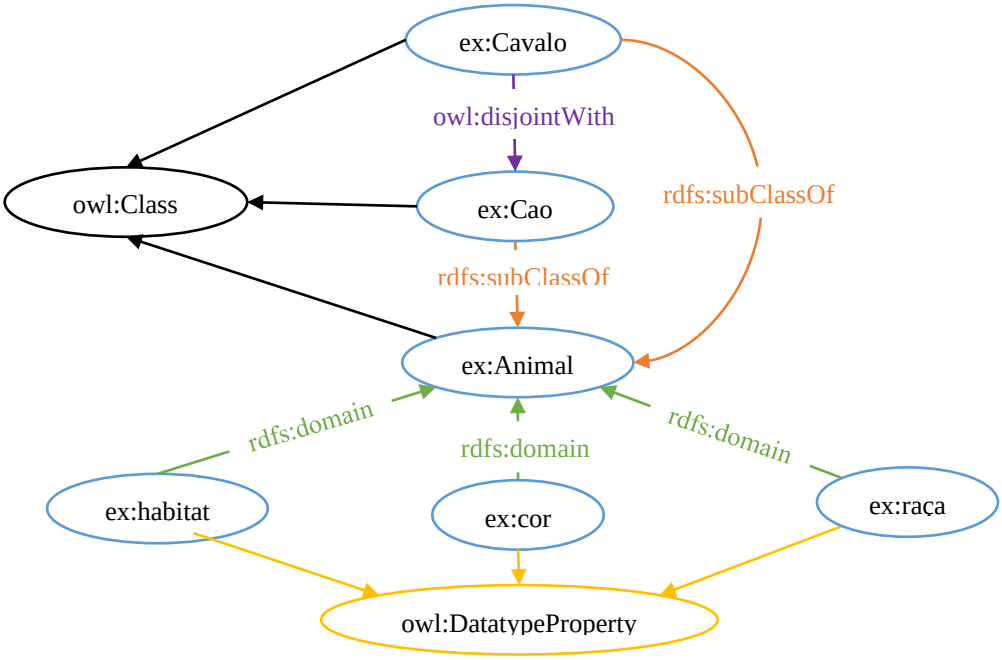

Figura 9 - Representação gráfica de um grafo utilizando os conceitos de OWL

Segundo a recomendação do *W3C*, existem três versões da *OWL*: *OWL Lite*, *OWL DL* e *OWL Full* [35], que possuem diferentes poderes de expressividade. A versão menos expressiva é a *OWL Lite* e a mais expressiva é a *OWL Full*. De acordo com Pinheiro [23], as linguagens menos expressivas estão contidas nas mais expressivas. Assim, uma ontologia que esteja definida, por exemplo, na linguagem *OWL Lite* é completamente aceite pela linguagem *OWL DL*.

2.4.6 *SPARQL*

Segundo Pinheiro [23], *SPARQL* é uma recomendação *W3C* que pode ser dividida em três partes: **(1)** uma linguagem de consulta declarativa; **(2)** um protocolo de acesso a dados em *RDF*; e, por último, **(3)** um formato de resultados.

SPARQL é uma linguagem de consulta declarativa que permite obter e manipular os dados armazenados em formato *RDF*, sendo que o local de armazenamento destes dados designa-se por repositório de triplos. Esta linguagem de consulta é muito semelhante à linguagem *SQL*, que é utilizada para consultar e manipular os dados contidos em Bases de Dados relacionais, pois também possui uma estrutura de *Select-From-Where* [20].

Uma cláusula de consulta *SPARQL* é construída com base em padrões de triplos *RDF*, sendo que um padrão de triplos *RDF* não é mais que um triplo *RDF* que possui variáveis no lugar do sujeito, do predicado e/ou do objeto. Por exemplo:

ex:websemantica.html	*ex:autor*	*?autor*

é um padrão de triplo *RDF* que possui a variável *?autor* no lugar do objeto. Na linguagem *SPARQL*, para se declarar uma variável utiliza-se "?" ou "$". De acordo com Vinuto [14], a um conjunto de padrões de triplos *RDF* dá-se o nome de padrão de grafo.

Uma consulta *SPARQL* segue uma estrutura definida. Segundo [36] e [14], uma consulta *SPARQL* é ou pode ser constituída pelas seguintes secções:

- **Declaração de prefixos:** é uma secção opcional, sendo utilizada para a abreviação de prefixos. O exemplo de uma cláusula pertencente a esta secção é ***PREFIX***;
- **Definição de *datasets*:** tal como a secção anterior, é opcional e é utilizada para indicar quais são os grafos *RDF* sobre os quais irá incidir a consulta *SPARQL*. Um exemplo de cláusula *SPARQL* é : ***FROM*;**
- **Cláusula de retorno:** esta secção é obrigatória e especifica qual a informação que deve ser retornada pela cláusula *SPARQL*. Segundo Ferreira [37], a linguagem *SPARQL* oferece quatro cláusulas de consulta de retorno, sendo estas: ***SELECT***, ***CONSTRUCT***, ***ASK*** e ***DESCRIBE***. Dependendo da cláusula, o retorno vai ser diferente. Por exemplo, no caso da cláusula *ASK* o retorno é sempre um valor booleano, enquanto que no caso da cláusula *CONSTRUCT*, o retorno é sempre um grafo *RDF* que é construído a partir de um *template*;
- **Padrão de consulta:** também é uma secção obrigatória que especifica os padrões de triplos que serão utilizados para consultar o *dataset*. Toda esta secção encontra-se espelhada dentro da cláusula *SPARQL* ***WHERE***;
- **Modificadores de consulta:** é uma secção opcional que tem como objetivo limitar, ordenar ou alterar os resultados de uma dada consulta *SPARQL*. Por exemplo, a cláusula ***LIMIT*** tem como objetivo limitar os resultados de uma consulta, enquanto que a cláusula ***ORDER BY*** tem como objetivo ordenar os resultados de uma consulta.

De seguida, apresentamos cada uma das cláusulas de consultas de retorno disponível na linguagem *SPARQL*.

2.4.6.1 Cláusula SELECT

Identifica todos os nomes das variáveis cujos valores serão retornados no resultado [14]. Por exemplo, se quisermos obter o autor do artigo *ex:websemantica.html* e tivermos os seguintes triplos *RDF*:

(*ex:websemantica.html* *ex:autor* *Maria*)
(*ex:websemantica.html* *ex:autor* *Manuel*)
(*ex:artigo2.html* *ex:autor* *Rita*)

temos de substituir o autor por uma variável. Deste modo ficamos com o seguinte padrão de triplo *RDF*:

ex:websemantica.html *ex:autor* *?autor* .

Podemos afirmar que este padrão de triplo *RDF* representa a condição que queremos que seja satisfeita, logo basta adicioná-lo à cláusula *WHERE* da consulta *SPARQL*. Além disso, temos de colocar a variável *?autor* na cláusula *SELECT*, de modo a identificar qual a variável que queremos que seja retornada da consulta. A consulta *SPARQL* para este exemplo fica como apresentada na Figura 10 a).

```
1. PREFIX ex: <http://www.example.org/artigos/>
2. SELECT ?autor
3. WHERE {
4.       ex:websemantica.html ex:autor ?autor .
5. }
6. LIMIT 1
```

a) Consulta *SPARQL* com cláusula *SELECT*

autor
"Maria"

b) Resultado da consulta

Figura 10 - Exemplo de uma consulta e respetivo resultado utilizando a clausula SELECT

Na Figura 10 a), podemos observar um exemplo de uma consulta utilizando a cláusula *SELECT* em que todas as cláusulas *SPARQL* estão destacadas a negrito. Na linha 1, utilizámos a cláusula *PREFIX* para abreviar os *IRI* dos recursos que serão mencionados na cláusula *WHERE*. Na linha 2, especificamos quais as variáveis que queremos retornar no resultado, sendo que neste exemplo é apenas a variável *?autor*. A cláusula *WHERE* (linhas 3 a 5) é utilizada para definir os requisitos de procura, ou seja, indicamos ao *SPARQL* quais as condições que têm de ser satisfeitas para que um recurso esteja presente no retorno da consulta. No nosso exemplo, apenas possuímos uma condição, no entanto, podíamos adicionar quantas quiséssemos. Por último, na linha 6 utilizámos a cláusula *LIMIT* para indicar ao *SPARQL* que apenas queremos receber 1 resultado de retorno, como mostra a Figura 10 b). Caso não

tivéssemos colocado esta cláusula na consulta *SPARQL*, o resultado desta seria composto por dois valores, Maria e Manuel e não apenas o valor Maria.

2.4.6.2 Cláusula CONSTRUCT

Permite transformar dados descritos de acordo com um esquema em novos triplos *RDF*, logo o seu retorno é um grafo *RDF* que é construído a partir de um *template* [38]. Esta cláusula permite, por exemplo, efetuar o mapeamento entre diferentes ontologias, sendo que por cada mapeamento, seja ele de classe ou propriedade, é criado um novo triplo *RDF*.

Para que fique mais percetível, vamos exemplificar como se cria uma consulta *SPARQL* utilizando o *CONSTRUCT*. Vamos utilizar os mesmos triplos *RDF* descritos anteriormente, mas mudar a condição, isto é, neste momento queremos obter todos os autores que escreveram qualquer artigo. Para tal, o padrão de triplo *RDF* passa a ser:

?s ex:autor ?autor .

Além disso, como o retorno da cláusula *CONSTRUCT* é um grafo *RDF* que é construído a partir de um *template*, temos de indicar na cláusula *CONSTRUCT* qual o *template* que queremos utilizar, sendo este construído a partir de padrões de triplos *RDF*. A Figura 11 a) mostra a consulta *SPARQL* utilizando a cláusula *CONSTRUCT*, com o *template* para criar os triplos *RDF* que respondem corretamente ao que foi pedido.

```
1. PREFIX ex: <http://www.example.org/artigos/>
2. PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3. CONSTRUCT { ex:autores foaf:name ?autor . }
4. WHERE {
5.        ?s ex:autor ?autor .
6. }
```

a) Consulta *SPARQL* com cláusula *CONSTRUCT*

```
ex:autores foaf:name Maria
ex:autores foaf:name Manuel
ex:autores foaf:name Rita
```

b) Resultado da consulta

Figura 11 - Exemplo de uma consulta e respetivo resultado utilizando a clausula CONSTRUCT

Tal como podemos observar na Figura 11 a), a sintaxe é idêntica à cláusula *SELECT*, mudando apenas a forma como declaramos o retorno, pois no exemplo anterior o retorno era apenas o valor da variável, enquanto que na cláusula *CONSTRUCT* o retorno é sempre um novo grafo *RDF*, que é construído a partir de um determinado *template*. No nosso exemplo, os novos triplos *RDF* são construídos a partir do seguinte *template*:

(*ex:autores foaf:name ?autor*)

Como no exemplo descrito na Figura 11 a) retirámos a cláusula *LIMIT*, logo no resultado da consulta aparecerão todos os triplos *RDF* que satisfazem a condição, tal como podemos observar na Figura 11 b).

2.4.6.3 Cláusula ASK

Permite verificar se existe pelo menos um resultado que satisfaça as condições determinadas, sendo que o seu retorno é um valor booleano. Por exemplo, se quisermos saber se a Rita escreveu algum artigo, podemos escrever uma consulta semelhante à apresentada na Figura 12 a). O resultado desta consulta é apresentado na Figura 12 b) sendo que, tal como podemos observar, o resultado é *true*, uma vez que a Rita é a autora da página *ex:artigo2.html*.

```
PREFIX ex: <http://www.example.org/artigos/>
ASK { ?s ex:autor “Rita” }
```

a) Consulta *SPARQL* com cláusula *ASK*

```
true
```

b) Resultado da consulta

Figura 12 - Exemplo de uma consulta e respetivo resultado utilizando a clausula ASK

2.4.6.4 Cláusula DESCRIBE

Devolve um grafo *RDF* composto por dados *RDF* sobre os recursos, sendo que a descrição dos recursos é determinada pelo serviço de consulta [38]. Deste modo, o retorno desta consulta pode ser diferente dependendo do *SPARQL Endpoint* que a executa. Por exemplo, se quisermos obter uma descrição acerca dos recursos que compõem *ex:Animal* (exemplo utilizado na Figura 9), podemos utilizar uma consulta semelhante à apresentada na Figura 13 a). O resultado desta consulta é composto por todos os recursos que compõem *ex:Animal*, tal como pode ser observado na Figura 13 b).

```
DESCRIBE <http://example.org/Animal>
```

a) Consulta *SPARQL* com cláusula

```
@prefix ex: <http://example.org/> .
@prefix rdfs:
<http://www.w3.org/2000/01/rdf-schema#> .

ex:Cavalo rdfs:subClassOf ex:Animal .
ex:Cao rdfs:subClassOf ex:Animal .
ex:habitat rdfs:domain ex:Animal .
ex:cor rdfs:domain ex:Animal .
ex:raca rdfs:domain ex:Animal .
```

b) Resultado da consulta

Figura 13 - Exemplo de uma consulta utilizando a clausula DESCRIBE

2.4.6.5 Cláusulas presentes em consultas SPARQL

A Tabela 2 exemplifica algumas cláusulas da linguagem *SPARQL* que são utilizadas na construção de consultas.

Tabela 2 – Exemplo de cláusulas presentes nas consultas SPARQL

Cláusulas	Definição
PREFIX	Palavra reservada que, tal como vimos anteriormente, é utilizada com o objetivo de abreviar os *IRI* dos recursos que serão mencionados no corpo da consulta *SPARQL*.
FROM	Cláusula opcional que especifica quais as fontes de dados *RDF* que serão consultadas. Caso não seja especificada, a procura será efetuada num documento *RDF/RDF-S* particular [20].
WHERE	Cláusula obrigatória que especifica quais as condições que devem ser satisfeitas, para que um triplo localizado na fonte de dados *RDF* seja considerado como um resultado da pesquisa [14]. As condições são escritas por meio de padrões de triplos *RDF*. Por exemplo, na Figura 10 a), apenas queríamos obter os autores que escreveram um artigo em particular.
DISTINCT	Cláusula que identifica um modificador que elimina os resultados duplicados, tal como acontece na linguagem *SQL*.

LIMIT	Cláusula utilizada para limitar o número máximo de resultados, tal como pudemos observar na Figura 10. Além deste modificador, existem outros que podem ser usados para organizar e restringir os resultados de uma consulta *SPARQL* [39], tais como: ***FILTER***, ***ORDER BY***, ***GROUP BY*** e ***OFFSET***.
FILTER	Modificador utilizado para restringir os resultados de uma consulta através de determinadas condições [39]. Por exemplo, podemos especificar que apenas queremos autores que comecem com a letra "a". Caso aplicássemos este caso ao exemplo da Figura 10, não iria haver qualquer resultado da pesquisa. Existem diversos operadores e funções que podem ser utilizadas dentro desta cláusula, sendo que as principais irão ser explicadas na secção 0.
BIND	Cláusula que permite atribuir um valor a uma variável, sendo muito utilizado nos exemplos deste trabalho, nomeadamente quando é necessário gerar um *URI* nos mapeamentos de classes semanticamente não semelhantes (ver secção 4.3.2 para mais informações e exemplo de utilização). Por exemplo, BIND((2*3) AS ?p) atribui o valor 6 à variável p.
ORDER BY	Modificador que possui o mesmo objetivo que na linguagem *SQL*, ou seja, especificar qual o critério de ordenação.
GROUP BY	Modificador que especifica de que forma é que vão ser agrupados os resultados, tal como acontece na linguagem *SQL*.
OFFSET	Cláusula que identifica um modificador que permite especificar que pretendemos ignorar os N primeiros resultados, tal como se sucede na linguagem *SQL*.

2.4.6.6 Operadores e funções

Ao especificar os filtros na linguagem *SPARQL*, podem estar presentes operadores e funções. Dentro dos operadores incluem-se os operadores lógicos, matemáticos e de comparação, enquanto que as funções podem ser agrupadas em vários grupos, dos quais destacamos as funções genéricas, de *string* e as de datas. Na Tabela 3, apresentamos os diversos operadores, enquanto que na Tabela 4 exemplificamos algumas funções.

Na Tabela 3, considere ?a, ?b, ?c, ?d e ?e como variáveis *SPARQL*, sendo a variável ?a = true, ?b = false, ?c = 10, ?d = 5 e ?e = 10.

Tabela 3 - Operadores permitidos na linguagem SPARQL

Tipo	Cláusulas	Significado	Exemplo
Lógicos	&&	"E" lógico	?a && ?b = false
	||	"Ou" lógico	?a || ?b = true
Matemáticos	+	Adição	?c + ?d = 15
	-	Subtração	?c - ?d = 5
	*	Multiplicação	?c * ?d = 50
	/	Divisão	?c / ?d = 2
Comparação	=	É igual a	?c = ?d irá retornar false
	!=	É diferente de	?c != ?d irá retornar true
	>	É maior que	?c > ?e irá retornar false
	>=	É maior ou igual que	?c >= ?e irá retornar true
	<	É menor que	?d < ?c irá retornar true
	<=	É menor ou igual que	?e <= ?c irá retornar true

Na Tabela 4 considere ?g1, ?g2, ?g3, ?s1, ?s2, ?s3, ?s4, ?s5 e ?data como variáveis *SPARQL*, sendo a variável ?g1 = 123, ?g2 = "Bom Dia", ?g3 = (123 , 1, 7), ?s1 = "Web", ?s2 = "Semântica", ?s3 = "Web123Semantics", ?s4 = "123", ?s5 = "_" e ?data = "2019-08-28T14:45:13.815-05:00".

Tabela 4 - Algumas funções que são permitidas na linguagem SPARQL

Tipo	Cláusulas	Significado	Exemplo
Genéricas	IN NOT IN	Determina se um elemento existe ou não num conjunto	?g1 **IN(**?g3**)** = true
	STR	Converte valores numa string, sendo que também permite resolver os prefixos	**STR(**?g1**)** = "123"
	ENCODE-FOR-URI	Converte os caracteres reservados de acordo com a função xPath **fn:encode-for-uri**[9]**.**	**ENCODE-FOR-URI(**?g2**)** = "Bom%20Dia"
De *string*	CONCAT	Retorna a concatenação de duas *strings*	**CONCAT(**?s1**,** ?s2**)** irá retornar a *string*: "WebSemantica"
	LCASE	Retorna uma *string* em letras minúsculas, sendo que o oposto é possível através da cláusula **UCASE**.	**LCASE(**?s1**)** irá retornar a *string:* "web"
	REPLACE	Retorna uma *string* depois de substituir todas as ocorrências de uma *string* noutra.	**REPLACE(**?s3**,** ?s4**,** ?s5**)** irá retornar a string: "Web_Semantics"
De datas	NOW	Retorna a data e a hora de hoje. Esta cláusula não possui qualquer parâmetro.	
	YEAR	Retorna o ano do valor da data, sendo que o parâmetro tem que ser passado em *dateTime*.	**YEAR(**?data**)** = 2019

[9] fn:encode-for-uri https://www.w3.org/TR/xpath-functions/#func-encode-for-uri

2.5 Considerações finais do capítulo

Neste capítulo, apresentámos uma visão geral dos principais conceitos, tecnologias e padrões utilizados na *Web Semântica* que servem como uma fundamentação teórica base para o entendimento desta dissertação.

Ao falarmos sobre a *Web Semântica*, tivemos que primeiramente efetuar um paralelo entre a *Web de Documentos* e a *Web de Dados*, não só para compreender as suas diferenças, mas também para que fosse percetível o porquê do surgimento da *Web Semântica* e em que contexto.

Na Tabela 5, está esquematizado de uma forma muito resumida, os pontos essenciais em que a *Web de Documentos* e a *Web de Dados* assentam, sendo possível observar que apesar de utilizarem o mesmo mecanismo de identificação de recursos e de acesso aos mesmos, diferem nos padrões utilizados. Além disso, a *Web de Dados* necessita ainda da linguagem de consulta *SPARQL* para obtenção e manipulação dos dados que se encontram armazenados em formato *RDF*.

Tabela 5 - Web de Documentos vs Web de Dados

	Web de Documentos	**Web de Dados**
Mecanismo de Identificação	*URI*	
Mecanismo de Acesso	*HTTP*	
Padrão para a representação de conteúdo	*HTML*	*RDF*
Acesso ao conteúdo	*Browsers HTML*	*Browsers RDF*
Navegação entre diversos conteúdos	Hiperligações	*Links RDF*
Linguagem de consulta para acesso aos conteúdos	-	*SPARQL*

A *Web Semântica* tem como principal objetivo tentar resolver os problemas da *Web de Documentos*, isto é, proporcionar semântica aos dados de modo a que estes possam ser compreendidos e manipulados por agentes computacionais. Para que tal seja possível, o *W3C* considera seis princípios principais para a consolidação da *Web Semântica*, sendo estes:

1. **Qualquer recurso pode ser identificado por um identificador universal:** Ao longo deste capítulo, foi possível observar que para identificar os recursos, os padrões utilizam o *URI* como identificador universal;

2. **Os recursos e as relações podem ser tipificados:** Segundo Koivunen e Miller [40], os recursos e as relações podem ter tipos que definem conceitos que podem ser interpretados por máquinas. Por exemplo, na Figura 7 observamos a relação *rdfs:subClassOf* entre *ex:Cao* e *ex:Animal* e assim foi identificado que o recurso *ex:Cao* é uma versão do recurso *ex:Animal*;
3. **Tolerância à quebra de relações:** Atualmente, a *Web de Documentos* possui recursos vinculados que deixam de existir, fazendo com que se obtenha erro 404 quando se tenta aceder a esses mesmos recursos. O mesmo pode suceder com a *Web Semântica*, logo as suas ferramentas precisam de tolerar a quebra de relações e funcionar à parte disso [40];
4. **A confiança não necessita de ser absoluta:** Nem tudo o que se encontra na *Web de Documentos* é verdadeiro e o mesmo acontece na *Web Semântica*, logo da mesma forma que os humanos avaliam a confiabilidade das informações que se encontram na *Web de Documentos*, as aplicações que processam as informações na *Web Semântica* tem que avaliar a confiabilidade das mesmas através do seu contexto, ou seja, quem disse as informações, quando o disse e quais as credenciais que tinha para o fazer;
5. **Permitir a evolução:** De acordo com Koivunen e Miller [40], a *Web Semântica* fornece ferramentas que permitem uma combinação de informações relativas a um recurso, que podem ou não estar definidas em diferentes vocabulários. Além disso, novas informações podem ser adicionadas sem que haja a necessidade de modificar o que já existe. Uma ontologia é um bom exemplo da evolução, pois permite definir as relações entre diferentes vocabulários, logo podemos afirmar que a camada "*OWL*" da *Web Semântica* é responsável por garantir a evolução;
6. **Implementação minimalista.**

3 Trabalhos Relacionados

Neste capítulo apresentamos uma visão geral dos trabalhos que têm sido efetuados de modo a mitigar os problemas existentes em relação à heterogeneidade semântica entre ontologias. Esta heterogeneidade ocorre quando possuímos diversas ontologias, cada uma com as suas características, para uma mesma área de conhecimento. A heterogeneidade semântica entre ontologias impossibilita a recuperação de informação por parte dos agentes computacionais sem que haja intervenção humana. Além disso, também impossibilita o cumprimento do grande objetivo da *Web de Dados*, isto é, a criação de um espaço global de dados homogéneos que permita a descoberta e a integração de novas fontes de dados.

Para mitigar os problemas relacionados com a heterogeneidade entre ontologias, existem diversas soluções, sendo as mais comuns: o *Mapeamento entre Ontologias* e o *Alinhamento de Ontologias*. Além disso, podemos conjugar as soluções de heterogeneidade entre ontologias com os *Padrões de Projeto em Ontologias*, de modo a facilitar a resolução destes problemas. Por exemplo, no nosso trabalho utilizamos os *Padrões de Projetos em Ontologias* para definir os padrões que serão utilizados nos mapeamentos.

As soluções enumeradas anteriormente serão abordadas, respetivamente, nas secções 3.1 e 3.2. Na secção 3.3, será introduzido o conceito de *Padrões de Projeto em Ontologias* e, por último, na secção 3.4, são apresentadas as considerações finais deste capítulo.

3.1 Mapeamento entre Ontologias

O mapeamento entre ontologias é um processo que permite relacionar semanticamente duas ontologias, ou seja, permite especificar que um determinado termo de uma ontologia fonte é semanticamente equivalente ou semelhante a um termo da ontologia alvo, sendo que este termo pode representar uma classe ou propriedade. Este processo tende a ser lento e complexo pois, além de ter de ser efetuado um mapeamento por cada termo, os mapeamentos podem não ser diretos. Por exemplo, dados semelhantes semanticamente podem ser definidos numa ontologia como sendo uma classe e noutra ontologia como sendo uma propriedade. O resultado de um mapeamento entre ontologias é uma estrutura única composta pelas diversas ligações entre os termos das duas ontologias. Existem diversas pesquisas no âmbito do mapeamento entre ontologias, sendo que salientamos a abordagem ***Juma*** [16], a ferramenta ***SPINMap*** [17], o *framework* ***LDIF*** [15] e a ferramenta ***RBA*** [14]. Além disso, em termos de linguagens que permitem o mapeamento entre ontologias, destacamos o ***SPARQL*** [41] e o ***R2R*** [42], pois o ***R2R*** é utilizado no ***LDIF*** e na ***RBA*** e o ***SPARQL*** é utilizado tanto na nossa proposta como na

Juma e na ***SPINMap***. Um *survey* sobre mapeamentos entre ontologias pode ser encontrado em: [43] e [44].

De seguida, apresentamos, de forma resumida, cada uma das abordagens e a linguagem de mapeamento *R2R*.

3.1.1 *Juma*

Juma é uma abordagem baseada em *block metaphor*[10], que permite representar mapeamentos em dados vinculados. Esta abordagem possui três aplicações [16]:

- ***Juma R2RML*:** Foi desenvolvido com o intuito de refletir o vocabulário *RDB to RDF Mapping Language* (*R2RML)*[11]. Esta aplicação retorna como *output*, a geração de mapeamentos *R2RML*.
- ***Juma Uplift*:** Segundo o autor, foi desenvolvida com um nível mais alto de abstração que o *Juma R2RML*, de modo a possuir a capacidade de gerar mapeamentos utilizando diferentes linguagens de mapeamento. Segundo Junior em [16], as linguagens de mapeamento suportadas por esta aplicação são: *R2RML* e *Sparqlification Mapping Language* (*SML*)[12].
- ***Juma Interlink:*** Suporta a geração de mapeamentos na forma de consultas *SPARQL* com a cláusula *CONSTRUCT*. Na Figura 14, podemos observar o diagrama desta aplicação e perceber que esta efetua a distinção entre os *mapeamentos simples*, que efetuam o mapeamento de uma entidade com outra (**1:1** – *um-para-um*) e *mapeamentos complexos*, que efetuam o mapeamento entre diversas entidades (**1:N** – *um-para-muitos*, **N:1** – *muitos-para-um* e **N:M** – *muitos-para-muitos*).

[10] *Block metaphor* combina a representação em árvore com elementos visuais. Segundo Junior [71], tornou-se muito popular em linguagens de programação visuais, sendo designada por paradigma de bloco.
[11] R2RML https://www.w3.org/TR/r2rml/
[12] SML http://events.linkeddata.org/ldow2015/papers/ldow2015_paper_09.pdf

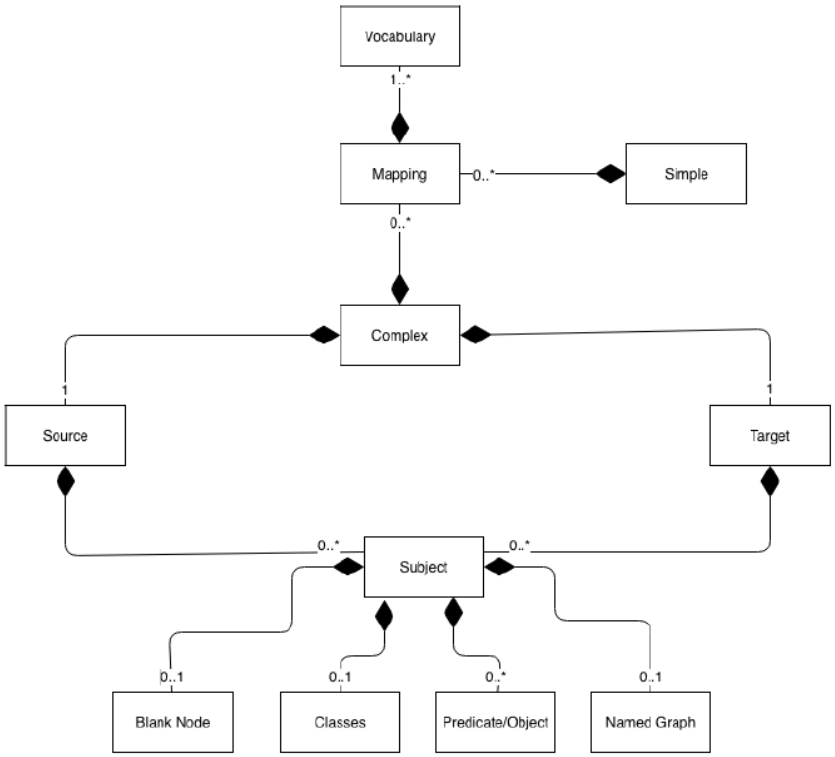

Figura 14 - Diagrama da aplicação Juma Interlink [16]

A aplicação ***Juma Interlink*** assemelha-se à nossa proposta na medida em que ambas geram mapeamentos *SPARQL* através da cláusula *CONSTRUCT*, no entanto, o conceito e resultado desta aplicação são muito diferentes da nossa proposta. De seguida, serão enumerados os aspetos em que esta aplicação e a nossa proposta divergem:

- **Conceito:** *Juma* foi proposta como uma solução que permite definir mapeamentos, não só entre dados de uma base de dados relacional para uma ontologia (através das aplicações ***Juma R2RML*** e ***Juma Uplift***), como também entre duas ontologias (***Juma Interlink***). Por sua vez, a ferramenta *SMA* foi proposta como uma solução para definir mapeamentos entre duas ontologias.

  Em relação aos tipos de mapeamentos permitidos, na ferramenta *SMA* nós lidamos com mapeamentos simples e complexos apenas para os casos mais usuais. Além disso, nós não lidamos com mapeamentos **N:M** (*muitos-para-muitos*), ao contrário do *Juma Interlink*.

  No exemplo da Figura 15, podemos visualizar um mapeamento **N:1**, ou seja, duas propriedades alvo são mapeadas a partir da mesma propriedade fonte. O mapeamento aqui apresentado é considerado um mapeamento complexo, em que o valor da propriedade fonte *foaf:name* é fracionado, através de uma função de

transformação, para compor os valores das propriedades *foaf:familyName* e *foaf:GivenName*. Na ferramenta *SMA*, nós não lidamos diretamente com mapeamentos **N:1**, ou seja, este mapeamento específico não é possível de efetuar da forma como é representado aqui. Para conseguir efetuar mapeamentos **N:1** na ferramenta *SMA*, é necessário criar mapeamentos distintos, tantos quanto o número representado por N.

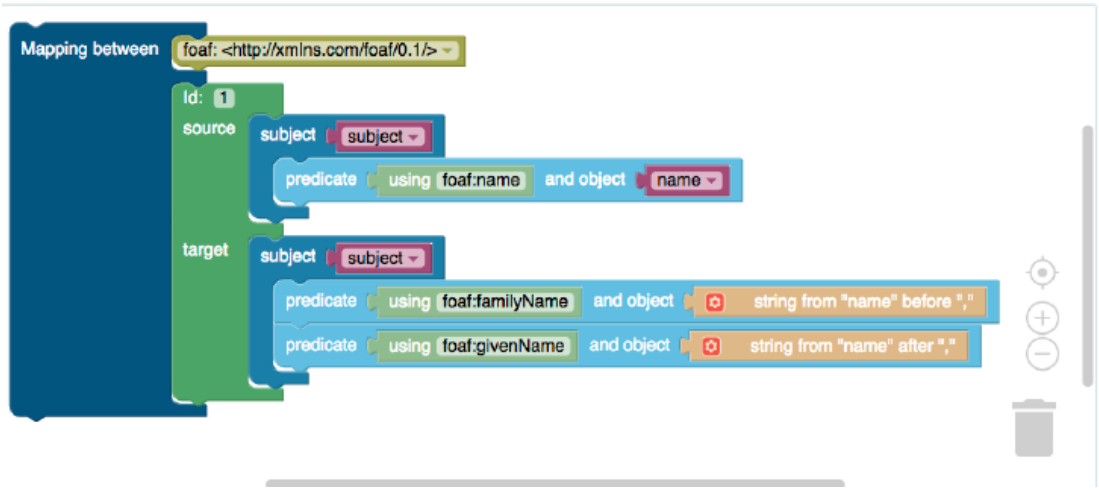

Figura 15 - Representação visual de um mapeamento complexo na aplicação Juma Interlink [16]

No nosso ponto de vista, as aplicações criadas com base no conceito *block metaphor*, podem dificultar a leitura dos mapeamentos quando muitos conceitos devem ser mapeados, o que pode tornar essa abordagem menos intuitiva que a ferramenta *SMA*. Por exemplo, na Figura 15, se tivessemos uma expressão de caminho, o bloco azul claro referente à propriedade fonte teria que ser expandido, de modo a albergar essa informação, o que, por sua vez, iria dificultar a leitura das diversas informações.

- **Resultado:** *Juma Interlink* gera os mapeamentos *SPARQL*, mas não os triplos *RDF*. A ferramenta *SMA* gera, não só os mapeamentos *SPARQL* e os triplos *RDF*, como também as regras de mapeamentos e as assertivas de mapeamento.

3.1.2 *SPINMap*

SPINMap[13] é um recurso disponibilizado por uma solução empresarial designada por *TopBraid*[14], pertencente à empresa *TopQuadrant*. Este recurso permite representar mapeamentos entre ontologias *RDF/OWL*, através da linguagem *SPARQL* [17].

SPINMap é um vocabulário composto por uma coleção de padrões de projeto reutilizáveis, sendo estes padrões construídos utilizando a linguagem *SPARQL Inferencing Notation* (*SPIN*)[15] [45]. Como o *SPINMap* é apenas um vocabulário, toda a interface gráfica disponibilizada ao utilizador é fornecida através do ambiente *TopBraid*. Na Figura 16, podemos observar a interface gráfica disponibilizada para efetuar mapeamentos.

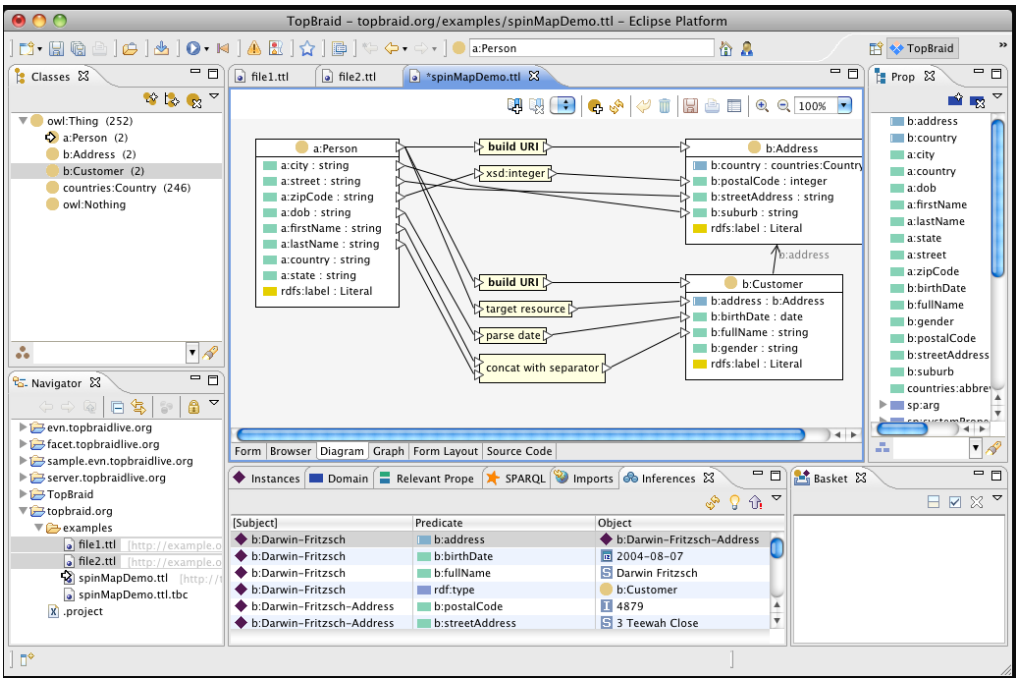

Figura 16 - Interface gráfica do ambiente TopBraid [45]

O *SPINMap* é semelhante à nossa proposta porque para efetuar mapeamentos entre ontologias utiliza a linguagem *SPARQL* e, além disso, também utiliza o conceito de padrões de projeto para permitir esses mapeamentos. Entretanto, não foi possível efetuar um levantamento mais pormenorizado deste vocabulário, especialmente no que toca aos padrões de mapeamento disponíveis, pois como se trata de uma solução empresarial, a informação disponibilizada é pouca. Além disso, não nos foi possível testar as nossas ontologias de estudo para apurar se este recurso possuía as mesmas funcionalidades que a nossa ferramenta, porque apenas tivemos

[13] SPINMap https://www.topquadrant.com/spinmap-sparql-based-ontology-mapping-with-a-graphical-notation/
[14] TopBraid – Trata-se de um ambiente de modelação. Para mais informações, consultar: https://www.topquadrant.com/products/topbraid-composer/
[15] *SPIN* https://www.w3.org/Submission/spin-overview/

acesso à versão *demo* da solução empresarial que não contempla interface gráfica utilizada em todos os tutoriais que estão disponíveis para consulta acerca deste recurso.

3.1.3 *LDIF*

LDIF é um *framework* desenvolvido em *Scala* e licenciado pela *Apache Software License*, que define um *workflow* para a integração de dados em *Linked Data*, gerindo os fluxos de dados entre cinco etapas diferentes [15], como podemos observar na Figura 17.

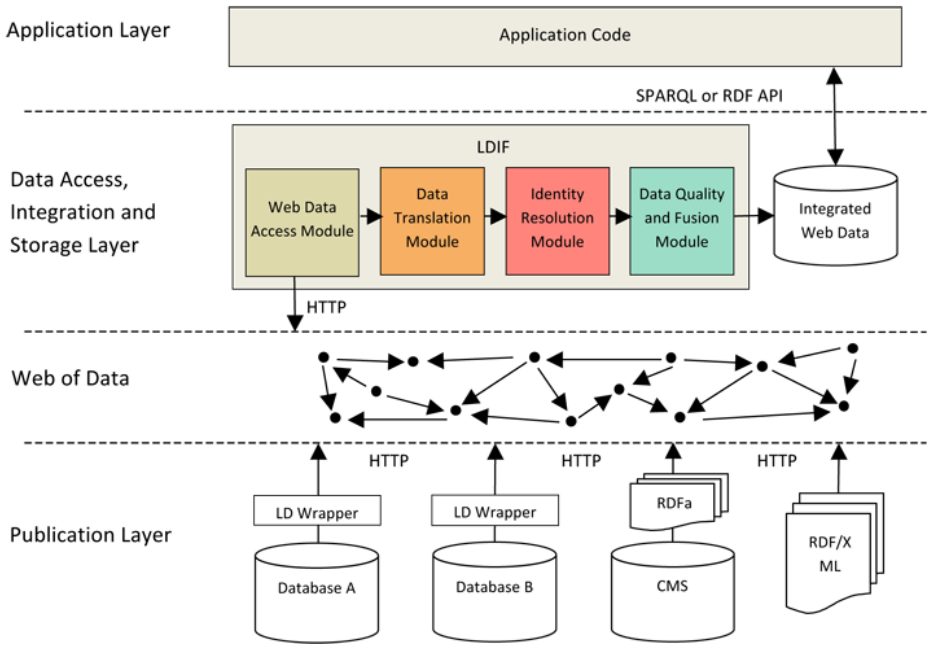

Figura 17 - Arquitetura do framework LDIF [15]

De uma forma muito resumida, as cinco etapas são [15]:

1. **Recolher dados:** é utilizado o módulo *Web Data Access Module* para coletar dados de diversas fontes como, por exemplo, dados provenientes do *SPARQL Endpoint* ou de ficheiros com um dos seguintes formatos: *RDF/XML*, *N-Triples*, *N-Quads* e *Turtle*;
2. **Mapear dados para o *schema*:** o módulo correspondente é o *Data Translation Module* e, é responsável por efetuar a tradução de dados da *web* expressados em termos de diferentes vocabulários, num único vocabulário de destino. Para tal, emprega a linguagem de mapeamento *R2R* (abordada na secção 3.1.5).
3. **Resolver identidades:** o módulo responsável por resolver identidades é o *Identity Resolution Module*. O *LDIF* faz uso do *framework Silk Link Discovery*[16] para descobrir os diferentes *URIs* que são utilizados em diversas fontes de dados

[16] *Silk Link Discovery* https://app.assembla.com/spaces/silk/wiki/Home

para identificar uma mesma entidade. O resultado do *framework Silk* é um conjunto de duplicados, sendo que o *LDIF* pega nesse conjunto e substitui todos os *alias* por um único *URI* de destino nos dados de saída;

4. **Avaliar a Qualidade e Fusão dos Dados:** *LDIF* emprega o *Sieve*[17] para fornecer uma limpeza e avaliação da qualidade dos dados. O procedimento de limpeza de dados funciona em duas etapas: **Avaliação de Qualidade** e **Fusão dos Dados.**

   Segundo Mendes et al. [46], a avaliação de qualidade dos dados é efetuada de forma configurável pelo utilizador pois este possui a liberdade de decidir quais os indicadores de qualidade em que o *Sieve* se deve focar. A segunda etapa, a fusão de dados, recebe como entrada as pontuações de qualidade e resolve os conflitos existentes entre valores de propriedades, consoante as pontuações recebidas.
5. **Saída de dados:** Atualmente o *LDIF* suporta dois formatos de ficheiros: ***N-Quads*** e ***N-Triples***.

Apesar deste *framework* se ter destacado dos demais devido ao seu bom desempenho e *performance*, possui uma grande limitação em relação à criação de mapeamentos entre ontologias. Estes mapeamentos são efetuados de forma manual pelo utilizador, o que leva a que o mesmo tenha de conhecer a linguagem de mapeamento *R2R*. Além disso, como o mapeamento é efetuado pelo utilizador, está mais suscetível a erros, sobretudo quando é necessário efetuar mapeamentos mais complexos, por exemplo, entre uma classe e uma propriedade.

3.1.4 *RBA*

A ferramenta *RBA* foi uma solução criada por Vinuto [14] com o objetivo de simplificar a tarefa de gerar mapeamentos *R2R* através da utilização de *AMs* e padrões de mapeamento. A nossa abordagem parte do mesmo princípio da ferramenta *RBA*, logo utilizamos os padrões de mapeamento propostos nessa ferramenta, adaptando os seus *templates* de mapeamento para a linguagem *SPARQL*.

Segundo Vinuto [14], a ferramenta *RBA* pode ser vista como um complemento para o *framework LDIF*, já que se preocupa essencialmente com a criação de mapeamentos entre ontologias de forma mais automática e *user-friendly* para o utilizador que a que está disponível no *LDIF*.

[17] *Sieve* http://sieve.wbsg.de/

Na Figura 18, é apresentada a arquitetura da ferramenta *RBA*, que é composta por quatro módulos: ***Graphical User Interface (GUI)***, ***GRM***, ***GR2R*** e o ***Mecanismo de mapeamento R2R***. O módulo *GUI*, *GRM* e *GR2R* são executados pelo *framework RBA*, enquanto que o módulo designado por mecanismo de mapeamento *R2R* é executado pelo *framework R2R*.

De acordo com Vinuto [14], os quatro módulos são responsáveis por:

- ***GUI*:** módulo onde o utilizador configura os esquemas fonte e alvo, sendo o resultado deste módulo um conjunto de *AMs* que são enviadas para o módulo *GRM* e o *GR2R*;
- ***GRM:*** módulo responsável por gerar as regras de mapeamento;
- ***GR2R:*** módulo responsável por gerar automaticamente os mapeamentos *R2R*, através de algoritmos que viabilizam esse processo. Por sua vez, estes mapeamentos *R2R* gerados são enviados para o último módulo, o mecanismo de mapeamento *R2R*;
- **Mecanismo de mapeamento *R2R:*** é o único que é executado no *framework R2R*. Possui como parâmetro de entrada o resultado dos mapeamentos provenientes do módulo *GR2R* e executa-os para posteriormente os enviar para uma base de dados de triplos *RDF* local.

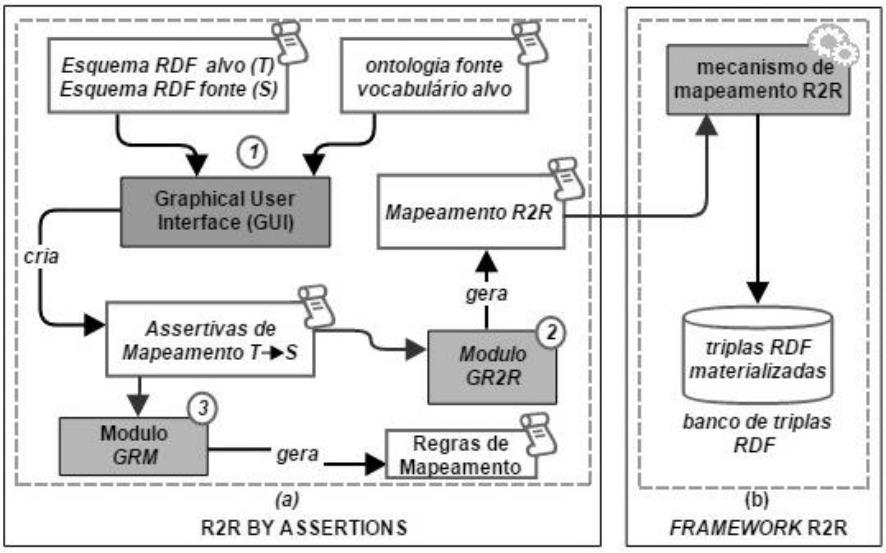

Figura 18 - Arquitetura da ferramenta RBA [14]

A arquitetura da nossa ferramenta (chamada *SMA*) foi desenvolvida tendo por base a arquitetura da ferramenta *RBA*, sendo que ambas possuem um módulo ***GUI*** e um módulo de criação de regras de mapeamento (designado por ***GRM*** na arquitetura da ferramenta *RBA* e ***CMS*** na nossa ferramenta). A grande diferença entre as duas arquiteturas reside no facto da *RBA* possuir um módulo responsável pela criação e execução de mapeamentos *R2R*, enquanto que a *SMA* possui um módulo para lidar com mapeamentos *SPARQL*.

Apesar da nossa proposta ser uma extensão do que foi desenvolvido pelo Vinuto em [14], preferimos criar a ferramenta *SMA* de raiz ao invés de utilizar a ferramenta *RBA*, devido a duas razões principais: **1)** o facto de ser necessário instalar um servidor de Base de Dados para a *RBA* funcionar adequadamente; e **2)** a ferramenta *RBA* ter sido desenhada como uma *desktop application*, sendo esta solução pouco apelativa e atrativa para os utilizadores dos dias atuais. Deste modo decidimos que, para abrangermos um maior número de utilizadores e tornar a aplicação mais atrativa, devíamos construí-la com uma *web application* e desenvolvê-la num formato mais apelativo e chamativo para os utilizadores. Além disso, decidimos também utilizar uma base de dados no modo embutido, de modo a que não fosse necessário instalar um servidor de Base de Dados.

3.1.5 Linguagem de Mapeamento R2R

R2R é uma linguagem declarativa, baseada na linguagem *SPARQL*, que foi projetada para publicar mapeamentos entre diferentes vocabulários *RDF* como *Linked Data* na *web* [42]. De acordo com Schultz e Bizer [42], os principais elementos da linguagem de mapeamento *R2R* são: os **Mapeamentos** (*mappings*), as **Transformações** (*transformations*) e os **Padrões Alvo e Modificadores de variáveis** (*target patterns and modifiers*).

3.1.5.1 Mapeamentos

Um mapeamento é sempre do tipo *r2r:mapping*, sendo que o seu objetivo é definir correspondências entre termos de um vocabulário alvo com termos do vocabulário fonte. Podem existir dois tipos de mapeamento: os **mapeamentos de classe** e os **de propriedade**.

Um mapeamento de classe define como uma classe no vocabulário alvo é expressa no vocabulário fonte, sendo especificado através da cláusula *r2r:classMapping*, enquanto que um mapeamento de propriedade define como uma propriedade no vocabulário alvo é expressa no vocabulário fonte, sendo especificado pela cláusula *r2r:propertyMapping* [42].

Segundo Pequeno et al. [47], qualquer mapeamento *R2R* possui duas cláusulas: *r2r:sourcePattern* e *r2r:targetPattern*, sendo que a primeira corresponde ao padrão de termos do vocabulário fonte e a segunda corresponde ao padrão de destino.

Na Tabela 6, estão expostas as principais cláusulas que são utilizadas nos mapeamentos *R2R* e respetivas definições.

Tabela 6 - Principais cláusulas dos mapeamentos R2R

Cláusulas	Definição
r2r:sourcePattern	Cada mapeamento deve possuir exatamente um *r2r:sourcePattern*, que expressa o padrão de termos do vocabulário fonte [42]. Quase todas as expressões válidas na cláusula *WHERE* da linguagem *SPARQL* são também permitidas nesta cláusula. Em todas as cláusulas *r2r:sourcePattern* está presente a variável de instância ?SUBJ que possui o objetivo de representar as instâncias que são o foco do mapeamento.
r2r:targetPattern	Cada mapeamento pode possuir um ou mais *r2r:targetPattern*, sendo que cada *r2r:targetPattern* pode conter um conjunto de triplos ou caminhos do vocabulário alvo.
r2r:preffixDefinitions	Um *r2r:preffixDefinitions* tem como principal objetivo abreviar *URIs.* Assim a definição dos prefixos que serão utilizados no mapeamento pode ser efetuada nesta cláusula de modo a tornar as demais cláusulas do mapeamento menos verbosas.
r2r:transformation	Define o modo como os valores que estão incluídos na cláusula *r2r:sourcePattern* serão transformados em valores da ontologia alvo [14]. Mais detalhes acerca desta cláusula serão apresentados na secção 3.1.5.2.
r2r:classMappingRef	Refere-se a uma cláusula *r2r:classMapping* definida anteriormente, sendo utilizada principalmente para reduzir a redundância [14].

3.1.5.2 Transformações

As transformações, como foi dito na Tabela 6, servem para definir o modo como os valores que estão incluídos na cláusula *r2r:sourcePattern* serão transformados em valores na ontologia alvo [14], sendo a cláusula *r2r:transformation* utilizada para o efeito. De acordo com Schultz e Bizer [42], as transformações são necessárias se o tipo de dados do valor existente no vocabulário fonte for diferente do tipo de dados que se deseja no vocabulário alvo. A linguagem *R2R* fornece um conjunto de funções que são utilizadas habitualmente, tais como funções aritméticas, de *String*, entre outras [42]. A Tabela 7 apresenta alguns exemplos de funções disponíveis na linguagem *R2R*.

Tabela 7 - Operadores permitidos na linguagem R2R

Tipo	Cláusulas	Significado	Exemplo
Aritméticos	*add()*	Soma dos vários valores passados como argumento	***add*(2 , 4 , 1) = 7**
	mod()	Resto da divisão de um número por outro	***mod*(5 , 2) = 1**
String	*concat()*	Concatenação dos vários valores passados como argumentos	***concat*('A', 'B') = 'AB'**
	split()	Divide uma *string* inicial em vários valores numa lista, dependendo da função *regex* definida	***split*('-', 'AB-C-D') = { 'AB', 'C', 'D' }**

Além do conjunto de funções, esta linguagem também disponibiliza uma sintaxe para efetuar diferentes ações dependendo de determinada condição. Basicamente tem o mesmo significado e aplicação que um construtor *If-Then-Else* definido em linguagens de programação, tais como o *Java*. A sintaxe disponibilizada é :

condição ? expressão se verdadeiro : expressão se falso

3.1.5.3 Padrões Alvo e Modificadores de variáveis

De acordo com Schultz e Bizer [42], um padrão alvo é um conjunto de triplos e/ou caminhos, em que os triplos podem conter *URIs*, literais, nós vazios e variáveis. Ao longo deste documento já foi possível entender o que são os *URIs*, os literais e as variáveis, faltando apenas explicar o significado de nós vazios (*blanked nodes*). Os nós vazios, como o próprio nome indica, são nós que não possuem qualquer valor e, apesar de o seu uso não ser recomendado, podem ser inseridos nos grafos *RDF*.

Os modificadores de variáveis definem a estrutura alvo e o vocabulário que será mapeado [14], sendo que a linguagem *R2R* oferece os seguintes modificadores de variáveis [42]:

- **Modificador de *URI:*** em que os valores das variáveis tornam-se *URIs*;

- **Modificador de literais:** em que os valores das variáveis tornam-se literais, ou seja, é o oposto dos modificadores de *URI*;
- **Modificador de idioma:** permitem adicionar uma *tag* de idioma a um literal;
- **Modificador de tipo de dados:** especificam qual o tipo de dados que os valores das variáveis devem assumir. Estes modificadores devem ser utilizados em conjunto com os modificadores de literais.

3.1.5.4 Exemplo de um Mapeamento em R2R vs SPARQL

Nesta secção, será apresentado um exemplo de um mapeamento de classe escrito tanto na linguagem *R2R* (Figura 19) como na linguagem *SPARQL* (Figura 20).

```
1. p: dbo_Agent_to_foaf_Agent
2. a r2r:classMapping;
3.        r2r:prefixDefinitions “dbo:<...>.foaf:<...>”;
4.        r2r:sourcePattern “?SUBJ a dbo:Agent”;
5.        r2r:targetPattern “?SUBJ a foaf:Agent”.
```

Figura 19 - Exemplo de um mapeamento de classe em R2R entre dbo:Agent e foaf:Agent

```
1. PREFIX foaf: <...>
2. PREFIX dbo: <...>
3. CONSTRUCT { ?SUBJ a foaf:Agent . }
4. WHERE { ?SUBJ a dbo:Agent . }
```

Figura 20 - Exemplo de um mapeamento de classe em SPARQL entre dbo:Agent e foaf:Agent

Na Figura 19 apresentamos o mapeamento *R2R* entre as classes *dbo:Person* e *foaf:Person*, que fazem parte, respetivamente, da ontologia *DBpedia*[18] e do vocabulário *FOAF*[19] *(Friend of a Friend)*. A linha 3 identifica quais os prefixos que serão utilizados nas cláusulas *r2r:sourcePattern* e *r2r:targetPattern*. Na Figura 20, podemos observar este mesmo mapeamento escrito em *SPARQL*. Este exemplo trata-se de um mapeamento de classe muito simples, em que todas as instâncias de *dbo:Person* serão mapeadas em instâncias de *foaf:Person*.

Na comparação anterior podemos verificar que, apesar da linguagem *SPARQL* não ter sido desenvolvida exclusivamente como linguagem de mapeamento, tal como aconteceu com

[18] DBpedia http://dbpedia.org/ontology/
[19] FOAF http://xmlns.com/foaf/spec/

a linguagem *R2R*, consegue ser expressiva o suficiente para expressar mapeamentos entre ontologias.

3.2 Alinhamento de Ontologias

Alinhamento de Ontologias (do inglês *ontology alignment* e, em alguns casos, também conhecida por *ontology matching*) é um processo que determina correspondências entre termos de duas ontologias, para posteriormente gerar vínculos que definem qual o elemento de uma ontologia é similar ao elemento de outra [48] [49].

De acordo com Vinuto [14], a esmagadora maioria dos trabalhos que são realizados nesta área propõem estratégias que permitem a geração de correspondências entre as ontologias de forma semiautomáticas.

Contrariamente ao mapeamento entre ontologias, o resultado de um alinhamento de ontologias é composto por duas ontologias separadas com ligações entre os termos equivalentes ou similares. Um *overview* sobre várias propostas para realizar alinhamento de ontologias pode ser encontrado em [48] [49] [18]. No presente trabalho vamos apenas apresentar a ferramenta *COMA 3.0* [50].

3.2.1 *COMA 3.0*

COMA 3.0[20] é uma extensão das ferramentas *COMA* e *COMA ++* e possui uma administração do *workflow* aprimorado e recursos adicionais como, por exemplo, fusão de ontologias [50].

Na Figura 21 podemos observar que a arquitetura desta ferramenta está dividida em quatro módulos, sendo estes: ***Storage***, ***Match Execution***, ***Mapping Processing*** e ***User Connection***.

[20] COMA 3.0: https://dbs.uni-leipzig.de/de/Research/coma.html

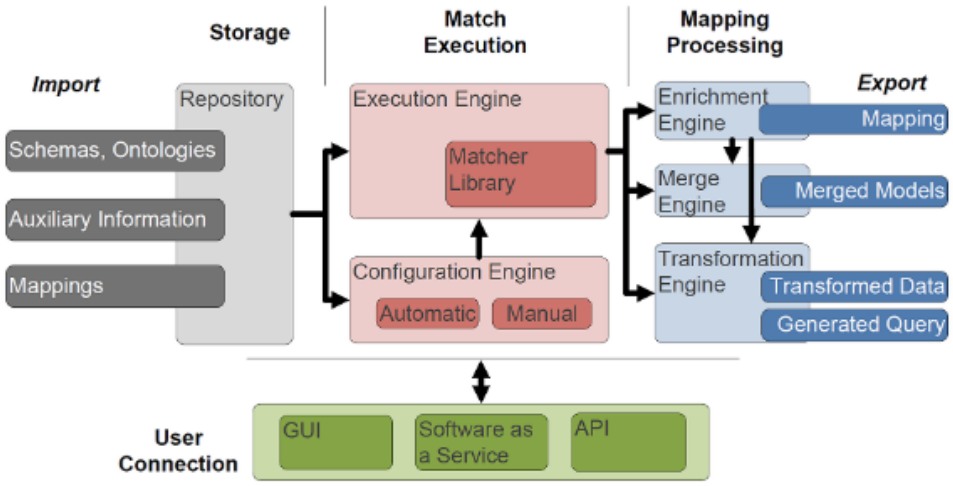

Figura 21 - Arquitetura da ferramenta COMA 3.0 [50]

De um modo resumido, os quatro módulos são responsáveis por [50]:

- **Armazenamento *(Storage)*:** módulo responsável por carregar os esquemas, ontologias, mapeamentos existentes e quaisquer informações adicionais no repositório, onde serão armazenados de forma persistente;
- **Execução de correspondência *(Match Execution)*:** módulo responsável por aplicar estratégias de correspondências e apresentar resultados de correspondências entre ontologias. Este módulo engloba o principal componente da ferramenta designado por *mecanismo de execução* (*execution engine*). O *mecanismo de execução* obtém como *input* dois esquemas ou ontologias, executa vários algoritmos de correspondências e calcula o resultado da mesma. Por outras palavras, este módulo retorna uma lista de correspondências entre as ontologias;
- **Processamento de Mapeamento *(Mapping Processing)*:** é o módulo responsável por executar tarefas adicionais após o resultado do mecanismo de correspondência como, por exemplo, detetar correspondências complexas para enriquecer automaticamente mapeamentos;
- **Conexão do Utilizador *(User Connection)*:** módulo composto por três soluções para que os utilizadores possam utilizar a ferramenta, sendo estas: **Interface Gráfica *(GUI)***, ***Software as a Service (SaS)*** e ***API***. Para a grande maioria dos utilizadores, a interface gráfica é a solução mais fácil e prática de utilizar esta ferramenta.

É importante notar que a ferramenta *COMA 3.0* tem como objetivo estabelecer uma correspondência entre os termos das ontologias, enquanto que a nossa ferramenta (*SMA*), visa

fazer o mapeamento entre uma ontologia alvo e uma ontologia fonte. Assim, para além de fazer a "correspondência" entre os vocabulários das ontologias envolvidas, necessitamos definir como obtemos os triplos *RDF* para a ontologia alvo a partir dos triplos *RDF* da ontologia fonte. Deste modo, podemos afirmar que apesar do contexto de utilização da ferramenta *COMA 3.0* e da *SMA* serem diferentes, estas ferramentas complementam-se.

3.3 Padrões de Projeto em Ontologias

Sempre que existe um problema em qualquer área de conhecimento, é muito raro que sejam propostas novas soluções completamente diferentes das que já existem, já que na maioria das vezes investiga-se soluções de problemas parecidos, de modo a reutilizá-las na resolução do novo problema. Deste modo, podemos afirmar que um padrão de projeto descreve um problema particular que acontece em determinados contextos e apresenta uma solução que é baseada em soluções utilizadas para problemas similares. O termo Padrão de Projeto de Ontologias (do inglês, *Ontology Design Patterns*) pode ser definido como uma solução de modelação para resolver um problema comum [51] [52].

OntologyDesignPatterns.org[21] é um portal dedicado ao desenvolvimento e criação de padrões para a Web Semântica, tendo sido iniciado no âmbito do projeto *NeOn*[22] [52]. Este portal permite que os utilizadores consultem as listas de padrões existentes, bem como os tipos de padrões, domínios, entre outros. Além das diversas funcionalidades em torno do consultar, também disponibiliza novas notícias e/ou eventos dentro da área e permite contribuir para a expansão deste portal através, por exemplo, do envio de um problema de modelação que até hoje não foi resolvido. Neste portal, os padrões relacionados com a nossa proposta encontram-se na categoria: ***Category:AlignmentOP***.

De acordo com Vinuto [14], existem poucos trabalhos de padrões de projeto na área de alinhamento de ontologias. Em relação à área de mapeamento entre ontologias, destacamos dois projetos, *Relational Database to RDF Mapping Patterns* apresentado em [53] e a ferramenta *RBA* (abordado na secção 3.1.4).

3.4 Considerações finais do capítulo

Neste capítulo apresentámos alguns dos trabalhos que pretendem mitigar os problemas relacionados com a heterogeneidade entre ontologias, em que o nosso foco se baseou nas

[21] OntologyDesignPatterns.org http://ontologydesignpatterns.org/wiki/Main_Page
[22] NeOn http://neon-project.org/nw/Welcome_to_the_NeOn_Project.html

seguintes abordagens: *Mapeamento entre Ontologias* e *Alinhamento de Ontologias*. Estas abordagens, embora semelhantes, podem ser distinguidas com base em dois critérios: os dados de entrada e o resultado.

Em termos de dados de entrada, o mapeamento entre ontologias apenas aceita duas ontologias, enquanto que no caso do alinhamento de ontologias podemos ter duas ou mais ontologias.

Em termos de resultados, o alinhamento de ontologias retorna as mesmas ontologias com vínculos de correspondência entre os termos, enquanto que o mapeamento entre ontologias retorna uma única e nova ontologia.

Neste capítulo também introduzimos os *Padrões de Projeto em Ontologias* que visam auxiliar a resolução de problemas relacionados com a heterogeneidade entre ontologias. A nossa abordagem, tal como referido anteriormente, assenta na abordagem da ferramenta *RBA*, tratando de uma extensão a esse mesmo trabalho. Aos padrões originalmente criados, nós acrescentámos *templates* que permitam a geração de mapeamentos utilizando a linguagem *SPARQL*.

4 Representação e Padrões de Mapeamentos

Este capítulo é constituído por dois tópicos: **(1)** introdução do formalismo usado para definir mapeamentos entre ontologias; e **(2)** apresentação dos padrões de mapeamentos de ontologias. O formalismo usado e os padrões de mapeamentos foram desenvolvidos por Vinuto em [14]. O formalismo é apresentado nesta dissertação apenas para facilitar a compreensão da nossa proposta. Os padrões de mapeamento foram estendidos nesta proposta por acrescentar *templates* para a criação de mapeamentos usando a linguagem *SPARQL*.

A organização deste capítulo segue a seguinte estrutura:

- **Secção 4.1 – Representação de Mapeamentos:** é introduzido o formalismo utilizado na definição dos mapeamentos entre ontologias;
- **Secção 4.2 – Estudo de caso:** é apresentado o nosso estudo de caso, em que são introduzidos os fragmentos das ontologias que iremos utilizar ao longo desta dissertação para exemplificar os diversos tipos de mapeamentos expostos;
- **Secção 4.3 – Padrões de Mapeamento:** são introduzidos os padrões de mapeamento. Para não tornar os padrões excessivamente verbosos, e assim facilitar a sua leitura e compreensão, optámos por não apresentar os *templates R2R* definidos em [14]. Ao invés destes *templates*, apresentamos os *templates SPARQL*, pois são a principal contribuição deste trabalho;
- **Secção 4.4 – Considerações finais do capítulo:** são apresentadas as considerações finais deste capítulo.

4.1 Representação de Mapeamentos

O mapeamento entre ontologias tem como objetivo transformar as instâncias de uma ontologia fonte em instâncias de uma ontologia alvo. O formalismo de mapeamento usado por Vinuto em [14] utiliza regras de mapeamento para realizar essa tarefa. Segundo o autor, estas são muito mais simples de serem utilizadas que outras linguagens de mapeamento tais como, por exemplo, o *SWRL*[23] ou *R2R*, no entanto, são suficientes para capturar mapeamentos expressivos e complexos.

De seguida, iremos apresentar e transcrever o formalismo introduzido em [14].

"Seja **V** um *vocabulário*, que é um conjunto de *classes* e *propriedades*. Uma *ontologia* é um par $\mathbf{O} = (\mathbf{V}, \Sigma)$**,** em que Σ representa um conjunto finito de fórmulas em **V**, ou seja, as *restrições* de **O**.

[23] SWRL https://www.w3.org/Submission/SWRL/

Seja $\mathbf{V_T}$ um *vocabulário alvo* e $\mathbf{O_S} = (\mathbf{V_S}, \Sigma_\mathbf{S})$ uma *ontologia fonte* com $\mathbf{V_S}$ e $\Sigma_\mathbf{S}$ sendo respetivamente, o vocabulário da fonte e o conjunto de restrições de $\mathbf{O_S}$. Seja $\mathcal{X}$ um conjunto de variáveis. Seja $\mathcal{C}$ um alfabeto de primeira ordem consistindo de um conjunto **F** de símbolos de função e um conjunto $\mathcal{P}$ de símbolos de predicado, respetivamente chamados de *símbolos de função concreto* e *símbolos de predicados concretos*.

Os símbolos de função 0-ários são chamados de *constantes*, que incluem os IRIs e valores de tipo de dados. Assumimos que os símbolos em $\mathcal{C}$ têm uma interpretação fixa. Por último, nós supomos que $\mathcal{X}$ e $\mathcal{C}$ são mutuamente disjuntos e que $\mathcal{C}$ também é disjunto de $\mathbf{V_T}$ e $\mathbf{V_S}$.

Um *termo* é uma expressão recursivamente construída a partir de símbolos de função, constantes e variáveis, como de costume. Um *literal* é uma expressão de uma das formas:

- Uma *classe literal* da forma $C(t)$: onde C é uma classe de $\mathbf{V_T} \cup \mathbf{V_S}$ e t é um termo;
- Uma *propriedade literal* $P(t, u)$: onde P é uma propriedade em $\mathbf{V_T} \cup \mathbf{V_S}$ e t e u são termos;
- $u = f(t_1, \ldots, t_n)$: onde f é um símbolo de função n-ária em **F** e $u, t_1, \ldots, t_n$ são termos;
- $\mathbf{p}(t_1, \ldots, t_n)$: onde $\mathbf{p}$ é um símbolo de predicado n-ário de $\mathcal{P}$ e $t_1, \ldots, t_n$ são termos.

Os literais que usam símbolos de função binária (ou predicado) concretos podem ser escritos em notação infixa, por conveniência sintática. Um padrão de *triplo* é uma classe ou propriedade literal. Dizemos que um *triplo* $\boldsymbol{t}$ se relaciona com um *padrão de triplo* $\mathbf{p}$ sse:"

- $\mathbf{p}$ é uma classe literal da forma $C(x)$: onde x é uma variável e $\boldsymbol{t}$ é da forma (*s* rdf:type C);
- $\mathbf{p}$ é uma propriedade literal da forma $P(x, y)$: onde x e y são variáveis e $\boldsymbol{t}$ é da forma (*s P o*).

"É necessário ter atenção que um triplo não relaciona um literal da forma $u = f(t_1, \ldots, t_n)$ ou $\mathbf{p}(t_1, \ldots, t_n)$, onde f é um símbolo de função n-ária em **F** e $\mathbf{p}$ um símbolo de predicado n-ário em $\mathcal{P}$. O *corpo de uma regra B* é uma lista de literais, separados por vírgulas. Quando necessário, usa-se "$B[x_1, \ldots, x_k]$" para indicar que as variáveis $x_1, \ldots, x_k$ ocorrem em *B*. Dizemos que *B* é sobre um vocabulário **V** sse todas as classes e propriedades que ocorrem em *B* são de **V**.

Como uma conveniência de notação, o *corpo de uma regra de B* pode incluir:

1. Alguns *caminhos de propriedades SPARQL*, seja em notação prefixa ou infixa;
2. Alguns operadores *SPARQL unários*, *binários* ou *ternários*, seja em notação prefixa ou infixa."

Na Tabela 8 apresentamos os caminhos de propriedade que são permitidos neste formalismo, logo são permitidos também na nossa proposta. "Nesta tabela, $P, P_1, P_2, \dots, P_k$ são propriedades e $x_1, x_2, \cdots, x_k$ são variáveis que não ocorrem no corpo da regra *B*." Os operadores lógicos usados e permitidos neste formalismo são os mesmos que são utilizados na linguagem *SPARQL*. Na página 44 desta dissertação, foram apresentados os principais operadores lógicos usados na linguagem *SPARQL*.

Tabela 8 - Expressões de caminho permitidos [14]

Tipo de Propriedade de Caminho	**Notação**	**Tradução**
Caminho inverso	$P\hat{}(t_1, t_2)$	$P(t_2, t_1)$
	$t_1\ P\hat{}\ t_2$	
Caminho de Sequência	$P_1\ /\ P_2\ /\ \dots\ /\ P_k(t_1, t_2)$	$P_1(t_1, x_2); P_2(x_2, x_3); \dots Pk(x_k, t_2)$
	$t_1 P_1\ /\ P_2\ /\ \dots\ /\ P_k t_2$	
Caminho de comprimento fixo ($k \geq 0$)	$P\{k\}(t_1, t_2)$	$P(t_1, x_2); P(x_2, x_3); \dots P(x_k, t_2)$ (P repetidas k vezes)
	$t_1 P\{k\} t_2$	

"Uma *regra de mapeamento* de $\mathbf{O_S} = (\mathbf{V_S}, \Sigma_\mathbf{S})$ para $\mathbf{V_T}$ é uma expressão de uma das seguintes formas:

- $C(x) \leftarrow B[x]$**:** é designado de *mapeamento de classe*, onde C é uma classe em $\mathbf{V_T}$ e B[x] é o corpo de uma regra sobre $\mathbf{V_S}$;
- $P(x, y) \leftarrow B[x, y]$**:** é designado por *mapeamento de propriedade*, em que P é uma propriedade em $\mathbf{V_T}$ e $B[x, y]$ é o corpo de uma regra sobre $\mathbf{V_S}$."

A expressão que se encontra à esquerda da seta é designada por *alvo*, enquanto que a expressão que se encontra à direta é designada por *fonte*.

"Um *mapeamento simples* é uma regra de mapeamento que pode assumir uma das seguintes formas", dependendo se se trata de um mapeamento de classes ou propriedades:

- **Mapeamento de classe:** "$C_\mathbf{T}(x) \leftarrow C_\mathbf{S}(x)$, onde $C_\mathbf{T}$ é uma classe em $\mathbf{V_T}$ e $C_\mathbf{S}$ é uma classe em $\mathbf{V_S}$";

- **Mapeamento de propriedade:** "$P_{\mathbf{T}}(x, y) \leftarrow C_{\mathbf{S}}(x); P_{\mathbf{S}}(x, y)$, onde $P_{\mathbf{T}}$ é uma propriedade em $\mathbf{V_T}$, $P_{\mathbf{S}}$ é uma propriedade em $\mathbf{V_S}$ e $C_{\mathbf{S}}$ é o domínio de $P_{\boldsymbol{S}}$, definido na ontologia fonte $\mathbf{O_S}$."

4.2 Estudo de caso

A fim de clarificar os conceitos recém expostos, elaboramos um estudo de caso que assenta em duas ontologias: **MyBook** e **DBpedia**[24] e retrata um cenário acerca de obras literárias. Neste estudo, **MyBook** é a ontologia alvo e **DBpedia** é a ontologia fonte. Os fragmentos das ontologias **DBpedia** e **MyBook** estão apresentados, respetivamente, na Figura 22 e Figura 23. Em relação à notação utilizada, tanto a ontologia fonte como a alvo estão representadas através de diagramas *Unified Modeling Language* (*UML)*, em que os retângulos representam as classes e, dentro dos mesmos, estão descritas as respetivas propriedades de tipo de dados associadas às mesmas. Além disso, as propriedades de objeto estão apresentadas na forma de setas entre o seu domínio e contradomínio. Por outras palavras, a origem da seta representa o *rdfs:domain* da propriedade, enquanto que o destino da seta representa o seu *rdfs:range*.

A ontologia fonte escolhida, a **DBpedia**, cobre diversas áreas de conhecimento, tais como a área da música, do desporto, entre outros. Das diversas áreas, decidimos abordar a área de conhecimento que envolve as obras literárias, sendo que o fragmento por nós utilizado permite-nos obter informações acerca dos livros e respetivos autores e editoras. Tal como observado na Figura 22, este fragmento é composto por quatro classes, sendo estas: *dbo:Book*, *dbo:Person*, *dbo:Language* e *dbo:Agent*. A classe *dbo:Book* contém informações acerca da obra literária, tais como o *isbn* e o nome da obra. A classe *dbo:Language* contém informações relativas ao idioma em que a obra literária está escrita. A classe *dbo:Person*, apesar de possuir informações acerca de todas as pessoas, foi restringida para apenas manter informações acerca dos autores das obras literárias e a classe *dbo:Agent* mantém informação acerca da editora.

[24] DBpedia http://dbpedia.org/ontology/

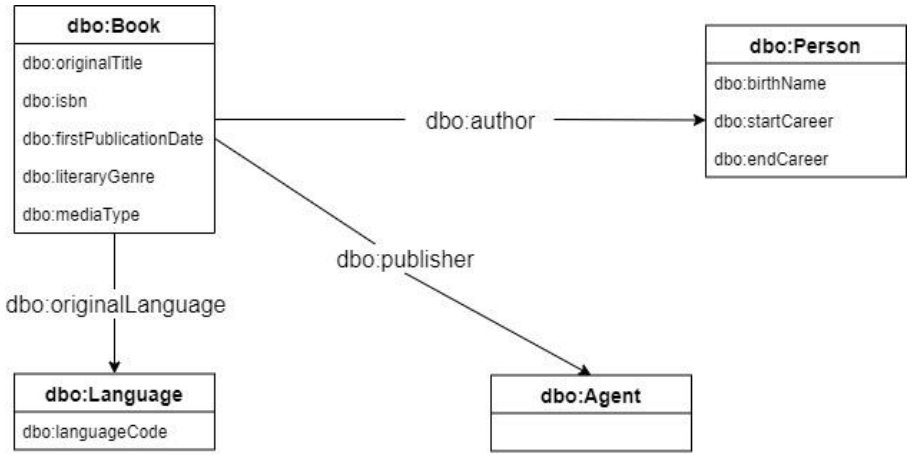

Figura 22 - Fragmento da ontologia fonte DBpedia

A ontologia alvo, a **MyBook**, permite manter informações acerca das características dos livros e respetivos autores. Esta ontologia utiliza três vocabulários, sendo estes: *Book*[25], *Dublin Core (DC)*[26] e o *FOAF*[27]. Além disso, utilizámos o prefixo "ex" para identificarmos o novo termo que definimos na ontologia **MyBook**, o *ex:careerDuration*. Este termo guarda informações sobre a duração da carreira de um autor. Na Figura 23, podemos constatar que **MyBook** é constituída por quatro classes: *bo:Book*, *dc:MediaType*, *bo:Author* e *foaf:Agent*. A classe *bo:Book* permite manter os dados acerca dos livros. A classe *dc:MediaType* permite identificar qual o formato em que o livro foi disponibilizado. Por último, as classes *bo:Author* e *foaf:Agent* permitem manter, respetivamente, as informações relativas aos autores dos livros e às editoras dos mesmos.

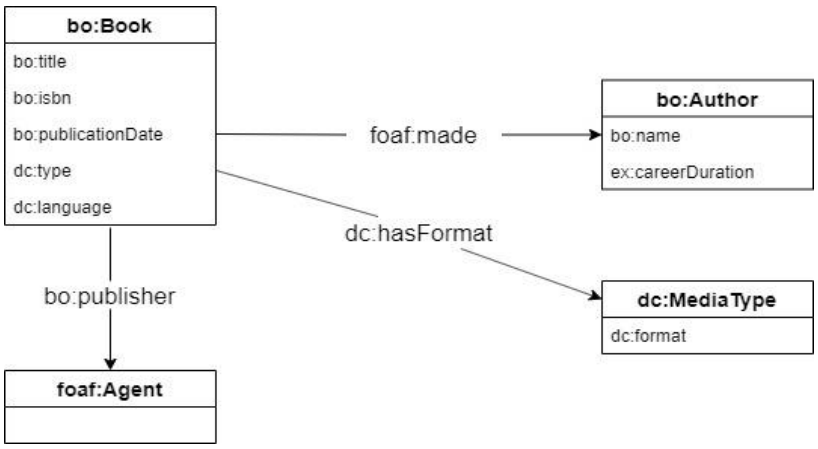

Figura 23 - Fragmento da ontologia alvo MyBook

[25] Book http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl
[26] Dublin Core https://www.dublincore.org/specifications/dublin-core/dcmi-terms/
[27] FOAF http://xmlns.com/foaf/spec/

4.2.1 *Exemplo de Regras de Mapeamento*

A partir do estudo de caso introduzido anteriormente, iremos explanar alguns exemplos de regras de mapeamento que têm por base a representação dos mapeamentos definidos na secção 4.1. Na Figura 24, são apresentados dois exemplos de regras de mapeamentos, designados por **R3** e **R6**:

(R3) *bo:Book(s)* ← *dbo:Book(s)* ; *dbo:literaryGenre(s,v)* != 'Romance'

(R6) *bo:isbn(s,v)* ← *dbo:Book(s)* ; *dbo:isbn(s,v)*

Figura 24 - Exemplo de regras de mapeamentos

A regra de mapeamento **R3** espelha um mapeamento complexo de classes, pois possui uma condição a ser satisfeita. **R3** mapeia a classe *bo:Book* para a classe *dbo:Book*, com a condição que o valor de *dbo:literaryGenre* não seja um romance. Na prática, este mapeamento permite definir que cada triplo de *dbo:Book*, que satisfaça o predicado *dbo:literaryGenre != "Romance"* irá produzir um triplo em *bo:Book*.

A regra de mapeamento **R6** é considerada uma regra de mapeamento simples de propriedades, por não possuir qualquer condição a ser satisfeita ou transformação a aplicar aos dados. **R6** mapeia a propriedade *bo:isbn*, cujo domínio é *bo:Book*, para a propriedade *dbo:isbn*, cujo domínio é *dbo:Book*. Este mapeamento permite indicar que cada triplo de *dbo:isbn* em *dbo:Book* irá produzir um triplo em *bo:isbn*, sendo que o objeto deste triplo irá ser o valor de *dbo:isbn*.

Além destes mapeamentos podemos definir uns ainda mais complexos, devido ao facto, por exemplo, de terem de ser efetuados através da utilização de funções de transformação. A Figura 25 apresenta dois exemplos de regras de mapeamentos mais complexos, designados por **R11** e **R4**:

(R11) *ex:careerDuration(s,v)* ← *dbo:Person(s)* ; *dbo:startCareer(s, v_1)* ;
dbo:endCareer(s, v_2) ; concat(v_1, '-', v_2, *v*)

(R4) *dc:MediaType(u)* ← *dbo:Book(s)* ; *dbo:mediaType(s,v)* ;
concat(*s*, xpath:encode-for-uri(*v*), *u*)

Figura 25 - Exemplo de regras de mapeamentos mais complexas

À semelhança da regra de mapeamento **R6**, a regra **R11** é uma regra de mapeamento de propriedades, sendo que a grande diferença entre elas reside no facto da regra de mapeamento **R11** permitir identificar que o valor da propriedade *ex:careerDuration*, cujo domínio é *bo:Author*, é conseguida através de uma função de transformação entre os valores das

propriedades *dbo:startCareer* e *dbo:endCareer*, cujo domínio de ambas é *dbo:Person*. Neste exemplo, esta função de transformação é apenas uma função de *string* em que concatenamos o valor de *dbo:startCareer* com o valor de *dbo:endCareer*.

A regra de mapeamento **R4** permite identificar um mapeamento entre uma propriedade e uma classe, em que a classe é *dc:MediaType* e a propriedade é *dbo:mediaType*, cujo domínio é *dbo:Book*. Este mapeamento indica que cada triplo *(s dbo:mediaType v)* produz um triplo *(u rdf:type dc:MediaType)*, em que o valor *u* é obtido através das funções *concat()* e *xpath:encode-for-uri()*. Estas duas funções permitem gerar um novo *URI* que tem por base o sujeito e o objeto do triplo de *dbo:mediaType*, logo existe a garantia que o *URI* gerado é sempre único.

Todas as regras de mapeamento possíveis entre os fragmentos das ontologias fonte e alvo apresentados, respetivamente, na Figura 22 e Figura 23, são mostradas no Anexo 01 – Regras de Mapeamento.

4.3 Padrões de Mapeamento

Tal como explicado na secção 3.3, um padrão de projeto de Ontologias pode ser definido como uma solução de modelação para resolver um problema comum. Ao adaptar esta frase para os padrões de mapeamento, podemos afirmar que cada padrão tem como objetivo representar uma solução genérica para um determinado problema de mapeamento.

Os padrões de mapeamento foram inicialmente propostos por Vinuto em [14]. Nesta secção iremos apresentar a nossa extensão a esses padrões, que consiste em permitir definir os mapeamentos utilizando a linguagem *SPARQL 1.1*. Esta versão da linguagem *SPARQL* traz inúmeras funcionalidades novas, incluindo, por exemplo, a atribuição de valores e de expressões de caminhos [41] utilizadas por nós aquando do desenvolvimento dos seguintes padrões.

Na secção 4.3.1 apresentamos o *template* proposto por Vinuto em [14] para descrever os padrões de mapeamento e na secção 4.3.2 introduzimos o catálogo de padrões de mapeamento.

4.3.1 *Template* para os padrões de mapeamento

Um *template* possui uma estrutura predefinida e tem como objetivo servir de modelo para o desenvolvimento e criação de conteúdo [54]. Apesar de ser uma definição genérica do conceito de *template*, podemos adaptá-la à nossa realidade afirmando que utilizar um *template*

para os padrões de mapeamento irá permitir, a partir de uma estrutura genérica, representar mapeamentos.

Na Tabela 9, podemos observar todos os constituintes do *template* para o padrão de mapeamento proposto. Este *template* foi adaptado do proposto por Vinuto em [14], sendo que por razões de clareza na leitura, decidimos não apresentar aqui os templates *R2R* presentes em [14].

Tabela 9 - Template para o padrão de mapeamento, adaptado de [14]

Nome	Indica qual o nome do padrão.
Problema	Representa uma breve descrição acerca dos objetivos do padrão.
Pré-condição	Especifica quais as condições necessárias para que seja possível aplicar o mapeamento.
Solução	Descreve a solução, através da utilização do formalismo de mapeamentos. Este formalismo é composto por: ➢ regras de mapeamento; ➢ assertivas de mapeamento; ➢ mapeamentos *SPARQL*.
Padrões Relacionados	Identifica os padrões relacionados com o mesmo.
Exemplos da solução proposta	Apresenta exemplos para a solução proposta.

Em relação ao formalismo que compõe os padrões de mapeamento (solução), já foram explicadas neste capítulo as regras de mapeamento e no Capítulo 2 foi introduzida a linguagem *SPARQL 1.1*. De acordo com Vinuto [14], uma Assertiva de Mapeamento (*AM*) é uma linguagem formal e declarativa utilizada para especificar mapeamentos entre ontologias e podem ser definidas de três maneiras diferentes: **Assertiva de Mapeamento de Classe** (*AMC*); **Assertiva de Mapeamento de Objeto** (*AMO*) e; **Assertiva de Mapeamento de Tipo de Dados** (*AMD*). Uma *AMC*, tal como o próprio nome indica, especifica um mapeamento de classe, enquanto que uma *AMO* e uma *AMD* especificam um mapeamento de propriedade. A principal diferença entre a *AMO* e a *AMD* é o tipo de predicado alvo, ou seja, a *AMO* especifica um mapeamento cujo predicado alvo é uma propriedade de objeto, e a *AMD* especifica um mapeamento cujo predicado alvo é uma propriedade de tipo de dados.

4.3.2 *Catálogos de padrões de mapeamento*

Segundo Vinuto [14], os padrões de mapeamento por ele definidos contemplam os tipos mais comuns de mapeamentos de ontologias. Estes padrões foram divididos em três categorias, como pode ser visualizado na Figura 26: **Padrões de Mapeamento de Classe**, **Padrões de Mapeamento de Propriedades de Objeto** e **Padrões de Mapeamento de Propriedades de Tipo de Dados**. Estes padrões serão detalhados nas secções a seguir.

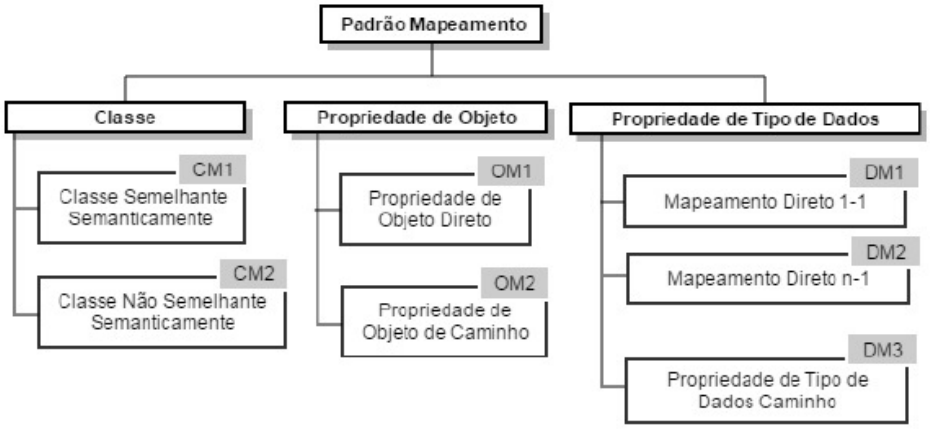

Figura 26 - Catálogo de Padrões de Mapeamento [14]

O catálogo de padrões de mapeamento definido por Vinuto em [14] está ligeiramente diferente do que apresentámos na Figura 26, pois em primeiro lugar adaptámos as siglas usadas pelo autor em inglês para português, ou seja, em vez de CM1 (Class Mapping 1), nós identificamos como MC1 (Mapeamento de Classe 1). Além disso, adaptámos o nome de alguns padrões como, por exemplo, o MO2, pois achamos que se torna mais percetível para o leitor.

Para que os *templates* sejam mais facilmente compreendidos, destacamos algumas conversões de formatação usadas nos mapeamentos *SPARQL*:

- O cardinal # antes dos elementos significa que os mesmos são comentários;
- Todas as palavras que se encontram a **negrito** são palavras reservadas do *SPARQL* como, por exemplo: **PREFIX**;
- As variáveis prefixExp, queryExp, uriExp e functionExp armazenam informações necessárias para a criação dos mapeamentos, em que prefixExp armazena os prefixos que estão presentes nos elementos de uma *AM* e queryExp, uriExp e functionExp armazenam todas as informações necessárias para obter os dados da ontologia fonte e transformá-los, quando necessário, para os dados da ontologia alvo. Isto inclui, por exemplo, construir um filtro, aplicar uma função de transformação, entre outros.

4.3.2.1 Padrões de Mapeamento de Classe

Nesta subsecção serão apresentados os dois mapeamentos de classe, que são: *Mapeamento de Classes Semelhantes Semanticamente* e o *Mapeamento de Classes Não Semelhantes Semanticamente*, sendo que correspondem, respetivamente, aos Padrões de Mapeamento CM1 e CM2 definidos em [14].

O *Mapeamento de Classes Semelhantes Semanticamente* ocorre quando o mapeamento é efetuado entre duas classes, ou seja, uma instância de uma classe representa o mesmo objeto que uma instância de uma outra classe, enquanto que o *Mapeamento de Classes Não Semelhantes Semanticamente* ocorre quando o mapeamento é efetuado entre instâncias que não representam o mesmo objeto, isto é, numa ontologia o objeto é representado por uma classe, enquanto que noutra ontologia é representado por uma propriedade.

Padrão de Mapeamento MC1

Nome: Padrão de Mapeamento de Classes Semelhantes Semanticamente (MC1)

Problema: "Como especificar o mapeamento de instância de uma classe numa ontologia fonte em instâncias de uma classe numa ontologia alvo, que representam o mesmo objeto do mundo real?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- **f** é um predicado de seleção sobre instâncias de $C_{\mathbf{S}}$, ou seja, identifica a condição que necessita de ser satisfeita sobre as instâncias de $C_{\mathbf{S}}$, sendo que se trata de uma cláusula opcional.

Solução:

- **Regra de Mapeamento:** $C_{\mathbf{T}}(s) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}\ (s)$
- **Assertiva de Mapeamento:** $\psi: C_{\mathbf{T}} \equiv C_{\mathbf{S}}\ /\ \mathbf{f}$

- ***Template* de Mapeamento *SPARQL*:**

```
# Template T1 – Mapeamento de Classe
# AMC ψ: C_T ≡ C_S / f
PREFIX prefixExp
CONSTRUCT { ?SUBJ a T: C_T . }
WHERE {
        ?SUBJ a  S: C_S queryExp
}
```

Padrões Relacionados: Não possui padrões relacionados.

Exemplos da solução proposta:

- **Regras de Mapeamento:**
  - $(\mathbf{R_1})$ *bo:Author*(*s*) ← *dbo:Person*(*s*)
  - $(\mathbf{R_3})$ *bo:Book*(*s*) ← *dbo:Book*(*s*) ; *dbo:literaryGenre*(*s*,*v*) != 'Romance'
- **Assertivas de Mapeamento:**
  - $(\boldsymbol{\psi_1})$ *bo:Author* ≡ *dbo:Person*
  - $(\boldsymbol{\psi_3})$ *bo:Book* ≡ *dbo:Book* / *dbo:literaryGenre* != "Romance"
- **Mapeamentos *SPARQL*:**

```
# Template T1 – Mapeamento de Classe
# AMC ψ1: bo: Author ≡ dbo: Person

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl#>
CONSTRUCT { ?SUBJ a bo:Author . }
WHERE { ?SUBJ a dbo:Person . }
```

```
# Template T1 – Mapeamento de Classe
# AMC ψ3: bo: Book ≡ dbo: Book / dbo: literaryGenre ! = "Romance"

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl#>
CONSTRUCT { ?SUBJ a bo:Book . }
WHERE {
        ?SUBJ a dbo:Book ;
            dbo:literaryGenre ?o .
        FILTER( ?o != "Romance" )
}
```

A *AMC* ψ_1 especifica que cada instância da classe *dbo:Person* produz uma instância para a classe *bo:Author*. Já a *AMC* ψ_3, especifica que cada instância da classe *dbo:Book* que satisfaça o filtro *dbo:literaryGenre != "Romance"*, irá produzir uma instância para a classe *dbo:Book*.

A variável prefixExp permite armazenar os valores dos prefixos das classes e propriedades que se encontram numa *AM*. No caso das *AMC* ψ_1 e *AMC* ψ_3, o valor da variável prefixExp irá ser "*dbo:<http://dbpedia.org/ontology/> PREFIX bo:<http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl#>"* .

A variável queryExp é muito dinâmica e varia consoante o *template* utilizado, sendo que armazena informações necessárias à criação de mapeamentos, tais como expressões de filtro e de caminho. Na *AMC* ψ_1, como não se possui qualquer filtro, a variável queryExp contêm apenas o valor *"."*, enquanto que, no caso da *AMC* ψ_3, esta variável possui o seguinte valor: *" ; dbo:literaryGenre ?o . FILTER(?o != "Romance") "*.

Padrão de Mapeamento MC2

Nome: Padrão de Mapeamento de Classes Não Semelhantes Semanticamente (MC2)

Problema: "Como especificar o mapeamento de instância de uma classe numa ontologia fonte em instâncias de uma classe numa ontologia alvo, que não representam o mesmo objeto do mundo real?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $A_1, \ldots, A_n$ são propriedades de tipos de dados em $C_{\mathbf{S}}$;

- **f** é um predicado de seleção sobre instâncias de C_S, ou seja, identifica a condição que necessita de ser satisfeita sobre as instâncias de C_S, sendo que se trata de uma cláusula opcional.

Solução:

- **Regra de Mapeamento:** $C_T(u) \leftarrow C_S(s); generateUri[\psi](s,u); \mathbf{f}(s)$
- **Assertiva de Mapeamento:** ψ: $C_T \equiv C_S[A_1, \ldots, A_n]$ / **f**
- ***Template* de Mapeamento *SPARQL*:**

```
# Template T2 – Mapeamento de Classe
# AMC  ψ: C_T ≡ C_S[A_1, ..., A_n] / f
PREFIX prefixExp
CONSTRUCT { ?generateURI a T: C_T . }
WHERE {
        ?SUBJ a S: C_S queryExp
        BIND(IRI(CONCAT( STR(?SUBJ),  uriExp )) AS ?generateURI)
}
```

Padrões Relacionados: Não possui padrões relacionados.

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_4})$ *dc:MediaType*(*u*) ← *dbo:Book*(*s*); *generateUri*[ψ_4](*s*,*u*)

- **Assertiva de Mapeamento:**

  $(\boldsymbol{\psi_4})$ *dc:MediaType* ≡ *dbo:Book* [*dbo:mediaType*]

➢ **Mapeamento *SPARQL*:**

```
# Template T2 – Mapeamento de Classe
# AMC ψ4: dc:MediaType ≡ dbo:Book [dbo:mediaType]

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/terms/>
CONSTRUCT { ?generateURI a dc:MediaType . }
WHERE {
        ?SUBJ a dbo:Book ;
            dbo:mediaType ? A1 .
        BIND( IRI( CONCAT( STR(?SUBJ), ENCODE-FOR-URI(? A1) ) ) AS
?generateURI)
}
```

A *AMC* ψ_4 especifica o mapeamento entre a classe *dc:MediaType* e a propriedade *dbo:mediaType*. Dizemos que *dbo:Book*[*dbo:mediaType*] é uma classe embutida da classe *dbo:Book*.

A variável prefixExp e queryExp utilizadas pelo *template* T2 no Padrão de Mapeamento MC2 foram obtidas conforme descrito no Padrão de Mapeamento MC1. Por outras palavras, para a *AMC* ψ_4, o valor de prefixExp é "*dbo:<http://dbpedia.org/ontology/> PREFIX dc: <http://purl.org/dc/terms/>"* e o valor da variável queryExp é *"; dbo:mediaType* ? A_1*."*.

A função *generateUri*, utilizada na regra de mapeamento, permite efetuar a geração de um *URI/IRI* único. Ao contrário do que aconteceu em [14], nós não criámos uma função própria para a geração de *URI/IRI* únicos. Nesta dissertação, optámos por utilizar as funções que o *SPARQL* disponibiliza. Para a geração de um *URI/IRI* único, efetuámos os seguintes passos:

(1) Preencher o valor da variável uriExp: A variável uriExp permite armazenar informações necessárias à criação de *URIs* únicos, sendo que o valor desta variável varia consoante se trate de apenas uma propriedade ou de várias.

Quando apenas existe a propriedade A_1, o valor da variável uriExp é constituído da seguinte forma: *ENCODE-FOR-URI*(z_1). No caso de existirem duas ou mais propriedades, necessitamos de utilizar a função *CONCAT*() para concatenar os valores das diversas propriedades, logo o valor da variável uriExp passa a ser constituído da seguinte forma: *ENCODE-FOR-URI*(*CONCAT*($z_1, \dots, z_n$)), em que $z_1, \dots, z_n$ correspondem, respetivamente, aos valores das propriedades $A_1, \dots, A_n$.

Na *AMC* ψ_4, como apenas existe uma propriedade, não é necessário proceder à concatenação de valores das propriedades, logo o valor da variável uriExp é *ENCODE-FOR-URI*($?A_1$);

(2) Concatenar o valor da instância da classe a ser mapeada com o valor da variável uriExp: No nosso exemplo, em primeiro lugar convertemos o valor da instância da classe *dbo:Book* numa *string*, através da função *STR()*. Após esta conversão, efetuamos a concatenação deste valor com o valor da variável uriExp;

(3) Converter o resultado da concatenação num *URI*: Através da função *IRI*(), convertemos o resultado da concatenação realizada no passo 2 num *URI*;

(4) Atribuir o *URI* gerado à variável *?generateURI*: Em *SPARQL*, utilizamos a função *BIND*() para este efeito.

4.3.2.2 Padrões de Mapeamento de Propriedades de Objeto

Os Padrões de Mapeamento de Propriedades de Objeto são constituídos por dois Padrões de Mapeamento: *Padrão de Mapeamento de Propriedades de Objeto Direto de Classes Semelhantes Semanticamente* e *Padrão de Mapeamento de Propriedades de Objeto Direto de Classe Embutida*, que correspondem, respetivamente, aos Padrões de Mapeamento OM1 e OM2 definidos por Vinuto em [14]. Tal como abordado na secção 2.4.5, as propriedades de objeto são definidas por *owl:ObjectProperty* e relacionam um recurso com outro que seja identificado por uma *URI* ou por uma *IRI*.

Padrão de Mapeamento MO1

Nome: Padrão de Mapeamento de Propriedades de Objeto Direto de Classes Semelhantes Semanticamente (MO1)

Problema: "Como especificar o mapeamento direto de propriedades de objeto em classes semelhantes semanticamente?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes semelhantes semanticamente, em que a *AMC* de $C_{\mathbf{T}}$ é da forma: ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}/\mathbf{f}$;
- $P_{\mathbf{T}}$ e $P_{\mathbf{S}}$ são propriedades de objeto em $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$, respetivamente, sendo que os contradomínios de $P_{\mathbf{T}}$ e $P_{\mathbf{S}}$ são $C_{\mathbf{T}}^*$ e $C_{\mathbf{S}}^*$;
- $C_{\mathbf{T}}^*$ e $C_{\mathbf{S}}^*$ são classes semelhantes semanticamente.

Solução:

- **Regra de Mapeamento:** $P_{\mathbf{T}}(s,v) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}(s); \varphi(s,t); P_{\mathbf{S}}(t,v); \mathbf{f}(v)$, onde o filtro **f** é opcional.
- **Assertiva de Mapeamento:** ψ: $C_{\mathbf{T}}$ / $P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi$ /$P_{\mathbf{S}}$ / **f**
  - **Restrição:**
    - **Quando $\boldsymbol{\psi}$ é da forma:** ψ: C_{T} / $P_{\mathrm{T}} \equiv C_{\mathrm{S}}$ / P_{S} / f. Deve existir uma *AMC* que relacione o domínio de $P_{\mathbf{T}}$ com $C_{\mathbf{S}}$ e outra *AMC* que relacione o contradomínio de $P_{\mathbf{T}}$ com o contradomínio de $P_{\mathbf{S}}$.
    - **Quando $\boldsymbol{\psi}$ é da forma:** ψ: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}/\varphi/\mathbf{f}$, onde φ representa o caminho de $C_{\mathbf{S}}$ para $C_{\mathbf{R}}$**.** Deve existir uma *AMC* que relacione o domínio de $P_{\mathbf{T}}$ com $C_{\mathbf{S}}$ e outra *AMC* que relacione o contradomínio de $P_{\mathbf{T}}$ com $C_{\mathbf{R}}$.
- ***Template* de Mapeamento *SPARQL*:**

```
#Template T3 – Mapeamento de Propriedade
# AMO / AMD ψ: C_T / P_T ≡ C_S / φ / P_S / f
PREFIX prefixExp
CONSTRUCT { ?SUBJ T: P_T ?P_S . }
WHERE {
        ?SUBJ a S: C_S ; queryExp
}
```

Padrões Relacionados: Padrão de Mapeamento de Classes Semelhantes Semanticamente (MC1).

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_{13}})$ *foaf:made(s,v) ← dbo:Book(s) ; dbo:author(s,v)*

- **Assertiva de Mapeamento:**

  $(\boldsymbol{\psi_{13}})$ *bo:Book / foaf:made ≡ dbo:Book / dbo:author*

- **Mapeamento *SPARQL*:**

```
# Template T3 – Mapeamento de Propriedade
# AMO / AMD ψ13: bo: Book / foaf: made ≡ dbo: Book / dbo: author

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { ?SUBJ foaf:made ?p . }
WHERE {
        ?SUBJ a dbo:Book ;
            dbo:author ?p ;
            dbo:literaryGenre ?o .
        FILTER( ?o != "Romance" )
}
```

A *AMO* ψ_{13} especifica que cada triplo que obedece aos padrões (*?SUBJ a dbo:Book*), (*?SUBJ dbo:author ?p*) e (*?SUBJ dbo:literaryGenre ?o*) e ao filtro (*?o != 'Romance'*) produz triplos *RDF* no vocabulário alvo na seguinte forma: (*?SUBJ foaf:made ?p*).

Para que este mapeamento possa ser efetuado, é necessário que previamente já exista uma *AMC* (Padrão de Mapeamento MC1) entre as classes dos domínios das propriedades e uma *AMC* (Padrão de Mapeamento MC1) entre as classes de contradomínio das propriedades. O domínio de *dbo:author* é *dbo:Book* e o de *foaf:made* é *bo:Book*. Assim, a assertiva ψ_3 tem de ser definida antes da assertiva ψ_{13}. O contradomínio de *dbo:author* é *dbo:Person* e o de *foaf:made* é *bo:Author*. Deste modo, a assertiva ψ_1 também tem de ser definida antes da assertiva ψ_{13}.

A variável prefixExp utilizada pelo *template* T3 no Padrão de Mapeamento MO1 foi obtida conforme descrito no Padrão de Mapeamento MC1. Para a *AMO* ψ_{13}, o valor de prefixExp é "*dbo:http://dbpedia.org/ontology/ PREFIX foaf: <http://xmlns.com/foaf/0.1/>* ".

No caso dos Padrões de Mapeamento de Propriedades, quer estes sejam de objetos ou tipos de dados, a variável queryExp possui as informações da ontologia fonte que sejam necessárias para a criação dos triplos da ontologia alvo. Assim, esta variável é formada por elementos oriundos da *AMC* de domínio das propriedades e por elementos da própria *AMD/AMO*. A *AMO* ψ_{13} é da forma $\psi: C_T / P_T \equiv C_S / P_S /$ f, o que significa que não existe qualquer caminho. Neste caso, a variável queryExp contêm elementos da assertiva ψ_3 (*AMC*

da classe que é o domínio da propriedade que está a ser mapeada, isto é, " *bo:Book* ≡ *dbo:Book* / *dbo:literaryGenre* != "Romance" ") e elementos da *AMO* ψ_{13}, ou seja, a variável queryExp é igual a "*dbo:author ?p; dbo:literaryGenre ?o . FILTER(?o != "Romance")*".

Padrão de Mapeamento MO2

Nome: Padrão de Mapeamento de Propriedades de Objeto Direto de Classe Embutida (MO2)

Problema: "Como especificar o mapeamento de propriedades de objeto em classe embutida?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes não semelhantes semanticamente (classe embutida), em que a *AMC* de $C_{\mathbf{T}}$ é da forma: ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \dots, A_n]$ / **f**;
- $P_{\mathbf{T}}$ é uma propriedade de objeto em $C_{\mathbf{T}}$, cujo contradomínio (*range*) é uma classe $C^*_{\mathbf{T}}$;
- $C^*_{\mathbf{T}}$ e $C^*_{\mathbf{S}}$ são classes não semelhantes semanticamente (classe embutida).

Solução:

- **Regra de Mapeamento:**
  $P_{\mathbf{T}}(s,u) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}(s); generateUri[\psi](s,u)$, onde o filtro **f** é opcional.
- **Assertiva de Mapeamento:** ψ: $C_{\mathbf{T}}/P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1 \dots A_n]/NULL$
  - **Restrição:** Deve existir uma *AMC* que relacione o domínio de $P_{\mathbf{T}}$ com $C_{\mathbf{S}}$ e outra *AMC* que relacione o contradomínio de $P_{\mathbf{T}}$ também com $C_{\mathbf{S}}$.
- ***Template* de Mapeamento *SPARQL*:**

```
#Template T5 – Mapeamento de Propriedade
#AMO ψ: C_T / P_T ≡ C_S[A_1 ... A_n]/NULL
PREFIX prefixExp
CONSTRUCT { ?SUBJ T: P_T ?generateURI . }
WHERE {
        ?SUBJ a S: C_S ; queryExp
        BIND(IRI(CONCAT( STR(?SUBJ), uriExp )) AS ?generateURI)
}
```

Padrões Relacionados: Padrão de Mapeamento de Classes Não Semelhantes Semanticamente (MC2).

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_{15}})$ *dc:hasFormat*(*s*,*u*) ← *dbo:Book*(*s*); *generateUri*[ψ_{15}](*s*,*u*)

- **Assertiva de Mapeamento:**

  $(\boldsymbol{\psi_{15}})$ *bo:Book / dc:hasFormat* ≡ *dbo:Book* [*dbo:mediaType*] / NULL

- **Mapeamento *SPARQL*:**

```
# Template T5 – Mapeamento de Propriedades
# AMO ψ15: bo: Book / dc: hasFormat ≡ dbo: Book [dbo: mediaType ] / NULL

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/terms/>
CONSTRUCT { ?SUBJ dc:hasFormat ?generateURI . }
WHERE {
        ?SUBJ a dbo:Book ;
            dbo:mediaType ?A1 ;
            dbo:literaryGenre ?o .
        FILTER( ?o != "Romance" )
        BIND( IRI( CONCAT( STR(?SUBJ), ENCODE-FOR-URI(?A1) ) ) AS
?generateURI)
}
```

A *AMO* ψ_{15} especifica que cada triplo que obedece aos padrões (*?SUBJ a dbo:Book*), (*?SUBJ dbo:mediaType* $?A_1$) e (*?SUBJ dbo:literaryGenre ?o*) e ao filtro (*?o != "Romance"*) produz triplos *RDF* no vocabulário alvo na seguinte forma: (*?SUBJ dc:hasFormat ?generateURI*).

Tal como acontece no Padrão de Mapeamento MO1, para que este mapeamento possa ser efetuado, é necessário que já exista previamente uma *AMC* (Padrão de Mapeamento MC2) que relacione a classe de domínio da propriedade da ontologia alvo com a classe da ontologia fonte e, uma *AMC* (Padrão de Mapeamento MC1) que relacione a classe de contradomínio da propriedade da ontologia alvo com a classe da ontologia fonte. O domínio de *dc:hasFormat* é *bo:Book* e o seu contradomínio é *dc:MediaType*. Como a classe da ontologia fonte é *dbo:Book*,

as assertivas que têm de ser definidas antes da assertiva ψ_{15} são, respetivamente, as assertivas ψ_3 e ψ_4.

As variáveis prefixExp, queryExp e uriExp utilizadas pelo *template* T5 no Padrão de Mapeamento MO2 foram obtidas conforme descrito no Padrão de Mapeamento MC2. Por outras palavras, para a *AMO* ψ_{15}, o valor de prefixExp é "*dbo:<http://dbpedia.org/ontology/> PREFIX dc: <http://purl.org/dc/terms/>*", o valor da variável queryExp é *"dbo:mediaType ?*A_1 *; dbo:literaryGenre ?o . FILTER(?o != "Romance")"* e o valor da variável uriExp é *ENCODE-FOR-URI(?*A_1*)*.

4.3.2.3 Padrões de Mapeamento de Propriedades de Tipo de Dados

Nos Mapeamentos de Propriedades de Tipo de Dados, distinguem-se três mapeamentos: *Mapeamento de Propriedades de Tipos de Dados Diretos 1-1 de Classes Semelhantes Semanticamente*, *Mapeamento de Propriedades de Tipos de Dados Diretos N-1 de Classes Semelhantes Semanticamente* e *Mapeamento de Propriedades de Tipos de Dados de Classe Embutida*, que correspondem, respetivamente, aos padrões de Mapeamento DM1, DM2 e DM3 definidos por Vinuto em [14].

As propriedades de tipo de dados, tal como abordado na secção 2.4.5, são definidas por *owl:DatatypeProperty* e relacionam um recurso com um literal ou com o tipo de dados do esquema *XML*.

Padrão de Mapeamento MD1

Nome: Padrão de Mapeamento de Propriedades de Tipos de Dados Diretos 1-1 de Classes Semelhantes Semanticamente (MD1).

Problema: "Como especificar o mapeamento 1-1 de propriedades de tipos de dados de classes semelhantes semanticamente?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes semelhantes semanticamente, em que a *AMC* de $C_{\mathbf{T}}$ é da forma: ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}/\mathbf{f}$;
- $[P_1, \dots, P_n]$ são propriedades de tipos de dados em $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$, sendo que os contradomínios de $P_{\mathbf{T}}$ e $P_{\mathbf{S}}$ são $C_{\mathbf{T}}^*$ e $C_{\mathbf{S}}^*$, respetivamente;
- T é uma função de transformação que mapeia valores de $C_{\mathbf{T}}^*$ em valores do tipo $C_{\mathbf{S}}^*$.

Solução:

- **Regra de Mapeamento:**

  $P_{\mathbf{T}}(s, v) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}(s); \varphi(s, o);\ P_{\mathbf{S}}(o, x); \mathbf{f}(x); \mathrm{T}(x, v)$, em que **f** e T são opcionais e correspondem, respetivamente, ao filtro e à função de transformação.

- **Assertiva de Mapeamento:** $\psi: C_{\mathbf{T}} \ / \ P_{\mathbf{T}} \equiv C_{\mathbf{S}} \ / \ \varphi \ / \ P_{\mathbf{S}} \ / \ \mathbf{f} \ / \ \mathrm{T}$
  - **Restrição:** Deve existir uma *AMC* que relacione o domínio de $P_{\mathbf{T}}$ com o domínio de $P_{\mathbf{S}}$.

- ***Template* de Mapeamento *SPARQL*:**

  Este padrão, à semelhança do que acontece em [15], possui dois *templates*, T3 e T6. Apesar de não o apresentarmos neste padrão, o *template* T3 pode ser observado e obtido aquando da explicação do Padrão de Mapeamento MO1. A única diferença entre os *templates* T3 e T6 está na utilização da variável functionExp.

```
#Template T6 – Mapeamento de Propriedade
# AMD  ψ: C_T / P_T ≡ C_S / φ / P_S / f / T
PREFIX prefixExp
CONSTRUCT { ?SUBJ T: P_T ?p . }
WHERE {
        ?SUBJ a S: C_S ; queryExp
        BIND( functionExp AS ?p )
}
```

Padrões Relacionados: Padrão de Mapeamento de Classes Semelhantes Semanticamente (MC1).

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_5})$ *bo:title*(*s*,*v*) ← *dbo:Book*(*s*) *; dbo:originalTitle*(*s*,*x*) *; lower-case(x,v)*

- **Assertiva de Mapeamento:**

  $(\boldsymbol{\psi_5})$ *bo:Book / bo:title* ≡ *dbo:Book / dbo:originalTitle / lower-case*(*dbo:originalTitle*)

- **Mapeamento SPARQL:**

```
# Template T6 – Mapeamento de Propriedade
# AMD ψ5: bo:Book / bo:title ≡ dbo:Book / dbo:originalTitle

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl#>
CONSTRUCT { ?SUBJ bo:title ?p . }
WHERE {
        ?SUBJ a dbo:Book ;
            dbo:originalTitle ?s ;
            dbo:literaryGenre ?o .
        FILTER( ?o != "Romance" )
        BIND( LCASE(?s) AS ?p )
}
```

A *AMD* ψ_5 especifica que cada triplo que obedece aos padrões (*?SUBJ a dbo:Book*), (*?SUBJ dbo:originalTitle ?s*) e (*?SUBJ dbo:literaryGenre ?o*) e ao filtro (*?o != "Romance"*) produz triplos *RDF* no vocabulário alvo na seguinte forma: (*?SUBJ bo:title ?p*), sendo que o valor de *?p* é transformado de modo a conter todas as letras em minúsculas.

No caso dos mapeamentos de propriedades de tipos de dados, é necessário que exista previamente uma *AMC* entre as classes (Padrão de Mapeamento MC1) dos domínios das propriedades. O domínio de *dbo:originalTitle* é *dbo:Book* e o de *bo:title* é *bo:Book*. Assim, a assertiva ψ_3 tem de ser definida antes da assertiva ψ_5.

A variável prefixExp utilizada pelo *template* T6 no Padrão de Mapeamento MD1 é obtida conforme descrito anteriormente no *template* T3 no Padrão de Mapeamento MO1. Assim, para a *AMD* ψ_5, o valor da variável prefixExp é "*dbo:<http://dbpedia.org/ontology/> PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl#>"*.

A variável queryExp irá conter informações relativas à *AMC* entre as classes (Padrão de Mapeamento MC1) dos domínios das propriedades e à *AMD* em questão, sendo que o valor guardado da parte da *AMC* é "*dbo:literaryGenre ?o . FILTER(?o != "Romance"*)" e o valor da parte da *AMD* é "*dbo:originalTitle ?p ;* ". Assim, o valor da variável queryExp é : "*dbo:originalTitle ?p ; dbo:literaryGenre ?o . FILTER(?o != "Romance"*) ".

A variável functionExp contêm informações relativas à função de transformação, logo sempre que a variável T possuir valor, o mesmo será transformado em linguagem *SPARQL* e armazenado em functionExp. Na *AMD* ψ_5 utilizámos uma função de transformação de modo a que o valor de *bo:title* seja uma *string* constituída apenas por letras minúsculas. Neste caso, o valor da variável functionExp é : "*LCASE*(*?s*)".

Padrão de Mapeamento MD2

Nome: Padrão de Mapeamento de Propriedades de Tipos de Dados Diretos N-1 de Classes Semelhantes Semanticamente (MD2).

Problema: "Como especificar o mapeamento N-1 de propriedades de tipos de dados de classes semelhantes semanticamente?" [14]

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes semelhantes semanticamente, em que a *AMC* de $C_{\mathbf{T}}$ é: $\psi: C_{\mathbf{T}} \equiv C_{\mathbf{S}}/\mathbf{f}$;
- $P_{\mathbf{T}}$ e $[P_{s_1}, \dots, P_{s_n}]$ são propriedades de tipos de dados em $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$, respetivamente, sendo que os contradomínios de $P_{\mathbf{T}}$ e $P_{\mathbf{S}}$ são, respetivamente, $C^*_{\mathbf{T}}$ e $C^*_{\mathbf{S}}$;
- T é uma função de transformação que mapeia os valores de $C^*_{\mathbf{T}}$ para valores do tipo de $C^*_{\mathbf{S}}$.

Solução:

- **Regra de Mapeamento:**

  $P_{\mathbf{T}}(s,t) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}(s); \varphi(s,o); P_{\mathbf{S}_1}(o, v_1), \dots, P_{\mathbf{S}_n}(o, v_n);$

  $concat(v_1, \dots, v_n, v);\ \mathrm{T}(v,t),$

  em que o filtro **f** e a função T são opcionais.
- **Assertiva de Mapeamento:** $\psi: C_{\mathbf{T}} \,/\, P_{\mathbf{T}} \equiv C_{\mathbf{S}} \,/\, \varphi \,/\, \{P_{\mathbf{S}_1}, \dots, P_{\mathbf{S}_n}\} \,/\, \mathrm{T}$
  - **Restrição:** Deve existir uma *AMC* que relacione o domínio de $P_{\boldsymbol{T}}$ com o domínio de $P_{\mathbf{S}_1}, \dots, P_{\mathbf{S}_n}$

- ***Template* de Mapeamento *SPARQL*:**

  À semelhança do Padrão de Mapeamento MD1, o Padrão de Mapeamento MD2 também possui dois *templates* T7 e T8, que apenas possuem como diferença a existência da variável functionExp. Ou seja, sempre que a variável T exista na *AMD*, é aplicado o *template* T8, caso contrário, é aplicado o *template*.

#*Template* T7 – Mapeamento de Propriedade

AMD ψ: $C_{\mathbf{T}} \ / \ P_{\mathbf{T}} \equiv C_{\mathbf{S}} \ / \ \varphi \ / \ \{P_{\mathbf{S}_1}, \dots, P_{\mathbf{S}_n}\}$

PREFIX prefixExp

CONSTRUCT { ?SUBJ **T**: $P_{\mathbf{T}}$?p . **}**

WHERE {

?SUBJ a **S**: $C_{\mathbf{S}}$ **;** queryExp

BIND(CONCAT(? $P_{\mathbf{S}_1}$, … , ? $P_{\mathbf{S}_n}$**) AS** ?p **)**

}

#*Template* T8 – Mapeamento de Propriedade

AMD ψ: $C_{\mathbf{T}} \ / \ P_{\mathbf{T}} \equiv C_{\mathbf{S}} \ / \ \varphi \ / \ \{P_{\mathbf{S}_1}, \dots, P_{\mathbf{S}_n}\}$ / T

PREFIX prefixExp

CONSTRUCT { ?SUBJ **T**: $P_{\mathbf{T}}$?v . **}**

WHERE {

?SUBJ a **S**: $C_{\mathbf{S}}$ **;** queryExp

BIND(CONCAT(? $P_{\mathbf{S}_1}$, … , ? $P_{\mathbf{S}_n}$**) AS** ?p **)**

BIND(functionExp **AS** ?v **)**

}

Padrões Relacionados: Padrão de Mapeamento de Classes Semelhantes Semanticamente (MC1).

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_{11}})$ *ex:careerDuration(s,v)* ← *dbo:Person(s); dbo:startCareer(s, v1); dbo:endCareer (s, v2); concat(v1, '-', v2, v)*

- **Assertiva de Mapeamento:**

  $(\boldsymbol{\psi_{11}})$ *bo:Author / ex:careerDuration* ≡ *dbo:Person / { dbo:startCareer , dbo:endCareer }*

- **Mapeamento *SPARQL*:**

```
# Template T7 – Mapeamento de Propriedade
# AMD ψ11: bo: Author / ex: careerDuration ≡ dbo: Person /
                    {dbo: startCareer , dbo: endCareer }

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX ex: <http://example.org/>
CONSTRUCT { ?SUBJ ex:careerDuration ?p . }
WHERE {
        ?SUBJ a dbo:Person ;
            dbo:startCareer ?s ;
            dbo:endCareer ?t .
        BIND(CONCAT(?s, ' –', ?t) AS ?p )
}
```

A *AMD* ψ_{11} especifica que cada conjunto de triplos *RDF* que obedecem aos padrões de triplos (*?SUBJ dbo:startCareer ?s*) e (*?SUBJ dbo:endCareer ?t*) produzem triplos *RDF* no vocabulário alvo na seguinte forma: (*?SUBJ ex:careerDuration ?p*), através das funções *BIND* e *CONCAT*: "*BIND(CONCAT(?s, ' –', ?t) AS ?p)*", em que a variável ?p irá possuir como valor o resultado da concatenação de ?s, '-' e ?t. Como a *AMD* ψ_{11} não possui definida a variável T, foi aplicado o *template* T7.

As variáveis prefixExp, queryExp e functionExp utilizadas pelos *templates* T7 e T8 do Padrão de Mapeamento MD2 são obtidas conforme descrito anteriormente no *template* T6 do Padrão de Mapeamento MD1. Logo, para a *AMD* ψ_{11}, o valor da variável prefixExp é "*dbo:<http://dbpedia.org/ontology/> PREFIX ex:<http://example.org/>"* e o valor da variável queryExp é: "*dbo:startCareer ?s ; dbo:endCareer ?t* .".

Para que este mapeamento seja possível, é necessário que seja criado previamente uma *AMC* entre as classes (Padrão de Mapeamento MC1) de domínio das propriedades. Na *AMD* ψ_{11}, o domínio de *ex:careerDuration* é *bo:Author* e o domínio de *dbo:startCareer* e *dbo:endCareer* é *dbo:Person*. Assim, a assertiva ψ_1 tem de ser definida antes da assertiva ψ_{11}.

Padrão de Mapeamento MD3

Nome: Padrão de Mapeamento de Propriedades de Tipos de Dados de Classe Embutida (MD3).

Problema: Como especificar o mapeamento 1-1 de propriedades de tipos de dados de classes não semelhantes semanticamente?

Pré-condição:

- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes em $\mathbf{V_T}$ e $\mathbf{V_S}$, respetivamente;
- $C_{\mathbf{T}}$ e $C_{\mathbf{S}}$ são classes não semelhantes semanticamente, em que a *AMC* de $C_{\mathbf{T}}$ é da forma: ψ: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1, \dots, A_n]$ / **f**;

Solução:

- **Regra de Mapeamento:**

  $P_{\mathbf{T}}(u, v) \leftarrow C_{\mathbf{S}}(s); \mathbf{f}(s); P_{\mathbf{S}}(s, v); generateUri[\psi](s, u)$, em que **f** é opcional e corresponde ao filtro.
- **Assertiva de Mapeamento:** ψ: $C_{\mathbf{T}}$ / $P_{\mathbf{T}} \equiv C_{\mathbf{S}}[A_1 \dots A_n]$ / A_i / **f**
  - **Restrição:** Deve existir uma *AMC* que relacione o domínio de $P_{\mathbf{T}}$ com $C_{\mathbf{S}}[A_1 \dots A_n]$
- ***Template* de Mapeamento *SPARQL*:**

```
#Template T4 – Mapeamento de Propriedade
# AMD ψ: C_T / P_T ≡ C_S[A_1 ... A_n] / A_i / f
PREFIX prefixExp
CONSTRUCT { ?generateURI T: P_T ?A_i . }
WHERE {
        ?SUBJ a S: C_S ; queryExp
        BIND(IRI(CONCAT( STR(?SUBJ), uriExp )) AS ?generateURI)
}
```

Padrões Relacionados: Padrão de Mapeamento de Classes Não Semelhantes Semanticamente (MC2).

Exemplos da solução proposta:

- **Regra de Mapeamento:**

  $(\mathbf{R_{10}})$ *dc:format*(*u*,*v*) ← *dbo:Book*(*s*) *; dbo:mediaType*(*s*,*v*);
  generateUri[ψ_{10}](*s*,*u*)

- **Assertiva de Mapeamento:**

  ($\boldsymbol{\psi_{10}}$) *dc:MediaType / dc:format* ≡ *dbo:Book* [*dbo:mediaType*] / *dbo:mediaType*

- **Mapeamento *SPARQL*:**

```
# Template T4 – Mapeamento de Propriedade
# AMD ψ10: dc:MediaType / dc:format ≡ dbo:Book [dbo:mediaType] / dbo:mediaType

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/terms/>
CONSTRUCT { ?generateURI dc:format ?A1 . }
WHERE {
        ?SUBJ a dbo:Book ;
                dbo:mediaType ?A1 .
        BIND( IRI( CONCAT( STR(?SUBJ), ENCODE-FOR-URI(?A1) ) ) AS
        ?generateURI)
}
```

A *AMD* ψ_{10} especifica que cada triplo *RDF* que obedece ao padrão de triplo (*?SUBJ dbo:mediaType* $?A_1$) produz triplos no vocabulário alvo na seguinte forma: (*?generateURI dc:format* $?A_1$).

Para que este mapeamento seja possível, é necessário que seja criado previamente uma *AMC* (Padrão de Mapeamento MC2) entre as classes de domínio das propriedades. O domínio de *dc:format* é *dc:MediaType* e o domínio de *dbo:mediaType* é *dbo:Book*. Logo, a assertiva ψ_4 tem de ser definida antes da assertiva ψ_{10}.

As variáveis prefixExp, queryExp e uriExp utilizadas pelo *template* T4 do Padrão de Mapeamento MD3 foram obtidas conforme descrito no Padrão de Mapeamento MC2. Por outras palavras, para a *AMO* ψ_{10}, o valor de prefixExp é "*dbo:<http://dbpedia.org/ontology/> PREFIX dc: <http://purl.org/dc/terms/>"* , o valor da variável queryExp é *"dbo:mediaType* $?A_1$ *."* e o valor da variável uriExp é *ENCODE-FOR-URI*($?A_1$).

4.4 Considerações finais do capítulo

Neste capítulo introduzimos e apresentamos o formalismo e os padrões de mapeamento. O formalismo é o que utilizamos na nossa proposta para a especificação de mapeamentos de ontologias, enquanto que os padrões de mapeamento se referem aos problemas e soluções genéricas para os diferentes tipos de mapeamentos entre ontologias. No caso das soluções, mostramos que podemos definir, através dos exemplos, mapeamentos mais complexos e específicos a partir dos padrões de mapeamento. Por exemplo, podemos definir filtros (cláusula *FILTER* do *SPARQL*), expressões de caminhos e utilizar funções de transformação para alterar o formato dos dados nas propriedades de tipos de dados.

5 Implementação da Proposta

Neste capítulo apresentamos a ferramenta proposta, *SPARQL Mapping with Assertions (SMA)*, que tem como principal objetivo facilitar a geração de mapeamentos *SPARQL* através da utilização dos padrões de mapeamentos que foram apresentados no Capítulo 4.

A ferramenta *SMA* foi implementada como uma aplicação *web*, baseada na linguagem Java e, além disso, não possui qualquer dependência para a sua execução, ou seja, não é necessário, por exemplo, possuir uma Base de Dados para conseguir executar a aplicação. Tal como qualquer aplicação *web*, possuímos um *front-end* e um *back-end*, em que o primeiro se refere à interface de interação com o utilizador, enquanto que o segundo se refere à implementação de todas as regras de negócio que estão por "detrás" do *front-end*.

Antes de descrevermos a arquitetura e módulos da nossa ferramenta, iremos primeiramente apresentar, na secção 5.1, as tecnologias e ferramentas que foram utilizadas na implementação da *SMA*. Na secção 5.2, introduzimos a arquitetura da nossa ferramenta e explicamos detalhadamente os módulos que a compõem. Na secção 5.3, apresentamos os algoritmos necessários para a criação de mapeamentos *SPARQL*. Na secção 5.4, apresentamos, usando o estudo de caso apresentado na secção 4.2, os passos necessários não só para gerar *AMs* e, consequentemente, as regras de mapeamento e os mapeamentos *SPARQL*, como também para obter os triplos *RDF* a partir dos mapeamentos *SPARQL* criados anteriormente. Por fim, na secção 5.5, apresentamos as considerações finais deste capítulo.

5.1 Tecnologias e Ferramentas

Nesta secção, as tecnologias e ferramentas utilizadas foram agrupadas em 3 (três) categorias, nomeadamente:

- ***Front-end*:** para a construção da interface de interação com o utilizador fizemos uso do *HTML*, *Cascade Style Sheets (CSS)*, *Bootstrap* e do *AngularJS*;
- ***Back-end*:** para implementar todas as regras de negócio que estão por "detrás" do *front-end* utilizámos o *Spring Boot*, *H2* e alguns módulos do *framework Spring*. No caso dos módulos do *Spring* utilizados, iremos apenas destacar o *Spring Data JPA*, pois é o que efetua a "comunicação" entre a implementação propriamente dita das regras de negócio (através do *Spring Boot*) e a Base de Dados (no nosso caso, *H2*);
- **Web Semântica:** utilizámos o Apache Jena para conseguir efetuar a manipulação dos dados que se encontram armazenados em *RDF*.

5.1.1 *Front-end*

5.1.1.1 HTML

HTML é uma linguagem de Marcação de Hipertexto que define o significado e estrutura de um conteúdo da *web* [6]. Cada página *web* é considerada como um "documento" que é interpretado pelos diversos *browsers* existentes nos dispositivos.

A palavra "Hipertexto" refere-se às hiperligações que ligam as páginas da *web* entre si, quer seja num único site ou entre vários sites da *web*, enquanto que a palavra "Marcação" se refere às *tags* utilizadas para anotar texto, imagens ou outro conteúdo [6]. Por exemplo, para indicarmos que queremos efetuar um parágrafo basta colocar o texto entre a *tag* **<p>**, tal como exemplificado de seguida:

```
<p>Exemplo de um parágrafo</p>
```

Na ferramenta *SMA* fizemos uso da versão *HTML* 5.

5.1.1.2 CSS

CSS (do inglês, *Cascade Style Sheets*) foi desenvolvido pelo *W3C* com o objetivo de separar a formatação da página do conteúdo da mesma. Assim, o *HTML* voltou a desempenhar apenas a sua função de estruturar um documento, deixando que o *CSS* se encarregue da apresentação e formatação do mesmo.

CSS possui como principais vantagens [6]:

- **Controlo da apresentação dos vários documentos a partir de uma única folha de estilos *CSS*:** podemos definir um padrão num único ficheiro *CSS* para formatar os conteúdos de diversas páginas *HTML*;
- **Aplicação de diferentes apresentações de acordo com o dispositivo utilizado:** por outras palavras, permite criar uma aplicação responsiva que possa ser utilizada em qualquer *browser* independentemente das características do *hardware* do dispositivo utilizado;
- **Possibilidade de manter a formatação em diferentes *browsers*;**
- **Maior precisão no controlo de apresentações:** como a formatação da apresentação se encontra concentrada em ficheiros *CSS*, o controlo de alterações é muito mais facilitado pois para mudar, por exemplo, a cor e o tipo de letra de todas as *tags* **"<p>"**, basta fazê-lo uma única vez.

CSS pode ser adicionado às páginas *HTML* de três formas diferentes:

1) **Utilizando o atributo *style* dentro de qualquer *tag HTML*:** esta opção é habitualmente utilizada quando apenas queremos modificar um elemento isoladamente.

   Por exemplo:

   ```
   <p style="color:blue;">Exemplo de um parágrafo</p>
   ```

   permite definir que este parágrafo em específico vai ser apresentado com a cor do texto a azul;

2) **Utilizando a *tag style*:** ao contrário do exemplo anterior, esta opção permite que as formatações sejam aplicadas a todos os elementos da página *HTML* onde é inserida.

   Por exemplo:

   ```
   <style type="texto/css">
       p {color:blue}
   </style>
   ```

   define que todos os elementos *HTML* que estejam definidos com *a tag* **<p>** serão afetados, ou seja, todos os parágrafos irão ser apresentados com a cor do texto a azul;

3) **Através de um *link* que faz referência a um ficheiro *CSS*:** esta é a opção mais adequada e mais utilizada, além de que é a única que garante todas as vantagens enumeradas anteriormente.

   ```
   <head>
       <link rel="stylesheet" type="text/css" href="style.css" />
   </head>
   ```

   O trecho de código anterior permite-nos adicionar um *link* que faz referência ao ficheiro *CSS* designado por *style.css*.

Na ferramenta *SMA* fizemos uso da versão *CSS3*. Em relação às diferentes formas de utilizar o *CSS*, optámos por utilizar somente a primeira e a terceira forma.

5.1.1.3 Bootstrap

Bootstrap é um *framework* de *front-end* gratuito que foi criado inicialmente por dois engenheiros do *Twitter*, Jacob Thorton e Mark Otto, com o objetivo de adotar uma estrutura única para otimizar o desenvolvimento da sua plataforma [55].

As principais vantagens e características deste *framework* são [56]:

- **Capacidade de criar aplicações responsivas:** Uma aplicação responsiva consiste numa aplicação que se ajusta de acordo com as dimensões do ecrã onde é visualizada, isto é, quer seja visualizada num *smartphone*, num *tablet* ou num computador. Assim, o programador tem a sua vida facilitada no que toca à criação de aplicações responsivas, pois este *framework* agiliza toda a programação necessária para esta funcionalidade através dos vários *plugins JavaScript* que possui;
- **Modelos de design baseados em *HTML* e *CSS*:** o *Bootstrap* fornece diversos modelos que facilitam a implementação de recursos como, por exemplo, um menu *dropdown*;
- **Facilidade na utilização:** este *framework* é bastante fácil de utilizar, sendo que basta que o programador tenha conhecimentos básicos em *HTML* e *CSS*;
- **Compatibilidade dos browsers:** a última versão deste *framework*, a versão 4, é compatível com todos os *browsers* modernos.

Na ferramenta *SMA* fizemos uso da versão do *Bootstrap* mais atual na altura do desenvolvimento, ou seja, utilizámos a versão 4.3.1.

5.1.1.4 AngularJS

AngularJS é um *framework open-source* em *JavaScript* mantida pelo Google que auxilia no desenvolvimento de aplicações *web* [57]. A linguagem de marcação *HTML* não foi concebida para criar aplicações com visualizações dinâmicas, sendo que o objetivo do *AngularJS* é claramente mitigar este problema. Deste modo, o *AngularJS* estende o vocabulário da linguagem de marcação *HTML*, o que permite obter um vocabulário muito mais expressivo [57].

As principais características e recursos do *AngularJS* são o facto de [57]:

- **Basear-se no padrão *Model-View-Controller (MVC)*:** no caso do *AngularJS*, o *Model* corresponde aos dados da aplicação *AngularJS*; a *View* corresponde às páginas *HTML* e, por último, o *Controller* é o controlador da aplicação *AngularJS* responsável pelas interações entre o *Model* e a *View*;

- **Basear-se no conceito de *Single-Page Application (SPA)*:** De acordo com Seshadri e Green [57], o conceito inerente às *SPA* envolve maior programação do lado do cliente do que do lado do servidor, permitindo assim construir uma aplicação dinâmica, em que os recursos apenas são carregados à medida que são necessários;
- **Utilizar recurso de diretivas:** Uma diretiva é uma extensão da linguagem *HTML* que permite ampliar o comportamento dos elementos *HTML* [57]. As principais diretivas existentes neste framework são a ***ng-app*** e a ***ng-controller***. A ***ng-app*** permite definir qual o elemento raíz da aplicação *AngularJS*, ou seja, qual a porção de *DOM* (*Document Object Model*) que será tratada como uma aplicação *AngularJS*. Por sua vez, a ***ng-controller*** é uma diretiva que pertence a uma dada ***ng-app*** e permite definir qual o controlador que será responsável pelo controlo dos dados. Uma aplicação *AngularJS* (***ng-app***) pode ser composta por vários controladores (***ng-controller***). Por exemplo, na nossa ferramenta possuímos três ***ng-controller*** definidos na mesma ***ng-app***.
- **Utilizar o recurso *Two-Way Data Binding*:** O recurso *Two-Way Data Binding* define que quaisquer alterações ao *model* se irão refletir na *view*, assim como quaisquer alterações à *view* se irão refletir no *model*. A ligação de dados no *AngularJS* trata-se da sincronização entre o *Model* e a *View* do padrão *MVC*, ou seja, quando os dados da aplicação *AngularJS* (*Model)* são alterados, a página *HTML* (*View)* reflete a alteração e vice-versa. Existem duas diretivas que juntas permitem este recurso, sendo estas: o ***ng-model*** e o ***ng-bind***. O ***ng-model*** associa o valor dos campos de preenchimento das páginas *HTML* aos dados da aplicação, enquanto que o ***ng-bind*** faz o processo inverso, ou seja, associa os dados da aplicação aos dados das páginas *HTML* [57];
- **Utilizar o conceito de injeção de dependência:** Neste *framework* as dependências dos componentes ficam explícitas, o que facilita a tarefa de gestão. Existem três formas de declarar dependências: através de *Arrays*, *$inject* ou usando diretamente os parâmetros de uma função [57];
- **Integração com outros *frameworks*:** O *AngularJS* permite ainda a integração com outros *frameworks* como, por exemplo, o *Bootstrap* e o *Apache Cordova*[28] [58].

[28] Apache Cordova https://cordova.apache.org/

Na ferramenta *SMA* fizemos uso da versão 1.6.6 do *AngularJS*.

5.1.2 *Back-end*

5.1.2.1 Spring Boot

Spring Boot é um *framework open-source* baseada em Java que tem como objetivo de simplificar a configuração de aplicações baseadas em Spring[29] [59]. Ao usar este *framework*, apenas devemos indicar quais os módulos queremos utilizar e o *Spring Boot* reconhece e configura-os automaticamente. Deste modo, conseguimos obter uma aplicação baseada em *Spring* a executar em pouco tempo, o que permite que o desenvolvedor se concentre mais na parte de negócio e menos na infraestrutura que suporta a aplicação.

Os principais benefícios e recursos oferecidos por este *framework* são o facto de [59]:

- Fornecer uma maneira flexível de configurar *Java Beans*[30], configurações *XML* e transações de Bases de Dados;
- Fornecer processamento em lote e efetuar a gestão de *REST endpoints*, isto é, efetuar a gestão dos *URIs* onde o serviço *REST* pode ser acedido;
- Funcionar com pouquíssima configuração manual, sendo que esta assume a forma de anotações, ao invés de ser em *XML* como seria de esperar numa aplicação *Spring*;
- Facilitar a gestão de dependências;
- Incluir um *Servlet Container* incorporado, sendo que por defeito é o *Apache Tomcat*[31]. Isto permite-nos executar uma aplicação *Spring Boot* sem que haja a necessidade de possuir um *Servlet Container* externo.

O ponto inicial de uma aplicação desenvolvida em *Spring Boot* é a classe que contenha a anotação ***@SpringBootApplication*** e o método principal. De acordo com [60], a anotação ***@SpringBootApplication*** inclui outras anotações, sendo estas:

- ***@Configuration*:** sinaliza a classe como a origem de definições de *beans* para o contexto da aplicação;
- ***@EnableAutoConfiguration*:** informa o *Spring Boot* que necessita de efetuar a inclusão de *beans* com base em diversas configurações e/ou outros *beans*;

[29] Spring https://spring.io/projects/spring-framework
[30] Java Beans https://www.javatpoint.com/java-bean
[31] Apache Tomcat http://tomcat.apache.org/

- ***@EnableWebMvc*:** sinaliza a aplicação como uma aplicação *web*, de modo a ativar comportamentos importantes, tais como, a configuração do *DispatcherServlet*[32];
- ***@ComponentScan*:** indica ao *Spring* que deve procurar por outros componentes, configurações e/ou serviços que se encontrem no pacote onde esta anotação foi inserida.

Na ferramenta *SMA* fizemos uso da versão 2.1.3 do *Spring Boot*.

5.1.2.2 Spring Data JPA

Spring Data JPA é um módulo disponibilizado pelo *Spring,* que pertence ao grupo *Spring Data*. *Spring Data* é um grupo de projetos que tem como objetivo facilitar o acesso a dados, tendo em conta as características de cada tipo de armazenamento subjacente. Além do *Spring Data JPA*, o *Spring Data* alberga, por exemplo, os módulos *Spring Data MongoDB* e o *Spring Data JDBC*. Na ferramenta *SMA* fizemos uso do módulo *Spring Data JPA* para facilitar a implementação de repositórios baseados em *Java Persistence API (JPA)* [61].

Antes de explicar o que é o *Spring Data JPA* é necessário entender o conceito base, ou seja, o que é o *JPA*. *JPA* é uma especificação que facilita o mapeamento objeto-relacional com o objetivo de efetuar a gestão de dados relacionais em aplicações Java [62]. Como se trata apenas de uma especificação, tem que ser implementado por outras ferramentas tais como, *Hibernate*[33], *Spring Data JPA*, entre outros.

O *Spring Data JPA* adiciona uma camada de abstração sobre o *JPA*, ou seja, por outras palavras, o *Spring Data JPA* utiliza todos os recursos que estão definidos pela especificação e adiciona os seus próprios recursos com o objetivo de reduzir significativamente o esforço da implementação [63] [61].

Em relação à versão do *Spring Data JPA* utilizado na nossa ferramenta, como é um módulo disponibilizado pelo *Spring*, a versão coincide com a versão do *framework*, ou seja, a versão é a mesma que a do *Spring Boot.*

5.1.2.3 H2

H2 é um sistema de gestão de Base de Dados relacional *open-source* desenvolvido em Java, e tem como principais características [64]:

[32] Dispatcher Servlet https://www.baeldung.com/spring-dispatcherservlet
[33] Hibernate http://hibernate.org/

- **Ser rápido e possuir um tamanho reduzido**;
- **Aplicação de gestão da Base de Dados a partir do *browser***;
- **Permitir que a Base de Dados seja de memória**, isto é, permite criar uma Base de Dados e respetivas tabelas em memória;
- **Possuir dois modos: modo embutido e o de servidor**. No modo embutido, quando a execução da aplicação onde se encontra a Base de Dados termina, os dados armazenados na Base de Dados deixam de existir. O modo servidor é o modo habitual de qualquer Base de Dados, ou seja, é o modo que permite a persistência dos dados.

De acordo com [64], devido às suas características, esta Base de Dados é uma ótima solução para todas as aplicações que apenas necessitam de uma Base de Dados simples, no entanto, não é recomendada a sua utilização em ambientes de produção.

Na ferramenta *SMA* optámos por utilizar esta Base de Dados no modo embutido, sendo que a versão do *H2* utilizada foi a 1.4.197.

5.1.3 *Web Semântica*

5.1.3.1 Apache Jena

Apache Jena (Jena) é um *framework open-source* desenvolvida em *Java* para criar aplicações de *Web Semântica* e de *Linked Data*, sendo composto por diferentes *Application Programming Interface* (*APIs*) que interagem entre si de modo a processar dados *RDF* [65].

A Figura 27 apresenta parte do *framework* Jena, apenas aquela que é essencial para o funcionamento da ferramenta *SMA*. Nesta figura é possível observar as diversas serializações que este *framework* suporta, bem como as três *APIs* necessárias para o funcionamento da nossa ferramenta.

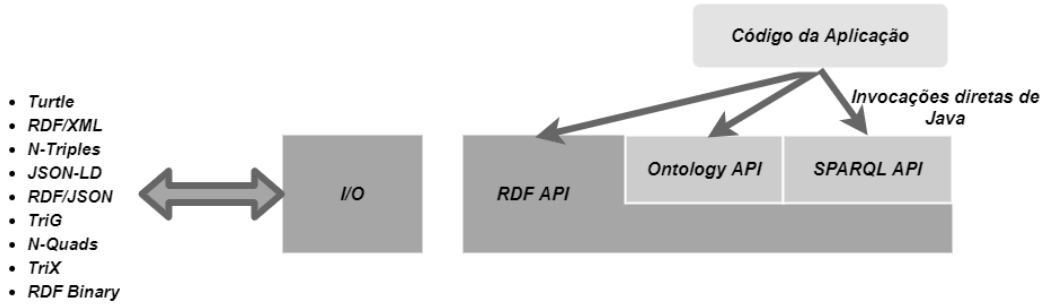

Figura 27 - Parte da Arquitetura do framework Apache Jena (Adaptado de [65])

As três *APIs* importantes para esta dissertação são: *API RDF* (em inglês, *RDF API*), *API OWL* (em inglês, *Ontology API)* e *API SPARQL* (em inglês, *SPARQL API*), sendo que a *API*

RDF serve de base para as outras duas *APIs*, porque o *OWL* é uma extensão do *RDF* e o *SPARQL* é uma linguagem de consulta que permite obter e manipular os dados armazenados em formato *RDF*.

O *framework Apache Jena* mantêm refletida a relação entre o *OWL* e o *RDF*, ou seja, de acordo com Cordeiro [66], as classes e interfaces relacionadas com o *OWL* usam ou estendem as classes ou interfaces da *API RDF* do *framework*. O caso do *SPARQL* é semelhante, pois o *framework* também mantêm refletida a sua relação com o *RDF*.

De acordo com Fernandes [67], as principais funcionalidades deste *framework* são:

- A leitura, o processamento e a gravação de dados em *RDF*;
- Manipulação de ontologias em *OWL*;
- Transformar dados através de regras, o que permite um raciocínio em fontes de dados *RDF* ou *OWL*;
- Fornecer mecanismos para a consulta em *SPARQL*.

Na ferramenta *SMA* fizemos uso a versão 3.12.0 do *framework Apache Jena*.

5.2 Arquitetura da ferramenta *SMA*

Nesta secção introduzimos a arquitetura da nossa ferramenta e explicamos detalhadamente todos os módulos que a compõem.

Na Figura 28, podemos observar a arquitetura da ferramenta *SMA* e constatar que a mesma é constituída por quatro módulos, ***GUI***, ***CRM***, ***CMS*** e ***ER***. Cada módulo é responsável por uma funcionalidade da ferramenta, dentre as quais efetuam a exportação de diferentes resultados. Por exemplo, o módulo ***CRM*** permite exportar uma lista de regras de mapeamento, enquanto que o módulo ***CMS*** permite exportar uma lista de mapeamentos *SPARQL*.

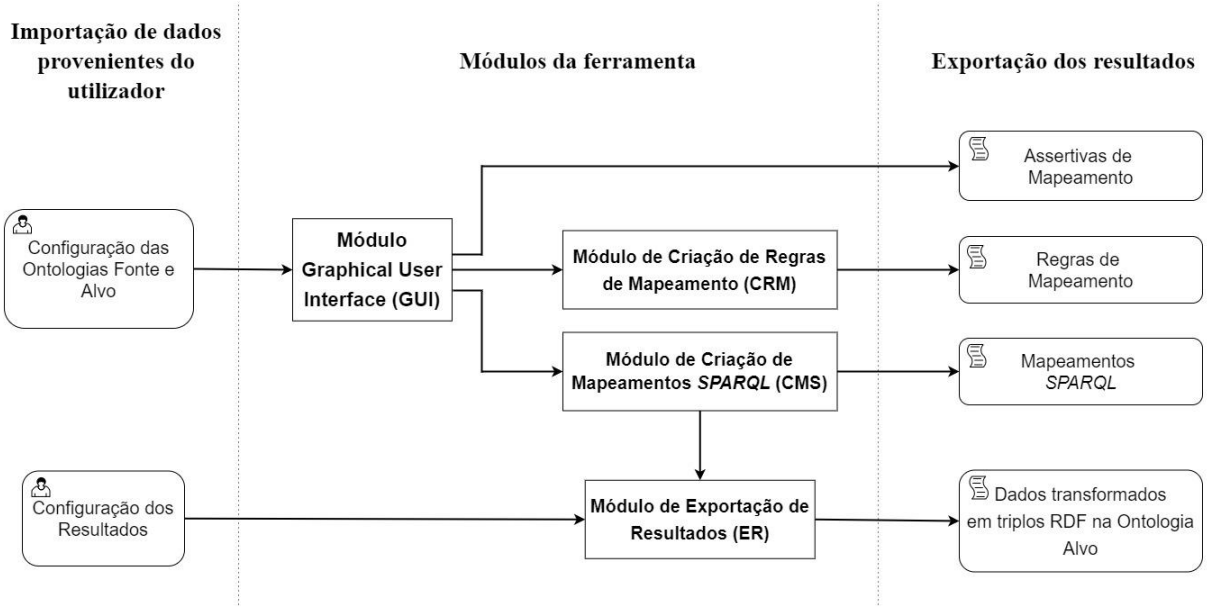

Figura 28 - Arquitetura da ferramenta SMA

De seguida, serão explanados os diversos módulos que compõem a nossa arquitetura, pela ordem em que os mesmos são utilizados, isto é, desde a configuração inicial das ontologias até à exportação de resultados.

- **Módulo *GUI*:** este módulo permite efetuar as configurações iniciais, ou seja, definir as ontologias fonte e alvo; indicar quais as linguagens de serialização em que cada uma das ontologias está escrita, de modo a que seja possível obter os dados das mesmas e; definir as *AMs*. A configuração inicial advém do preenchimento do formulário designado por *Configuração de um novo mapeamento* por parte do utilizador. Após esta configuração, tanto a ontologia fonte como a ontologia alvo irão ser apresentadas através de uma estrutura em árvore.
- **Módulo *CRM*:** módulo responsável pela criação de regras de mapeamento consoante cada *AM* que foi retornada pelo módulo *GUI*. O formalismo para a criação de regras de mapeamento foi apresentado no Capítulo 4. Este módulo permite exportar uma lista composta por todas as regras de mapeamento que foram definidas pelo utilizador.
- **Módulo *CMS*:** módulo responsável pela criação dos mapeamentos *SPARQL*, sendo que para cada *AM* retornada pelo módulo *GUI* é criado o mapeamento *SPARQL* (através dos padrões de mapeamento definidos no Capítulo 4). Este módulo, à semelhança do ***CRM***, também permite exportar uma lista de mapeamentos, neste caso, mapeamentos *SPARQL*.

- **Módulo *ER*:** módulo responsável por permitir a exportação dos resultados dos mapeamentos. A configuração dos resultados definidos pelo utilizador e os mapeamentos *SPARQL* gerados pelo módulo ***CMS*** são usados para apresentar os triplos *RDF*. A configuração por parte do utilizador é composta por três pontos: **(1)** um ficheiro com os triplos *RDF* que são instâncias da ontologia fonte; **(2)** a linguagem de serialização em que este ficheiro está definido e; **(3)** a linguagem de serialização que o utilizador pretende que seja utilizada aquando da exportação dos triplos *RDF*. Neste momento, as linguagens de serialização possíveis para exportação de resultados são as mesmas que para a importação das ontologias, ou seja, ***RDF/XML***, ***Turtle***, ***Notation3*** e ***N-Triples***.

5.3 Algoritmos necessários ao funcionamento da ferramenta

Nesta secção, iremos apresentar os dois algoritmos responsáveis por permitir a geração automática de mapeamentos *SPARQL*. Na Figura 29, apresentamos o primeiro algoritmo, designado por ***saveNewMapping***, que é chamado sempre que o utilizador adiciona uma nova *AM*. Este algoritmo é responsável por criar um novo mapeamento na Base de Dados (Algoritmo ***createMapping***), através da invocação dos algoritmos que permitem a criação de regras de mapeamento (**saveMapRule**) e mapeamentos *SPARQL*(**saveMapSPARQL**).

```
1. Método: saveNewMapping ()
2. Input: ψ
3.     mapRule = saveMapRule(ψ);
4.     mapSPARQL = saveMapSPARQL(ψ);
5.     createMapping(ψ, mapRule, mapSPARQL);
```

*Figura 29 – Algoritmo 1 (**saveNewMapping**) - Adicionar um novo mapeamento*

Como os módulos ***CRM*** e ***CMS*** apresentam algoritmos muito idênticos, apenas vamos apresentar o algoritmo ***saveMapSPARQL*** que pertence ao módulo ***CMS***, não apresentando assim, o algoritmo ***saveMapRule*** que pertence ao módulo ***CRM***. Na Figura 30, podemos observar este algoritmo e verificar que o *template* a ser utilizado irá variar consoante a *AM* definida pelo utilizador. Os *templates* invocados por este algoritmo correspondem aos que foram definidos anteriormente na secção 4.3.

```
1.  Método: saveMapSPARQL ()
2.  Input: ψ
3.  Output: mapeamento SPARQL
4.  prefixExp = getPrefixes(ψ);  ➔ Obtém os prefixos presentes na ψ
5.  queryExp = ".";
6.  uriExp = NULL;
7.  functionExp = NULL;
8.  if ψ é da forma ψc: C_T ≡ C_S / f then
9.      if f ≠ NULL then
10.         queryExp = getFilterExp(f);  ➔ Obtém o filtro f presente na ψ
11.     end if
12.     use o template T1;
13. else if ψ é da forma ψc: C_T ≡ C_S[A_1, ..., A_n] / f then
14.     queryExp = getQueryExp(ψc);
15.     if f ≠ NULL then
16.         queryExp = queryExp + getFilterExp(f);
17.     end if
18.     uriExp = getUriExp(ψc);  ➔ Obtém os valores das propriedades
19.     use o template T2;          A_1, ..., A_n presentes na ψ e concatena-os
20. else if ψ é da forma ψp: C_T / P_T ≡ C_S / φ / P_S / f / T then
21.     queryExp = getDomain(ψp); ➔ Obtém a AMC de domínio das
22.     if φ ≠ NULL then              propriedades
23.         queryExp = queryExp + getPath(φ) + " ; S: P_S ?P_S ."; ➔ Obtém o
24.     else                                                        caminho
25.         queryExp = queryExp + " ; S: P_S ?P_S .";               presente
26.     end if                                                      em ψ
27.     if f ≠ NULL then
28.         queryExp = queryExp + getFilterExp(ψp);
29.     end if
30.     if T ≠ NULL then
31.         functionExp = getFunctionExp(ψp);  ➔ Obtém a função de
32.         use o template T6;                    transformação T
33.     else                                      presente na ψ
34.         use o template T3;
35.     end
```

*Figura 30 - Algoritmo 2 (**saveMapSPARQL**) - Geração de Mapeamentos SPARQL*

```
36. else if ψ é da forma ψ_p: C_T / P_T ≡ C_S / φ / {P_S1, … , P_Sn} / T then
37.     queryExp = getDomain(ψ_p);
38.     if φ ≠ NULL then
39.         queryExp = queryExp + getPath(φ) + "S: P_S1 ? P_S1 ; … ; S: P_Sn ? P_Sn";
40.     else
41.         queryExp = queryExp + "S: P_S1 ? P_S1 ; … ; S: P_Sn ? P_Sn";
42.     end
43.     if T ≠ NULL then
44.         functionExp = getFunctionExp(ψ_p);
45.         use o template T7;
46.     else
47.         use o template T8;
48.     end
49. else if ψ é da forma ψ_p: C_T / P_T = C_S[A_1 … A_n] / A_i / f then
50.     queryExp = getDomain(ψ_p);
51.     if f ≠ NULL then
52.         queryExp = queryExp + getFilterExp(f)
53.     end if
54.     uriExp = getUriExp(ψ_p);
55.     ?Ai = A_i;
56.     use o template T4;
57. else
58.     queryExp = getDomain(ψ_p);
59.     uriExp = getUriExp(ψ_p);
60.     use o template T5;
61. end if
```

*Figura 31 - Algoritmo 2 (**saveMapSPARQL**) - Geração de Mapeamentos SPARQL (Continuação)*

O Algoritmo ***saveMapSPARQL*** (Figura 30 e Figura 31) assume como parâmetro de entrada a *AM* definida pelo utilizador. Em primeiro lugar, o algoritmo verifica qual o tipo de *AM* definida pelo utilizador. Caso se trate de uma *AMC* o processo é mais simples, pois basta verificar se se trata de uma *AMC* Semelhante Semanticamente (linha 8) ou Não Semelhante Semanticamente (linha 13), aplicar o filtro caso exista, carregar a variável *uriExp* no caso da *AMC* Não Semelhante Semanticamente (linha 18) e, por fim, aplicar o *template* correspondente.

No caso de uma *AMD/AMO* o processo é mais complexo, pois envolve a necessidade de obter diversas informações, entre elas, a *AMC* de domínio. A *AMC* de domínio das propriedades é obtida através da invocação da função ***getDomain(ψ_p)***. Além desta informação,

estas *AMs* podem possuir uma expressão de caminho φ. Neste caso, é necessário converter a expressão de caminho para a sintaxe de caminho da linguagem *SPARQL 1.1*, através da função ***getPath(*ψ_p*)***, e adicioná-la à variável *queryExp*.

A *AMD* ainda possui uma particularidade que não se encontra na *AMO*, que é o facto de poder ser necessário usar uma função de transformação, representada aqui por T. Neste caso, a variável *functionExp* irá possuir como valor uma função de transformação disponível na linguagem *SPARQL 1.1*. Este valor é obtido através da invocação da função ***getFunctionExp(***T***)***.

Seja qual for a *AM* definida pelo utilizador, o algoritmo irá obter os prefixos utilizados através da invocação da função ***getPrefixes(*ψ*)*.** Caso a *AM* definida pelo utilizador possua uma expressão de filtro **f**, o algoritmo converte a expressão para estar de acordo com a sintaxe da linguagem *SPARQL*, através da função ***getFilterExp*(f)**.

5.3.1 *Geração de um Mapeamento de Classes*

A Figura 32, apresenta um mapeamento *SPARQL* entre as classes *bo:Book* e *dbo:Book*, tendo como filtro "*dbo:literaryGenre != 'Romance'* ". Esse mapeamento foi obtido após a chamada ao algoritmo ***saveNewMapping*()** (apresentado na Figura 29), em que foi passado como parâmetro a ψ_3 (apresentada no Capítulo 4, aquando a explicação do Padrão de Mapeamento MC1). O algoritmo ***saveNewMapping*()** chama o algoritmo ***saveMapSPARQL*()** para criar o mapeamento *SPARQL* (apresentado nas Figura 30 e Figura 31). Como esta *AMC* ψ_3 é da forma: ψ_c: $C_{\mathbf{T}} \equiv C_{\mathbf{S}}$ / **f**, o algoritmo utilizou o *template* T1 para a criação do mapeamento *SPARQL*.

De seguida, apresentamos uma breve comparação entre o *template* T1 apresentado no Capítulo 4 e o mapeamento *SPARQL* utilizado como exemplo (Figura 32).

```
1  # Template T1 - Mapeamento de Classes
2  PREFIX dbo: <http://dbpedia.org/ontology/>
3  PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl/>
4  CONSTRUCT { ?SUBJ a bo:Book . }
5  WHERE {?SUBJ a dbo:Book ; dbo:literaryGenre ?o . FILTER(?o != 'Romance') }
```

Figura 32 - Mapeamento SPARQL ψ_3 gerado com o template T1

A cláusula ***PREFIX*** que aparece nas linhas 2 e 3 da Figura 32 corresponde a cláusula "**PREFIX** prefixExp" no *template* T1, sendo que o valor da variável *prefixExp* foi obtido através da função ***getPrefixes*(**ψ_3**)** que recebeu como parâmetro a *AMC* ψ_3.

A cláusula ***CONSTRUCT***, na linha 4, corresponde a "**CONSTRUCT {** ?SUBJ a **T**: $C_{\mathbf{T}}$ **.** **}**" no *template* T1, em que a classe **T**: $C_{\mathbf{T}}$ foi substituída pela classe da ontologia alvo, *bo:Book*.

Por fim, na linha 5, a cláusula ***WHERE*** corresponde a "**WHERE {** ?SUBJ a **S**: C_S queryExp **}** " no *template* T1, em que a classe **S**: C_S foi substituída pela classe da ontologia fonte, *dbo:Book* e a variável *queryExp* foi convertida em " *; dbo:literaryGenre ?o . FILTER(?o != 'Romance')* " utilizando a função ***getFilterExp*(**ψ_3**)** (linha 10 do algoritmo ***saveMapSPARQL*()**).

5.3.2 *Geração de um Mapeamento de Propriedades*

Na Figura 33, apresentamos um mapeamento *SPARQL* entre duas propriedades de tipos de dados, *bo:isbn* e *dbo:isbn*. Os domínios destas propriedades correspondem, respetivamente, a *bo:Book* e *dbo:Book*. Esse mapeamento foi obtido após a chamada ao algoritmo ***saveNewMapping*()** (apresentado na Figura 29), em que foi passado como parâmetro a ψ_5 (apresentada no Capítulo 4, aquando a explicação do Padrão de Mapeamento MD1). O algoritmo ***saveNewMapping*()** chama o algoritmo ***saveMapSPARQL*()** para criar o mapeamento *SPARQL* (apresentado nas Figura 30 e Figura 31). Como esta *AMD* é da forma: ψ_p: C_T / $P_T \equiv C_S$ / φ / P_S / **f** / T e não possui nenhuma função de transformação T, o algoritmo utilizou o *template* T3. Caso possuísse uma função de transformação T, o algoritmo iria utilizar o template T6. Tal como no exemplo anterior, iremos efetuar um breve paralelismo entre o *template* T3 apresentado no Capítulo 4 e o mapeamento *SPARQL* que foi gerado pela nossa ferramenta (Figura 33).

A cláusula ***PREFIX***, tal como no exemplo anteriormente, corresponde à cláusula "**PREFIX** prefixExp" no template T3, sendo o valor da variável *prefixExp* obtido da mesma maneira.

```
1  # Template T3 - Mapeamento de Propriedade
2  PREFIX dbo: <http://dbpedia.org/ontology/>
3  PREFIX bo: <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl/>
4  CONSTRUCT { ?SUBJ bo:isbn ?p . }
5  WHERE { ?SUBJ a dbo:Book ; dbo:isbn ?p; dbo:literaryGenre ?o .
6   FILTER(?o != 'Romance') }
```

Figura 33 - Mapeamento SPARQL ψ_6 gerado com o template T3

A cláusula ***CONSTRUCT***, na linha 4, corresponde a "**CONSTRUCT {** ?SUBJ **T**: P_T ?p . **}**" no *template* T3, em que a propriedade **T**: P_T foi substituída pela propriedade da ontologia alvo, *bo:isbn*.

Por fim, a cláusula ***WHERE***, na linha 5, corresponde a "**WHERE{** ?SUBJ a **S**: P_S queryExp **}**" no *template* T3. Ao contrário dos mapeamentos de classes, os mapeamentos de propriedades possuem mais particularidades. Assim, como se pode observar no algoritmo

***saveMapSPARQL*(ψ_5)** (Figura 30 e Figura 31), o valor da variável *queryExp* pode ser obtido através de três etapas:

- Obtenção do mapeamento de classes da classe de domínio das propriedades, através da função ***getDomain*(ψ_6)**;
- Obtenção das expressões de caminhos, através da função ***getPath*(φ)**, caso exista um caminho na φ *AMD/AMO*;
- Obtenção do filtro f, através da função ***getFilterExp*(f)**, caso exista um filtro f na *AMD/AMO*.

No nosso exemplo, o algoritmo ***getDomain*()** obtém o mapeamento entre *bo:Book* e *dbo:Book*, através da *AMC* ψ_3. A *AMD* ψ_6 permite expressões de caminho e filtro, logo o algoritmo tem de verificar se a ψ_6 as possui, de modo a convertê-las para a linguagem *SPARQL* e adicioná-las ao conteúdo da variável *queryExp*. Além disso, a *AMD* ψ_6 também permite função de transformação, logo o algoritmo tem de verificar a existência da função na ψ_6 (a sua existência altera o *template* utilizado pelo algoritmo, como se pode ver na linha 30 do algoritmo ***saveMapSPARQL*()**), de modo a convertê-la para a linguagem *SPARQL* e adicioná-la a variável *functionExp*. A *AMD* ψ_6 não possui expressões de caminho, filtro, nem função de transformação, logo irá ser utilizado o *template* T3 e o conteúdo da variável *queryExp* é obtido apenas pelo algoritmo ***getDomain*()**. Assim, o valor da variável *queryExp* é: "*dbo:literaryGenre ?o . FILTER(?o != 'Romance')*".

5.4 Interface Gráfica

Nesta secção abordamos a interface gráfica da *SMA*, em que demonstramos todos os passos que são necessários para gerar os mapeamentos *SPARQL* entre duas ontologias. Para apresentar a nossa ferramenta, iremos utilizar o estudo de caso apresentado no Capítulo 4.

De uma forma muito resumida, o processo começa com a configuração inicial das duas ontologias, que será apresentada na secção 0. Após tal configuração, é possível criar diversos mapeamentos entre as mesmas, o qual será apresentado na secção 5.4.2. Para que seja possível proceder à extração dos resultados dos mapeamentos, é necessário efetuar uma configuração final, que será apresentada na secção 5.4.3.

5.4.1 Configuração inicial das ontologias

Antes de iniciar a criação de mapeamentos entre ontologias, é necessário indicar à partida quais as ontologias fonte e alvo, através da importação dos ficheiros onde as mesmas estão descritas e a indicação do nome das ontologias e da linguagem de serialização em que os ficheiros estão escritos. O nome das ontologias fonte e alvo apenas é relevante na extração dos resultados, dado que o nome do ficheiro gerado corresponde à concatenação do nome da ontologia fonte e alvo. Utilizando o estudo de caso apresentado anteriormente, o nome da ontologia fonte poderia ser "DBpedia_S" e o da ontologia alvo "MyBook_T", logo nesta situação o nome do ficheiro gerado será "DBpedia_S_to_MyBook_T".

Na Figura 34, podemos observar o formulário disponibilizado ao utilizador para a inserção dos dados referentes à configuração das ontologias fonte e alvo.

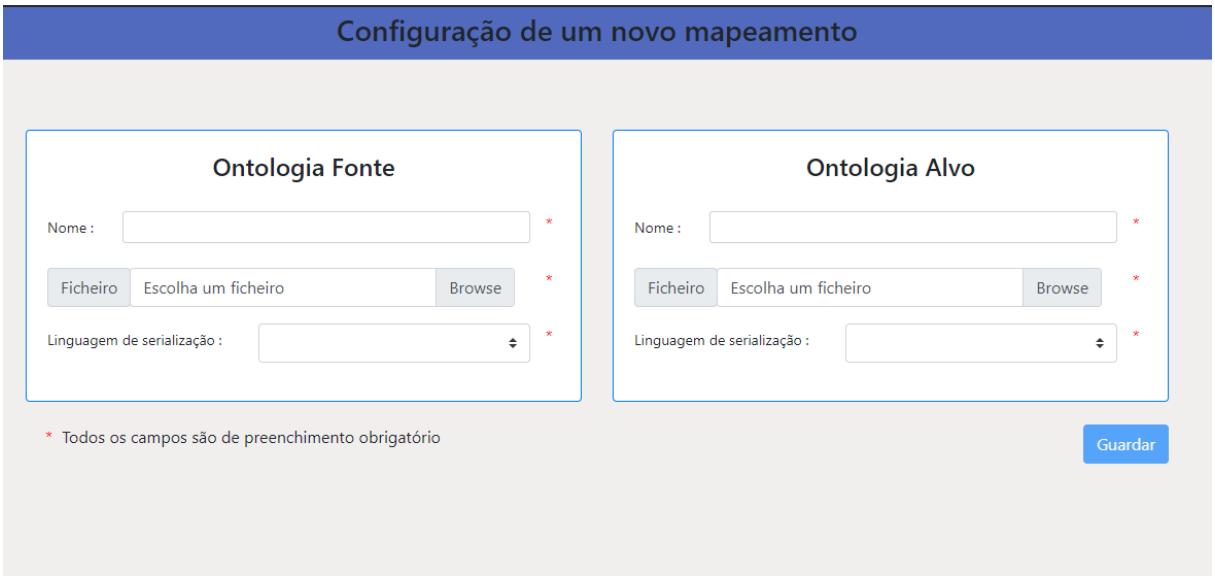

Figura 34 - Página inicial de configuração de ontologias

Após a conclusão da configuração inicial, o módulo ***GUI*** apresenta a página de edição de mapeamentos, em que à esquerda é apresentada, através de uma estrutura em árvore, a ontologia alvo e, à direita, são apresentadas as diversas funcionalidades disponibilizadas na ferramenta, tal como pode ser observado na Figura 35.

Figura 35 - Página de edição de mapeamentos

Inicialmente a página possui as funcionalidades desabilitadas, com exceção das funcionalidades do menu, designadas por **Exportar**, **Obter resultados** e **Ajuda**, sendo obrigatório escolher uma classe ou propriedade da ontologia alvo para poder começar a efetuar os mapeamentos.

5.4.2 Criação de Mapeamentos

Na criação de mapeamentos coexistem três módulos, **GUI**, **CRM** e **CMS**, sendo que o **GUI** é responsável pela criação de AMs (etapas 1 e 2), enquanto que os módulos **CRM** e **CMS** só são utilizados na etapa 3, após a criação da assertiva. De seguida, descrevemos o que é efetuado em cada etapa:

Etapa 1 - Seleção do elemento da ontologia alvo e do elemento da ontologia fonte:

Para realizar esta etapa é necessário, em primeiro lugar, escolher uma classe ou propriedade da ontologia alvo e clicar no botão "Ontologia Fonte". Como se pode observar na Figura 36, após ser escolhido um elemento da ontologia alvo, o botão "**Ontologia Fonte**" aparece habilitado (aparece com uma cor mais escura do que estava antes, compare com a Figura 35).

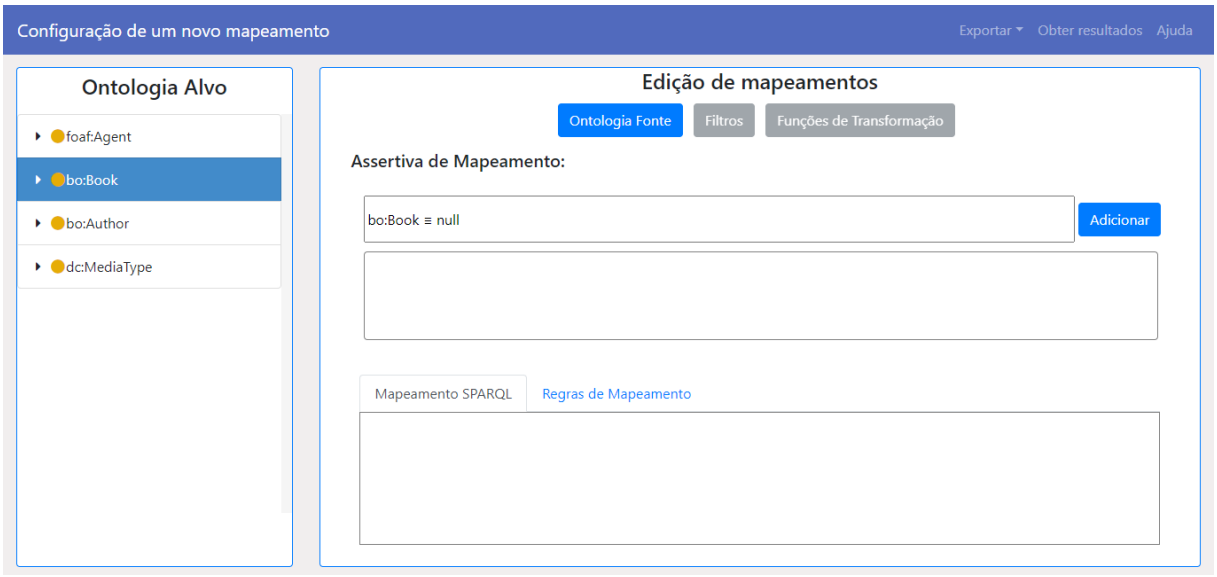

Figura 36 - Classe da ontologia fonte escolhida

Após essa operação, irá aparecer uma modal que contém a hierarquia de classes e propriedades que compõem a ontologia fonte, tal como se pode observar na Figura 37.

Figura 37 - Estrutura em árvore da ontologia fonte

Na Figura 37, abrimos a classe *dbo:Language* de modo a que fosse perceptível visualizar a forma como aparecia as propriedades que possuem como domínio esta classe. Além disso, podemos observar que o botão "Selecionar" está desabilitado até que seja escolhido algum elemento desta ontologia.

Para concluir esta etapa, basta escolher uma classe ou propriedade e clicar no botão "Selecionar". No nosso exemplo, vamos ilustrar a criação do mapeamento entre classes semelhantes semanticamente (Padrão de Mapeamento MC1), sendo que iremos focar-nos no exemplo com filtros ($\boldsymbol{\psi_3}$). De seguida, a *AM* pode ser visualizada na caixa de texto que possui um botão "Adicionar" à frente (Figura 38).

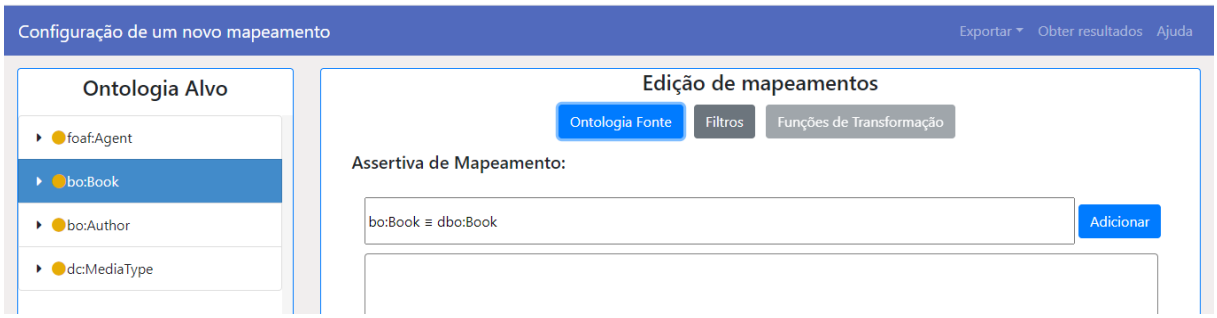

Figura 38 - Assertiva de mapeamento em construção

Tal como podemos observar na Figura 38, as funcionalidades "**Filtros**" e "**Adicionar**" já se encontram habilitadas (a cor dos botões "**Filtros**" e "**Adicionar**" estão com uma cor mais escura do que originalmente, compare com a Figura 35). A funcionalidade "**Funções de Transformação**" continua desabilitada, pois tal como se pode observar no Capítulo 4, apenas os Padrões de Mapeamento de Tipos de Dados é que possuem a possibilidade de indicar a função de transformação a aplicar ao valor da(s) propriedade(s) da ontologia fonte para obter o valor pretendido na ontologia alvo.

Etapa 2 - inclusão de filtros e/ou funções de transformação na assertiva:

Esta etapa é opcional, logo caso não se queira adicionar filtros nem funções de transformação na assertiva em questão, basta ignorar esta etapa e passar para a terceira e última etapa. Voltando ao nosso exemplo, queremos apenas mapear as instâncias da classe *dbo:Book* que não são livros de romance, assim basta clicar no botão "Filtros" e preencher o formulário que aparece na Figura 39. Para tal, escolhemos a propriedade "dbo:literaryGenre", o operador "!=" e escrevemos o valor "Romance".

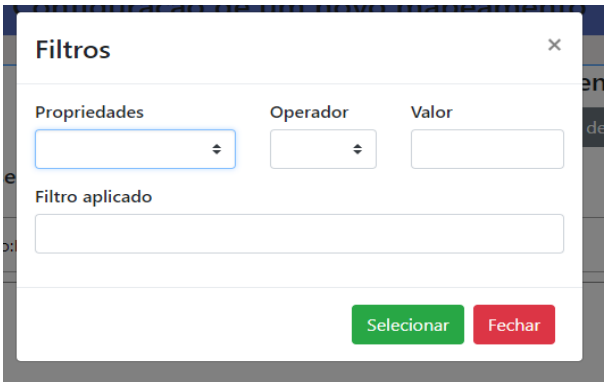

Figura 39 - Opções permitidas na funcionalidade filtro

De seguida, clicamos no filtro aplicado, o qual retorna a seguinte condição: "dbo:literaryGenre!= 'Romance'" (Figura 40). Este é o filtro que será aplicado à *AM* que está a ser construída.

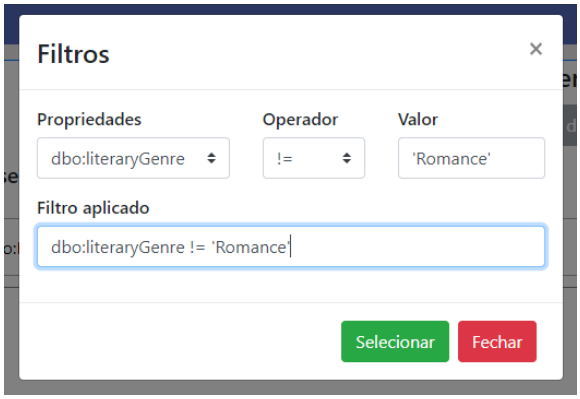

Figura 40 - Filtro aplicado

Etapa 3 - inclusão da *AM* na lista de todas as assertivas e criação dos mapeamentos:

Após a conclusão da *AM*, basta clicar no botão "Adicionar" para que a mesma fique adicionada à lista de todas as assertivas. Quando a assertiva é adicionada, irão ser chamados os módulos **CRM** e **CMS** para proceder, respetivamente, à criação da regra de mapeamento e do mapeamento *SPARQL* correspondente.

Uma vez que a *AM* foi adicionada à lista de assertivas de mapeamento, esta passará a constar na caixa de texto representada com um "2" na Figura 41. Quando carregamos na assertiva, é possível observar o mapeamento *SPARQL* e a regra de mapeamento associada à mesma. Na Figura 41, na caixa de texto indicada com um "3", podemos observar o mapeamento *SPARQL* associado à assertiva " *bo:Book* ≡ *dbo:Book* / *dbo:literaryGenre* != "Romance" " e na Figura 42, a correspondente regra de mapeamento é apresentada (após se ter clicado na tab "Regras de Mapeamento").

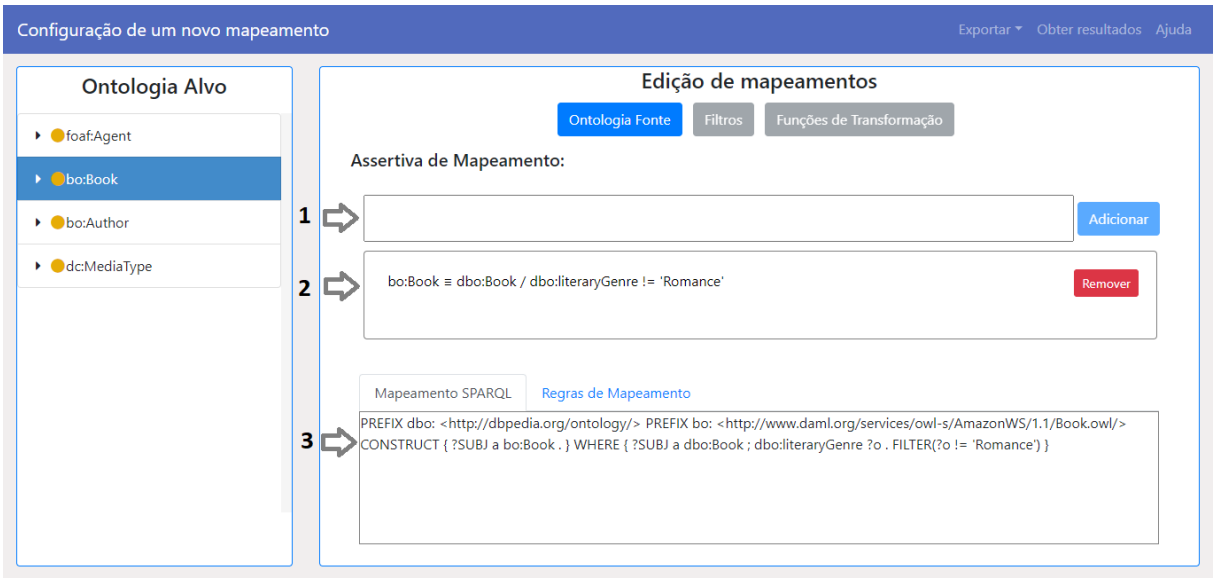

Figura 41 - Assertiva adicionada com sucesso

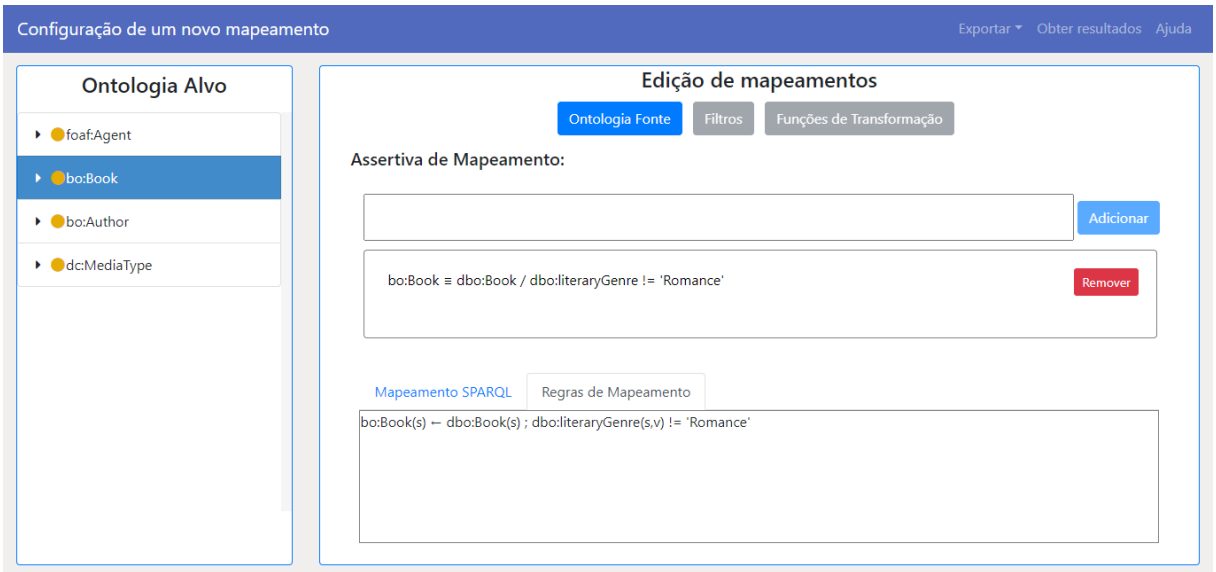

Figura 42 - Regra de Mapeamento

Apesar de não ser possível editar uma assertiva que já tenha sido adicionada à lista, é possível removê-la. Temos duas maneiras de fazer isto: **1)** a partir do botão remover ao lado da caixa de texto na janela "Edição de Mapeamentos" (Figura 41, número "2") e; **2)** a partir do botão remover que aparece ao lado de cada *AM* na janela "Lista de Assertivas" que é apresentada na Figura 44.

5.4.3 Geração de Mapeamentos *SPARQL*

Figura 43 – Menu superior da ferramenta SMA

No menu superior da ferramenta *SMA* (apresentado na Figura 43), quando carregamos na opção "Obter resultados", irá aparecer uma *modal* com a Listagem das Assertivas que foram criadas até ao momento (tal como pode ser observado na Figura 44).

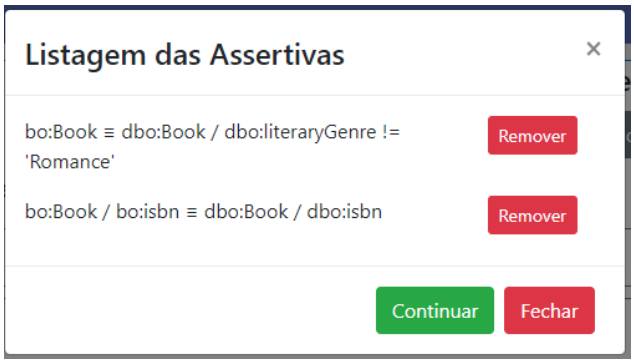

Figura 44 - Listagem de todas as assertivas

A partir desta *modal*, para além de podermos visualizar todas as *AMs* criadas e de removê-las, se quisermos, podemos ainda gerar o *tripleset* da ontologia alvo. Para isso, carregamos no botão "Continuar".

Ao carregar no botão "Continuar", irá aparecer um formulário (Figura 45) para obter toda a informação necessária à obtenção dos triplos *RDF* da ontologia fonte, nomeadamente: o ficheiro que contém os triplos *RDF* que são instâncias da ontologia fonte; a linguagem de serialização em que o mesmo está escrito e; a linguagem de serialização que o utilizador pretende que seja utilizada no momento da exportação dos resultados. De seguida, ao carregar no botão "Guardar", é iniciada a execução do módulo ***ER***, o responsável pela exportação do *tripleset* da ontologia alvo**.**

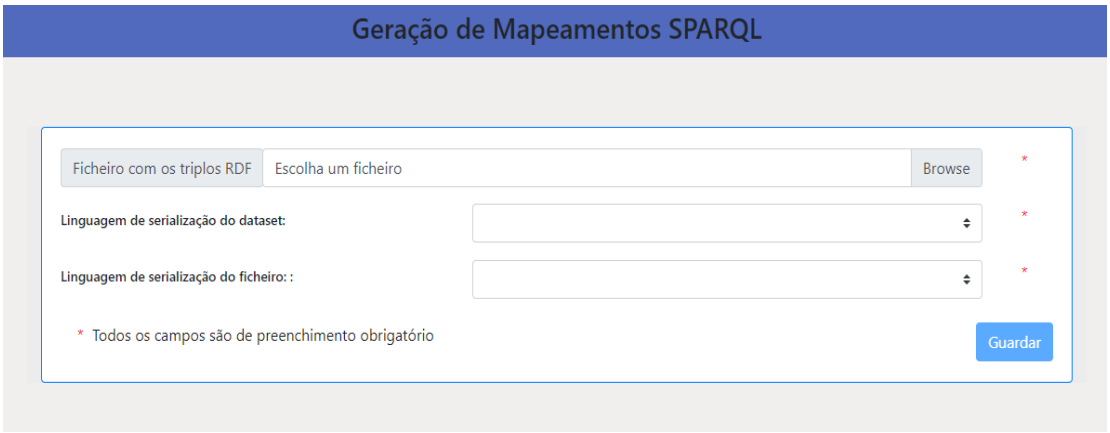

Figura 45 - Formulário de configuração final

A Figura 46 apresenta um fragmento do *tripleset* gerado a partir da nossa ferramenta para o estudo de caso mostrado nas secções 5.3.1 e 5.3.2. Nesta figura, a listagem foi apresentada na linguagem *Turtle*, no entanto, a nossa ferramenta também permite gerar os triplos em *RDF/XML*, *Notation3* e *N-Triples.*

```
@prefix dbo:    <http://dbpedia.org/ontology/> .
@prefix bo:     <http://www.daml.org/services/owl-s/AmazonWS/1.1/Book.owl/> .

dbo:As_Ruinas  a  bo:Book ;
        bo:isbn  "978-85-9807-820-5" .

dbo:O_Povo_do_Abismo  a  bo:Book ;
        bo:isbn  "978-97-2608-350-4" .

dbo:Os_100  a    bo:Book ;
        bo:isbn  "978-14-4476-688-2" .

dbo:Perdida_no_espaço
        a        bo:Book ;
        bo:isbn  "978-97-2253-742-1" .

dbo:O_exorcista  a  bo:Book ;
        bo:isbn  "978-85-2203-157-3" .
```

Figura 46 – Fragmento do tripleset gerado pela SMA para o estudo de caso da secção 5.4

5.4.4 Exportação de listagens

Quando carregamos na *dropdow* "Exportar" do menu superior (Figura 43), são apresentadas três opções como podemos ver na Figura 47.

Cada opção irá gerar um ficheiro de texto com uma listagem. Assim sendo, podemos afirmar que ao carregar na opção:

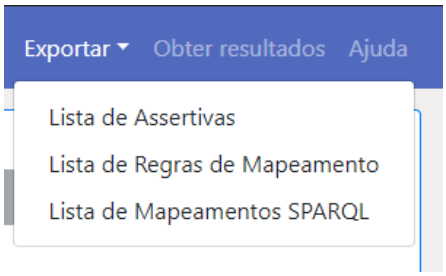

Figura 47 - Opção "exportar"

- **"Lista de Assertivas"** irá ser gerado um ficheiro de texto constituído por todas as assertivas criadas na ferramenta até ao momento. O módulo responsável por esta geração é o **GUI;**
- **"Lista de Regras de Mapeamento"** irá ser gerado num ficheiro de texto uma lista com as regras de mapeamento existentes, sendo o módulo **CRM** responsável por esta geração;
- **"Listagem de Mapeamentos *SPARQL*"** irá ser gerado um ficheiro de texto com todos os mapeamentos *SPARQL*. O módulo responsável pela geração desta listagem é o **CMS**.

5.5 Considerações finais do capítulo

Em relação às tecnologias e ferramentas, a nossa proposta incluía três requisitos base, sendo estes:

- **Ser uma aplicação web:** deste modo não é necessário instalar uma aplicação num ambiente como, por exemplo, no caso de uma aplicação *Desktop*. Além disso, ao ser uma aplicação *web* possui uma maior visibilidade;
- **Possuir uma Base de Dados em que os dados não fossem persistidos:** pois apenas necessitávamos de armazenar os dados enquanto estivessem a decorrer os mapeamentos entre as duas ontologias, ou seja, por outras palavras, não necessitávamos de uma Base de Dados em que os dados fossem persistidos, mas sim de uma em que os dados se mantivessem disponíveis enquanto a aplicação estivesse a funcionar;
- **Não possuir dependências:** deste modo pode ser executada em qualquer ambiente, sem que haja necessidade de possuir um servidor aplicacional e um servidor de Base de Dados.

Em relação ao primeiro requisito, ou seja, o facto de ser uma aplicação *web*, tal foi conseguido através do *HTML*, *CSS*, *Bootstrap*, *AngularJS* e *Spring Boot*. Como é sabido, o *HTML*, o *CSS* e o *JavaScript* são as "linguagens" de base para a criação de páginas *web*, em que o *HTML* tem como objetivo criar conteúdo estático na *web*; o *CSS* permite definir a aparência/apresentação de uma página *web*; e por último, o *JavaScript* permite descrever o comportamento de uma página *web* [6]. Na nossa proposta, utilizámos o *Bootstrap* para facilitar a definição da apresentação das páginas *web* e decidimos utilizar o *AngularJS* ao invés do *JavaScript* puro, pois este *framework* permite, entre outros, facilitar a programação em

JavaScript. Além disso, o *Spring Boot* permite criar aplicações *web* e colocá-las em funcionamento em pouco tempo.

Por último, em relação ao segundo e terceiro requisito, tal foi conseguido devido ao facto de termos desenvolvido a aplicação em *Spring Boot* e por possuirmos uma Base de Dados embutida (*H2*). Em relação ao *Spring Boot*, já sabemos que possui incorporado um *servlet container*, ou seja, podemos executar uma aplicação *Spring Boot* utilizando apenas a *Java Virtual Machine* (*JVM*). Em relação à Base de Dados, como o *H2* possui o modo embutido, foi-nos permitido embutir esta Base de Dados na nossa aplicação sem que haja a necessidade de possuir um servidor de Base de Dados. O facto do *H2* estar no modo embutido também nos permite satisfazer o segundo requisito, pois tal como foi explicado anteriormente, no momento em que a execução da aplicação termina, a Base de Dados e consequentemente os dados lá armazenados deixam de existir.

Após apresentarmos todas as tecnologias e ferramentas que foram usadas no desenvolvimento da nossa ferramenta, expusemos a nossa arquitetura, descrevendo todos os módulos pertencentes à mesma. Além disso, apresentámos também os algoritmos necessários à criação de mapeamentos *SPARQL*.

Por último, mostrámos as principais funcionalidades da nossa ferramenta, em que explicámos passo a passo todo o processo necessário para a criação de mapeamentos entre ontologias, sendo que utilizámos como base o estudo de caso apresentado no Capítulo 4, que retrata um cenário acerca de obras literárias.

Apesar de não terem sido mostrados todos os tipos de *AMs*, inclusive com funções de transformação, na ferramenta foram implementados todos os padrões definidos no Capítulo 4.

O código fonte da ferramenta *SMA*, bem como um pequeno manual de utilização encontra-se em: https://github.com/verap95/SMA

6 Conclusão

A área de pesquisa desta dissertação prende-se essencialmente com as soluções para a mitigação dos problemas relacionados com a heterogeneidade entre ontologias. Existem diversas abordagens para lidar com a heterogeneidade entre ontologias, dependendo do tipo de problema que se deseja tratar. Neste trabalho, enumeramos as duas abordagens mais comuns: o *Mapeamento entre Ontologias* e o *Alinhamento de Ontologias*, tendo sido a primeira abordagem o focus do corrente trabalho.

Para lidar com a heterogeneidade entre ontologias, o presente trabalho propôs uma extensão ao trabalho proposto por Vinuto em [14]. Em [14], Vinuto propõe padrões de mapeamento para solucionar os problemas de mapeamentos mais comuns. Para cada padrão de mapeamento é apresentado um ou mais *templates* que permitem a geração de mapeamentos na linguagem *R2R*. A nossa extensão consiste em acrescentar aos padrões de mapeamento já existentes, *templates* que permitam a geração de mapeamentos *SPARQL 1.1*.

Para proporcionar suporte à extensão efetuada, criámos uma ferramenta, designada por *SMA*, que visa auxiliar e simplificar a tarefa de geração de mapeamentos *SPARQL 1.1*. Essa abordagem difere de algumas ferramentas encontradas na literatura, tais como, a ferramenta *RBA* e o *framework LDIF*, que utilizam a linguagem *R2R* para realizar o mapeamento entre as ontologias. Além disso, a *SMA* foi construída de modo a simplificar também a forma como pode ser utilizada, ou seja, desenvolvemos uma ferramenta em que não é necessário qualquer configuração para a sua execução e é apelativa na medida em que foi desenvolvida como uma aplicação *web*.

A ferramenta *SMA* foi implementada com a mesma base da ferramenta *RBA*, isto é, utiliza *AMs*, regras de mapeamento e padrões de mapeamento para especificar os mapeamentos entre as ontologias. Os padrões de mapeamentos que foram implementados na nossa ferramenta foram apresentados no Capítulo 4 e, no Capítulo 5, apresentamos passo a passo como é que podemos criar mapeamentos *SPARQL*, utilizando, para exemplificar, o estudo de caso introduzido no Capítulo 4.

Não se chegou a publicar nenhum artigo sobre o trabalho aqui apresentado, no entanto, espera-se algumas publicações sobre a parte teórica e sobre a ferramenta *SMA* em conferências nacionais ou internacionais.

7 Trabalho futuro

Ao longo do desenvolvimento desta dissertação foram colocadas em prática várias funcionalidades, no entanto, ainda existem inúmeras possibilidades que visam a continuação do desenvolvimento da nossa ferramenta, sendo estas:

- Permitir a possibilidade de introduzir ficheiros com listas de *AMs*, regras de mapeamento ou ainda mapeamentos *SPARQL* aquando a configuração inicial das ontologias: como podemos efetuar a exportação de três listas diferentes (*AMs*, regras de mapeamento, mapeamento *SPARQL*) faria sentido permitir que as mesmas fizessem parte do input da ferramenta *SMA*. Deste modo, o utilizador poderia ir aprimorando os mapeamentos à medida que tivesse necessidade sem ter de começar todos os mapeamentos do zero;
- A implementação de todas as possibilidades de filtros e de funções de transformação: de todas as funções e filtros permitidos pelo *SPARQL*, apenas contemplámos alguns dos que considerámos como sendo os usualmente utilizados (apresentados no Capítulo 2). Além disso, é necessário também ter em atenção quais os filtros e funções de transformação que são permitidos no *framework Apache Jena*, pois é a partir desta que são executados os mapeamentos *SPARQL* para a exportação dos resultados;
- Permitir que as ontologias possam ser provenientes de um *URL*: neste momento, a nossa ferramenta apenas contempla a definição da ontologia através de um ficheiro, no entanto, deveria também ser possível adicionar a definição da ontologia através de um *URL*;
- Permitir a edição de uma *AM*: depois de adicionada uma *AM* à lista, apenas é permitido removê-la. Seria interessante permitir também que a mesma possa ser editada;
- Implementar validações para minimizar o erro humano: a nossa ferramenta possui poucas validações para minimizar o erro humano, sendo que seria importante adicioná-las de modo a que o resultado corresponda realmente ao que o utilizador pretendia aquando da criação dos mapeamentos. Por exemplo, ao adicionar um filtro ou uma função de transformação à assertiva de mapeamento, atualmente a ferramenta não verifica se o utilizador está a introduzir tipos de dados incorretos. No caso de um filtro, o tipo da variável da ontologia fonte pode ser uma *string* e o utilizador estar a tentar igualar a um valor inteiro. Este caso

em específico não vai incorrer num erro da aplicação, mas o utilizador também não vai obter os resultados pretendidos.

8 Bibliografia

[1] A. Fernandes. (2018, Março) O que é API? Entenda de uma maneira simples. Acedido em 21 Julho 2019. [Online]. Disponível em: https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples/

[2] Entenda o que é um Framework. Acedido em 21 Julho 2019 [Online]. Disponível em: https://gaea.com.br/entenda-o-que-e-framework/

[3] (2019, jan) HTTP. Acedido em 23 Fevereiro 2019. [Online]. Disponível em: http://developer.mozilla.org/pt-PT/docs/Web/HTTP

[4] F. C. Santos e C. L. d. Carvalho, "Aplicações de Suporte à Web Semântica", Universidade Federal de Goiás, Relatório Técnico, 2007

[5] E. Brito. (2014, Novembro) Java: Entenda para que serve o software e os problemas da sua ausência. Acedido em 02 Agosto 2019. [Online]. Disponível em: https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2014/11/java-entenda-para-que-serve-o-software-e-os-problemas-da-sua-ausencia.html

[6] J. C. Meloni. HTML, CSS and JavaScript (2ª Edição). Indiana, USA: Sams, 2015. ISBN: 13-978-0-672-337-14-1

[7] (2008) Introdução aos JavaBeans. Acedido em 10 Agosto 2019. Disponível em: https://www.devmedia.com.br/introducao-aos-javabeans/8621

[8] S. Ladd, D. Davison, S. Devijver, C. Yates. Expert Spring MVC and Web Flow. Apress, 2006.

[9] K. U. Idehen. (2018, Agosto) What is a SPARQL Endpoint, and why is it important? Acedido em 28 Março 2020. [Online]. Disponível em: https://medium.com/virtuoso-blog/what-is-a-sparql-endpoint-and-why-is-it-important-b3c9e6a20a8b

[10] Tour of Scala: Introduction. Acedido em 21 Julho 2019. [Online]. Disponível em: https://docs.scala-lang.org/tour/tour-of-scala.html

[11] R. Wang. (2013, Março) What is a Servlet Container? Acedido em 04 Agosto 2019. [Online]. Disponível em: https://dzone.com/articles/what-servlet-container

[12] T. Berners-Lee, “Www: Past, Present and Future,” Computer, vol. 29, no. 10, pp. 69–77, 1996. [Online]. Disponível em: https://www.w3.org/People/Berners-Lee/1996/ppf.html

[13] D. S. Miranda, L. L. Azevedo e R. P. Magalhães, “Consumindo Linked Data na Web,” em Encontro Unificado de Computação em Parnaíba, Novembro 2012.

[14] T. d. S. Vinuto, “Uma abordagem para a geração semiautomática de mapeamentos R2R baseado em padrões,” Dissertação de Mestrado, Universidade Federal do Ceara, Brasil, 2017.

[15] A. Schultz, A. Matteini, R. Isele, P. N. Mendes, C. Bizer e C. Becker, “LDIF – Linked Data Integration Framework,” em Proceedings of the Second International Conference on Consuming Linked Data, ser. COLD’11, vol. 782. Aachen, Germany, Germany: CEUR-WS.org, 2011, pp. 125– 130.

[16] A. C. Junior, “A Jigsaw Puzzle Metaphor for Representing Linked Data Mappings,” Tese de Doutoramento, Trinity College Dublin.School of Computer Science & Statistics, 2019.

[17] H. Knublauch. (2011, Abril) SPINMap: SPARQL-based Ontology Mapping with a Graphical Notation. Acedido em 22 Fevereiro 2020. [Online]. Disponível em: https://www.topquadrant.com/spinmap-sparql-based-ontology-mapping-with-a-graphical-notation/.

[18] X. Xue e J. Chen, “A Compact co-Firefly Algorithm for Matching Ontologies,” em 2019 IEEE Symposium Series on Computational Intelligence (SSCI), 2019, pp. 2629–2632.

[19] G. Kück, “Tim Berners-Lee's Semantic Web,” South African Journal of Information Management, vol. 6, 12 2004.

[20] D. R. B. Cunha, D. Y. Souza, e B. F. Loscio, “Linked Data: da Web de Documentos para a Web de Dados,” Novembro 2011. [Online]. Disponível em: https://www.cin.ufpe.br/ daise/publications.html

[21] P. Maio. (2015, Junho) Vamos Ligar e Partilhar? – Introdução à Web Semântica. Acedido em 02 Fevereiro 2019. [Online]. Disponível em: https://pplware.sapo.pt/internet/ligar-partilhar-introducao-web-semantica/.

[22] T. Berners-Lee. (2006, Julho) Linked Data. Acedido em 02 Fevereiro 2019. [Online]. Disponível em: https://www.w3.org/DesignIssues/LinkedData.html.

[23] J. C. Pinheiro, “Processamento de consulta em um Framework baseado em mediador para integração de dados no padrão de Linked Data,” Tese de Doutoramento, Universidade Federal do Ceará, 2011.

[24] E. A. M. Morais e A. P. L. Ambrósio, “Ontologias: conceitos, usos, tipos, metodologias, ferramentas e linguagens,” Universidade Federal de Goias, Relatório Técnico INF ´ 001/07, dec2007.

[25] M. Rocha. (2016, Janeiro) Vamos Ligar e Partilhar? – Introdução às Ontologias. Acedido em 20 Fevereiro 2019. [Online]. Disponível em: https://pplware.sapo.pt/internet/ligar-partilhar-introducao-as-ontologias/.

[26] T. U. Consortium, The Unicode Standard - Version 13 - Core Specification. Mountain View, CA: Unicode Consortium, mar 2020, acedido em 09 Fevereiro 2019.

[27] T. A. S. Foundation. An Introduction to RDF and the Jena RDF API. Acedido em 20 Abril 2019. [Online]. Disponível em: https://jena.apache.org/tutorials/rdf_api.html.

[28] M. Mealling and R. Denenberg, “Report from the joint w3c/ietf URI planning interest group: Uniform resource identifiers (URIs), URLs, and uniform resource names (URNs): Clarifications and recommendations,” Relatório Técnico, 2002.

[29] S. Griffiths. (2017, Junho) Namespace registration for international standard book number. Acedido em 20 Abril 2019. [Online]. Disponível em: https://www.iana.org/assignments/urn-formal/isbn

[30] J. C. d. Lima e C. L. d. Carvalho, “Resource Description Framework (RDF),” Universidade Federal de Goias, Relatório Técnico RT-INF ´ 003 − 05, jun2005.

[31] D. Beckett e T. Berners-Lee. (2011, Março) Turtle - Terse RDF Triple Language. Acedido em 01 Maio 2019. [Online]. Disponível em: https://www.w3.org/TeamSubmission/turtle/.

[32] M. Rocha. (2015, Agosto) Vamos Ligar e Partilhar? – Introdução ao RDFS (Parte I). Acedido em 10 Fevereiro 2019. [Online]. Disponível em: https://pplware.sapo.pt/internet/ligar-partilhar-introducao-ao-rdfs-parte-i/.

[33] (2015, Setembro) Vamos Ligar e Partilhar? – Introdução ao RDFS (Parte II). Acedido em 10 Fevereiro 2019. [Online]. Disponível em: https://pplware.sapo.pt/internet/ligar-partilhar-introducao-ao-rdfs-parte-ii/

[34] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider e L. A. Stein. (2004, Fevereiro) OWL Web Ontology Language Reference. Acedido em 26 Agosto 2019. [Online]. Disponível em: https://www.w3.org/TR/owl-ref/

[35] D. L. McGuinness e F. v. Harmelen. (2004, Fevereiro) . OWL Web Ontology Language. Acedido em 18 Maio 2019. [Online]. Disponível em: https://www.w3.org/TR/owl-features/.

[36] L. Feigenbaum, SPARQL By Example - A tutorial, VP Technology & Standards, Cambridge Semantics, jun 2009. Acedido em 23 Marco 2019. [Online]. Disponível em: https://www.w3.org/2008/09/sparql-byexample/Overview.html

[37] J. A. Ferreira, "Wikis semânticos: da Web para a Web Semântica," Dissertação de Mestrado, Universidade Estadual Paulista (UNESP), 2014.

[38] E. Prud'hommeaux e A. Seaborne. (2008, Janeiro) SPARQL Query Language for RDF. Acedido em 02 Junho 2019. [Online]. Disponível em: https://www.w3.org/TR/rdf-sparql-query/

[39] M. Rocha. (2015, Outubro) Vamos Ligar e Partilhar? – Introdução ao SPARQL. Acedido em 02 Junho 2019. [Online]. Disponível em: https://pplware.sapo.pt/internet/ligar-partilhar-introducao-ao-sparql/

[40] M.-R. Koivunen e E. Miller, "W3C Semantic Web Activity," em Semantic Web Kick-Off in Finland: Vision, Technologies, Research, and Applications, Maio 2002, pp. 27–43.

[41] W. S. W. Group. (2013, Março) SPARQL 1.1 Overview. Acedido em 01 Setembro 2019. [Online]. Disponível em: https://www.w3.org/TR/sparql11-overview/

[42] A. Schultz e C. Bizer, The R2R Framework - Transforming RDF Datasets, University of Mannheim, School of Business Informatics and Mathematics, Julho 2011. Acedido em 16 Março 2019.

[43] N. Choi, I.-Y. Song e H. Han, "A Survey on Ontology Mapping," SIGMOD Record, vol. 35, pp. 34-41, Setembro 2006.

[44] Y. K. Hooi, M. F. Hassan e A. M. Shariff, "A Survey on Ontology Mapping Techniques," Lecture Notes in Electrical Engineering, vol. 279, Janeiro 2014.

[45] T. team. (2014, Setembro) Reshaping Relational Data using SPINMap. Acedido em 22 Fevereiro 2020. [Online]. Disponível em: https://www.topquadrant.com/spin/SPINMapRDBMS/.

[46] P. N. Mendes, H. Müleisen, V. Bryl e C. Bizer, “Sieve - Linked Data Quality Assessment and Fusion,” em Proceedings of the 2012 Joint EDBT/ICDT Workshops, ser. EDBT-ICDT ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 116–123.

[47] V. M. Pequeno, V. M. P. Vidal e T. Vinuto, “Towards Semi-Automatic Generation of R2R Mappings,” em ICEIS 2018: 20th International Conference on Enterprise Information Systems, Funchal, 2018.

[48] A. F. M. Silva, “Uma Abordagem para a Junção de Ontologias e sua utilização no Desenvolvimento de Ontologias de Aplicaçção,” Dissertação de Pós-Graduação, Universidade Federal do Maranhão, Brasil, 2014.

[49] M. Granitzer, V. Sabol, K. W. Onn, D. Lukose e K. Tochtermann, “Ontology Alignment—A Survey with Focus on Visually Supported Semi-Automatic Techniques,” Future Internet, vol. 2, Setembro 2010.

[50] S. Massmann, S. Raunich, D. Aumueller, P. Arnold, E. Rahm. "Evolution of the COMA Match System OM-2011", em The Sixth International Workshop on Ontology Matching, 2011

[51] K. N. Valcarel, “A utilização de padrões de projeto de ontologias na modelagem de um cenário real,” Projeto de Graduação, Universidade Federal do Estado do Rio de Janeiro, Brasil, 2014.

[52] M. J. V. Jorente, M. C. Padua e J. E. S. Segundo, “Criação de padrões na web semântica: perspectivas e desafios,” em Em Questão, vol. 23, Porto Alegre, 2017, pp. 157-178.

[53] J. Sequeda, F. Priyatna e B. Villazón-Terrazas, “Relational Database to RDF Mapping Patterns,” em Proceedings of the 3rd International Conference on Ontology Patterns, ser. WOP’12, vol. 929, Aachen, DEU: CEUR-WS.org, 2012, pp. 97-108.

[54] F. Scharffe, O. Zamazal e D. Fensel. Ontology alignment design patterns. Knowledge and Information Systems, vol. 40, Springer, 2014, pp. 1-28.

[55] L. Barbiere. (2017, Julho) O que é Bootstrap e para que serve? Acedido em 26 Julho 2019. [Online]. Disponível em: https://www.ciawebsites.com.br/dicas-e-tutoriais/o-que-e-bootstrap/

[56] Bootstrap 4 Get Started. Acedido em 26 Julho 2019. [Online]. Disponível em: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp

[57] S. Seshadri, B. Green. AngularJS Up & Running: Enhanced Productivity with Structured Web Apps. O'Reilly Media, 2014.

[58] J. Sampaio. (2015) O que é o AngularJS. Acedido em 22 de Julho 2019. [Online]. Disponível em: https://www.devmedia.com.br/implementando-servicos-com-angularjs/32715

[59] F. Gutierrez. Pro Spring Boot. New Mexico, USA: Apress, 2016

[60] Spring team. Building an Application with Spring Boot. Acedido em 02 Agosto 2019. [Online]. Disponível em: https://spring.io/guides/gs/spring-boot/

[61] Spring team. Spring Data JPA. Acedido em 04 Agosto 2019. [Online]. Disponível em: https://spring.io/projects/spring-data-jpa

[62] JavaTpoint. JPA Tutorial. Acedido em 04 Agosto 2019. [Online]. Disponível em: https://www.javatpoint.com/jpa-tutorial

[63] T. Janssen. What is Spring Data JPA? And why should you use it? Acedido em 04 Agosto 2019. [Online]. Disponível em: https://thoughts-on-java.org/what-is-spring-data-jpa-and-why-should-you-use-it/

[64] TutorialsPoint. H2 Database Tutorial. Acedido em 05 Agosto 2019. [Online]. Disponível em: https://www.tutorialspoint.com/h2_database/index.htm

[65] Apache Jena. Getting started with Apache Jena. Acedido em 01 Maio 2019. [Online]. Disponível em: https://jena.apache.org/getting_started/index.html

[66] A. H. D. N. Cordeiro. Apache Jena. Acedido em 02 Junho 2019. [Online]. Disponível em: https://www.cin.ufpe.br/~in1099/132/Apache%20Jena.pdf

[67] R. d. M. Fernandes, “Integração de Dados Baseada em Ontologia e Raciocínio Automático: Estudo de Caso com Dados Públicos de Saúde,” Dissertação de Mestrado, Universidade Federal de Pernambuco, Brasil, 2012.

[68] A. C. Junior, C. Debruyne e D. O'Sullivan, "An Editor that Uses a Block Metaphor for Representing Semantic Mappings in Linked Data," em *The Semantic Web: ESWC 2018 Satellite Events*, 2018.

9 Anexo 01 – Regras de Mapeamento

Mapeamento de Classe	**(R1)** bo:Author(s) ← dbo:Person(s)
	(R2) foaf:Agent(s) ← dbo:Agent(s)
	(R3) bo:Book(s) ← dbo:Book(s) ; dbo:literaryGenre(s,v) != 'Romance'
	(R4) dc:MediaType(u) ← dbo:Book(s) ; dbo:mediaType(s,v) ; generateUri[ψ_4](s,u)
Mapeamento de Propriedades de Tipos de Dados	**(R5)** bo:title(s,v) ← dbo:Book(s) ; dbo:originalTitle(s,v) ; fn:lower-case (v) as xs:string
	(R6) bo:isbn(s,v) ← dbo:Book(s) ; dbo:isbn(s,v)
	(R7) bo:publicationDate(s,v) ← dbo:Book(s) ; dbo:firstPublicationDate(s,v)
	(R8) dc:type(s,v) ← dbo:Book(s) ; dbo:literaryGenre (s,v)
	(R9) dc:language(s,v) ← dbo:Book(s) ; dbo:originalLanguage / dbo:languageCode(s,v)
	(R10) dc:format(u,z) ← dbo:Book(s) ; dbo:mediaType (s,z) ; generateUri[ψ_{10}](s,u)
	(R11) ex:careerDuration(s,v) ← dbo:Person(s); dbo:startCareer(s, v1); dbo:endCareer(s, v2); concat(v1, '-', v2, v)
	(R12) bo:name(s,v) ← dbo:Person(s) ; dbo:birthName(s,v)
Mapeamento de Propriedades de Objetos	**(R13)** foaf:made(s,v) ← dbo:Book(s) ; dbo:author(s,v)
	(R14) bo:publisher(s,v) ← dbo:Book(s) ; dbo:publisher(s,v)
	(R15) dc:hasFormat(s,u) ← dbo:Book(s) ; dbo:mediaType(s,v) generateUri[ψ_{15}](s,u)