FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**JONATHAN SCHNEIDER**

# CRIAÇÃO E CONSUMO DE DADOS NOS PADRÕES DA WEB SEMÂNTICA, UTILIZANDO RDF E JENA

MARÍLIA
2011

JONATHAN SCHNEIDER

# CRIAÇÃO E CONSUMO DE DADOS NOS PADRÕES DA WEB SEMÂNTICA, UTILIZANDO RDF E JENA

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Me. RODOLFO BARROS CHIARAMONTE

MARÍLIA
2011

Schneider, Jonathan
Criação e consumo de dados nos padrões da Web Semântica, utilizando RDF e Jena / Jonathan Schneider; orientador: Rodolfo Barros Chiaramonte. Marília, SP:[s.n], 2011 109 f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2011.

1. Web Semântica 2. Jena 3. RDF

CDD:004.6782

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
*BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO*

**TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL**

Jonathan Schneider

CRIAÇÃO E CONSUMO DE DADOS NOS PADRÕES DA WEB SEMÂNTICA, UTILIZANDO RDF E JENA

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 10.0 (dez)

Orientador: Rodolfo Barros Chiaramonte

1º. Examinador: Fabio Lucio Meira

2º. Examinador: Leonardo Castro Botega

Marília, 29 de novembro de 2011.

# AGRADECIMENTOS

*Em primeiro lugar, agradeço aos meus docentes, sendo eles:*

*Profa. Juliana, por me ensinar que o limite de uma constante é a própria constante;*

*Prof. Rodelo, por sempre ouvir minhas dúvidas, sugestões e lamentações;*

*Prof. Santarém, por me mostrar com quantos vértices e arestas se faz um K5;*

*Prof. Dacencio, por me ensinar Assembler (a coisa mais "legal" que aprendi na faculdade);*

*Prof. Elvis, por me fazer gostar ainda mais do Java;*

*Prof. Rodolfo, por me ensinar Prolog (a segunda coisa mais "legal" que aprendi na faculdade) e pela dedicação como orientador neste trabalho;*

*Prof. Petruzza, por me mostrar qual a finalidade da até então misteriosa máscara de sub-rede;*

*Prof. Galante, por me deixar ir tomar coca no shopping;*

*Prof. Celso, por me ensinar que o desvio padrão é a raiz quadrada da variância;*

*Prof. Elton, por me mostrar que não é tão fácil ficar rico;*

*Prof. Botega, por me ensinar os conhecimentos necessários para desenvolver o Marília Street Hot;*

*Prof. Marcelo, por me ensinar que posso baixar arquivos da internet sem ser preso;*

*Prof. Paulo, por me ensinar os segredos do Windows Server;*

*Profa. Renata, por me ensinar que "pressione" se escreve com "SS" e não "C";*

*Prof. Mauricio, por me ensinar de forma fantástica como usar ponteiros no C;*

*Prof. Meira, por me ensinar 100 formas diferentes de fazer o caso de uso PokaRopa;*

*Prof. Bugatti, por me ensinar, além de várias teorias, o verdadeiro "espírito" da Ciência da Computação;*

*Em segundo lugar, agradeço aos demais funcionários do Univem, afinal, cada um deles teve uma parcela de participação na formação que recebi nesta instituição.*

*Agradeço aos companheiros de turma, em especial os amigos: Martins, Negrisoli e Piazentin pelas críticas e sugestões realizadas neste trabalho.*

*Agradeço também, os amigos:*

*Sanvido, pela ajuda técnica;*

*Petranski, pelos ensinamentos de CD, utilizados nas varias ilustrações deste trabalho;*

*E não poderia deixar de agradecer a pequena Bia e a grande Teca, por compreenderem a minha ausência em várias ocasiões no decorrer deste trabalho.*

SCHNEIDER, Jonathan. **Criação e consumo de dados nos padrões da Web Semântica, utilizando RDF e Jena.** 2011 109 f. Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2011.


RESUMO

A Web Semântica tem como objetivo ser uma nova forma de organizar o conteúdo da Web, tornando este entendível não só por humanos como acontece hoje, mas também por agentes computacionais. O foco deste trabalho é o estudo dos principais padrões e tecnologias da Web Semântica, onde primeiramente são abordados os frameworks RDF (*Resource Description Framework)* e RDF-Schema (*Resource Description Framework Schema*) que tem por finalidade a criação de conteúdo semântico; passando em seguida para a linguagem de consulta para documentos RDF "SPARQL" (*Protocol and RDF Query Language)*; e na sequencia é abordado o Jena, framework Java para manipulação de documentos RDF. No final do trabalho é apresentada a implementação de um APC (Assistente Pessoal de Compras) que é um agente nos padrões da Web Semântica que tem por objetivo ajudar um usuário humano a escolher um produto, onde a fonte de dados consultada são bases RDF provenientes de vários fornecedores.



**Palavras-chave:** Web Semântica, Jena, RDF

SCHNEIDER, Jonathan. **Criação e consumo de dados nos padrões da Web Semântica, utilizando RDF e Jena.** 2011 109 f. Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2011.


ABSTRACT

The Semantic Web aims to be a new way to organize Web content, making this not only understandable by humans as it does today, but also by computational agents. The focus of this work is the study of the major standards and technologies of Semantic Web, which are first addressed the frameworks RDF (Resource Description Framework) and RDF Schema (Resource Description Framework Schema), which aims to create semantic content, then moving on to the query language for RDF document "SPARQL" (Protocol and RDF Query Language) and in the sequel we shall address the Jena Java framework for handling RDF documents. At the end of the paper presents the implementation of a PCA (Personal Shopping Assistant) is an agent on the Semantic Web standards that aims to help a human user to choose a product where the source data bases are accessed from RDF multiple vendors.



**Keywords:** Semantic Web, Jena, RDF

# LISTA DE ILUSTRAÇÕES

# LISTA DE ABREVIATURAS E SIGLAS

**Ajax:** *Asynchronous Javascript and XML*

**API:** *Application Programming Interface*

**BSD:** *Berkeley Source Distribution*

**DBA:** *Database administrator*

**DL:** Lógica de Descrição

**FIA:** *Fédération Internationale de l'Automobile*

**FOAF:** *Friend of a Friend*

**HTML:** *Hypertext Markup Language*

**OO:** Orientação a Objetos

**OWL:** *Web Ontology Language*

**RDF :** *Resource Description Framework*

**RDF-Schema:** *Resource Description Framework Schema*

**SPARQL:** *Protocol and RDF Query Language*

**SQL:** *Structured Query Language*

**SWRL:** *Semantic Web Rule Language*

**URI:** *Uniform Resource Identifier*

**W3C:** *World Wide Web Consortium*

**Web:** *World Wide Web*

**WS:** Web Semântica

**WSDL:** *Web Services Description Language*

**XML:** *Extensible Markup Language*

# SUMÁRIO

# INTRODUÇÃO

Este trabalho apresenta o que é, e como funciona a Web Semântica, esta idealizada por Tim Bernes Lee, Ora Lassila e James Hendler, no artigo intitulado "*The Semantic Web*" publicado na revista *Scientific American*, no mês de Maio de 2001, onde os autores fizeram a proposta de criar uma nova forma de organizar o conteúdo da Web, tornando este entendível não só por humanos como acontece hoje, mas também por agentes computacionais.

Apesar da proposta inicial já ter completado 10 anos, nos dias atuais ainda são poucas as aplicações Web que utilizam este padrão. Ao longo destes anos, já foram desenvolvidos e definidos vários dos padrões necessários, e existem diversas ferramentas para auxiliar no desenvolvimento desta nova forma de Web, o que falta é o envolvimento de profissionais da Web com os padrões, para que se possa gradualmente colocá-los em pratica.

Este trabalho tem como principal objetivo servir como introdução para as tecnologias da Web Semântica, onde são apresentados vários exemplos práticos visando sempre destacar os benefícios que trazem quando aplicados.

O seu conteúdo está dividido em duas grandes partes, onde nos dois primeiros capítulos são abordados o funcionamento e as tecnologias da Web Semântica, com foco nos frameworks RDF (*Resource Description Framework*) e RDF-Schema (*Resource Description Framework Schema*), que servem basicamente para a criação de conteúdo semântico.

Na segunda parte, nos capítulos três e quatro o foco são nas tecnologias que vão "consumir" o conteúdo semântico, e são abordados o SPARQL (*Protocol and RDF Query Language*) e o framework Jena respectivamente.

E no último capítulo é apresentada a implementação de uma aplicação na linguagem Java utilizando o framework Jena. Esta aplicação é um agente semântico, mais especificamente um APC (Assistente Pessoal de Compras) que tem por objetivo auxiliar um usuário humano a procurar por informações sobre produtos, buscar avaliações de outros usuários sobre o produto e pesquisar preços em sites de comércio eletrônico, onde a base de pesquisa é composta por documentos semânticos (RDF) provenientes de várias fontes.

# CAPÍTULO 1 - INTERNET E A WORLD WIDE WEB

Neste primeiro capítulo são abordados os conceitos de Web e Web Semântica apresentando suas definições e mudanças através dos anos. Também são apresentados os padrões e tecnologias da Web Semântica.

## 1.1 Web

Segundo Stout (1997), a Internet em seus primórdios era apenas um modo para que computadores pudessem transferir dados através de uma rede, de forma robusta e confiável, ou seja, a Internet é a estrutura física da grande rede mundial de computadores em conjunto com seus protocolos de comunicação. No início não existia uma aplicação padrão para se navegar por todo conteúdo que a Internet dispunha a seus usuários, mas em novembro de 1990 com o objetivo de facilitar o uso da Internet, Tim Berners-Lee, publicou a proposta *WorldWideWeb: Proposal for a HyperText Project* , que propôs uma estrutura composta por um servidor, um navegador e documentos *hypertext* contendo *hyperlinks*, assim criando conectividade entre documentos na internet, como definido por Wall (1997)

> A Web é um sistema de hipertexto, ou seja, não é organizada de forma linear como um livro tradicional ou outro tipo comum de apresentação. A Web não tem 'princípio' nem 'fim'. Trata-se de uma nuvem de dados através da qual você pode encontrar seu caminho da forma que melhor desejar.

Apesar de muitos especialistas e pesquisadores da área não concordarem com o emprego de versões na Web, esta é uma maneira informal de dividir as suas principais evoluções, seja na forma como ela vem sendo utilizada ou pelo emprego de novas tecnologias, assim esta primeira Web baseada em documentos e *hyperlinks* intitula-se Web 1.0.

Graças ao advento da Web, a Internet passou a ser utilizada por uma grande porcentagem da população mundial. No início a grande maioria dos usuários da Web era formada apenas por consumidores de conteúdo, ficando a tarefa de criar conteúdo a cargo de empresas dos mais variados setores, e alguns poucos usuários com nível técnico mais avançado.

Com o passar dos anos e o crescente número de usuários, surgiu naturalmente o desejo destes em colaborar na produção de conteúdo, este desejo foi suprido com a criação de ferramentas como o Blog, que ofereceu uma forma fácil para o usuário padrão da Web (sem conhecimento técnico) publicar seu diário, ideias ou qualquer tipo de conteúdo na Web.

Várias outras ferramentas foram criadas com o mesmo propósito, pode-se citar: wikis, fóruns de discussão, álbuns de fotos, redes sociais entre outros. Assim nascia uma nova Web, intitulada como Web 2.0, que foi definida por O'Reilly (2006), (tradução nossa)

> Web 2.0 é a revolução de negócios na indústria de informática causada pela mudança para a internet como plataforma, e uma tentativa de entender as regras para o sucesso nessa nova plataforma. O principal entre essas regras é o seguinte: se tornarem melhores quanto mais são usados pelas pessoas (Isto é o que eu tenho chamado de "o aproveitamento da inteligência coletiva.)

Apesar de existirem documentos informais incluindo tecnologias como o Ajax (*Asynchronous Javascript and XML*) como sendo ícone da Web 2.0, nenhuma nova tecnologia realmente foi criada, o que realmente e informalmente define a Web 2.0, são os conteúdos colaborativos, onde o usuário comum tem o poder de criar e opinar.

## 1.2 Web 3.0 - Web Semântica

Do termo Web Semântica, já se definiu o que é "Web" na seção anterior, e para definir "semântica" é utilizado o exemplo abaixo:

- Tweety é um pássaro.
- Existe um pássaro chamado Tweety.

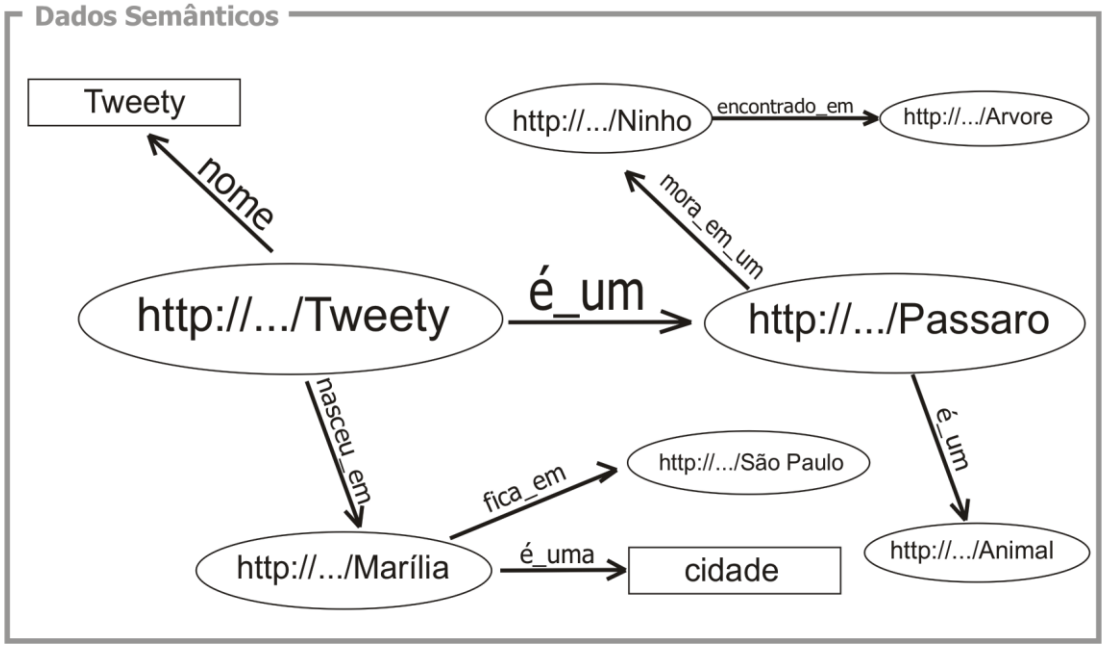
Cada uma das sentenças é um conjunto de dados, que unidos formam uma informação, apesar das sentenças serem diferentes, ambas expressam a mesma ideia, assim pode-se afirmar que ambas são semanticamente equivalentes, apesar de serem sintaticamente diferentes.

Pessoas com a capacidade de ler e interpretar o português, concluem que "Tweety" trata-se de um nome próprio, e "pássaro" é um animal, assim adquirem o conhecimento de que existe um pássaro chamado Tweety, e mesmo que não esteja explícito na sentença, mas baseado em conhecimentos anteriores sobre pássaros, pode-se afirmar também que Tweety provavelmente pode voar, que ele pode ter um ninho, penas, e com certeza nasceu de um ovo colocado por sua mãe.

Recriar esta habilidade que as pessoas possuem de inferir conhecimentos através de sentenças em agentes computacionais, é o principal objetivo da Web Semântica, ou seja, criar padrões para que todo o conteúdo da Web possa ser entendido não somente por humanos, mas também por agentes, como foi proposto por Berners-Lee (2001).

Para ficar mais clara a proposta da WS (Web Semântica) e seu funcionamento, é utilizada a figura 1, que ilustra de forma simplificada um grafo com dados semânticos.

Figura 1 - Grafo com Dados Semânticos




Fonte: Próprio Autor

Supondo-se que exista uma página Web que esteja de acordo com os padrões da WS, e nesta tenha a frase “Tweety é um pássaro”, e o nome próprio Tweety seja referenciado com o URI (*Uniform Resource Identifier*) “http://.../Tweety”, isso significa que existe um recurso único na Web referenciado por este URI. Supondo-se também que um usuário esteja navegando nesta página através de um “agente semântico”, e deseja obter mais informações sobre Tweety, e o seu agente tenha habilidades para entender conteúdos semânticos, mesmo que para o usuário só apareça à frase “Tweety é um pássaro”, o agente de forma oculta para o usuário cria um grafo como o da figura 1, e começa a percorrê-lo trazendo informações para o usuário, como exemplificado na conversa abaixo entre o usuário e o agente:

Usuário: Quem é Tweety?

Agente: é um pássaro, nascido em Marília.

Usuário: Onde fica Marília?

Agente:  Marília é uma cidade do estado de São Paulo.

Usuário: Tweety mora em uma árvore?

Agente: Tweety mora em um ninho, que pode ser encontrado em árvores.

Apesar de este diálogo estar “maquiado” com emprego de processamento de linguagem natural, serve como exemplo básico da proposta da WS.

Como pode ser observado o agente percorre o grafo em busca de informações, muitas veze a informação esta a muitos nós de distância, como na pergunta “Tweety mora em

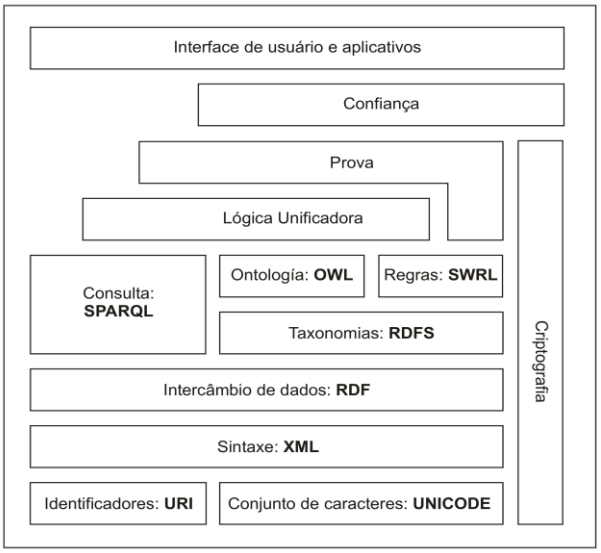
uma árvore?" o agente fez o seguinte caminho até encontrar a resposta "Tweety→Passaro→Ninho→Arvore".

O estudo dos padrões recomendados pelo W3C(*World Wide Web Consortium*), para a produção e consumo de um conteúdo semântico, como este que foi exemplificado é o foco deste trabalho.

## 1.3 Principais Tecnologias e Padrões da Web Semântica

Nesta seção são abordadas as principais tecnologias e padrões da WS, dando uma breve descrição de cada uma delas. Na figura 2 é apresentada a estrutura da Web Semântica, esta dividida em camadas, onde cada camada faz o uso de uma tecnologia e tem uma função específica.

Figura 2 - Estrutura de camadas da Web Semântica




Fonte: Hebeler(2009),(tradução nossa)

- **Identificadores – URI:** (*Uniform Resource Identifier*) , Segundo Hebeler (2009) "é simplesmente uma maneira padronizada de nomear recursos". Exemplificando, "`http://www.univem.edu.br/cursos#bcc`" e "`http://www.w3.org/2000/01/rdf-schema`" são URIs, geralmente um URI esta ligado a algum tipo de arquivo publicado na Web (página, foto, som...), mas seu uso na Web Semântica é apenas identificar recursos;
- **Conjunto de caracteres – Unicode:** Como encontrado em Unicode.org (2011), Unicode é um padrão internacional para representação de caracteres, fornecendo um número único para cada caractere, este foi o padrão adotado

pela WS a fim de garantir a internacionalização das informações, pois seu uso permite a representação de forma consistente de caracteres de qualquer sistema de escrita ou plataforma de software existente;

- **Sintaxe – XML:** (*Extensible Markup Language*) baseado em Ray (2001) pode ser definida, como um protocolo para conter e gerenciar informações ou até mesmo servir como base para tecnologias que podem ir mais além como formatar documentos e até filtrar dados, o XML serviu como base para outras linguagens da WS;
- **Intercâmbio de Dados – RDF:** (*Resource Description Framework*) foi definido como padrão para representar a informação (metadados) sobre recursos através de afirmações em triplas (recurso, propriedade e valor), sendo assim denominado como a subcamada semântica da arquitetura, fornece a capacidade para descrever recursos, usufruindo de uma sintaxe baseada em XML;
- **Taxonomias – RDFS:** (*Resource Description Framework Schema*) framework que fornece meios para a criação de vocabulários RDF, onde é possível descrever novas classes e propriedades e os relacionamentos entre elas;
- **Consulta – SPARQL:** (*SPARQL Protocol and RDF Query Language*) protocolo e linguagem para consultas em modelos RDF, possui sintaxe muito similar à linguagem SQL;
- **Ontologias – OWL:** (*Web Ontology Language*) como descrito por Pollock (2010) é uma extensão do RDF, onde é possível a inclusão de mais termos de vocabulário para descrever classes, fatos sobres estas classes, relacionamento entre classes e características destas relações através de axiomas lógicos. A OWL baseia-se em linguagens de representação do conhecimento, também conhecidas como DL (Lógica de Descrição), estas fornecem uma estrutura para descrição de conceitos e semânticas baseadas em lógica, para um domínio especifico;
- **Regras – SWRL:** *(Semantic Web Rules Language)* regras que visam inferir novos conhecimentos em modelos RDF/OWL;

Dando início aos conceitos sobre criação de conteúdo semântico, no próximo capítulo são abordados o RDF e RDF-Schema.

# CAPÍTULO 2 - RDF – RESOURCE DESCRIPTION FRAMEWORK

O primeiro passo para tornar a Web entendível por maquinas, é organizar e armazenar o seu conteúdo de forma semântica, neste capítulo são abordados o RDF e RDF-SCHEMA que são os padrão do W3C para esta finalidade.

## 2.1 RDF – Conceitos Básicos

RDF (*Resource Description Framework*), como explicado por Pollock (2010) é a especificação padrão para modelagem e intercambio de dados na Web Semântica. E uma recomendação aprovada pelo W3C, sendo sua última revisão publicada em fevereiro de 2004.

O seu papel dentro da Web Semântica é descrever um recurso, atribuindo a este, propriedades com seus respectivos valores, esta atribuição é realizada por uma instrução em forma de tripla (recurso, propriedade e valor). O RDF é indicado para situações em que a informação precisa ser processada por máquinas e não somente apresentada, pois fornece uma interface que permite que informações sejam trocadas entre aplicações sem perda de consistência. Pelo fato de utilizar URI para identificar recursos e vocabulários, oferece um meio para a eliminação de ambiguidades, mesmo em informações provenientes de várias fontes diferentes, esta é uma das características que o tornaram como padrão para representação e intercâmbio de informação na Web Semântica.

## 2.2 Modelo RDF

Como mencionado anteriormente, o RDF tem a função de descrever um recurso atribuindo a este propriedades, esta atribuição é realizada por uma afirmação em forma de tripla, contendo recurso, propriedade e valor.

O recurso (sujeito) é aquilo que a instrução descreve, pode ser um objeto físico (Ayrton Senna, Petróleo Brasileiro S.A, Volkswagem Fusca) ou abstrato (Romantismo, Copa do Mundo FIFA). Pode-se afirmar que recurso é tudo aquilo que tenha propriedades e possa ter um URI, ou seja, tudo aquilo que possa ser catalogado em uma enciclopédia pode ser considerado um recurso.

A propriedade (predicado) indica uma informação sobre um recurso, por exemplo o recurso Ayrton Senna poderia ter as propriedades: nome, nacionalidade, dataNascimento e altura.

O valor (objeto) é o valor atribuído a uma propriedade, o valor pode ser um literal, ou pode ser também outro recurso, um exemplo de valor para a propriedade "nacionalidade" seria o literal "brasileiro", para a propriedade "homepage" seria o recurso "http://www.senna.com.br".

Para exemplificar o modelo RDF, será realizada a identificação do recurso, propriedade e valor da frase "Ayrton Senna nasceu em 21 de março de 1960".

O primeiro passo é identificar o recurso da frase, que neste caso é Ayrton Senna, em seguida deve ser verificado qual é o seu URI, que no exemplo será definido como "http://f1drivers.com/Senna".

O próximo passo é encontrar um vocabulário que contenha uma propriedade equivalente a "data de nascimento". Vocabulários RDF-Schema serão apresentados na seção 2.4, mas trata-se de um conjunto de termos (classes e propriedades) utilizadas nas declarações RDF, neste exemplo será utilizado o vocabulário FOAF (*Friend of a Friend*) que possui a propriedade *birthday*(data de nascimento) que atende a esta necessidade. Assim a declaração RDF seria:

- **Recurso:** `http://f1drivers.com/Senna`
- **Propriedade:** `foaf:birthday`
- **Valor:** `"21/03/1960"`

## 2.3 Sintaxe RDF

Nas subseções seguintes é apresentada a sintaxe RDF em conjunto com as principais propriedades do vocabulário básico RDF.

### 2.3.1 Grafos RDF

Segundo Manola (2004) o modelo conceitual RDF é um grafo, e grafos RDF são utilizados para apresentar declarações RDF para leitores humanos pela sua facilidade de interpretação, a seguir é apresentada a sintaxe para este tipo de representação:

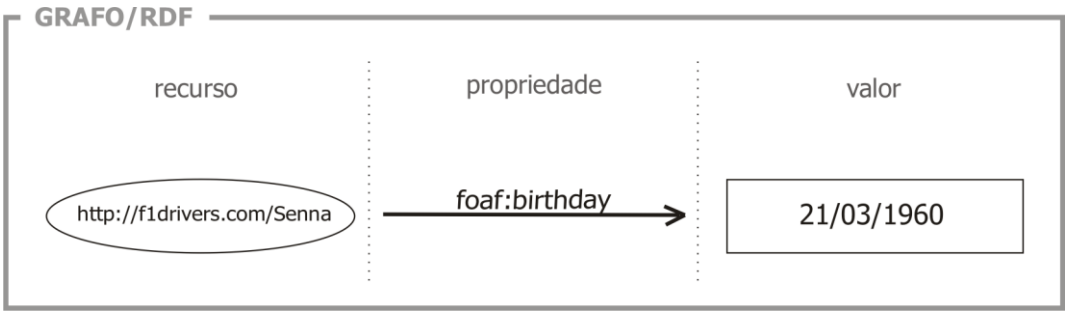
**Recursos:** São representados por ovais rotulados com o seu respectivo URI.

**Propriedades:** São representadas por arestas rotuladas.

**Valor:** Quando literais, são representadas por retângulos rotulados com o valor em questão, mas quando o valor é um recurso, é utilizado um oval.

A figura 3 ilustra a frase "Ayrton Senna nasceu em 21 de março de 1960", em forma de grafo RDF, este ilustra um caso onde o valor da propriedade é um literal.

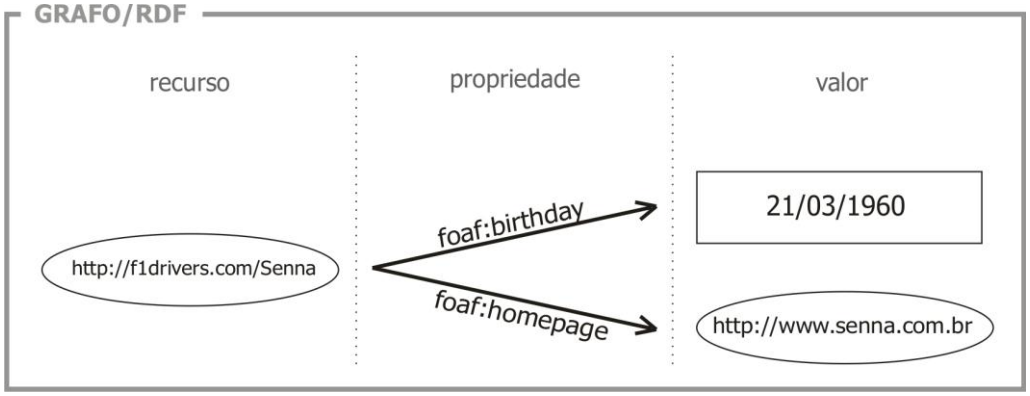
Figura 3 - Exemplo de Grado RDF




Fonte: Próprio Autor

Na figura 4 é ilustrado o grafo RDF da frase "Ayrton Senna nasceu em 21 de março de 1960 e sua homepage é www.senna.com.br.", este grafo difere do da figura 3 por dois aspectos, primeiro é declarado um recurso como valor da propriedade `foaf:homepage`, a segunda diferença é que neste são feitas duas declarações, ou seja são atribuídas duas propriedades: `foaf:birthday` e `foaf:homepage` para o mesmo recurso.

Figura 4 - Exemplo Grafo RDF 2




Fonte: Próprio Autor

Por apresentar grande facilidade de leitura, a maioria dos exemplos de declarações RDF neste trabalho, estarão sempre acompanhados do seu respectivo grafo, e de uma frase em linguagem natural da afirmação.

### 2.3.2 XML/RDF

Existem vários formatos para documentos RDF, os mais populares além de grafos são: N3, Turtle, N-Triplas e XML/RDF. Este último apesar de ser o mais complexo, será o utilizado no restante deste trabalho por ser o mais utilizado na bibliografia sobre o assunto.

A figura 5 ilustra a declaração "Ayrton Senna nasceu em 21 de março de 1960" no formato XML/RDF, (esta é a mesma declaração ilustrada em forma de grafo na figura 3).

Figura 5 - Exemplo da sintaxe XML/RDF

RDF/XML

```
1  <?xml version="1.0"?>
2
3  <rdf:RDF
4    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
5    xmlns:foaf="http://xmlns.com/foaf/0.1/#">
6
7    <rdf:Description rdf:about="http://f1drivers.com/Senna">
8        <foaf:birthday>1960/03/21</foaf:birthday>
9    </rdf:Description>
10
11 </rdf:RDF>
```

Fonte: Próprio Autor

A fim de tornar mais fácil o entendimento da sintaxe XML/RDF abaixo segue o significado de cada linha deste documento:

- Linha 1: é a declaração XML, indica que o conteúdo a seguir é XML na sua versão 1.0, (declaração que é padrão e idêntica para todos os documentos, e será omitida nos exemplos seguintes);
- Linha 3: é aberta a tag `<rdf:RDF` que indica que o XML seguinte até a sua tag de fechamento na linha 11 é RDF;
- Linhas 4 e 5: são declarados o `xmlns` (*XML Name Space*) para o prefixo `rdf` e `foaf` isso significa que no documento será utilizado o vocabulário "`http://www.w3.org/1999/02/22-rdf-syntax-ns`" e o seu prefixo neste documento será `rdf`, e também o vocabulário `http://xmlns.com/foaf/0.1/` com o prefixo `foaf`;
- Linha 7: é aberta a tag `<rdf:Description` que indica que se deseja fazer afirmações sobre um recurso já existente, e o recurso em questão é indicado por `rdf:about="URI_do_recurso"` que recebe o URI do recurso, desta forma todas as afirmações que vierem até a sua tag de fechamento `</rdf:Description>` na linha 9 são referentes ao recurso indicado;
- Linha 8: indica que o recurso tem a propriedade `foaf:birthday` e esta tem o valor literal "1960/03/21", vale destacar que a propriedade `birthday` faz parte do vocabulário `foaf` declarado na linha 5;

Uma aplicação com a capacidade para interpretação de documentos XML/RDF chegaria na seguinte conclusão sobre este documento:

- Vocabulários utilizados:
  - `rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns`
  - `foaf=http://xmlns.com/foaf/0.1/`

- Recursos:
  - *http://f1drivers.com/Senna*
    - *propriedades:*
      - *foaf:birthday*
        - *valores:*
          - *1960/03/21*

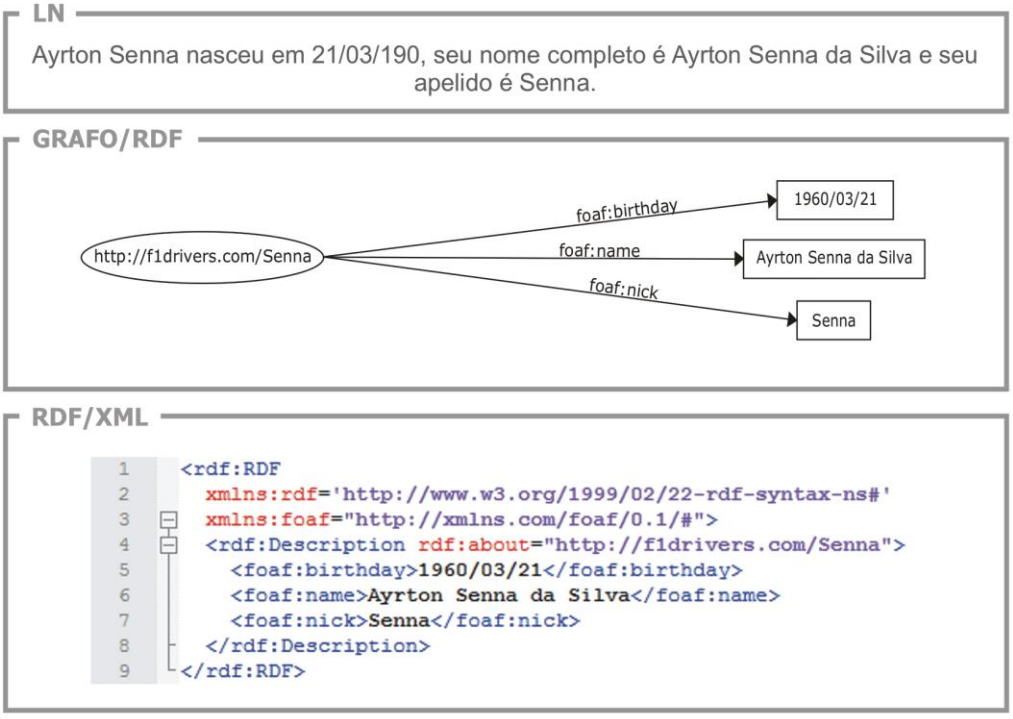
Como pode ser observado no exemplo anterior e afirmado por Breitman(2005) RDF também pode ser definido como uma lista de recursos com suas respectivas propriedades e valores.

### 2.3.3 Afirmações Compostas

O exemplo da figura 6 ilustra uma afirmação atribuindo três propriedades para um único recurso (*http://f1drivers.com/Senna*), na linha 4 é aberta a tag *<rdf:Description rdf:about="http://f1drivers.com/Senna">* que informa que todas as declarações de propriedades(linhas 5,6,7) até a sua tag de fechamento na linha 8 refere-se a este recurso.

A sintaxe da figura 6 é utilizada para declarações compostas por mais de uma propriedade para um mesmo recurso. Outra forma de fazer a mesma declaração, mas com uma sintaxe ligeiramente diferente é ilustrada na figura 7, onde é aberta uma tag *<rdf:Description* para cada uma das propriedades.

Figura 6 - Exemplo de Afirmação Composta




Fonte: Próprio Autor

Figura 7 - Exemplo de Afirmação Composta 2

RDF/XML

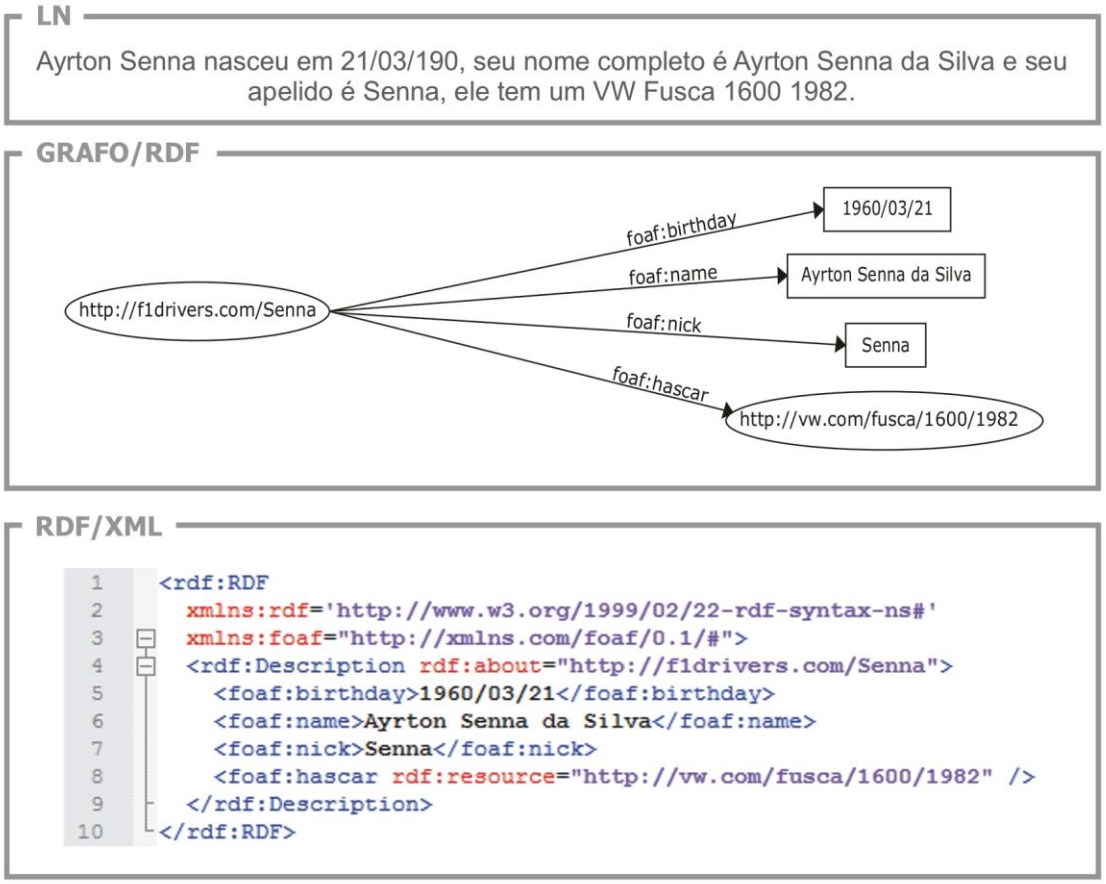
```
1  <rdf:RDF
2    xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
3    xmlns:foaf="http://xmlns.com/foaf/0.1/#">
4    <rdf:Description rdf:about="http://f1drivers.com/Senna">
5      <foaf:birthday>1960/03/21</foaf:birthday>
6    </rdf:Description>
7
8    <rdf:Description rdf:about="http://f1drivers.com/Senna">
9      <foaf:name>Ayrton Senna da Silva</foaf:name>
10   </rdf:Description>
11
12   <rdf:Description rdf:about="http://f1drivers.com/Senna">
13     <foaf:nick>Senna</foaf:nick>
14   </rdf:Description>
15 </rdf:RDF>
```

Fonte: Próprio Autor

### 2.3.4 Recurso Como Valor de Uma Propriedade

Como mencionado anteriormente as propriedades podem receber valores literais ou recursos. Para exemplificar a sintaxe para este tipo de afirmação será declarado que Ayrton Senna possui um carro modelo VW Fusca 1600 ano 1982. Esta declaração poderia ser feita com uma propriedade de valor literal do tipo String, por exemplo *foaf:hascar="VW Fusca 1600 ano 1982"* , mas neste caso este valor só teria significado para leitores humanos, um agente da web semântica entenderia isso apenas como um conjunto de caracteres e nada mais, ou seja, nenhuma informação adicional poderia ser inferida sobre o veículo. Assim para dar mais expressividade (semântica) ao documento a ser criado, será utilizada uma base RDF fictícia que a montadora VW criou com todos os veículos já fabricados por ela, inclusive o VW Fusca 1600 ano 1982, e este possui o URI *http://vw.com/fusca/1600/1982* . Como mostra a figura 8, na linha 8 a propriedade *foaf:hascar* não tem um valor literal, mas a declaração *rdf:resource="http://vw.com/fusca/1600/1982"* que indica que o valor desta propriedade é o recurso identificado pelo URI *http://vw.com/fusca/1600/1982*.

Figura 8 - Exemplo de Recurso Como Valor de Uma Propriedade




Fonte: Próprio Autor

### 2.3.5 Valores Literais Tipados

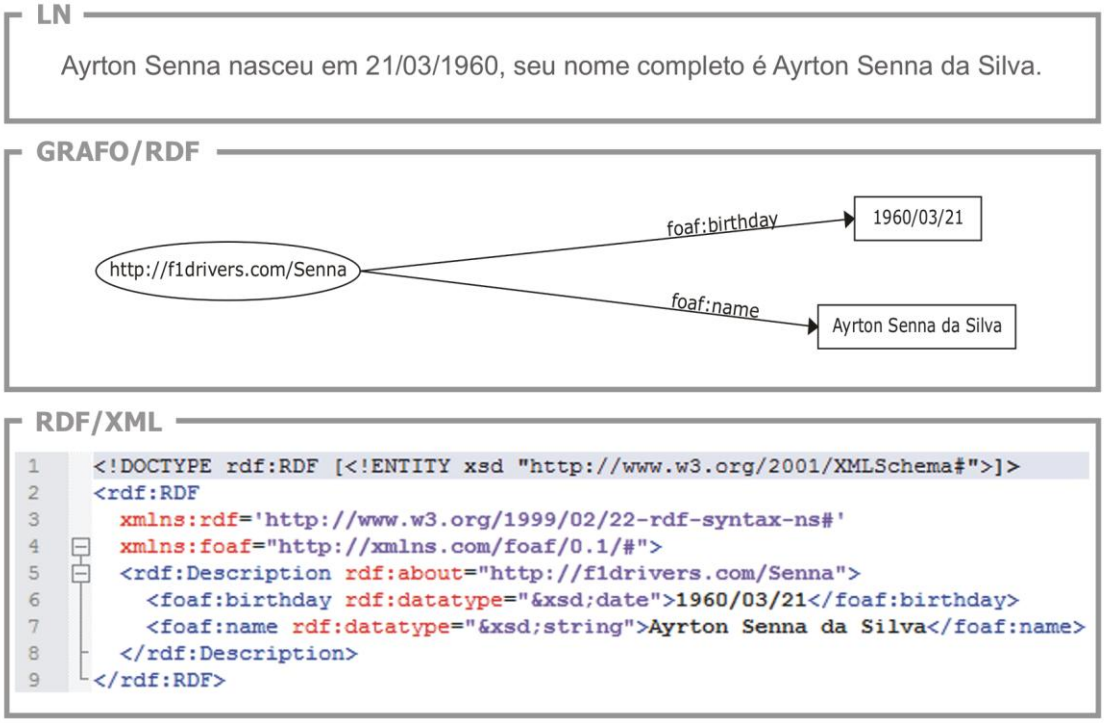
O modelo RDF possui uma sintaxe que atribui mais significado às propriedades literais através da declaração dos tipos destes valores. Vale ressaltar que esta prática é opcional e não obrigatória. Os tipos básicos para valores literais são do XML-Schema. Como apontado por Manola (2004) esta pratica é altamente recomendável sempre que for possível, pois aumenta a legibilidade do documento, facilitando a sua leitura por outros aplicativos, uma vez que documentos RDF são muitas vezes convertidos em objetos de uma linguagem de programação, como Java ou PHP, e a atribuição de tipos facilita esta conversão.

A figura 9 ilustra a sintaxe para a declaração de tipos literais, iniciando na linha 1 com a tag: `<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>` na linha 6 é aberta a tag `<foaf:birthday` que indica a declaração desta propriedade e nesta mesma tag tem a declaração `rdf:datatype="&xsd;date"` , onde `rdf:datatype` indica que a propriedade tem um tipo e

`&xsd;` é o prefixo para o vocabulário `http://www.w3.org/2001/XMLSchema#`, e `date` é o tipo da propriedade.

Existem vários tipos de dados no XML-Schema, os principais são aqueles também encontrados na maioria das linguagens de programação como: *String, boolean, decimal, float e double*; para maiores esclarecimentos sobre os tipos XML-Schema recomenda-se a leitura de Biron (2004) .

Figura 9 - Exemplo de Valores Literais Tipados




Fonte: Próprio Autor

### 2.3.6 Blank Nodes

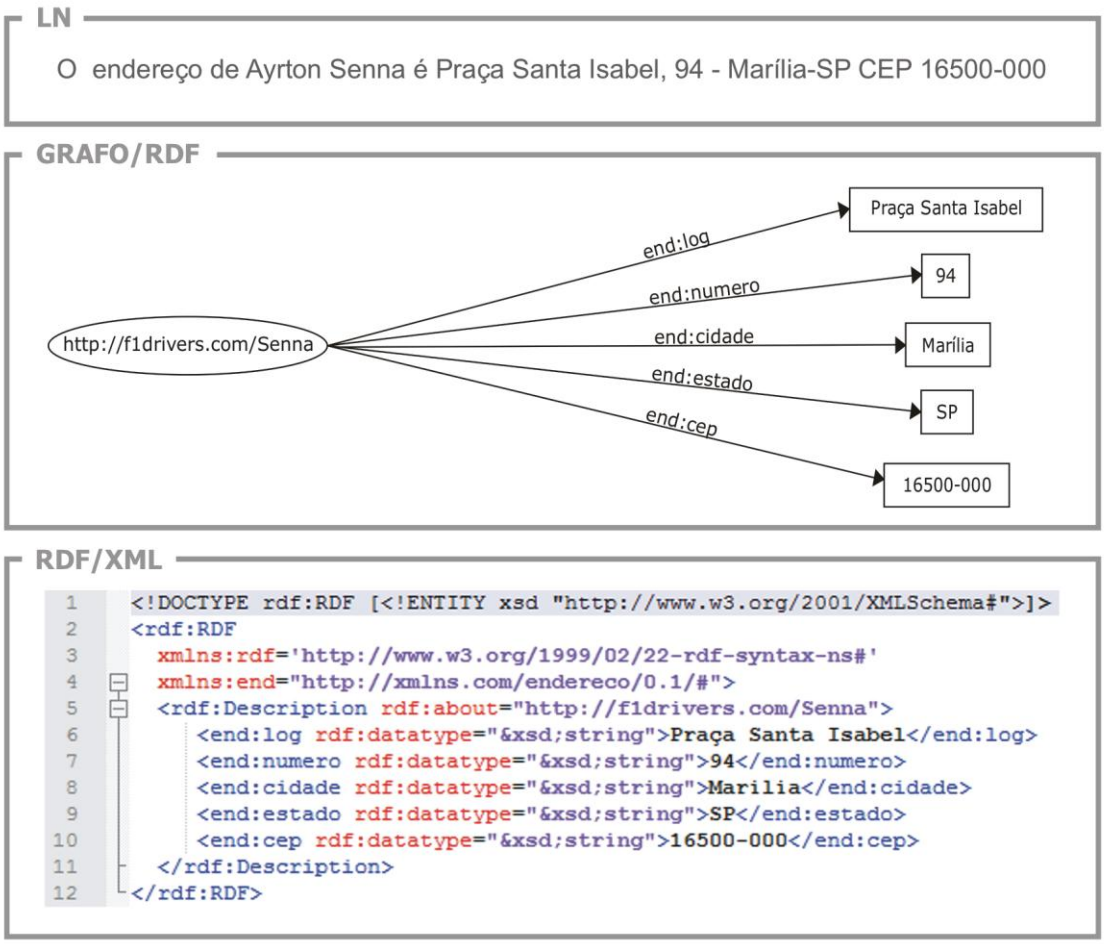
Em certos casos na criação de modelos RDF, o simples uso de triplas nem sempre é a melhor escolha, por exemplo, como exemplificado por Manola (2004), para descrever o endereço de uma pessoa, onde endereço corresponde ao conjunto de dados (logradouro, número, cidade, estado e CEP).

Com base nos exemplos anteriores pode-se presumir duas soluções:

A primeira seria adicionar uma propriedade “endereço” com um valor literal String com todos os itens em sequência “Pça Santa Isabel,94 – Marília-SP -16500-000”, esta solução tornaria estes dados legíveis para um humano, mas não para um agente da WS;

Uma segunda solução que além de resolver este problema ainda traria semântica aos dados, seria a utilização de um vocabulário com propriedades para descrever cada um dos itens do conjunto endereço, como ilustrado na figura 10, onde utiliza-se um vocabulário fictício.

Figura 10 - Exemplo de representação de um endereço em RDF



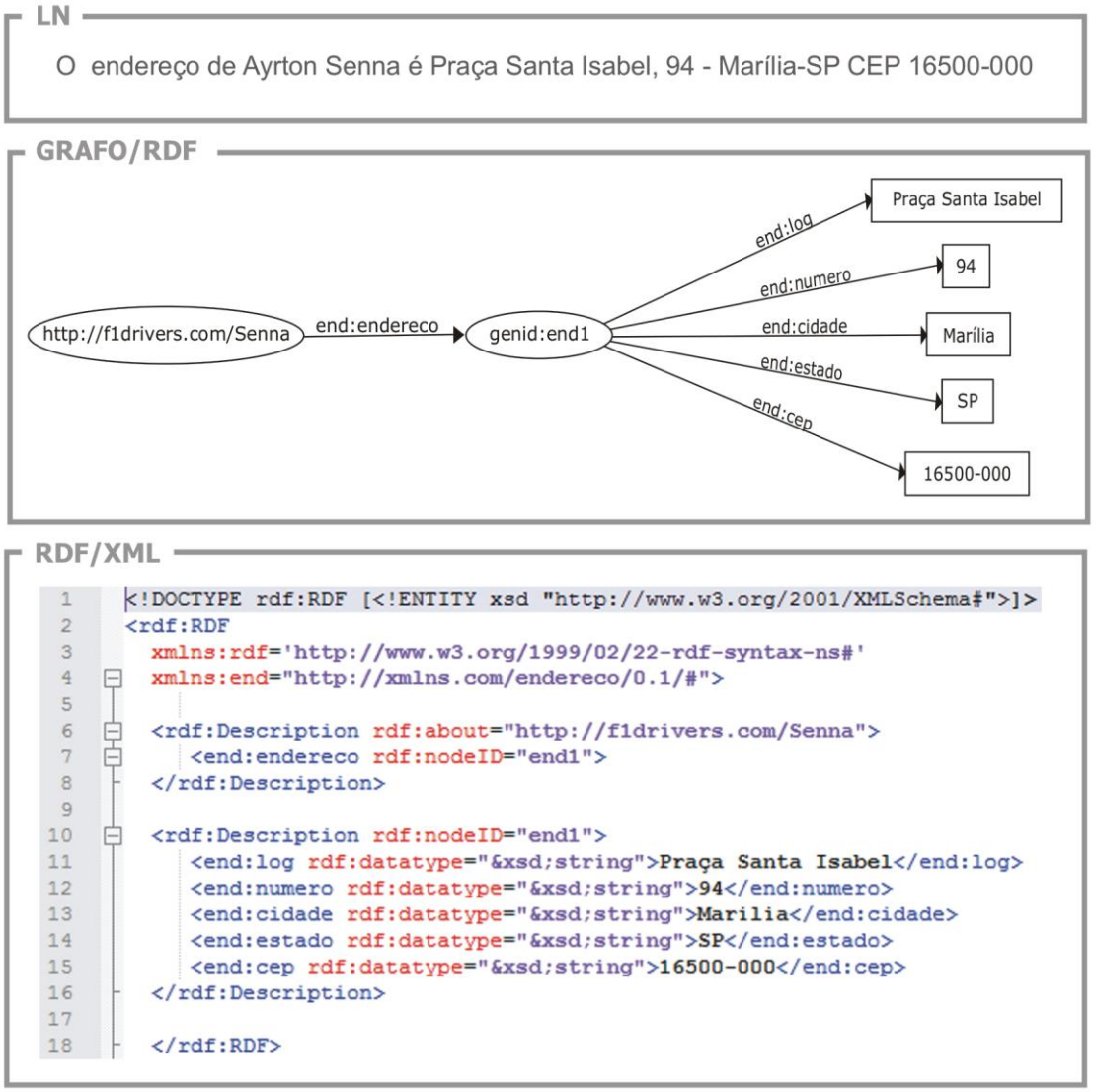

Fonte: Próprio Autor

A solução da figura 10 parece resolver o problema, pois torna os dados do endereço semânticos e entendíveis por um agente, mas pode ocorrer de uma pessoa ter mais de um endereço, o que torna esta segunda solução imprópria, pois cada propriedade teria que ser duplicada com os valores do endereço1 e endereço2, assim um agente não teria como saber qual propriedade pertence a qual endereço. Para resolver este tipo de impasse cada endereço poderia ser descrito como um recurso e com um URI próprio, e este URI atribuído a uma propriedade, mas o endereço de uma pessoa geralmente não é uma entidade de grande importância para receber um URI próprio, e provavelmente só será referido no documento

que descreve o seu detentor como recurso. Para casos como este o modelo RDF disponibiliza uma sintaxe denominada *blank nodes* (nós em branco).

A sintaxe do uso de *blank nodes* é ilustrada na figura 11. Como pode ser observado na linha 10 é criado um recurso, mas ao invés de declarar um URI para este recurso é atribuído um `nodeID`. Esta declaração tem visibilidade apenas dentro do documento onde este é declarado, ao invés de uma visibilidade "global" como no caso de um URI, na linha 7 a propriedade `end:endereco` recebe como valor o mesmo valor declarado para `nodeID` na linha 10 fazendo assim a ligação entre ambos os recursos.

Figura 11 - Exemplo de Blank Nodes




Fonte: Próprio Autor

*Blank nodes* resolveu o problema de se agrupar todas as propriedades do conjunto endereço, mas se for necessário atribuir mais de um endereço para a mesma pessoa, apenas o

uso de *blank nodes* não seria suficiente para este tipo de declaração, o mais indicado seria o uso conjunto de *blank nodes* e *Containers* RDF, este último é abordado na próxima seção.

### 2.3.7 Containers

Quando surge a necessidade de atribuir mais de um valor para uma mesma propriedade, existe uma sintaxe RDF denominada *Containers.* Por exemplo, atribuir nomes dos atores que participaram de um filme em uma propriedade "elenco", entre várias outras situações onde o valor de uma propriedade não é um valor único, mas um grupo de valores.

A principio existem três formas de *Containers*, que apesar de serem sintaticamente iguais apresentam algumas diferenças para um interpretador RDF.

Abaixo segue uma descrição de cada uma das formas:

- **rdf:Bag** : agrupa recursos (incluindo até mesmo *blank nodes*) ou literais, onde são permitidos valores duplicados, e a ordem que são declarados não tem importância;
- **rdf:Seq** : agrupa recursos (incluindo até mesmo *blank nodes*) ou literais, onde são permitidos valores duplicados. Seu uso é recomendado quando a ordem que os valores são declarados tem importância.
- **rdf:Alt** : agrupa recursos (incluindo até mesmo *blank nodes*) ou literais. O uso do "Alt" é recomendando quando se deseja recuperar apenas um valor, pode ser definido como um "valor alternativo" do grupo e não o grupo por completo, como ocorre com *Bag* e *Seq*.

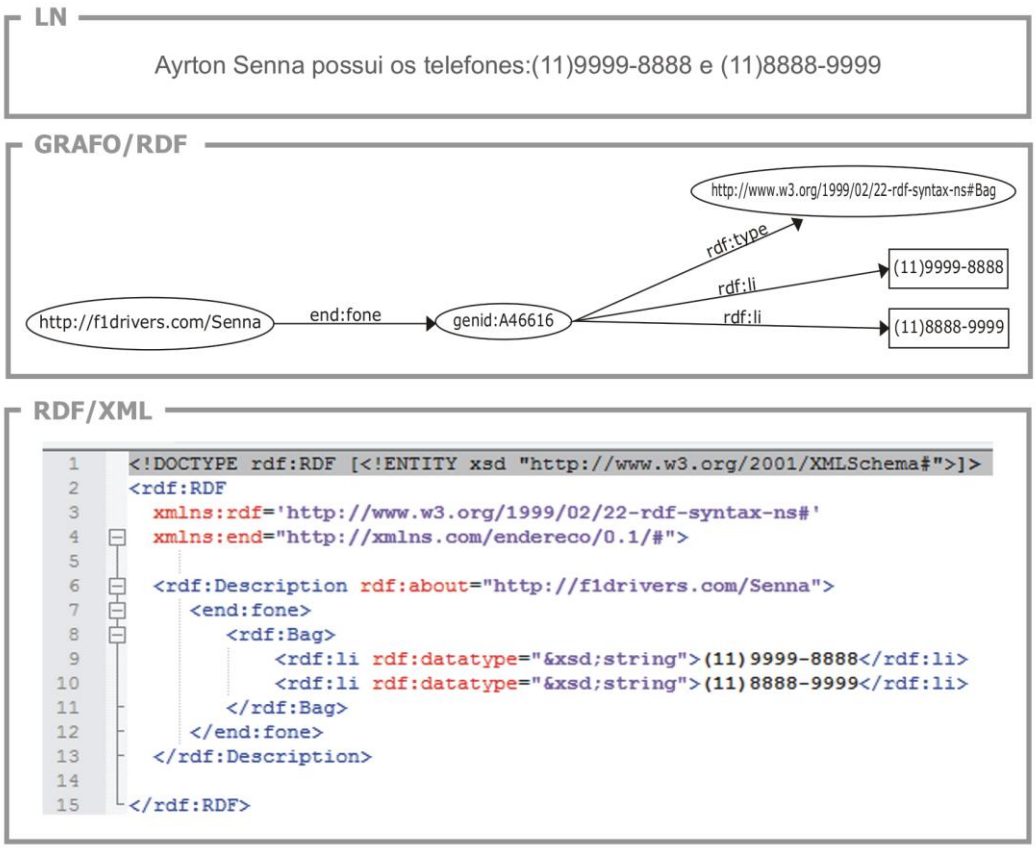
Para exemplificar a sintaxe dos *Containers* RDF, no exemplo da figura 12, são atribuídos dois valores para a propriedade `end:fone`. Como pode ser observado na linha 7 é aberta a tag `<end:fone>` e logo abaixo, na linha 8 a tag `<rdf:Bag>`, isso significa que todas as declarações seguintes até o fechamento da tag `</rdf:Bag>` na linha 11, são os valores da propriedade `end:fone`, e nas declarações dos valores é utilizado o elemento `rdf:li`, sendo este parte do vocabulário padrão do RDF. O elemento `li` indica que se trata de uma lista.

Apesar do exemplo da figura 12 ter utilizado o elemento `Bag`, a sintaxe para os elementos `Seq` e `Alt` é idêntica, como mencionado anteriormente.

A única diferença entre eles é a interpretação que um leitor RDF fará sobre suas listas. Outro fato interessante sobre os *Containers* RDF e interpretadores RDF é que apesar de não existir nenhuma especificação de como os *Containers* devem ser interpretados por estes, na maioria dos casos, os interpretadores vão tratar os *Containers* como um novo recurso, mais

precisamente como um nó em branco. Como pode ser observado no grafo da figura 12 que simula a interpretação do código RDF, e este nó em branco também recebe implicitamente um tipo que no exemplo é `Bag`, mas poderia ser `Seq` ou `Alt` sem nenhuma alteração em sua sintaxe.

Figura 12 - Exemplo de Container BAG com telefones




Fonte: Próprio Autor

## 2.4 RDF Schema

Baseado nos exemplos anteriores é possível reafirmar que o RDF provê meios para a descrição de recursos através da atribuição de propriedades com seus respectivos valores, mas nenhuma propriedade propriamente dita foi criada através do RDF. Utilizou-se vocabulários previamente criados como o “FOAF” e o próprio vocabulário padrão “RDF”.

Nesta seção é abordado o RDF Schema, que tem o objetivo de complementar o RDF, como afirmado por Santarém Segundo (2010),

> [...] a linguagem RDF recebeu como complemento o RDF-Schema, que dá à linguagem RDF o poder de construção de estruturas como hierarquias, propriedades e subpropriedades, entre outros, que a linguagem RDF até então não possibilitava.

O RDF Schema permite a criação de vocabulários próprios, ou seja, a definição de novas classes e propriedades para um domínio especifico, como afirmado por Breitman(2005) “O RDF-Schema não fornece as classes e propriedades propriamente ditas, e sim um framework no qual é possível descrevê-las”.

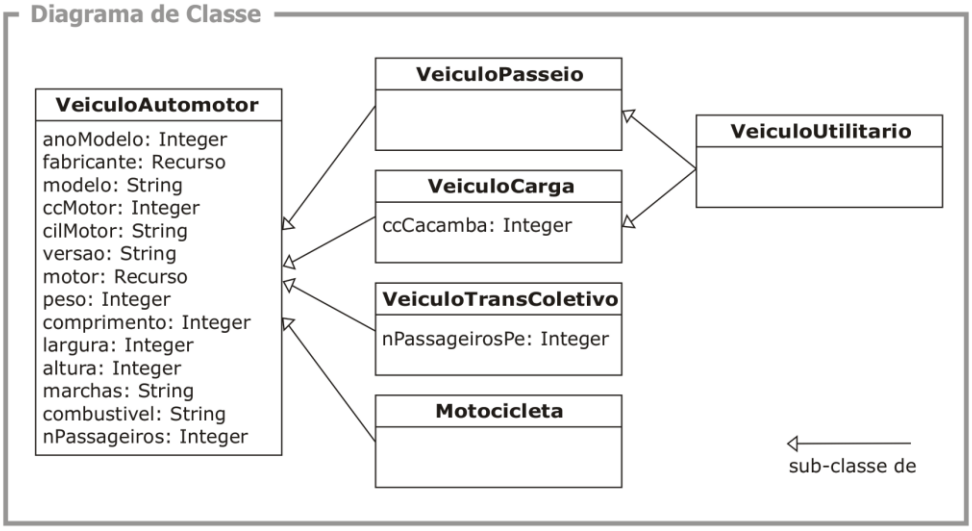
Além de descrever classes e propriedades RDF-Schema permite especificar também os relacionamentos entre classes e propriedades. O próprio RDF-Schema é um vocabulário definido para descrever outros vocabulários, os principais itens que compõem o vocabulário RDF-Schema e suas características podem ser encontrados no apêndice A.

Para exemplificar a criação de um vocabulário RDF-Schema, parte-se da necessidade de descrever o veículo VW Fusca 1600 1982. É um pré-requisito na criação de um vocabulário o conhecimento sobre o domínio onde este será aplicado, neste caso, quais as classes e propriedades uma descrição de veículo deveria ter.

Assim como em outras modelagens existentes na ciência da computação (entidade relacional, programação orientada a objetos,...) não existe uma forma correta e nem incorreta de se modelar um domínio.

Por fugir do escopo deste trabalho, não será coberta a modelagem em si, apenas será descrita a criação de um vocabulário, baseado no diagrama ilustrado na figura 13.

Figura 13 - Diagrama de Classes base para criação do vocabulário.




Fonte: Próprio Autor

### 2.4.1 Classes - RDF Schema

Para a criação de classes o vocabulário do RDF-Schema fornece os seguintes itens:

- **rdfs:Resource:** A classe de todos os recursos;
- **rdfs:Class:** A superclasse de todas as classes (toda classe é um recurso);
- **rdfs:Literal:** A classe de todos os literais;
- **rdf:Property:** A classe de todas as propriedades (é definida no vocabulário RDF);
- **rdfs:Container:** A classe de todos os Containers RDF;
- **rdfs:ContainerMembershipProperty:** A classe de todos os membros de Container;
- **rdfs:Datatype:** A classe de todos Datatype;

Quanto aos relacionamentos entre classes são fornecidos os itens:

- **rdfs:subClassOf:** Define o relacionamento de herança entre classes;
- **rdf:type:** Define que um recurso é a instância de uma classe (é definida no vocabulário RDF);

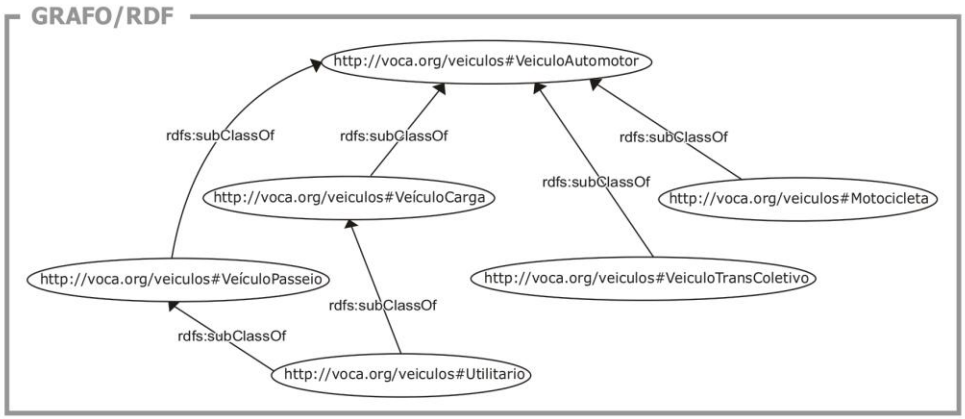
O primeiro passo para a criação de classes é a abertura de um novo vocabulário, como pode ser observado na figura 14, na linha 4. A instrução `xml:base="http://voca.org/veiculos"` define que será descrito um vocabulário, e sua URI base será `http://voca.org/veiculos`, assim todos as declarações seguintes farão parte deste vocabulário.

Seguindo a modelagem da figura 13, na linha 6 da figura 14, é feita a declaração `<rdfs:Class rdf:ID="VeiculoAutomotor"/>` onde é declarada a classe *VeículoAutomotor* , a tag `rdfs:Class` significa que uma classe será declarada e `rdf:ID=` tem como efeito atribuir um título para a classe e consequentemente atribui uma referencia URI para a classe, assim o URI desta classe em particular passa a ser `http://voca.org/veiculos#VeiculoAutomotor`, ou seja para todas as declarações de classes feitas neste vocabulário automaticamente será atribuído um URI, sempre composto por URIbase#ID (URI base do vocabulário # ID atribuído à classe).

Em RDF uma classe pode ser subclasse de uma ou mais classes (é permitido herança múltipla), nas declarações de heranças é utilizada a propriedade `rdfs:subClassOf`, como mostra na linha 9 da figura 14 onde é declarado que a classe *VeículoPasseio* é subclasse de *VeículoAutomotor*. A propriedade *subClassOf* é transitiva, ou seja, uma classe é

automaticamente uma sub-classe de todas as superclasses da qual ela herda, e assim sucessivamente, um exemplo pode ser encontrado na Lina 24 da figura 14, a classe *Utilitario* que é uma subclasse de *VeiculoCarga* e *VeículoPasseio* é automaticamente uma subclasse de *VeículoAutomotor*.

Figura 14 - Classes do Dominio Veículos Automotor




RDF/XML

```
1  <rdf:RDF
2    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
4    xml:base="http://voca.org/veiculos">
5
6  <rdfs:Class rdf:ID="VeiculoAutomotor"/>
7
8  <rdfs:Class rdf:ID="VeiculoPasseio">
9    <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
10 </rdfs:Class>
11
12 <rdfs:Class rdf:ID="VeiculoCarga">
13   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
14 </rdfs:Class>
15
16 <rdfs:Class rdf:ID="VeiculoTransColetivo">
17   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
18 </rdfs:Class>
19
20 <rdfs:Class rdf:ID="Motocicleta">
21   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
22 </rdfs:Class>
23
24 <rdfs:Class rdf:ID="Utilitario">
25   <rdfs:subClassOf rdf:resource="#VeiculoCarga"/>
26   <rdfs:subClassOf rdf:resource="#VeiculoPasseio"/>
27 </rdfs:Class>
28
29 </rdf:RDF>
```

Fonte: Próprio Autor

Vale destacar que em RDF toda classe é automaticamente um recurso, e como foi descrito anteriormente possui um URI, e consequentemente é uma subclasse de *rdfs:Resource*, mesmo que não esteja de forma explicita nas declarações.

### 2.4.2 Propriedades - RDF Schema

Para a criação de propriedades em um novo vocabulário são criadas instâncias da classe *Property* do vocabulário RDF, o RDF-Schema apenas fornece itens para descrever relacionamentos entre propriedades, estes são listados a seguir:

- **rdfs:subPropertyOf:** Define o relacionamento de herança entre propriedades (propriedades RDF são instâncias de *rdf:Property*);
- **rdfs:range:** Especifica o alcance da propriedade, ou seja, qual ou quais classes podem ser o valor da propriedade;
- **rdfs:domain:** Especifica o domínio da propriedade, ou seja, qual ou quais classes podem ter esta propriedade;

No RDF-Schema existem algumas propriedades auxiliares para a construção de vocabulários. Estas são listadas a seguir:

- **rdfs:comment** : Permite a inserção de um texto que descreve o recurso em linguagem natural;
- **rdfs:label** : Permite a inserção de uma etiqueta, geralmente o nome do recurso em linguagem natural;

Para exemplificar a modelagem de propriedades em RDF-Schema, usa-se a comparação com o modelo OO (orientação a objetos) que é um modelo bem conhecido na ciência da computação. Como foi apontado por Manola (2004), RDF-Schema é semelhante em alguns aspectos com o modelo de OO pois faz o uso de classes, propriedades e instâncias. Mas uma grande diferença entre OO e RDF-Schema, é o conceito de propriedades, uma vez que em OO uma classe possui um conjunto de propriedades especificas, em RDF-Schema uma propriedade pode pertencer a várias classes.

Quando existe a necessidade de restringir a atribuição de propriedade em uma ou mais classes especificas, é a própria propriedade quem define um conjunto de classes que fazem parte de seu domínio, através do seu *domain*.

A figura 15 traz um exemplo destas diferenças, onde são declaradas 3 classes: *Artigo*, *Livro* e *Autor*; onde as duas primeiras possuem as propriedades *autor* e *paginas*, e a última tem a propriedade *nome*.

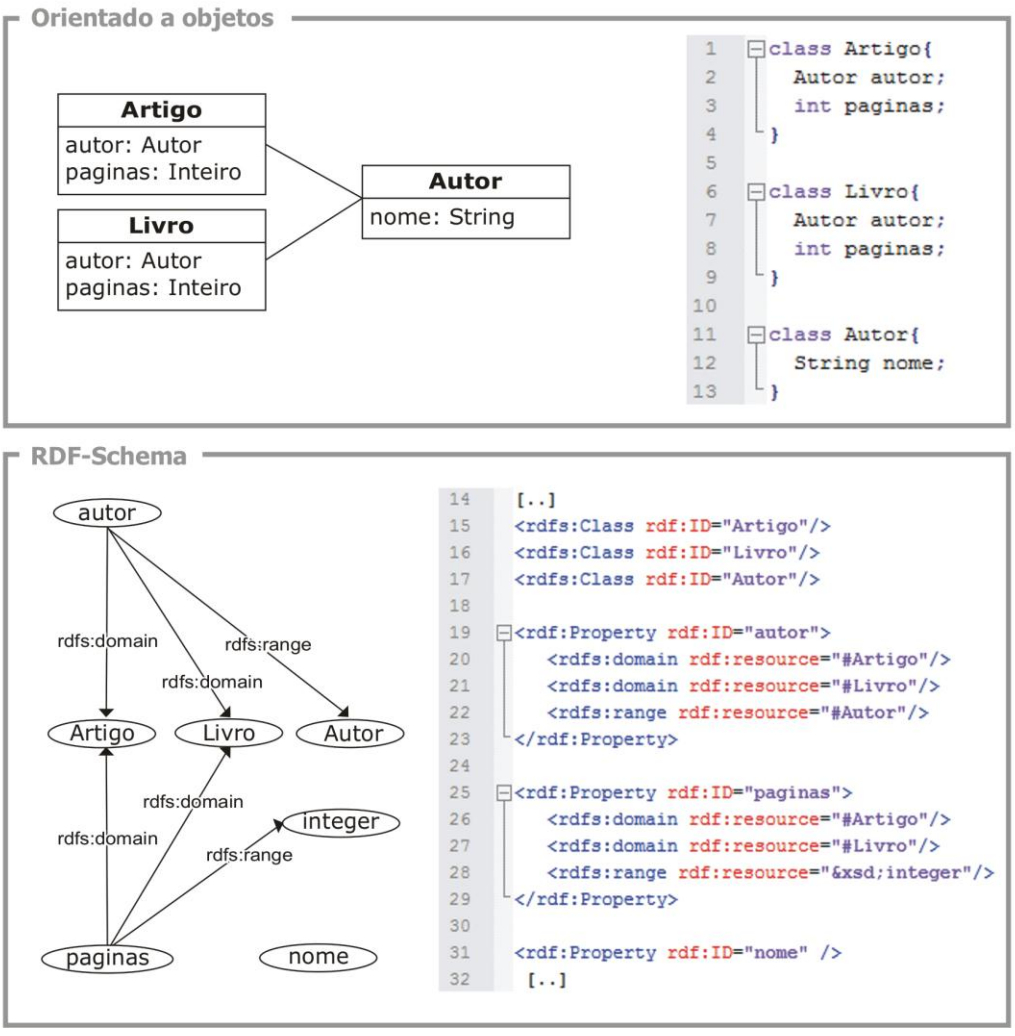
Como pode ser observado na figura 15, mais especificamente nas linhas 1 a 9 no modelo OO são as classes que definem suas propriedades e o tipo do valor destas. Já no modelo RDF-Schema nas linhas 15 e 16, são declaradas as mesmas classes (*Artigo* e *Livro*), mas nenhuma propriedade até então foi definida para elas. Nas linhas 19 a 23 é declarada a

propriedade *autor* e nas linhas 20 e 21 é definido o *domain* desta propriedade como sendo as classes *Artigo* e *Livro*, ou seja, é a própria propriedade que está definindo que ela pode ser uma propriedade de instâncias de *Artigo* e *Livro*. Na linha 22 é definido o *range* da propriedade *autor* como sendo a classe *Autor*, assim a propriedade recebe como valor instâncias desta classe, o mesmo acontece com a propriedade *paginas*, com a diferença que *paginas* tem como *range* o tipo literal *integer* e não uma classe.

Outra grande diferença entre OO e RDF-Schema é o caso da propriedade *nome* declarada na linha 31, que não tem *range* e nem *domain* definidos, isso indica que a propriedade pode ser de qualquer classe e pode receber qualquer tipo de valor.

Esta abordagem torna mais fácil estender o uso de definições de propriedades para casos não previstos na criação de um vocabulário, e é um dos fatores que torna o RDF-Schema flexível.

Figura 15 - Comparação OO x RDF-Schema




Fonte: Próprio Autor

Para exemplificar a sintaxe da definição de propriedades, será utilizado como fonte o diagrama ilustrado na figura 13, que em RDF-Schema é como ilustrado na figura 16, algumas linhas foram omitidas do código, mas vale considerar que este código é continuação do código da figura 14, onde são definidas as classes do vocabulário, e o código completo, com as classes e propriedades deste vocabulário, pode ser encontrado no apêndice B deste trabalho.

Figura 16 - Exemplo de declaração de classes RDF-Schema

RDF/XML

```
1   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
2   <rdf:RDF
3     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5     xml:base="http://ifmsa.net.br/aluno/veiculo">
24  [...]
25  <rdf:Property rdf:ID="anoModelo">
26     <rdfs:label>Ano de Lançamento do Modelo</rdfs:label>
27     <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
28     <rdfs:range rdf:resource="&xsd;integer"/>
29  </rdf:Property>
30
31  <rdf:Property rdf:ID="fabricante">
32     <rdfs:label>Fabrincante(Montadora) do Veículo</rdfs:label>
33     <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
34     <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
35  </rdf:Property>
52  [...]
53  <rdf:Property rdf:ID="ccCacamba">
54     <rdfs:label>Centímetros Cúbicos da Caçamba</rdfs:label>
55     <rdfs:domain rdf:resource="#VeiculoCarga"/>
56     <rdfs:range rdf:resource="&xsd;integer"/>
57  </rdf:Property>
58
59  <rdf:Property rdf:ID="nPassageirosPe">
60     <rdfs:label>Número de Passageiros em pé</rdfs:label>
61     <rdfs:domain rdf:resource="#VeiculoTransColetivo"/>
62     <rdfs:range rdf:resource="&xsd;integer"/>
63  </rdf:Property>
64  [...]
70
71  </rdf:RDF>
```

Fonte: Próprio Autor

Da mesma forma que ocorreu com as classes, o primeiro passo para a criação de propriedades é a abertura de um novo vocabulário (neste caso foi utilizado o mesmo vocabulário da figura 14), como pode ser observado na linha 4 da figura 16, a instrução *xml:base="http://voca.org/veiculos"* define que será descrito um vocabulário, onde a URI base deste será *http://voca.org/veiculos*, assim todos as declarações seguintes farão parte deste vocabulário, na linha 25 a tag *<rdf:Property rdf:ID="anoModelo">*

declara uma propriedade o qual o seu ID é `anoModelo` e todas as declarações até a sua tag de fechamento na linha 29 são desta propriedade.

Na linha 26 é adicionado um *label* à propriedade; na linha 27 é definido o *domain* desta propriedade como sendo a classe *VeiculoMotor*, ou seja, esta propriedade pode ser atribuída a qualquer instância de *VeículoMotor* e suas subclasses; na linha 28 é definido o *range* desta propriedade e este é um literal do tipo inteiro.

As demais declarações de propriedades ilustradas seguem a mesma sintaxe, mas com algumas diferenças, por exemplo: a propriedade *fabricante* como pode ser observado na linha 34, tem seu *range* declarado tendo como valor a classe *Resource*, ou seja, pode ser qualquer classe, como não foi definida uma classe *Fabricante* no vocabulário, não é possível prever qual será a classe atribuída nesta propriedade quando um veículo for instanciado.

Definir o *range* de uma propriedade como *Resource*, acaba por deixa-la aberta para receber qualquer subclasse de *Resource* como valor, isso torna o vocabulário mais flexível e é uma das vantagens do RDF.

Como pode ser observado nas seções 2.4.1 e 2.4.2, RDF-Shema fornece meios para descrever vocabulários RDF básicos, ficando alguns recursos apontadas como úteis por Manola (2004), não sendo possíveis de se descrever. Alguns deles são:

- Classificar por valores de propriedades, por exemplo, classificar uma pessoa como falecida, caso ela possua uma propriedade como “dataFalecimento”;
- Restrições de cardinalidade em propriedades, por exemplo, definir que uma pessoa tem apenas um pai;
- Determinar propriedades como sendo transitivas, por exemplo, se João é ancestral de Pedro, e Pedro é ancestral de Lucas, se a propriedade ancestral for transitiva é possível inferir que João é ancestral de Lucas;
- Especificar cardinalidade de uma propriedade dependendo do seu *domain*, um exemplo poderia ser a propriedade “njogadores”, quando seu domínio for “futebol” esta teria 11 valores, e quando o seu domínio for basquete esta teria 5 valores;
- Aplicar regras da teoria dos conjuntos (união, intersecção, disjunção);

Adicionar estes recursos a vocabulários, sem dúvida, traz mais expressividade para o RDF, e este é o alvo das linguagens de ontologias, onde na área da web semântica a que mais se destaca é a OWL.

Sendo o foco desse trabalho o funcionamento básico de um aplicativo na web semântica, e OWL não sendo necessária para uma aplicação básica, não será abordada neste trabalho. Mas vale destacar que este é de grande importância para inferências na web semântica.

## 2.5 Instâncias RDF

Nesta seção é exemplificada a descrição (instância) de um veículo em RDF/XML, utilizando-se o vocabulário criado nas seções anteriores. Todas as regras de sintaxe descritas na seção 2.3 são aplicáveis nesta descrição, com a diferença que nesta descrição será instanciada uma classe dando origem a um novo recurso, e nos exemplos da seção 2.3, foi utilizado como exemplo um recurso já existente.

Como pode ser observado na figura 17, nas linhas 2 e 3, são declarados os prefixos dos vocabulários RDF e RDFS, respectivamente, e na linha 4 é declarado o prefixo *dv* atribuído ao URI do vocabulário criado nas seções anteriores.

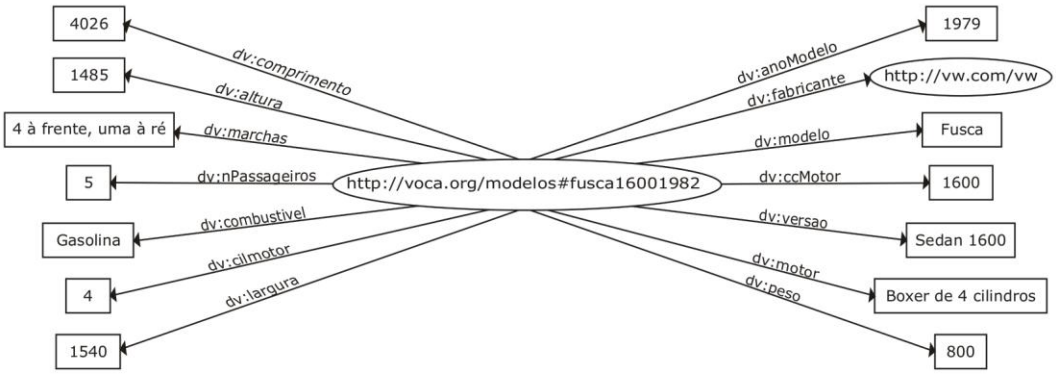
Na linha 5 é definido um URI base, e na linha 7 é declarada uma nova instância para a classe *VeiculoPasseio*, e atribuído a esta o *ID* "fusca16001982". Da mesma forma que ocorreu com as classes e propriedades criadas nos exemplos anteriores, esta instância se torna um novo recurso, e recebe o URI composto pelo seu URIbase#ID, ou seja, o URI deste recurso passa a ser "http://voca.org/modelos#fusca16001982". Entre as linhas 8 e 21, são valoradas as propriedades deste recurso, assim qualquer aplicação capaz de interpretar RDF poderia facilmente "entender" as características deste veículo.

Vale ressaltar, que o vocabulário criado para exemplificar é básico e muito simples. Um vocabulário para uma aplicação real capaz de descrever um veículo demandaria, um estudo detalhado sobre o domínio, e teria uma quantidade considerável de classes e propriedades muito mais complexas que estas criadas para os exemplos.

Uma vez que existam informações organizadas e armazenadas nos padrões da Web Semântica (RDF, por exemplo), é preciso uma forma de se recuperar estas informações, para esta finalidade no próximo capítulo é abordado o SPARQL.

Figura 17 - Exemplo de uma instância RDF

GRAFO/RDF




RDF/XML

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:dv="http://voca.org/veiculo#"
  xml:base="http://voca.org/modelos">

  <dv:VeiculoPasseio rdf:ID="fusca16001982">
    <dv:anoModelo>1982</dv:anoModelo>
    <dv:fabricante rdf:resource="http://vw.com/vw"/>
    <dv:modelo>Fusca</dv:modelo>
    <dv:ccMotor>1600</dv:ccMotor>
    <dv:versao>Sedan 1600</dv:versao>
    <dv:motor>Boxer de 4 cilindros</dv:motor>
    <dv:peso>800</dv:peso>
    <dv:comprimento>4026</dv:comprimento>
    <dv:largura>1540</dv:largura>
    <dv:altura>1485</dv:altura>
    <dv:marcahas>4 à frente, uma à ré</dv:marcahas>
    <dv:nPassageiros>5</dv:nPassageiros>
    <dv:combustivel>Gasolina</dv:combustivel>
    <dv:cilMotor>4</dv:cilMotor>
  </dv:VeiculoPasseio>

</rdf:RDF>
```

Fonte: Próprio Autor

# CAPÍTULO 3 - SPARQL – SPARQL PROTOCOL AND RDF QUERY LANGUAGE

Este capítulo da início a segunda parte do trabalho, onde são abordadas as tecnologias para a implementação de consumidores de conteúdo semântico, mais especificamente será abordado o SPARQL, o padrão W3C para consultas em documentos RDF.

## 3.1 SPARQL – Conceitos Básicos

SPARQL (*SPARQL Protocol and RDF Query Language*) é uma linguagem e protocolo de consulta para RDF, sendo um padrão recomendado pelo W3C, desde janeiro de 2008. Como definido por Clark (2008), SPARQL como protocolo faz o uso de WSDL (*Web Services Description Language*) para descrever um meio de transmissão para consultas, ou seja, um meio para o envio de consultas e também para o retorno dos seus respectivos resultados através de *Web Services*. Desta forma, SPARQL permite consultar documentos RDF remotamente através da sua linguagem de consulta que é sintaticamente semelhante ao SQL (*Structured Query Language*), sendo essa a linguagem de consulta mais popular entre os desenvolvedores.

Vale destacar, que além de ser o pilar da WS, o conjunto RDF+SPARQL por si só, já traz uma melhora significativa para um dos maiores desafios da Web atual, a recuperação da informação.

Para demonstrar o poder destas duas tecnologias juntas, é utilizado como exemplo, uma pesquisa onde o resultado é a lista de todas as corridas de Formula-1, onde o piloto Ayrton Senna chegou em primeiro lugar, e que o segundo colocado foi, o também piloto, Alain Prost.

Certamente o DBA (*Database administrator*) da FIA (*Fédération Internationale de l'Automobile*), ou de outra entidade ligada ao esporte, conseguiria esta lista consultando a sua base de dados. Mas se tratando de dados abertos na Web, e sendo o Google a ferramenta mais popular para buscas, foi realizada uma pesquisa utilizando o mesmo. Na busca foram utilizados alguns termos (palavras chaves), mas como já era esperado, nenhum trouxe o resultado esperado (pelo menos não nos doze primeiros resultados). Os termos utilizados e os resultados da pesquisa podem ser encontrados no apêndice C.

Outra forma para encontrar a resposta para a questão, seria o uso de tecnologias da Web 2.0 (fóruns, redes sociais, wikis, etc..). Em se tratando de fóruns, é provável que em algum especializado sobre o assunto, um usuário do mesmo, que tenha este conhecimento poderia responder a questão, mas a espera por sua resposta, demoraria um tempo indeterminado, e também seria difícil estimar a credibilidade da sua resposta.

Outra ferramenta da Web 2.0 que poderia trazer a resposta seria a Wikipedia.org, já que disponibiliza o resultado de todas as corridas dos pilotos citados na pergunta, mas seria necessária a inferência de um usuário humano nestes dados, para se chegar ao resultado esperado, que seria comparar a posição de chegada de cada piloto em todas as corridas, (apêndice D).

Apesar da WS ainda estar na sua fase inicial, é possível, de forma rápida e fácil (noções básica de SPARQL), encontrar a resposta para a pergunta: “Quais corridas Ayrton Senna chegou na primeira posição, seguido por Alain Prost na segunda posição”.

A figura 18 ilustra uma consulta SPARQL, realizada na base RDF da DBpedia, e que retornou corretamente a lista com todas as corridas que satisfazem a pergunta.

A DBpedia é uma comunidade que vem extraindo e estruturando dados da Wikipedia.org em RDF, no momento que este trabalho estava sendo escrito a base da DBpedia contava com cerca de 1 bilhão de triplas RDF e vem crescendo a cada dia. Maiores informações sobre a DBpedia podem ser encontradas em <http://wiki.dbpedia.org/About>.

Uma observação importante sobre o exemplo da consulta da figura 18, é que o resultado desta consulta, foi uma lista contendo todas as corridas [grande prêmio (*Grand Prix*)] que satisfizeram a consulta. Mas a lista não se trata de uma lista de Strings, como geralmente ocorre em consultas SQL a uma base de dados convencional. A lista em questão é uma lista de recursos, ou seja, cada grande prêmio da lista é um recurso, e cada um tem várias propriedades que os descrevem, sendo assim possível obter mais informações sobre cada uma das corridas, e, até mesmo, adicionar novos critérios de busca. Esta é uma das vantagens de se utilizar, sempre que possível, um recurso e não uma apenas um literal como valor de propriedade.

Figura 18 - Resultado da pesquisa “Quais corridas Ayrton Senna chegou na primeira posição, seguido por Alain Prost na segunda posição”


Fonte: Próprio Autor

Para mostrar um pouco mais do “poder” de RDF+SPARQL será executada uma segunda consulta, que além de trazer todas as corridas onde Ayrton Senna foi o primeiro e Alain Prost o segundo, mas que também traga informações sobre como estava o clima em cada uma das corridas.

Como pode ser observado na figura 19, o resultado satisfez a consulta, e só foi possível pelo fato de que, cada grande prêmio, foi descrito como sendo um recurso, e por existir a propriedade *dbpedia2:weather*, definida para cada um deles.

Figura 19 - Resultado da consulta “Quais corridas Ayrton Senna chegou na primeira posição, seguido por Alain Prost na segunda posição, e como estava o clima neste dia”

```
SELECT ?corrida ?clima
WHERE {
?corrida <http://dbpedia.org/ontology/firstDriver> <http://dbpedia.org/resource/Ayrton_Senna>.
?corrida <http://dbpedia.org/ontology/secondDriver> <http://dbpedia.org/resource/Alain_Prost>.
?corrida dbpedia2:weather ?clima
}
ORDER BY ?corrida
```

Results: Browse Go! Reset

SPARQL results:

| corrida | clima |
|---|---|
| :1988_Belgian_Grand_Prix | "Warm, dry and overcast"@en |
| :1988_Canadian_Grand_Prix | "Sunny and hot"@en |
| :1988_Detroit_Grand_Prix | "Warm, sunny"@en |
| :1988_German_Grand_Prix | "Wet and cool"@en |
| :1988_Hungarian_Grand_Prix | "Sunny and hot"@en |
| :1988_Japanese_Grand_Prix | "Cool and mainly dry, some rain toward the end"@en |
| :1988_San_Marino_Grand_Prix | "Sunny, warm"@en |
| :1989_Belgian_Grand_Prix | "Wet, cloudy, cool"@en |
| :1989_German_Grand_Prix | "Warm, cloudy"@en |
| :1989_Monaco_Grand_Prix | "Warm, dry, sunny"@en |
| :1989_San_Marino_Grand_Prix | "Warm, dry, sunny"@en |
| :1990_Belgian_Grand_Prix | "Dry"@en |
| :1990_Italian_Grand_Prix | "Hot, dry, sunny"@en |
| :1991_United_States_Grand_Prix | "Clear and Warm"@en |
| :1993_Australian_Grand_Prix | "Sunny"@en |
| :1993_Japanese_Grand_Prix | "Dry, warm, cloudy"@en |

Fonte: Próprio Autor

Com os exemplos anteriores, é possível notar a contribuição que a WS traz para as buscas na Web, e ainda vale ressaltar que as buscas em SPARQL, somente são possíveis em dados RDF, ou seja, existe a necessidade prévia da reestruturação de conteúdos da Web atual para o padrão RDF, para que se possa, então, desfrutar deste tipo de busca.

SPARQL permite quatro formas diferentes de consultas:

- **SELECT:** Forma de consulta que tem como retorno uma lista tabular, de acordo com uma condição definida (cláusula WHERE), semelhante com o SELECT do SQL;
- **CONSTRUCT:** Forma de consulta que tem como retorno um novo grafo, de acordo com uma condição definida (cláusula WHERE);
- **ASK:** Retorna um valor booleano (TRUE ou FALSE), de acordo com a existência ou não do padrão gráfico definido (cláusula WHERE) na base consultada;

- **DESCRIBE:** Retorna um grafo de acordo com suas restrições definidas na clausula WHERE. É uma ótima alternativa para recuperar todas as triplas onde um recurso é citado;

Pelo fato da documentação do SPARQL ser extensa, serão exemplificadas apenas as principais funcionalidades de cada uma das formas de consultas, com um foco maior na SELECT. [a documentação completa pode ser encontrada em Prud'hommeaux(2008)].

## 3.2 SELECT SPARQL

A sintaxe SELECT do SPARQL é similar à sintaxe SQL, uma consulta simples tem a sintaxe: `SELECT ?<variavel> WHERE { <condição> .}` ; onde variável é sempre precedida pelo símbolo "?", podendo ser mais de uma, separadas por espaço. A condição é na maioria das vezes uma tripla RDF(contendo os termos recurso, propriedade e valor, sempre nesta ordem e separados por espaço), onde um ou mais termos desta tripla é substituído por uma variável. Quando isso ocorre este, o termo substituído é denominado incógnita. O motor de consulta SPARQL tenta achar um tripla RDF em sua fonte de dados que corresponda com a tripla "condição", e atribui na variável (incógnita) o valor correspondente da tripla encontrada, ou seja, o motor cria um sub-grafo com a condição, e procura por ocorrências deste em sua base, este sub-grafo criado é denominado como Padrão Gráfico Básico, Prud'hommeaux(2008).

### 3.2.1 Consultas Simples SPARQL

O funcionamento de uma consulta SPARQL é ilustrado na figura 20, onde no quadro "GRAFO/RDF" existe a descrição do recurso `http://voca.org/modelos#fusca16001982`, e no quadro "SPARQL" existe uma consulta, esta deseja buscar o valor da propriedade `dv:anoModelo` do recurso citado. No quadro "Padrão Gráfico Básico" é ilustrada a consulta em forma de grafo, onde o valor da propriedade `dv:anoModelo`, é a variável(incógnita) `?ano`, desta forma o motor de busca SPARQL percorre o grafo fonte e encontra o valor da propriedade `dv:anoModelo` que foi atribuído ao recurso citado e atribui este a variável `?ano`.

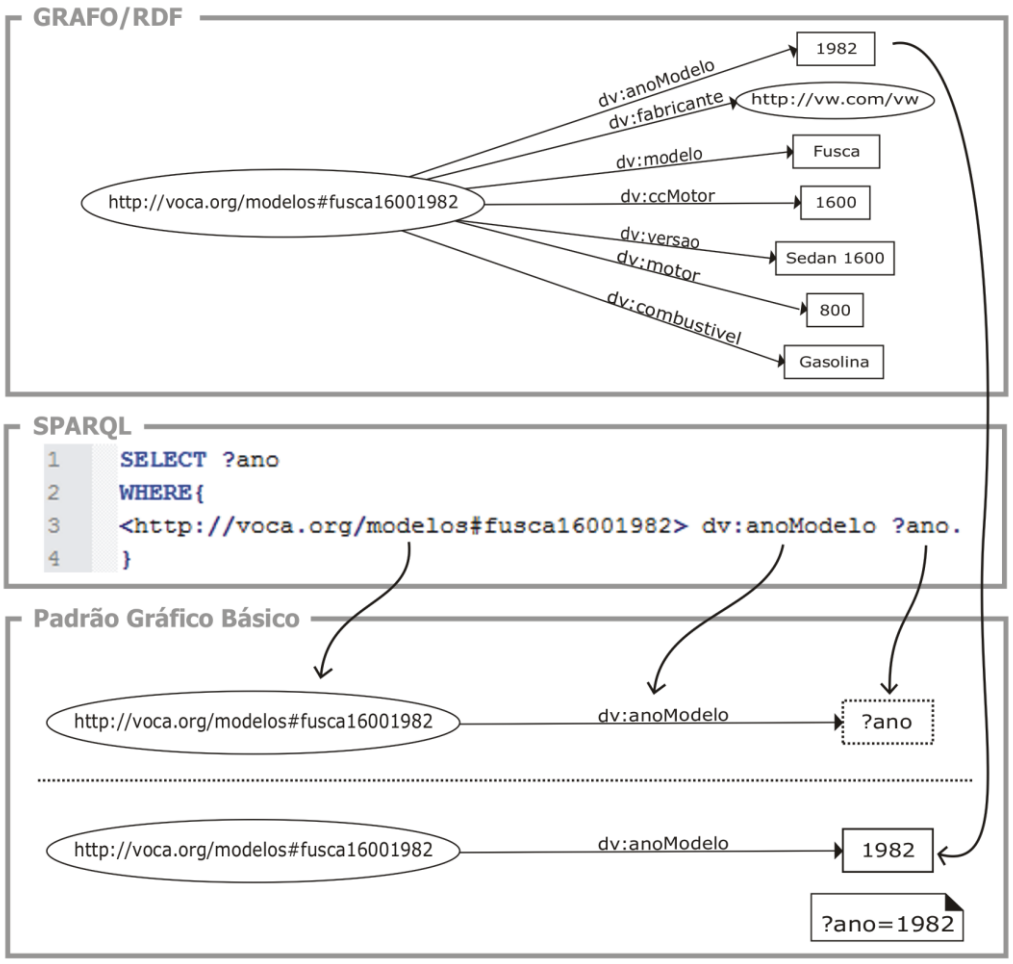
Neste exemplo o foco da busca foi o valor de uma propriedade, mas como citado anteriormente o foco da busca pode ser qualquer termo de uma tripla, ou seja, pode ser um recurso, uma propriedade ou um valor.

Outro exemplo de consulta tendo o grafo da figura 20 como base, seria buscar qual recurso tem o valor “1600” para a propriedade *dv:ccMotor*, o código SPARQL para esta consulta seria: *SELECT ?recurso WHERE{ ?recurso dv:ccMotor “1600”.};* ou para saber qual propriedade deste recurso tem o valor “Gasolina” o código seria:

*SELECT ?propriedade WHERE{ <http://voca.org/modelos#fusca16001982> ?propriedade “Gasolina”.}.*

Vale destacar que quando se usa um literal na consulta como no caso de “Gasolina” o uso de aspas duplas “..” é obrigatório.

Figura 20 - Exemplo de consulta simples SPARQL




Fonte: Próprio Autor

### 3.2.2 Consultas Compostas SPARQL

SPARQL permite consultas compostas por mais de uma variável, e permite também que o resultado seja composto por mais de uma ocorrência, ou seja, o resultado é composto

por quantas correspondências do padrão gráfico existirem no RDF fonte, e este é retornado em formato tabular.

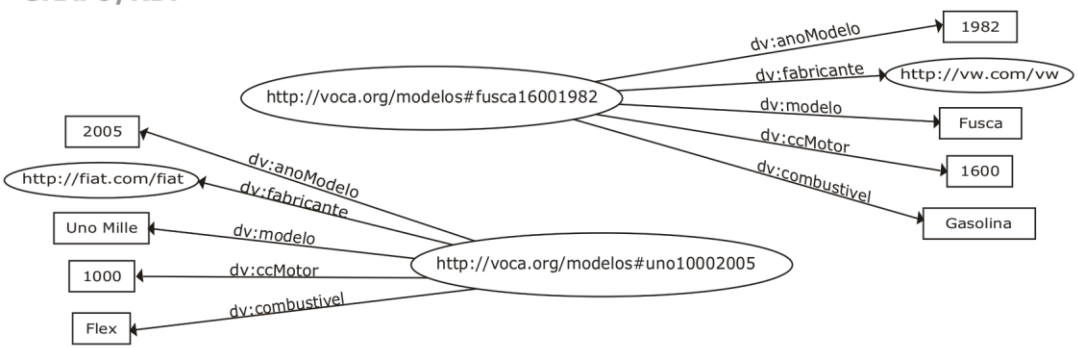
Como exemplo de consulta e retorno composto, é utilizada a figura 21, onde o RDF base é a descrição de dois veículos (recursos), e se consulta o ano, modelo e URI destes veículos, abaixo segue a descrição de cada linha do código apresentado na figura:

- Linha 1: na declaração `PREFIX dv: http://voca.org/veiculo#` que se trata da declaração do *namespace* do vocabulário utilizado na consulta, o conceito de *namespace* é o mesmo que na criação de documentos RDF já abordado anteriormente;
- Linha 2: a instrução `SELECT ?ano ?modelo ?recurso`, significa que se deseja como retorno estas três incógnitas;
- Linha 3 : a instrução `WHERE{` indica a abertura das condições de consulta;
- Linha 4: é declarada a primeira condição em forma de tripla `?recurso dv:anoModelo ?ano.` que cria uma tripla onde o recurso é a variável `?recurso`, a propriedade é `dv:anoModelo` e o valor é a variável `?ano`, ou seja, nesta tripla recurso e valor são as incógnitas, assim o motor de consulta do SPARQL trará como resultado todas as triplas onde um recurso tenha a propriedade `dv:anoModelo` definida;
- Linha 5 : o retorno esperado são os recursos que possuam a propriedade `dv:Modelo` definida, como ambas condições estão separadas por um ponto ".", este é equivalente ao operado "AND" do SQL, ou seja, serão retornadas apenas as triplas que se enquadram em ambas condições;
- Linha 6 : `}` indica o fechamento da clausula `WHERE`;

Outra sintaxe para esta consulta seria alterar a linha 2 para "`SELECT *`", o uso do asterisco após o SELECT traz uma tabela contendo todas as variáveis utilizadas na clausula WHERE e não somente as declaradas após o SELECT, da mesma forma como ocorre no SQL.

Figura 21 - Exemplo de consulta composta SPARQL

GRAFO/RDF




SPARQL

```
1  PREFIX dv: <http://voca.org/veiculo#>
2  SELECT ?ano ?modelo ?recurso
3  WHERE{
4  ?recurso dv:anoModelo ?ano.
5  ?recurso dv:modelo ?modelo.
6  }
```

Resultado

| ?ano | ?modelo | ?recurso |
|---|---|---|
| 1982 | Fusca | http://voca.org/modelos#fusca16001982 |
| 2005 | Uno Mille | http://voca.org/modelos#uno10002005 |

Fonte: Próprio Autor

## 3.3 CONSTRUCT - SPARQL

Esta forma de consulta é muito semelhante ao SELECT, inclusive o funcionamento e sintaxe da sua clausula WHERE. A principal diferença está no formato do retorno, enquanto SELECT retorna dados tabulares, CONSTRUCT retorna um novo documento RDF/XML (ou outros formatos dependendo do motor utilizado).

Para exemplificar uma consulta CONSTRUCT é utilizada a figura 22. Nesta, é realizada uma consulta onde se deseja ter como retorno todas as corridas de Formula-1 ocorridas no ano de 1987, em que Ayrton Senna foi o primeiro colocado. No quadro “SPARQL” da referida figura, mais precisamente na linha 3, é declarada que se trata de uma consulta “CONSTRUCT”, na linha 4 é definido o formato da tripla, que deverá ser retornada, ou seja, o que será o recurso, propriedade e valor respectivamente do documento retornado. Nas linhas 7 e 8 são declaradas as condições da consulta (clausula WHERE).

Como pode ser observado no quadro “Resultado” da figura 22, o motor retornou o resultado da consulta no formato RDF/XML. Consultas deste tipo são muito úteis quando é necessário extrair parcelas de informações de grafos extensos e é uma poderosa funcionalidade do SPARQL.

Figura 22 - Exemplo de consulta CONSTRUCT

SPARQL

```
PREFIX dbpedia2:<http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
CONSTRUCT {
?corridas dbpedia:ontology/firstDriver <http://dbpedia.org/resource/Ayrton_Senna>
}
WHERE {
?corridas dbpedia:ontology/firstDriver <http://dbpedia.org/resource/Ayrton_Senna>.
?corridas dbpedia2:yearOfRace 1987
}
```

Resultado

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:dbpedia-owl="http://dbpedia.org/ontology/#">

    <rdf:Description rdf:about="http://dbpedia.org/resource/1987_Detroit_Grand_Prix">
      <dbpedia-owl:firstDriver  rdf:resource="http://dbpedia.org/resource/Ayrton_Senna"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://dbpedia.org/resource/1987_Monaco_Grand_Prix">
      <dbpedia-owl:firstDriver rdf:resource="http://dbpedia.org/resource/Ayrton_Senna"/>
    </rdf:Description>
</rdf:RDF>
```

Fonte: Próprio Autor

## 3.4 DESCRIBE – SPARQL

DESCRIBE é a forma de consulta SPARQL para casos onde não se tem conhecimento prévio da estrutura dos dados ou vocabulários utilizados na descrição destes, ou seja, o retorno é composto por todas as triplas, onde o objeto de consulta esteja envolvido, seja este um recurso, propriedade ou valor. Para exemplificar esta forma de consulta, na figura 23, é ilustrada uma consulta DESCRIBE para o recurso Ayrton Senna.

O retorno da consulta foi limitado por questão de espaço, mas como pode ser observado, foram retornadas todas a triplas da base da DBpedia onde o recurso Ayrton Senna, era um dos termos (recurso, propriedade ou valor).

Figura 23 - Exemplo de consulta DESCRIBE

SPARQL

```
1   DESCRIBE <http://dbpedia.org/resource/Ayrton_Senna>
```

Resultado

| | | |
|---|---|---|
| :1985_Portuguese_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1991_Portuguese_Grand_Prix | dbpedia2:secondDriver | :Ayrton_Senna |
| :1988_Hungarian_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1988_Australian_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1991_Italian_Grand_Prix | dbpedia2:secondDriver | :Ayrton_Senna |
| :1989_Brazilian_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :Ayrton_Senna | dbpedia:ontology/birthDate | "1960-03-21" |
| :Ayrton_Senna | dbpedia:ontology/podiums | "80" |
| :1989_San_Marino_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :Ayrton_Senna | foaf:surname | "Senna" |
| :1988_Australian_Grand_Prix | dbpedia:ontology/secondDriver | :Ayrton_Senna |
| :Ayrton_Senna | rdfs:label | "Сенна, Айртон" |
| :1992_German_Grand_Prix | dbpedia2:secondDriver | :Ayrton_Senna |
| :1989_United_States_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1987_Japanese_Grand_Prix | dbpedia:ontology/secondDriver | :Ayrton_Senna |
| :Roberto_Moreno | dbpedia2:after | :Ayrton_Senna |
| :1985_Austrian_Grand_Prix | dbpedia2:secondDriver | :Ayrton_Senna |
| :1989_Italian_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1987_German_Grand_Prix | dbpedia:ontology/thirdDriver | :Ayrton_Senna |
| :1985_Portuguese_Grand_Prix | dbpedia2:fastDriver | :Ayrton_Senna |
| :1989_Belgian_Grand_Prix | dbpedia:ontology/poleDriver | :Ayrton_Senna |
| :1987_Italian_Grand_Prix | dbpedia:ontology/fastestDriver | :Ayrton_Senna |

[...]

Fonte: Próprio Autor

## 3.5 ASK – SPARQL

ASK é a forma de consulta que retorna apenas valores booleanos (TRUE ou FALSE), ou seja, é fornecido ao motor apenas a condição para consulta, e o motor SPARQL pesquisa a ocorrência desta tripla (padrão gráfico) em sua base, existindo retorna TRUE, caso contrário retorna FALSE.

ASK é totalmente dependente da sua clausula WHERE, que tem a mesma sintaxe e funcionamento, que seu uso no SELECT.

Como exemplo, segue na figura 24, uma consulta ASK na base da DBpedia, onde foi questionado se Ayrton Senna foi o primeiro colocado no GP de Mônaco de 1992.

Figura 24 - Exemplo de consulta ASK

SPARQL

```
1   PREFIX dbpedia: <http://dbpedia.org/>
2   ASK
3   WHERE {
4       <http://dbpedia.org/resource/1992_Monaco_Grand_Prix>
5       dbpedia:ontology/firstDriver
6       <http://dbpedia.org/resource/Ayrton_Senna>
7   }
```

Resultado

```
true
```

Fonte: Próprio Autor

Como pode ser observado na figura é passado na clausula WHERE um padrão gráfico, que neste exemplo por questão de espaço, o recurso, propriedade e valor não estão na mesma linha, e sim divididos nas linhas 4,5 e 6 respectivamente, e a resposta retornada foi o booleano `true`, o formato de retorno pode variar dependendo do motor SPARQL, mas sempre será um valor booleano.

Dando continuidade a tecnologias para consumidores de conteúdo semântico no próximo capítulo é abordado o framework Jena, que além de trazer um motor para consultas SPARQL, fornece uma coleção de funções para manipulação de arquivos RDF.

# CAPÍTULO 4 - JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA

Dando continuidade a segunda parte do trabalho e abordando tecnologias para implementação de consumidores de conteúdo semântico, neste capítulo será abordado o framework Jena, que fornece uma biblioteca para manipulação de arquivos nos padrões da WS, e ainda traz mecanismos para consultas SPARQL e raciocinadores.

## 4.1 Jena – Conceitos Básicos

Jena é um framework Java de código aberto para o desenvolvimento de aplicações da WS, fornece uma poderosa biblioteca para manipulação de arquivos RDF, RDF-Schema e OWL, também possui classes que permitem executar consultas SPARQL, e um motor de inferência.

Foi primeiramente desenvolvida por Brian McBride, funcionário da HP (*Hewlett-Packard*). Baseando-se num projeto denominado SIRPAC-API, que já se encontrava com alguns conceitos bem fundamentados, e que por decisões internas da HP foi descontinuado e acabou por dar origem ao Jena, sobre licença BSD (*Berkeley Source Distribution*).

Quando se utiliza um framework, o passo inicial é conhecer a sua API (*Application Programming Interface).* A API completa do Jena pode ser encontrada em <http://jena.sourceforge.net/javadoc/index.html>, mas entre as suas principais classes pode-se destacar:

- **Model:** classe que representa um documento RDF/OWL;
- **Statement:** classe que representa uma tripla RDF;
- **Resource:** classe que pode representar um recurso RDF;
- **Query:** classe que representa consultas SPARQL;
- **Reasoner:** classe que representa motores de inferência;

O funcionamento básico do Jena, quando se trabalha com arquivos RDF é carrega-los na memória, ou seja, Jena transforma o documento em objeto, e também oferece meios para se criar classes Java baseadas em RDF-Schema, ou seja, converte classes e propriedades deste documento em classes e propriedades Java, esta conversão é realizada através da classe “schemagen”.

## 4.2 Populando Um Arquivo RDF

Para exemplificar a representação de um arquivo RDF no Jena, foi realizado um exemplo básico, encontrado na figura 25, onde foi populado um arquivo RDF/XML do disco para a memória, e em seguida o mesmo foi impresso na tela, neste exemplo nenhuma manipulação foi realizada.

Na linha 4 é atribuída uma instância de `Model` para a variável `modelVeiculo`, a instância foi retornada pelo método `createDefaultModel()` que retorna um modelo padrão, este é um dos vários métodos da classe `ModelFactory` que provê métodos para criação de Modelos. Na linha 6, tem uma instrução básica Java que retorna um `InputStream` de um arquivo em disco, e na linha 8, o método `read(arquivo, null)`, faz a leitura do RDF/XML do arquivo passado como parâmetro para o modelo, outro formato comumente usado de parâmetro para este método, seria uma String contendo a url do arquivo, o segundo parâmetro é o URI base do arquivo, como no arquivo passado não se faz tal uso, este foi definido como `null`.

Uma vez que o arquivo esteja populado, na linha 10, a chamada ao método *write(System.out)*, escreve o modelo na tela, o argumento passado a este método pode ser qualquer sub-tipo de *OutputStream*.

Em se tratando de carregar arquivos RDF/XML, a classe *Model* dispõem de outros métodos úteis quando se deseja trabalhar com mais de um arquivo. São estes:

***Model add(Model m) :*** retorna um novo *Model* acrescido das triplas de *Model m* ao *Model* chamador;

***Model union(Model m):*** retorna um novo *Model* com a união do *Model m* ao *Model* chamador, mesma função do método *add()*, com a diferença de que triplas duplicadas são descartadas;

***Model intersection(Model m):*** retorna um novo *Model* com a intersecção do *Model m* com o *Model* chamador, ou seja, apenas triplas existentes em ambos;

***Model diferrence(Model m):*** retorna um novo *Model* com a disjunção do *Model M* com o *Model* chamador, ou seja, apenas triplas que sejam exclusivas do Model chamador;

Figura 25 - Exemplo de criação de um Model

Código Java

```java
public class Teste{
    public static void main (String args[]) {
        // cria um Model vazio
        Model modelVeiculo = ModelFactory.createDefaultModel();
        // abre arquivo veiculo.rdf no disco D: local
        InputStream arquivo = FileManager.get().open("D:\\veiculo.rdf");
        // faz a aleitura do arquivo para o Model
        modelVeiculo.read( arquivo, null );
        // imprime o arquivo na tela
        modelVeiculo.write(System.out);
    }
}
```

Saída

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dv="http://voca.org/veiculo#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://voca.org/modelos#fusca16001982">
    <dv:altura>1485</dv:altura>
    <dv:nPassageiros>5</dv:nPassageiros>
    <rdf:type rdf:resource="http://voca.org/veiculo#VeiculoPasseio"/>
    <dv:cilMotor>4</dv:cilMotor>
    <dv:motor>Boxer de 4 cilindros</dv:motor>
    <dv:combustivel>Gasolina</dv:combustivel>
    <dv:marcahas>4 </dv:marcahas>
    <dv:fabricante rdf:resource="http://vw.com/vw"/>
    <dv:comprimento>4026</dv:comprimento>
    <dv:ccMotor>1600</dv:ccMotor>
    <dv:largura>1540</dv:largura>
    <dv:versao>Sedan 1600</dv:versao>
    <dv:peso>800</dv:peso>
    <dv:modelo>Fusca</dv:modelo>
  </rdf:Description>
</rdf:RDF>
```

Fonte: Próprio Autor

## 4.3 Consultando Um Arquivo RDF

Uma vez que, se tenha uma base RDF populada na memória, em aplicações reais sempre será necessário consultar esta base. Para esta finalidade Jena oferece um motor de pesquisa SPARQL denominado ARQ.

A figura 26, traz o exemplo de uma consulta implementada em Java, onde se deseja buscar o URI, ano e modelo dos veículos de uma base RDF (foi utilizada a base da figura 25).

Antes de comentar o exemplo vale destacar que a documentação do ARQ é separada da documentação do Jena, assim algumas das classes utilizadas no próximo exemplo podem ser encontradas em < http://openjena.org/ARQ/javadoc/index.html >.

Os códigos até a linha 8 são relativos ao carregamento do arquivo em um objeto *Model*, de forma idêntica ao exemplo anterior. Na linha 10 é criada a `queryString` com o código SPARQL a ser executado. Na linha 16 é criado uma instância da classe *Query*, através do método `QueryFactory.create(queryString)`, que recebe como parâmetro a *String* com o código SPARQL, na linha 18 o método `QueryExecutionFactory.create(mquery, modelVeiculo)`, recebe como parâmetro o objeto criado na linha 18, e o objeto *Model* com a base que se deseja consultar, este método retorna um objeto *QueryExecution*, este objeto é quem realmente faz a consulta.

Figura 26 - Exemplo de consulta SPARQL com Jena

Código Java

```
1  public class Teste{
2      public static void main (String args[]) {
3          // cria um Model vazio
4          Model modelVeiculo = ModelFactory.createDefaultModel();
5          // abre arquivo veiculo.rdf no disco D: local
6          InputStream arquivo = FileManager.get().open("D:\\veiculo.rdf");
7          // faz a aleitura do arquivo para o Model
8          modelVeiculo.read( arquivo, null );
9          //cria a queryString com o codigo SPAQL
10         String queryString = "SELECT ?anos ?uri ?modelo " +
11                 "WHERE{" +
12                 "?uri <http://voca.org/veiculo#anoModelo> ?anos. " +
13                 "?uri <http://voca.org/veiculo#modelo> ?modelo " +
14                 "}";
15         // cria um objeto Query
16         Query mquery =  QueryFactory.create (queryString);
17         // cria um objeto Query Execution
18         QueryExecution qexec = QueryExecutionFactory.create(mquery, modelVeiculo);
19         // recolhe o resultado
20         ResultSet results = qexec.execSelect() ;
21         // percorre o resultado e imprime
22         while(results.hasNext()){
23             QuerySolution solucao = results.nextSolution() ;
24             System.out.println(" URI: "+solucao.get("uri"));
25             System.out.println(" Modelo: "+solucao.get("modelo"));
26             System.out.println(" Ano: "+solucao.get("anos"));
27         }
28     }
29 }
```

Saída

```
URI: http://voca.org/modelos#fusca16001982
Modelo: Fusca
Ano: 1982
```

Fonte: Próprio Autor

Na linha 20 é chamado o seu método `execSelect()`, que executa uma consulta SELECT. Vale destacar, que a classe QueryExecution, possui os métodos `execAsk()`, `execConstruct` e `execDescrible` , um para cada forma de consulta SPARQL.

No caso do `execSelect()`, o retorno é um objeto *ResultSet*, que traz o resultado em forma tabular. Este objeto implementa a interface *Iterator* do Java, e possui o método `hasNext()` que permite percorrer os resultados através de uma instrução `while()` como pode ser observado na linha 22. Na linha 23 o método `results.nextSolution()`, tem a função de percorrer cada "linha" do resultado e atribui cada uma para um objeto *QuerySolution*, este objeto por sua vez, oferece métodos para extrair os resultados. Neste exemplo foi utilizado o método `get(String varName)`, onde `varName` é o nome das variáveis declaradas na instrução SELECT em questão.

## 4.4 Inferindo Um Arquivo RDF

Outra funcionalidade importante do Jena é o seu motor de inferência, este permite inferir novos conhecimentos de um modelo baseado em regras OWL (*Web Ontology Language*) e também Jena Rules, na realidade no núcleo do framework, regras OWL são interpretadas como Jena Rules Reynolds(2010), assim por hora a única diferença entre as duas abordagens está na sintaxe. Nesta seção será abordada apenas o seu uso com Jena Rules uma vez que OWL não está sendo abordado neste trabalho.

Jena disponibiliza três formas de inferências:

- ***Foward Chaining:*** raciocínio no sentido fatos → conclusões, onde um conjunto de fatos ocorridos leva a uma conclusão;
- ***Backward Chainig:*** raciocínio no sentido ocorrido → fatos, parte de uma hipótese, que deve ser provada através de fatos;
- ***Hybrid Chainig:*** é a junção de *Backward Chainig* com *Foward Chaining*;

Graças a política de uso de interfaces em sua API, Jena permite a integração de raciocinadores implementados por terceiros, entre os mais populares, pode-se citar o Pallet(<http://clarkparsia.com/pellet>).

### 4.4.1 Jena Rules

Jena Rules é a sintaxe utilizada para descrever regras de propósitos gerais, essas são definidas em linguagem de primeira ordem, muito semelhantes à linguagem Prolog Hebeler(2009), são regras do tipo *if – then*. No caso de documentos RDF o funcionamento básico pode ser definido como: dada uma condição, se verdadeira, uma nova afirmação (tripla) é gerada.

As condições em uma regra são estabelecidas por funções primitivas, que fazem operações dos tipos:

- Teste de tipo de um objeto
- Igualdade
- Operações matemáticas
- Operações com Strings (regex, concatenação)
- Operações com datas
- Manipulação de Listas

No apêndice E pode ser encontrada uma tabela com as várias funções e suas respectivas descrições. A documentação da Jena Rules pode ser encontrada em <http://jena.sourceforge.net/inference/index.htm>.

A figura 27 traz como exemplo uma regra, onde se deseja adicionar a propriedade `dev:pesoCav`, que representa o peso do veículo dividido por sua potencia em cavalos vapor, esta propriedade não foi definida no modelo a ser inferido, ou seja, até que a regra seja inferida a propriedade não existe.

Na linha 1 são declarados os prefixos (*namespaces*) utilizados na regras, o conceito de *namespaces* em regras é o mesmo abordado anteriormente em RDF, RDF-Schema e SPARQL, na linha 2 é aberta uma regra pelo caractere "[", acompanhado pelo nome da regra e ":", cada regra é delimitada por "[" e "]", e um mesmo arquivo pode conter várias regras, na linha 3 a instrução `(?carro dev:cavalos ?cav)`, forma uma tripla e busca por recursos que tenham a propriedade `dev:cavalos`, e quando encontra atribui o recurso na variável `?carro` e o seu valor na variável `?cav`, de forma praticamente idêntica ao SPARQL, e como pode ser notado variáveis em regras também são precedidas com o operador "?". A linha 3 tem a mesma função da linha 2, apenas recolher valores para as suas variáveis e desta forma procuram por veículos que tenha a quantidade de cavalos e peso definidos, na linha 5 a primitiva `quotient(?peso, ?cav, ?cp)` atribui na variável `?cp` o valor resultante da operação `?peso/?cav`. Na linha 6 a instrução `->`, indica que será realizada a operação de inferência *Foward Chaining*, e esta tendo sucesso, a instrução da linha 7 é executada, que por sua vez adiciona o propriedade `dev:pesoCav` ao recurso atribuído na variável `?carro` com o valor armazenado em `?cp`.

Figura 27 - Exemplo de uma regra

Jena Rules

```
1  @prefix dev: <http://voca.org/veiculo#>.
2  [CavoloPeso:
3  ( ?carro dev:cavalos ?cav )
4  ( ?carro dev:peso ?peso )
5  quotient(?peso, ?cav, ?cp)
6  ->
7  ( ?carro dev:pesoCav ?cp )
8  ]
```

Fonte: Próprio Autor

A figura 28 traz a impressão de dois modelos, o primeiro é o modelo sem inferência, contendo a descrição de dois veículos, o segundo modelo é o primeiro modelo inferido pela regra da figura 27.

Como pode ser observado nas setas 2 e 3 foi criada a propriedade `dev:pesoCav` para cada um dos veículos, com  o seu respectivo valor, e de acordo com o calculo realizado na regra. Vale destacar que quando se utiliza primitivas Jena Rules é importante verificar a compatibilidade dos tipos que se deseja realizar as operações, por exemplo, tentar dividir um *Integer* por um *String*, mesmo que este represente um número pode fazer com que a regra simplesmente não traga resultado algum gerando um erro difícil de ser diagnosticado.

A seta 1 da figura 28 indica outra instrução que foi adicionada após a inferência da regra, esta indica o local físico da regra e o tipo de inferência aplicada.

O código da figura 29, traz um exemplo de aplicação de inferência utilizando Jena. Este código foi o responsável por gerar a saída da figura 28. Até a linha 8 como abordado anteriormente apenas foi carregado um arquivo RDF em um objeto *Model*, na linha 11 este é impresso na tela, na sua forma original de entrada, na linha 14 é atribuído o local físico do arquivo que contém as regras, este arquivo é o mesmo ilustrado na figura 27. Nas linhas 16,18 e 20 é criado um recurso que define as configurações de inferência informando o local do arquivo de regras e o tipo de motor de inferência que devera ser instanciado, que neste caso é `hybrid`. Na linha 22 é criado um objeto *Reasoner* que se trata do motor de inferência do Jena, e este recebe como parâmetro o recurso que contém as configurações. Na linha 25 é criado um novo Modelo, mas do tipo *InfModel*, na sua criação são passados como parâmetros o modelo original, e o objeto *Reasoner* criado na linha 22, este modelo ao ser criado já é inferido como pode ser observado no segundo modelo impresso na figura 28, que além das novas triplas inferidas (setas 2 e 3) o recurso `configuração` também foi adicionado no modelo (seta 1).

Figura 28 - Exemplo de um documento inferido

Saída

```
############################ Model original
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dv="http://voca.org/veiculo#"
    xmlns:j.0="http://jena.hpl.hp.com/2003/RuleReasoner#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
  <rdf:Description rdf:about="http://voca.org/modelos#unomille10002005">
    <dv:peso rdf:datatype="http://www.w3.org/2001/XMLSchema#int">900</dv:peso>
    <dv:anoModelo>2005</dv:anoModelo>
    <dv:cavalos rdf:datatype="http://www.w3.org/2001/XMLSchema#int">40</dv:cavalos>
    <dv:ccMotor>1000</dv:ccMotor>
    <dv:modelo>Uno Mille</dv:modelo>
    <rdf:type rdf:resource="http://voca.org/veiculo#VeiculoPasseio"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://voca.org/modelos#fusca16001982">
    <dv:peso rdf:datatype="http://www.w3.org/2001/XMLSchema#int">800</dv:peso>
    <dv:anoModelo>1982</dv:anoModelo>
    <dv:cavalos rdf:datatype="http://www.w3.org/2001/XMLSchema#int">40</dv:cavalos>
    <dv:ccMotor>1600</dv:ccMotor>
    <dv:modelo>Fusca</dv:modelo>
    <rdf:type rdf:resource="http://voca.org/veiculo#VeiculoPasseio"/>
  </rdf:Description>
</rdf:RDF>
############################ Model inferido
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dv="http://voca.org/veiculo#"
    xmlns:j.0="http://jena.hpl.hp.com/2003/RuleReasoner#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
1 <rdf:Description rdf:nodeID="A0">
    <j.0:ruleSet>D:\regras.txt</j.0:ruleSet>
    <j.0:ruleMode>hybrid</j.0:ruleMode>
  </rdf:Description>
  <rdf:Description rdf:about="http://voca.org/modelos#unomille10002005">
2   <dv:pesoCav rdf:datatype="http://www.w3.org/2001/XMLSchema#int">22</dv:pesoCav>
    <dv:peso rdf:datatype="http://www.w3.org/2001/XMLSchema#int">900</dv:peso>
    <dv:anoModelo>2005</dv:anoModelo>
    <dv:cavalos rdf:datatype="http://www.w3.org/2001/XMLSchema#int">40</dv:cavalos>
    <dv:ccMotor>1000</dv:ccMotor>
    <dv:modelo>Uno Mille</dv:modelo>
    <rdf:type rdf:resource="http://voca.org/veiculo#VeiculoPasseio"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://voca.org/modelos#fusca16001982">
3   <dv:pesoCav rdf:datatype="http://www.w3.org/2001/XMLSchema#int">20</dv:pesoCav>
    <dv:peso rdf:datatype="http://www.w3.org/2001/XMLSchema#int">800</dv:peso>
    <dv:anoModelo>1982</dv:anoModelo>
    <dv:cavalos rdf:datatype="http://www.w3.org/2001/XMLSchema#int">40</dv:cavalos>
    <dv:ccMotor>1600</dv:ccMotor>
    <dv:modelo>Fusca</dv:modelo>
    <rdf:type rdf:resource="http://voca.org/veiculo#VeiculoPasseio"/>
  </rdf:Description>
</rdf:RDF>
```

Fonte: Próprio Autor

Figura 29 - Exemplo de código que realiza inferência

Código Java

```java
public class Teste{
    public static void main (String args[]) {
        // cria um Model vazio
        Model modelVeiculo = ModelFactory.createDefaultModel();
        // abre arquivo veiculo.rdf no disco D: local
        InputStream arquivo = FileManager.get().open("D:\\veiculo.rdf");
        // faz a aleitura do arquivo para o Model
        modelVeiculo.read( arquivo, null );
        //cria a queryString com o codigo SPAQL
        System.out.println("############################ " +"Model original");
        modelVeiculo.write(System.out);
        System.out.println("############################ " +"Model inferido ");
        // define local do arqruivo com as regras
        String regras = "D:\\regras.txt";
        //cria um novo recurso que vai configurar o Reasoner
        Resource  configuracao = modelVeiculo.createResource();
        //define o tipo de inferencia
        configuracao.addProperty(ReasonerVocabulary.PROPruleMode, "hybrid");
        //passa o local das regras
        configuracao.addProperty(ReasonerVocabulary.PROPruleSet, regras);
        // cria a instancia de Reasoner
        Reasoner reasoner =
            GenericRuleReasonerFactory.theInstance().create(configuracao);
        // cria um novo modelo,o mesmo de modelVeiculo mas inferido
        InfModel infModel = ModelFactory.createInfModel(reasoner, modelVeiculo);
        // imprime o novo modelo inferido na tela
        infModel.write(System.out);
    }
}
```

Fonte: Próprio Autor

No próximo capítulo é apresentado o APC (Assistente Pessoal de Compras), que é uma implementação de consumidor de dados semântico e foi implementado utilizando o framework Jena.

# CAPÍTULO 5 - APC (ASSISTENTE PESSOAL DE COMPRAS)

Neste capítulo, é apresentado o funcionamento do APC (Assistente Pessoal de Compras), este desenvolvido em Java, utilizando o framework Jena, com o objetivo de ser uma demonstração prática de todas as teorias apresentadas no trabalho.

## 5.1 Especificações do APC

O APC, na sua função de assistente pessoal de compras da WS tem por objetivo receber o URI de um produto e realizar as seguintes tarefas:

- Buscar dados do produto junto ao seu fabricante;
- Encontrar produtos similares;
- Buscar avaliações do produto feitas por amigos do usuário;
- Buscar lojas que comercializem o produto;

Para realizar as tarefas listadas acima o APC deverá realizar consultas a dados RDF fornecidos por terceiros. Estes dados são provenientes de quatro categorias de fornecedores de conteúdo:

- Redes sociais;
- Fabricantes;
- Avaliação de produtos;
- Comércio eletrônico;

## 5.2 Simulação de um Ambiente Web Semântico

Para simular um ambiente web semântico foi criada uma estrutura como ilustrada pela figura 30, onde o APC tem acesso aos dados semânticos de vários fornecedores e cada categoria de fornecedor utiliza um vocabulário especifico para descrever a sua base dados, como mostra a tabela de fornecedores da figura 30.

Neste teste não será abordada à forma como o APC colhe as bases RDF. Nesta simulação os dados RDF já se encontram dentro de uma pasta no diretório raiz da aplicação. E é importante destacar que não existe nenhuma restrição quanto à forma que um agente obtém dados semânticos, sendo que esta operação pode ocorrer de inúmeras maneiras.

Figura 30 - Ilustração de um ambiente Web Semântico

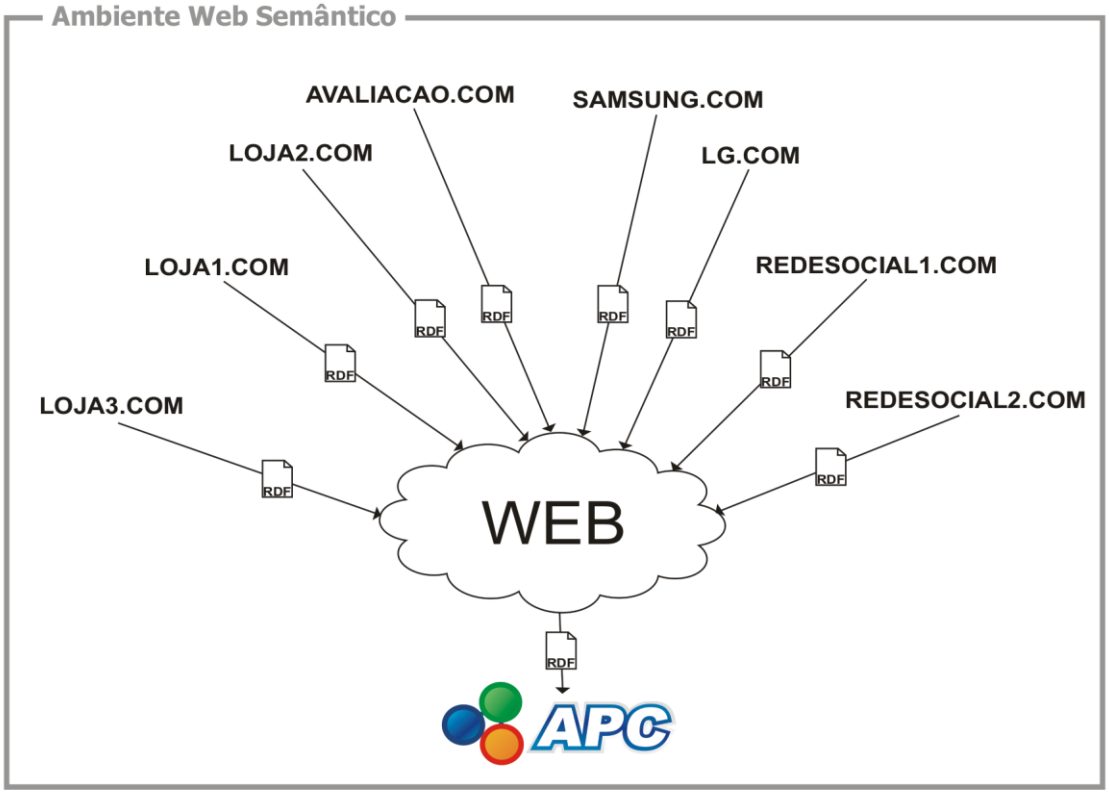



Tabela de Fornecedores

| Fornecedor | Categoria | Vocabulário | Arquivo de dados |
|---|---|---|---|
| REDESOCIAL1.COM | Rede Social | foaf | redesocial1.rdf |
| REDESOCIAL2.COM | Rede Social | foaf | redesocial2.rdf |
| LOJA1.COM | Comercio eletrônico | lojaproduto | loja1.rdf |
| LOJA2.COM | Comercio eletrônico | lojaproduto | loja2.rdf |
| LOJA3.COM | Comercio eletrônico | lojaproduto | loja3.rdf |
| SAMSUNG.COM | Fabricante | tv | samsungtv.rdf |
| LG.COM | Fabricante | tv | lgtv.rdf |
| AVALIACAO.COM | Avaliação de produtos | temproduto | avaliacao1.rdf |

Fonte: Próprio Autor

Neste ambiente simulado, nenhuma aplicação fornecedora foi implementada, o que realmente foi criado, foram os vocabulários e bases RDF, para simular os dados semânticos que aplicações reais destas categorias, poderiam facilmente extrair de suas bases de dados já existentes.

## 5.3 Desenvolvimento do APC

Para desenvolver o APC, primeiramente foram criados os vocabulários e bases RDF para cada uma das categorias de fornecedores de conteúdo.

Os fornecedores da categoria “Rede Social” (REDESOCIAL1.COM e REDESOCIAL2.COM), fornecem uma base RDF cada uma, onde são descritos dados pessoais (nome, e-mail, foto, etc..) e relação de amizade entre alguns usuários fictícios. As bases de dados podem ser encontradas no apêndice F.

O fornecedor “AVALIACAO.COM” da categoria “Avaliação de produtos”, faz o papel de um site onde usuários fazem avaliações de produtos, e a base de dados fornecida por ele contém o URI do usuário, do produto avaliado, a nota que o usuário atribuiu ao produto e um pequeno texto, com a opinião do usuário sobre o produto. A base foi criada utilizando o vocabulário “temproduto”. A base e o vocabulário podem ser encontrados nos apêndices G e H respectivamente.

LOJA1.COM, LOJA2.COM e LOJA3.COM fornecem os seus respectivos estoques, onde são descritos o URI, link e preço de cada produto, as bases de cada loja e o vocabulário utilizado podem ser encontrados nos apêndices I e J respectivamente.

Fornecedores da categoria “Fabricante” (SAMSUNG.COM e LG.COM), fornecem uma base onde são descritos os produtos que cada empresa produz. Neste teste, o único tipo de produto existente são aparelhos de TV. As bases e vocabulário utilizado podem ser encontrados nos apêndices K e L.

O funcionamento básico do APC pode ser definido como um grupo de classes Java implementadas utilizando o framework Jena (capítulo 4). Mais especificamente, foi criada uma classe para cada tarefa, cada classe popula as bases RDF fornecidas pelos fornecedores e realiza consultas SPARQL, de acordo com a sua tarefa.

Para apresentar o resultado da consulta para o usuário foi criada uma interface gráfica em HTML (*Hypertext Markup Language)* que faz requisições Ajax às classes Java. Por fugir do escopo deste trabalho, não será detalhado o funcionamento da interface gráfica, será abordado apenas o funcionamento das classes Java. Nas seções seguintes são detalhadas a implementação e funcionamento de cada uma das tarefas especificadas na seção 5.1.

### 5.3.1 Buscar Dados do Produto Junto ao seu Fabricante

Esta tarefa tem início quando o usuário inicia o APC informando o URI de um produto, o APC então faz uma requisição Ajax para a classe *BuscaProduto*, enviando o URI como parâmetro.

Como pode ser observado no código da figura 31, o método *pesquisarProduto* popula as bases RDF fornecidas pelos fabricantes e o vocabulário que foi utilizado nestas

bases (tv). Após ter todas as bases unidas, é realizada uma consulta SPARQL onde são retornados o *label* da superclasse do produto, o *label* do produto e o endereço de sua foto.

Figura 31 - Classe BuscaProduto

Código Java

```
1  public class BuscaProduto{
2      public String pesquisarProduto(String uriProduto){
3              ResourceBundle bundle = ResourceBundle.getBundle("config");
4              Model vocabulariotv = ModelFactory.createDefaultModel();
5              Model fabricante1 = ModelFactory.createDefaultModel();
6              Model fabricante2 = ModelFactory.createDefaultModel();
7              Model universal = ModelFactory.createDefaultModel();
8              InputStream arquivo = FileManager.get().open(bundle.getString("path_rdf_tv"));
9              vocabulariotv.read( arquivo, null );
10             arquivo = FileManager.get().open(bundle.getString("path_rdf_samsungtv"));
11             fabricante1.read( arquivo, null );
12             arquivo = FileManager.get().open(bundle.getString("path_rdf_lgtv"));
13             fabricante2.read( arquivo, null );
14             universal = vocabulariotv.union(fabricante1.union(fabricante2));
15             Resource  configuracao = universal.createResource();
16             configuracao.addProperty(ReasonerVocabulary.PROPruleMode, "hybrid");
17             configuracao.addProperty(ReasonerVocabulary.PROPruleSet, regras);
18             Reasoner reasoner =
19             GenericRuleReasonerFactory.theInstance().create(configuracao);
20             InfModel infModel = ModelFactory.createInfModel(reasoner, universal);
21             String queryString =
22                     "PREFIX tv: <http://voca.org/tv#> " +
23                     "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
24                     "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
25                     "SELECT * " +
26                     "WHERE{" +
27                     " <"+uriProduto+">  rdfs:label ?label . " +
28                     " <"+uriProduto+">  rdf:type ?classe . " +
29                     " <"+uriProduto+">  tv:foto ?foto . " +
30                     " ?classe  rdfs:label ?labelclasse . " +
31                     "}  ";
32             Query mquery =  QueryFactory.create (queryString);
33             QueryExecution qexec = QueryExecutionFactory.create(mquery, infModel);
34             ResultSet results = qexec.execSelect() ;
```

Fonte: Próprio Autor

Na figura 32 é ilustrado o retorno do método na interface gráfica, onde também são geradas duas perguntas para o usuário.

A primeira pergunta “Deseja que eu procure mais informações sobre este produto” é um complemento da tarefa já iniciada (Buscar dados do produto junto ao seu fabricante) e que faz uma nova requisição Ajax desta vez para a classe *BuscaDetalhes*.

A segunda pergunta “Ou prefere que eu procure mais produtos iguais a este” tem o objetivo de descobrir a superclasse do produto e listar todos os produtos pertencentes a mesma.

Figura 32 - Retorno do método BuscaProduto.pesquisarProduto()


Fonte: Próprio Autor

Dando continuidade à tarefa “Buscar dados do produto junto ao seu fabricante”, será simulado um clique positivo na primeira pergunta, gerando assim uma requisição Ajax à classe *BuscaDetalhes*. Como pode ser observado na figura 33 a classe *BuscaDetalhes* popula as bases RDF fornecidas pelos fabricantes e realiza uma consulta SPARQL que retorna o *label* e valor de todas as propriedades definidas para o produto em questão.

Nesta classe é importante destacar que ela faz o uso de regras (Jena Rules), que foram criadas para “ensinar” o APC, que produtos que tenham a propriedade “Conexão de Entrada” com os valores “HDMI” ou “VGA” podem ser conectados a um PC. Assim o APC adiciona a propriedade “Conexão com PC” ao produto. Também foi ensinado ao APC

distinguir a qualidade da resolução da tela do produto, atribuindo a propriedade “Qualidade da Resolução”, os valores “HD” ou “Full HD”, de acordo com o intervalo do valor da propriedade “Resolução Vertical”.

O arquivo com as regras utilizadas para ensinar o APC, pode ser encontrado no apêndice M.

Figura 33 - Classe BuscaDetalhes

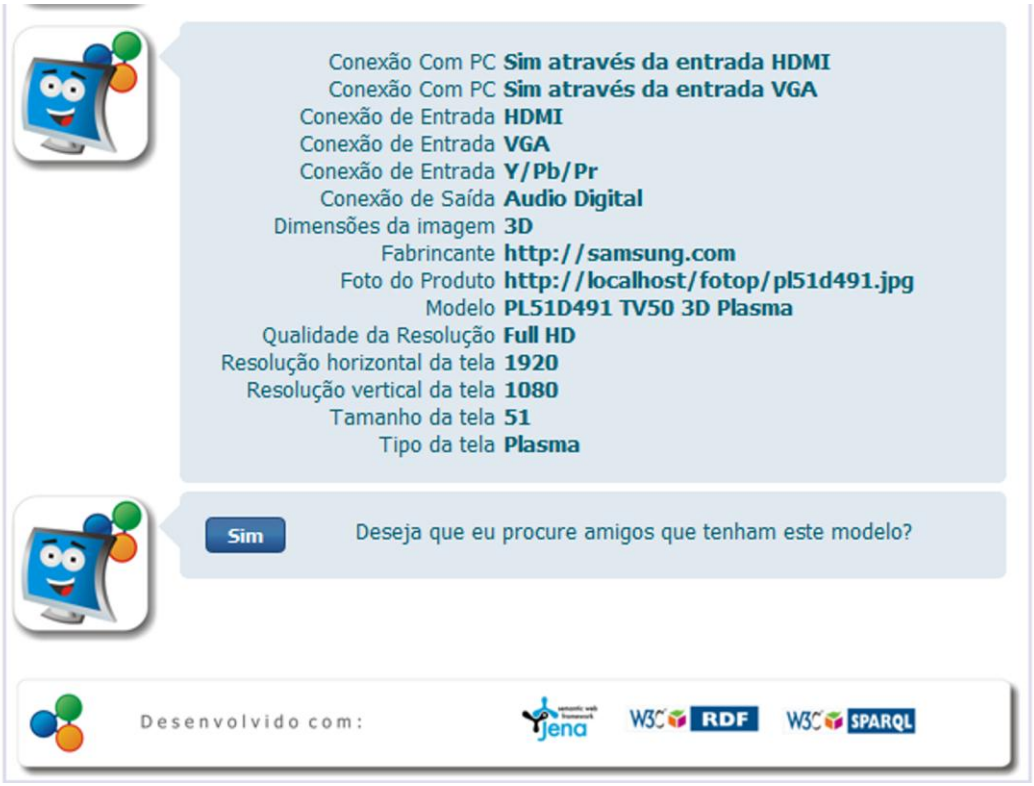
Código Java

```
1  public class BuscaDetalhes{
2      public String pesquisarDetalhes(String uriUsuario, String uriProduto){
3          ResourceBundle bundle = ResourceBundle.getBundle("config");
4          Model vocabulariotv = ModelFactory.createDefaultModel();
5          Model fabricante1 = ModelFactory.createDefaultModel();
6          Model fabricante2 = ModelFactory.createDefaultModel();
7          Model universal = ModelFactory.createDefaultModel();
8          InputStream arquivo = FileManager.get().open(bundle.getString("path_rdf_tv"));
9          vocabulariotv.read( arquivo, null );
10         arquivo = FileManager.get().open(bundle.getString("path_rdf_samsungtv"));
11         fabricante1.read( arquivo, null );
12         arquivo = FileManager.get().open(bundle.getString("path_rdf_lgtv"));
13         fabricante2.read( arquivo, null );
14         universal = vocabulariotv.union(fabricante1.union(fabricante2));
15         String regras = bundle.getString("path_rdf_regrastv");
16         Resource  configuracao = universal.createResource();
17         configuracao.addProperty(ReasonerVocabulary.PROPruleMode, "hybrid");
18         configuracao.addProperty(ReasonerVocabulary.PROPruleSet, regras);
19         Reasoner reasoner =
20             GenericRuleReasonerFactory.theInstance().create(configuracao);
21         InfModel infModel = ModelFactory.createInfModel(reasoner, universal);
22         String queryString =
23             "PREFIX tv: <http://voca.org/tv#> " +
24             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
25             "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
26             "SELECT * " +
27             "WHERE{" +
28             " <"+uriProduto+">  ?prop ?valor . " +
29             " ?prop  rdfs:label ?label ." +
30             " ?prop rdfs:range ?tipo "+
31             "} ORDER BY ?label ";
32         Query mquery =  QueryFactory.create (queryString);
33         QueryExecution qexec = QueryExecutionFactory.create(mquery, infModel);
34         ResultSet results = qexec.execSelect() ;
```

Fonte: Próprio Autor

A figura 34, ilustra o retorno da classe *BuscaDetalhes* onde são retornados o *label* e valor de todas propriedades do produto.

Figura 34 - Retorno do método BuscaDetalhes.pesquisarDetalhes()




Fonte: Próprio Autor

### 5.3.2 Encontrar Produtos Similares

A tarefa “Encontrar produtos similares”, tem início quando o usuário responde positivamente à segunda pergunta “Ou prefere que eu procure mais produtos iguais a este “, após a resposta do usuário, uma requisição Ajax é enviada para a classe *BuscaCategoria* (figura 35), passando como parâmetro o URI do produto.

A classe popula os dados RDF e realiza uma pesquisa SPARQL para descobrir qual é a superclasse do produto, e retorna todos os produtos da base que sejam, também, subclasse desta classe, (o retorno da classe é ilustrado na figura 36).

Figura 35 - Classe BuscaCategoria

Código Java

```
1  public class BuscaCategoria{
2      public String pesquisarCategoria(String uriProduto){
3              ResourceBundle bundle = ResourceBundle.getBundle("config");
4              Model vocabulariotv = ModelFactory.createDefaultModel();
5              Model fabricante1 = ModelFactory.createDefaultModel();
6              Model fabricante2 = ModelFactory.createDefaultModel();
7              Model universal = ModelFactory.createDefaultModel();
8              InputStream arquivo = FileManager.get().open(bundle.getString("path_rdf_tv"));
9              vocabulariotv.read( arquivo, null );
10             arquivo = FileManager.get().open(bundle.getString("path_rdf_samsungtv"));
11             fabricante1.read( arquivo, null );
12             arquivo = FileManager.get().open(bundle.getString("path_rdf_lgtv"));
13             fabricante2.read( arquivo, null );
14             universal = vocabulariotv.union(fabricante1.union(fabricante2));
15             String queryString =
16                 "PREFIX tv: <http://voca.org/tv#> " +
17                 "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
18                 "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
19                 "SELECT * " +
20                 "WHERE{" +
21                 " <"+uriProduto+">  rdf:type ?classe . " +
22                 " ?uri  rdf:type ?classe . " +
23                 " ?uri  rdfs:label ?label . " +
24                 " ?uri  tv:foto ?foto . " +
25                 "}  ";
26         Query mquery =  QueryFactory.create (queryString);
27         QueryExecution qexec = QueryExecutionFactory.create(mquery, universal);
28         ResultSet results = qexec.execSelect() ;
```

Fonte: Próprio Autor

Figura 36 - Retorno do método BuscaCategoria.pesquisarCategoria()


Fonte: Próprio Autor

### 5.3.3 Buscar Avaliações do Produto Feitas por Amigos do Usuário

A tarefa “Buscar avaliações do produto feitas por amigos do usuário” tem início quando o usuário responde de forma positiva à pergunta “Deseja que eu procure amigos que tenham este modelo?”. A primeira operação é o envio de uma solicitação Ajax para a classe *BuscaAmigos* (figura 37), onde o seu método *pesquisaramigos* recebe por parâmetro o URI do produto e URI do usuário. O seu método *pesquisaramigos* popula as bases RDF fornecidas pelas Redes Sociais e pelo site de avaliação, em seguida faz o cruzamento de URI para descobrir quais usuários conhecem o usuário principal, nas bases das redes sociais. E depois procura por avaliações, onde o produto seja o informado por parâmetro e os usuários sejam os conhecidos pelo usuário principal.

Figura 37 - Classe BuscaAmigos

Código Java

```
public class BuscaAmigos{
    public String pesquisarAmigos(String uriUsuario, String uriProduto){
        ResourceBundle bundle = ResourceBundle.getBundle("config");
        Model redes1 = ModelFactory.createDefaultModel();
        Model redes2 = ModelFactory.createDefaultModel();
        Model avaliacao = ModelFactory.createDefaultModel();
        Model redesuniversal = ModelFactory.createDefaultModel();
        InputStream arquivo = FileManager.get().open(bundle.getString("path_rdf_redesocial1"));
        redes1.read( arquivo, null );
        arquivo = FileManager.get().open(bundle.getString("path_rdf_redesocial2"));
        redes2.read(arquivo, null);
        redesuniversal = redes1.union(redes2);
        arquivo = FileManager.get().open(bundle.getString("path_rdf_avaliacao1"));
        avaliacao.read(arquivo, null);
        redesuniversal = redesuniversal.union(avaliacao);
        String queryString =
            "PREFIX tp: <http://voca.org/temproduto#> " +
            "PREFIX tv: <http://voca.org/tv#> " +
            "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
            "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
            "PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
            "SELECT * " +
            "WHERE{" +
            "  <"+uriUsuario+">  foaf:knows ?uriamigo . " +
            " ?uriava tp:pessoa ?uriamigo . " +
            " ?uriamigo foaf:name ?nome . " +
            " ?uriamigo foaf:depiction ?foto . " +
            " ?uriava tp:produto <"+uriProduto+"> ." +
            " ?uriava tp:nota ?nota ." +
            " ?uriava tp:opniao ?opniao " +
            "}";
        Query mquery =  QueryFactory.create (queryString);
        QueryExecution qexec = QueryExecutionFactory.create(mquery, redesuniversal);
        ResultSet results = qexec.execSelect();
```

Fonte: Próprio Autor

O retorno do método é ilustrado na figura 38, onde são listados os amigos que se enquadram nos parâmetros da busca. De cada amigo é retornado o nome, nota que atribuiu ao

produto, um texto com sua opinião sobre o produto e o endereço da sua foto (que o APC exibe para o usuário principal).

Figura 38 - Retorno do método BuscaAmigos.pesquisarAmigos()


Fonte: Próprio Autor

### 5.3.4 Buscar Lojas que Comercializem o Produto

A tarefa “Buscar lojas que comercializem o produto” é iniciada quando o usuário responde a pergunta “Deseja que eu procure este modelo nas lojas” (ilustrada na figura 38).

A classe *BuscaLojas* é a responsável por esta tarefa e possui o método *pesquisarLojas* que recebe o URI do produto como parâmetro, realizando assim uma nova consulta SPARQL nas bases fornecidas pelos fornecedores da categoria comércio eletrônico. O código da classe é apresentado na figura 39.

O retorno desta tarefa é ilustrado na figura 40, onde são retornados o preço, e link do produto na loja, o nome, slogan e endereço do logotipo da loja, sendo este último exibido o usuário.

Figura 39 - Classe BuscaLojas

Código Java

```
1  public class BuscaLojas{
2      public String pesquisarLojas(String uriProduto){
3          ResourceBundle bundle = ResourceBundle.getBundle("config");
4          Model loja1 = ModelFactory.createDefaultModel();
5          Model loja2 = ModelFactory.createDefaultModel();
6          Model loja3 = ModelFactory.createDefaultModel();
7          Model universal = ModelFactory.createDefaultModel();
8          InputStream arquivo = FileManager.get().open(bundle.getString("path_rdf_loja1"));
9          loja1.read( arquivo, null );
10         arquivo = FileManager.get().open(bundle.getString("path_rdf_loja2"));
11         loja3.read( arquivo, null );
12         arquivo = FileManager.get().open(bundle.getString("path_rdf_loja3"));
13         loja3.read( arquivo, null );
14         universal = loja1.union(loja2.union(loja3));
15         String queryString =
16             "PREFIX lp: <http://voca.org/lojaproduto#> " +
17             "PREFIX tp: <http://voca.org/temproduto#> " +
18             "PREFIX tv: <http://voca.org/tv#> " +
19             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
20             "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
21             "PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
22             "SELECT * " +
23             "WHERE{" +
24             " ?uripl  lp:produto <"+uriProduto+"> ." +
25             " ?uripl  lp:preco ?preco ." +
26             " ?uripl  lp:link ?link ." +
27             " ?uripl  lp:loja ?uriloja ." +
28             " ?uriloja lp:nomeLoja ?nomeloja ." +
29             " ?uriloja rdfs:label ?label ." +
30             " ?uriloja lp:logoLoja ?logo " +
31             "} ORDER BY ?preco";
32         Query mquery =  QueryFactory.create (queryString);
33         QueryExecution qexec = QueryExecutionFactory.create(mquery,universal);
34         ResultSet results = qexec.execSelect();
```

Fonte: Próprio Autor

Figura 40 - Retorno do método BuscaLojas.pesquisarLojas()


Fonte: Próprio Autor

Com a lista retornada, o usuário pode escolher qual loja deseja visitar e finalizar a compra, assim encerrando o caso de teste. No apêndice N pode ser encontrado o “diálogo” completo do caso de teste realizado.

É importante deixar claro que o APC é apenas um exemplo básico, e não uma proposta real para um Assistente Pessoal de Compra, tendo unicamente o objetivo de servir como uma demonstração prática de todas as teorias apresentadas no trabalho.

A principal diferença do APC para aplicações similares existentes na Web atual é que todas as informações que ele retornou para o usuário tiveram origem em bases RDF de diversas fontes e não de uma única base de dados como geralmente ocorre em aplicações da Web atual.

Vale destacar que as únicas exigências para as bases RDF consultadas pelo APC foram: a utilização de vocabulários específicos e o uso de URI para identificar cada recurso descrito, sendo estes os requisitos básicos de uma aplicação da WS.

# CONCLUSÕES

Durante os estudos realizados ficou claro que a Web Semântica traz inúmeros benefícios principalmente nas buscas como foi exemplificado na seção 3.1.

Um dos benefícios é a possibilidade de realizar tarefas com o uso de agentes, tais como o APC (Assistente Pessoal de Compras), sendo o único requisito na implementação, o conhecimento prévio do vocabulário, onde se atribui um significado para cada termo e consequentemente o agente se torna apto para consultar qualquer base definida, no mesmo vocabulário.

Foi observado que, apesar de ser possível criar uma aplicação da Web Semântica como foi exemplificado neste trabalho, existem algumas dificuldades que impedem a sua popularização. A principal seria a conversão de documentos HTML (o formato mais popular atualmente na Web) para documentos RDF, tornando este um tema de grande importância para trabalhos futuros.

Quanto aos objetivos inicias, propostos para elaboração do presente trabalho, todos foram alcançados.

O primeiro foi a elaboração de um documento introdutório sobre Web Semântica, e suas principais tecnologias.

O segundo objetivo, também foi alcançado através da implementação e testes realizados no APC, onde foi possível comprovar o funcionamento de uma aplicação moldada nos padrões propostos pela Web Semântica, realizando consultas e cruzando dados de diferentes origens.

# REFERÊNCIAS


BERNERS-LEE, T.; CAILLIAU, R. **WorldWideWeb: Proposal for a HyperText Project**. 1990. Disponível em: <http://www.w3.org/Proposal.html/ >. Acessado em: 20 fev. 2011.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. **The semantic web**: a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American. 2001. Disponível em: <http://www.scientificamerican.com/article.cfm?id=the-semantic-web>. Acesso em : 10 jan. 2011.

BIRON, P. V.; MALHOTRA A. **XML Schema Part 2:** Datatypes Second Edition. 2004. Disponível em:< http://www.w3.org/TR/xmlschema-2/>. Acesso em: 14 mar. 2011.

BREITMAN, K. K. **Web semântica:** a internet do futuro. Rio de Janeiro : LTC, 2005.

CLARK, K. G.; FEIGENBAUM, L.; TORRES, E. **SPARQL Protocol for RDF**. 2008. Disponível em: < http://www.w3.org/TR/rdf-sparql-protocol/ >. Acesso em: 20 jun. 2011.

HEBELER, J.;FISHER M.;BLACE R.; PEREZ-LOPEZ, A. **Semantic Web Programming**. United States of America: Wiley Publishing, 2009.

MANOLA, F.; MILLER, E.; MCBRIDE, B. **RDF Primer**. 2004. Disponível em: <http://www.w3.org/TR/rdf-primer/>. Acesso em: 16 dez. 2010.

MCBRIDE, B.; BOOTHY, D.; DOLLIN, C. **An Introduction to RDF and the Jena RDF API**. 2010. Disponível em :< http://jena.sourceforge.net/tutorial/RDF_API/ >. Acesso em: 10 jul. 2011.

O'REILLY, T. **Web 2.0 Compact Definition:** Trying Again. 2006. Disponível em: <http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html />. Acesso em: 15 mar. 2011.

POLLOCK, J. **Web Semântica Para Leigos** .Rio de Janeiro: Alta Books, 2010.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. 2008. Disponível em :< http://www.w3.org/TR/rdf-sparql-query/ >. Acesso em: 22 jun. 2011.

RAY, E. T. **Learning XML.** United States of America:O'Reilly Media, 2001.

REYNOLDS, D. **Jena 2 Inference Support**. 2010. Disponível em: <http://jena.sourceforge.net/inference/index.html >. Acesso em: 10 jul. 2011.

SANTAREM SEGUNDO, J. E. **Representação Iterativa:** um modelo para Repositórios Digitais. São Paulo: Faculdade de Filosofia e Ciências, Universidade Estadual Paulista, 2010.

STOUT, R. **Dominando a World Wide Web**. São Paulo: Makron Books, 1997.

WALL, David. **Usando a World Wide Web**. Rio de Janeiro: Campus, 1997.

# APÊNDICE A - Itens Dos Vocabulários RDF e RDF-Schema

| Nome da classe | Descrição |
|---|---|
| rdfs: Resource | O recurso de classe, tudo. |
| rdfs: Literal | A classe de valores literais. |
| rdf: XMLLiteral | A classe de valores literais XML. |
| rdfs: Class | A classe de classes. |
| rdf: Property | A classe de propriedades RDF. |
| rdfs: Datatype | A classe de tipos de dados RDF. |
| rdf: Statement | A classe de declarações RDF. |
| rdf: Bag | A classe conteiners (não ordenada). |
| rdf: Seq | A classe conteiners (ordenada). |
| rdf: Alt | A classe conteiners (alternativa). |
| rdfs: Container | A classe de RDF containers. |
| rdfs: ContainerMembershipProperty | A classe de membros de conteiners. |
| rdf: Lista | A classe de listas RDF. |

| Nome da propriedade | Descrição | domain | range |
|---|---|---|---|
| rdf: type | Indica a classe de uma instância. | rdfs: Resource | rdfs: Class |
| rdfs: subClassOf | O indica a super classe de uma subclasse . | rdfs: Class | rdfs: Class |
| rdfs: subPropertyOf | Indica a super classe de uma propriedade. | rdf: Property | rdf: Property |
| rdfs: domain | O domínio de uma propriedade( classes ). | rdf: Property | rdfs: Class |
| rdfs: range | O alcance(possíveis valores) de uma propriedade. | rdf: Property | rdfs: Class |
| rdfs: label | Etiqueta em linguagem natural. | rdfs: Resource | rdfs: Literal |
| rdfs: comment | Comentário . | rdfs: Resource | rdfs: Literal |
| rdfs: member | Um membro do recurso. | rdfs: Resource | rdfs: Resource |
| rdf: first | O primeiro item em uma lista RDF. | rdf: List | rdfs: Resource |
| rdf: rest | O item após o primeiro em uma lista RDF. | rdf: List | rdf: List |
| rdfs: seeAlso | Mais informações sobre o recurso. | rdfs: Resource | rdfs: Resource |
| rdfs: isDefinedBy | A definição do recurso assunto. | rdfs: Resource | rdfs: Resource |
| rdf: value | Propriedade idiomática para valores estruturados. | rdfs: Resource | rdfs: Resource |
| rdf: subject | O recurso de uma declaração RDF. | rdf: Statement | rdfs: Resource |
| rdf: predicate | A propriedade de uma declaração RDF. | rdf: Statement | rdfs: Resource |
| rdf: object | O valor de uma declaração RDF. | rdf: Statement | rdfs: Resource |

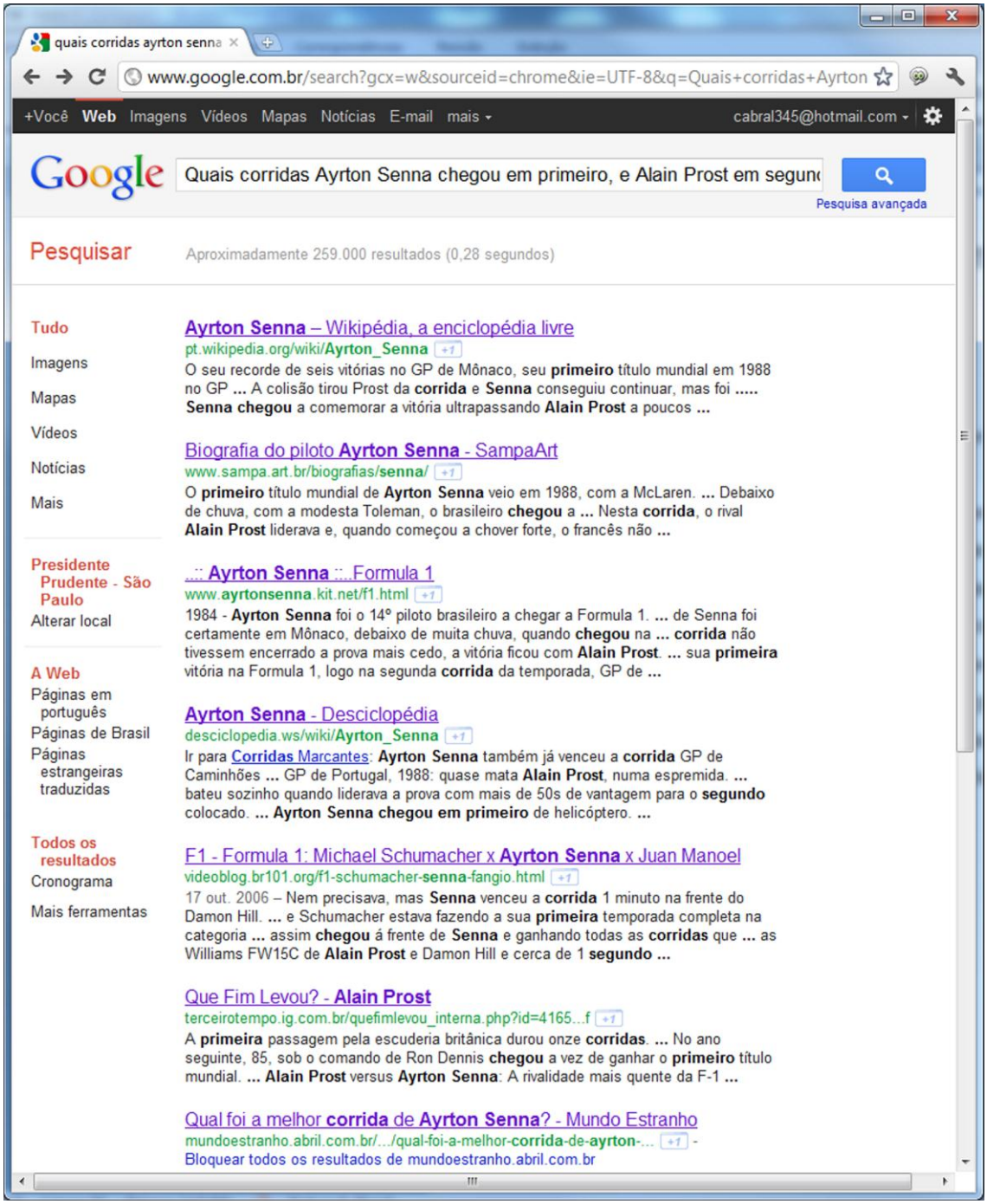
Adaptado de < http://www.w3.org/TR/rdf-schema/> (tradução nossa).

# APÊNDICE B – Código RDF/XML Do Vocabulário Da Seção 2.4.2

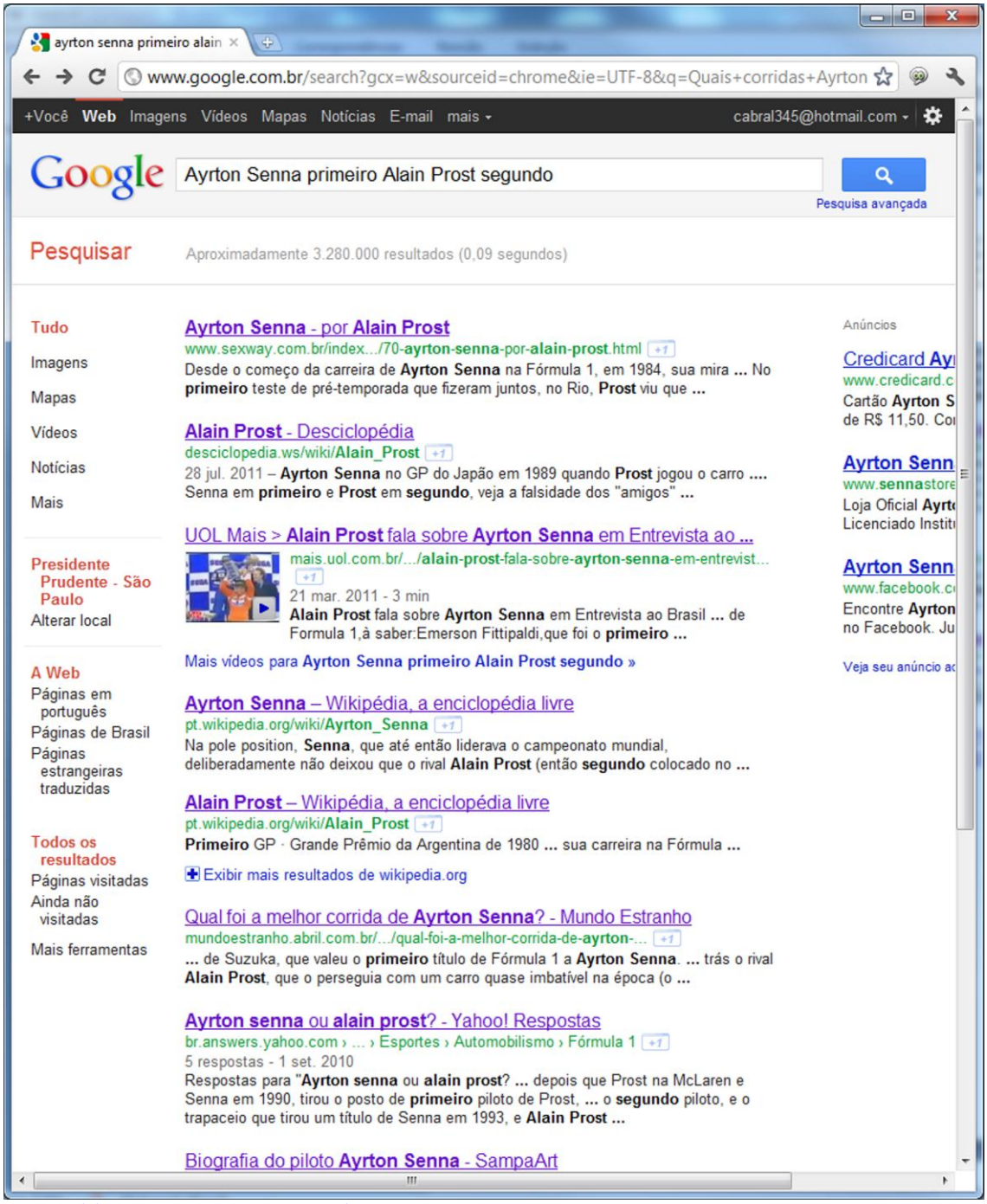
```
1: <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
2: <rdf:RDF
3:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4:   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
5:   xml:base="http://voca.org/veiculo">
6:
7: <rdfs:Class rdf:ID="VeiculoAutomotor"/>
8:
9: <rdfs:Class rdf:ID="VeiculoPasseio">
10:   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
11: </rdfs:Class>
12:
13: <rdfs:Class rdf:ID="VeiculoCarga">
14:   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
15: </rdfs:Class>
16:
17: <rdfs:Class rdf:ID="VeiculoTransColetivo">
18:   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
19: </rdfs:Class>
20:
21: <rdfs:Class rdf:ID="Motocicleta">
22:   <rdfs:subClassOf rdf:resource="#VeiculoAutomotor"/>
23: </rdfs:Class>
24:
25: <rdfs:Class rdf:ID="Utilitario">
26:   <rdfs:subClassOf rdf:resource="#VeiculoCarga"/>
27:   <rdfs:subClassOf rdf:resource="#VeiculoPasseio"/>
28: </rdfs:Class>
29:
30: <rdf:Property rdf:ID="anoModelo">
31:    <rdfs:label>Ano de Lançamento do Modelo</rdfs:label>
32:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
33:    <rdfs:range rdf:resource="&xsd;integer"/>
34: </rdf:Property>
35:
36: <rdf:Property rdf:ID="fabricante">
37:    <rdfs:label>Fabrincante(Montadora) do Veículo</rdfs:label>
38:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
39:    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
40: </rdf:Property>
41:
42: <rdf:Property rdf:ID="modelo">
43:    <rdfs:label>Nome do Veículo</rdfs:label>
44:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
45:    <rdfs:range rdf:resource="&xsd;string"/>
46: </rdf:Property>
47:
48: <rdf:Property rdf:ID="ccMotor">
49:    <rdfs:label>Cilindradas do Motor</rdfs:label>
50:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
51:    <rdfs:range rdf:resource="&xsd;integer"/>
52: </rdf:Property>
53:
54: <rdf:Property rdf:ID="cilMotor">
55:    <rdfs:label>Cilindros do motor</rdfs:label>
56:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
57:    <rdfs:range rdf:resource="&xsd;string"/>
58: </rdf:Property>
59:
60: <rdf:Property rdf:ID="versao">
61:    <rdfs:label>Versão do veículo</rdfs:label>
62:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
63:    <rdfs:range rdf:resource="&xsd;string"/>
64: </rdf:Property>
65:
```

```
66: <rdf:Property rdf:ID="motor">
67:    <rdfs:label>Versão do veículo</rdfs:label>
68:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
69:    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
70: </rdf:Property>
71:
72: <rdf:Property rdf:ID="peso">
73:    <rdfs:label>Peso bruto(kg)</rdfs:label>
74:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
75:    <rdfs:range rdf:resource="&xsd;integer"/>
76: </rdf:Property>
77:
78: <rdf:Property rdf:ID="comprimento">
79:    <rdfs:label>Comprimento(mm)</rdfs:label>
80:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
81:    <rdfs:range rdf:resource="&xsd;integer"/>
82: </rdf:Property>
83:
84: <rdf:Property rdf:ID="largura">
85:    <rdfs:label>Largura(mm)</rdfs:label>
86:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
87:    <rdfs:range rdf:resource="&xsd;integer"/>
88: </rdf:Property>
89:
90: <rdf:Property rdf:ID="altura">
91:    <rdfs:label>Altura(mm)</rdfs:label>
92:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
93:    <rdfs:range rdf:resource="&xsd;integer"/>
94: </rdf:Property>
95:
96: <rdf:Property rdf:ID="marchas">
97:    <rdfs:label>Marchas do câmbio</rdfs:label>
98:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
99:    <rdfs:range rdf:resource="&xsd;string"/>
100: </rdf:Property>
101:
102: <rdf:Property rdf:ID="combustivel">
103:    <rdfs:label>Combustível do veículo</rdfs:label>
104:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
105:    <rdfs:range rdf:resource="&xsd;string"/>
106: </rdf:Property>
107:
108: <rdf:Property rdf:ID="nPassageiros">
109:    <rdfs:label>Número de Passageiros Sentados</rdfs:label>
110:    <rdfs:domain rdf:resource="#VeiculoAutomotor"/>
111:    <rdfs:range rdf:resource="&xsd;integer"/>
112: </rdf:Property>
113:
114: <rdf:Property rdf:ID="ccCacamba">
115:    <rdfs:label>Centímetros Cúbicos da Caçamba</rdfs:label>
116:    <rdfs:domain rdf:resource="#VeiculoCarga"/>
117:    <rdfs:range rdf:resource="&xsd;integer"/>
118: </rdf:Property>
119:
120: <rdf:Property rdf:ID="nPassageirosPe">
121:    <rdfs:label>Número de Passageiros em pé</rdfs:label>
122:    <rdfs:domain rdf:resource="#VeiculoTransColetivo"/>
123:    <rdfs:range rdf:resource="&xsd;integer"/>
124: </rdf:Property>
125:
126: </rdf:RDF>
```

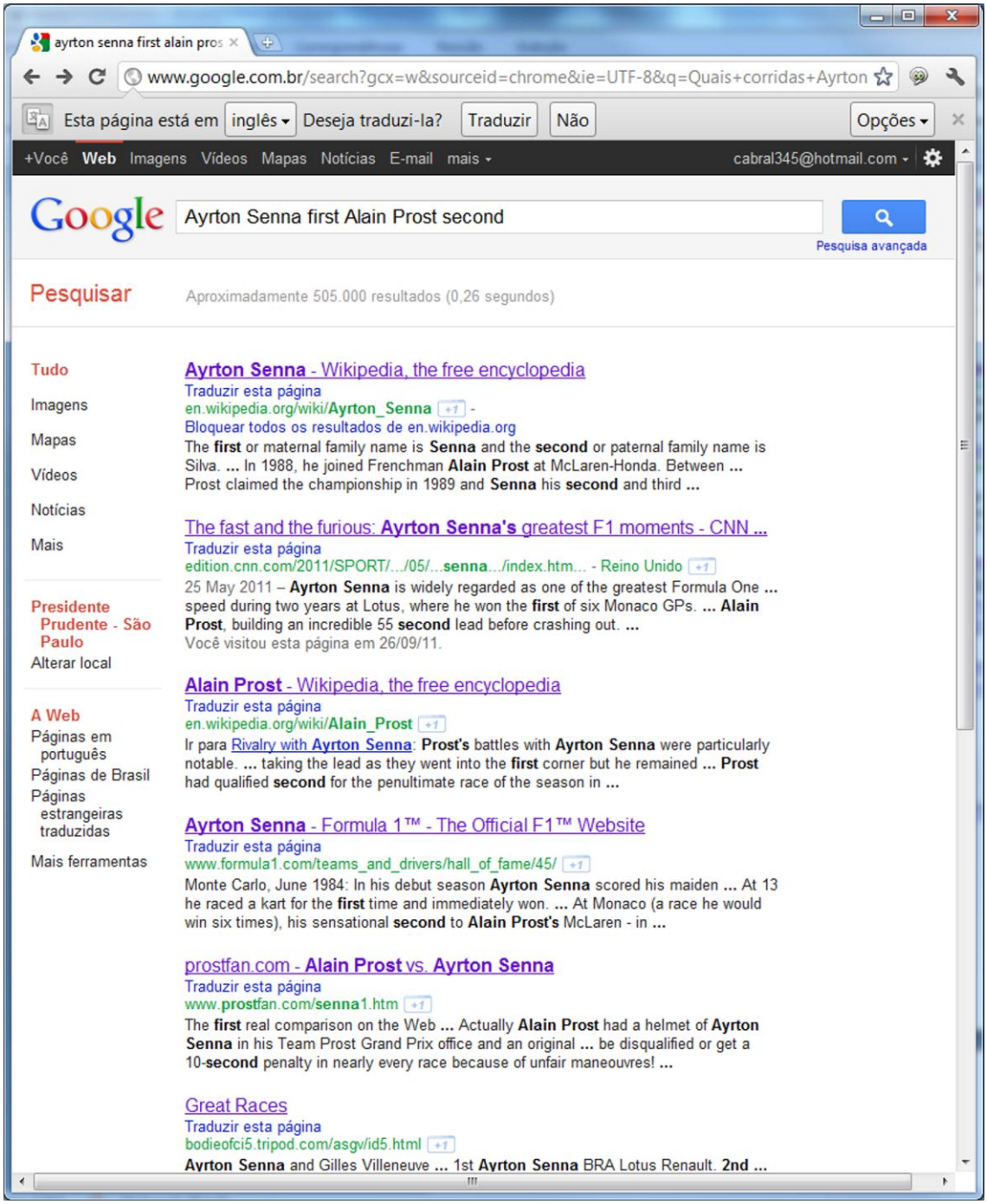
# APÊNDICE C – Resultado Das Pesquisas No Google




http://www.google.com.br/search?gcx=w&sourceid=chrome&ie=UTF-8&q=Quais+corridas+Ayrton+Senna+chegou+em+primeiro%2C+e+Alain+Prost+em+segundo




http://www.google.com.br/search?gcx=w&sourceid=chrome&ie=UTF-8&q=Quais+corridas+Ayrton+Senna+chegou+em+primeiro%2C+e+Alain+Prost+em+segundo#sclient=psy-ab&hl=pt-BR&source=hp&q=Ayrton%20Senna%20primeiro%20Alain%20Prost%20segundo&pbx=1&oq=&aq=&aqi=&aql=&gs_sm=&gs_upl=&bav=on.2,or.r_gc.r_pw.&fp=dda70b697e1da66b&biw=826&bih=936&pf=p&pdl=500




http://www.google.com.br/search?gcx=w&sourceid=chrome&ie=UTF-8&q=Quais+corridas+Ayrton+Senna+chegou+em+primeiro%2C+e+Alain+Prost+em+segundo#pq=ayrton+senna+primeiro+alain+prost+segundo&hl=pt-BR&sugexp=pfwc&cp=37&gs_id=1q&xhr=t&q=Ayrton+Senna+first+Alain+Prost+second&pf=p&sclient=psy-ab&source=hp&pbx=1&oq=Ayrton+Senna+first+Alain+Prost+second&aq=f&aqi=&aql=&gs_sm=&gs_upl=&bav=on.2,or.r_gc.r_pw.&fp=dda70b697e1da66b&biw=826&bih=936

# APÊNDICE D – Resultado Das Corridas Na Wikipedia.Org

Uma alternativa para encontrar a resposta para a pergunta “Quais corridas Ayrton Senna chegou na primeira posição, seguido por Alain Prost na segunda posição” através da Wikipédia seria cruzar esta tabela que contém os resultados de Ayrton Senna na formula 1 com a tabela com os resultados de Alain Prost.

Complete Formula One results

(key) (Races in **bold** indicate pole position; races in *italics* indicate fastest lap)

| Year | Team | Chassis | Engine | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | WDC | Points[1] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1984 | Toleman Group Motorsport | Toleman TG183B | Hart S4 (t/c) | BRA Ret | RSA 6 | BEL 6 | SMR DNQ | | | | | | | | | | | | | 9th | 13 |
| | | Toleman TG184 | Hart S4 (t/c) | | | | | FRA Ret | *MON*[2] 2 | CAN 7 | DET Ret | DAL Ret | GBR 3 | GER Ret | AUT Ret | NED Ret | ITA | EUR Ret | POR 3 | | |
| 1985 | John Player Special Team Lotus | Lotus 97T | Renault V6 (t/c) | BRA Ret | *POR* 1 | SMR 7 | MON Ret | *CAN* 16 | *DET* Ret | FRA Ret | GBR 10 | GER Ret | AUT 2 | NED 3 | ITA 3 | BEL 1 | EUR 2 | RSA Ret | AUS Ret | 4th | 38 |
| 1986 | John Player Special Team Lotus | Lotus 98T | Renault V6 (t/c) | BRA 2 | ESP 1 | SMR Ret | MON 3 | BEL 2 | CAN 5 | DET 1 | FRA Ret | GBR Ret | GER 2 | HUN 2 | AUT Ret | ITA Ret | POR 4 | MEX 3 | AUS Ret | 4th | 55 |
| 1987 | Camel Team Lotus Honda | Lotus 99T | Honda V6 (t/c) | BRA Ret | SMR 2 | BEL Ret | *MON* 1 | *DET* 1 | FRA 4 | GBR 3 | GER 3 | HUN 2 | AUT 5 | *ITA* 2 | POR 7 | ESP 5 | MEX Ret | JPN 2 | AUS DSQ | 3rd | 57 |
| 1988 | Honda Marlboro McLaren | McLaren MP4/4 | Honda V6 (t/c) | BRA DSQ | SMR 1 | *MON* Ret | MEX 2 | *CAN* 1 | DET 1 | FRA 2 | GBR 1 | GER 1 | HUN 1 | BEL 1 | ITA 10 | POR 6 | ESP 4 | *JPN* 1 | AUS 2 | 1st | 90 (94) |
| 1989 | Honda Marlboro McLaren | McLaren MP4/5 | Honda V10 | BRA 11 | SMR 1 | MON 1 | MEX 1 | *USA* Ret | CAN 7 | FRA Ret | GBR Ret | *GER* 1 | HUN 2 | BEL 1 | ITA Ret | POR Ret | *ESP* 1 | JPN DSQ | AUS Ret | 2nd | 60 |
| 1990 | Honda Marlboro McLaren | McLaren MP4/5B | Honda V10 | USA 1 | BRA 3 | SMR Ret | *MON* 1 | CAN 1 | MEX 20 | FRA 3 | GBR 3 | GER 1 | HUN 2 | BEL 1 | *ITA* 1 | POR 2 | ESP Ret | JPN Ret | AUS Ret | 1st | 78 |
| 1991 | Honda Marlboro McLaren | McLaren MP4/6 | Honda V12 | USA 1 | BRA 1 | SMR 1 | MON 1 | CAN Ret | MEX 3 | FRA 3 | GBR 4 | GER 7 | HUN 1 | BEL 1 | *ITA* 2 | POR 2 | ESP 5 | *JPN* 2 | AUS[‡] 1 | 1st | 96 |
| 1992 | Honda Marlboro McLaren | McLaren MP4/6B | Honda V12 | RSA 3 | MEX Ret | | | | | | | | | | | | | | | 4th | 50 |
| | | McLaren MP4/7A | Honda V12 | | | BRA Ret | ESP 9 | SMR 3 | MON 1 | CAN Ret | FRA Ret | GBR Ret | GER 2 | HUN 1 | BEL 5 | ITA 1 | *POR* 3 | JPN Ret | AUS Ret | | |
| 1993 | Marlboro McLaren | McLaren MP4/8 | Ford V8 | RSA 2 | BRA 1 | *EUR* 1 | SMR Ret | ESP 2 | MON 1 | CAN 18 | FRA 4 | GBR 5 | GER 4 | HUN Ret | BEL 4 | ITA Ret | POR Ret | JPN 1 | AUS 1 | 2nd | 73 |
| 1994 | Rothmans Williams Renault | Williams FW16 | Renault V10 | BRA Ret | PAC Ret | SMR Ret | MON | ESP | CAN | FRA | GBR | GER | HUN | BEL | ITA | POR | EUR | JPN | AUS | NC | 0 |

Resultado das corridas de Ayrton Senna.

## Complete Formula One results

(key) (Races in **bold** indicate pole position, races in *italics* indicate fastest lap)

| Year | Team | Chassis | Engine | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | WDC | Points[1] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1980 | Marlboro Team McLaren | McLaren M29B | Ford Cosworth DFV | ARG 6 | BRA 5 | RSA DNS | USW | | | | | | | | | | | | | 16th | 5 |
| | | McLaren M29C | | | | | | BEL Ret | MON Ret | FRA Ret | GBR 6 | GER 11 | AUT 7 | | | | | | | | |
| | | McLaren M30 | | | | | | | | | | | | NED 6 | ITA 7 | CAN Ret | USA DNS | | | | |
| 1981 | Equipe Renault Elf | Renault RE20B | Renault V6 (t/c) | USW Ret | BRA Ret | ARG 3 | SMR Ret | BEL Ret | | | | | | | | | | | | 5th | 43 |
| | | Renault RE30 | | | | | | | MON Ret | ESP Ret | *FRA* 1 | GBR Ret | **GER** 2 | AUT Ret | **NED** 1 | ITA 1 | CAN Ret | CPL 2 | | | |
| 1982 | Equipe Renault Elf | Renault RE30B | Renault V6 (t/c) | *RSA* 1 | ***BRA*** 1 | USW Ret | SMR Ret | **BEL** Ret | MON 7 | ***DET*** NC | CAN Ret | NED Ret | GBR 6 | FRA 2 | GER Ret | AUT 8 | ***SUI*** 2 | ITA Ret | **CPL** 4 | 4th | 34 |
| 1983 | Equipe Renault Elf | Renault RE30C | Renault V6 (t/c) | BRA 7 | | | | | | | | | | | | | | | | 2nd | 57 |
| | | Renault RE40 | | | USW 11 | ***FRA*** 1 | SMR 2 | **MON** 3 | **BEL** 1 | DET 8 | CAN 5 | *GBR* 1 | GER 4 | *AUT* 1 | NED Ret | ITA Ret | EUR 2 | RSA Ret | | | |
| 1984 | Marlboro McLaren International | McLaren MP4/2 | TAG V6 (t/c) | *BRA* 1 | RSA 2 | BEL Ret | SMR 1 | *FRA* 7 | **MON**‡ 1 | CAN 3 | DET 4 | DAL Ret | GBR Ret | ***GER*** 1 | AUT Ret | **NED** 1 | ITA Ret | EUR 1 | POR 1 | 2nd | 71.5 |
| 1985 | Marlboro McLaren International | McLaren MP4/2B | TAG V6 (t/c) | *BRA* 1 | POR Ret | SMR DSQ | MON 1 | CAN 3 | DET Ret | FRA 3 | *GBR* 1 | GER 2 | ***AUT*** 1 | *NED* 2 | ITA 1 | ***BEL*** 3 | EUR 4 | RSA 3 | AUS Ret | 1st | 73 (76) |
| 1986 | Marlboro McLaren International | McLaren MP4/2C | TAG V6 (t/c) | BRA Ret | ESP 3 | SMR 1 | ***MON*** 1 | *BEL* 6 | CAN 2 | DET 3 | FRA 2 | GBR 3 | GER 6 | HUN Ret | AUT 1 | ITA DSQ | POR 2 | MEX 2 | AUS 1 | 1st | 72 (74) |
| 1987 | Marlboro McLaren International | McLaren MP4/3 | TAG V6 (t/c) | BRA 1 | SMR Ret | *BEL* 1 | MON 9 | DET 3 | FRA 3 | GBR Ret | GER 7 | HUN 3 | AUT 6 | ITA 15 | POR 1 | ESP 2 | MEX Ret | *JPN* 7 | AUS Ret | 4th | 46 |
| 1988 | Honda Marlboro McLaren | McLaren MP4/4 | Honda V6 (t/c) | BRA 1 | *SMR* 2 | MON 1 | *MEX* 1 | CAN 2 | *DET* 2 | ***FRA*** 1 | GBR Ret | GER 2 | *HUN* 2 | BEL 2 | ITA Ret | **POR** 1 | *ESP* 1 | JPN 2 | *AUS* 1 | 2nd | 87 (105) |
| 1989 | Honda Marlboro McLaren | McLaren MP4/5 | Honda V10 | BRA 2 | *SMR* 2 | *MON* 2 | MEX 5 | USA 1 | **CAN** Ret | **FRA** 1 | GBR 1 | GER 2 | HUN 4 | *BEL* 2 | *ITA* 1 | POR 2 | ESP 3 | *JPN* Ret | AUS Ret | 1st | 76 (81) |
| 1990 | Scuderia Ferrari | Ferrari 641 | Ferrari V12 | USA Ret | BRA 1 | SMR 4 | MON Ret | | | | | | | | | | | | | 2nd | 71 (73) |
| | | Ferrari 641/2 | | | | | | CAN 5 | *MEX* 1 | FRA 1 | GBR 1 | GER 4 | HUN Ret | *BEL* 2 | ITA 2 | POR 3 | ESP 1 | JPN Ret | AUS 3 | | |
| 1991 | Scuderia Ferrari | Ferrari 642 | Ferrari V12 | USA 2 | BRA 4 | SMR DNS | *MON* 5 | CAN Ret | MEX Ret | | | | | | | | | | | 5th | 34 |
| | | Ferrari 643 | | | | | | | | FRA 2 | GBR 3 | GER Ret | HUN Ret | BEL Ret | ITA 3 | POR Ret | ESP 2 | JPN 4 | AUS | | |
| 1993 | Canon Williams Renault | Williams FW15C | Renault V10 | ***RSA*** 1 | **BRA** Ret | **EUR** 3 | ***SMR*** 1 | **ESP** 1 | ***MON*** 4 | CAN 1 | FRA 1 | GBR 1 | GER 1 | ***HUN*** 12 | ***BEL*** 3 | **ITA** 12 | POR 2 | ***JPN*** 2 | AUS 2 | 1st | 99 |

Resultado das corridas de Alain Prost.

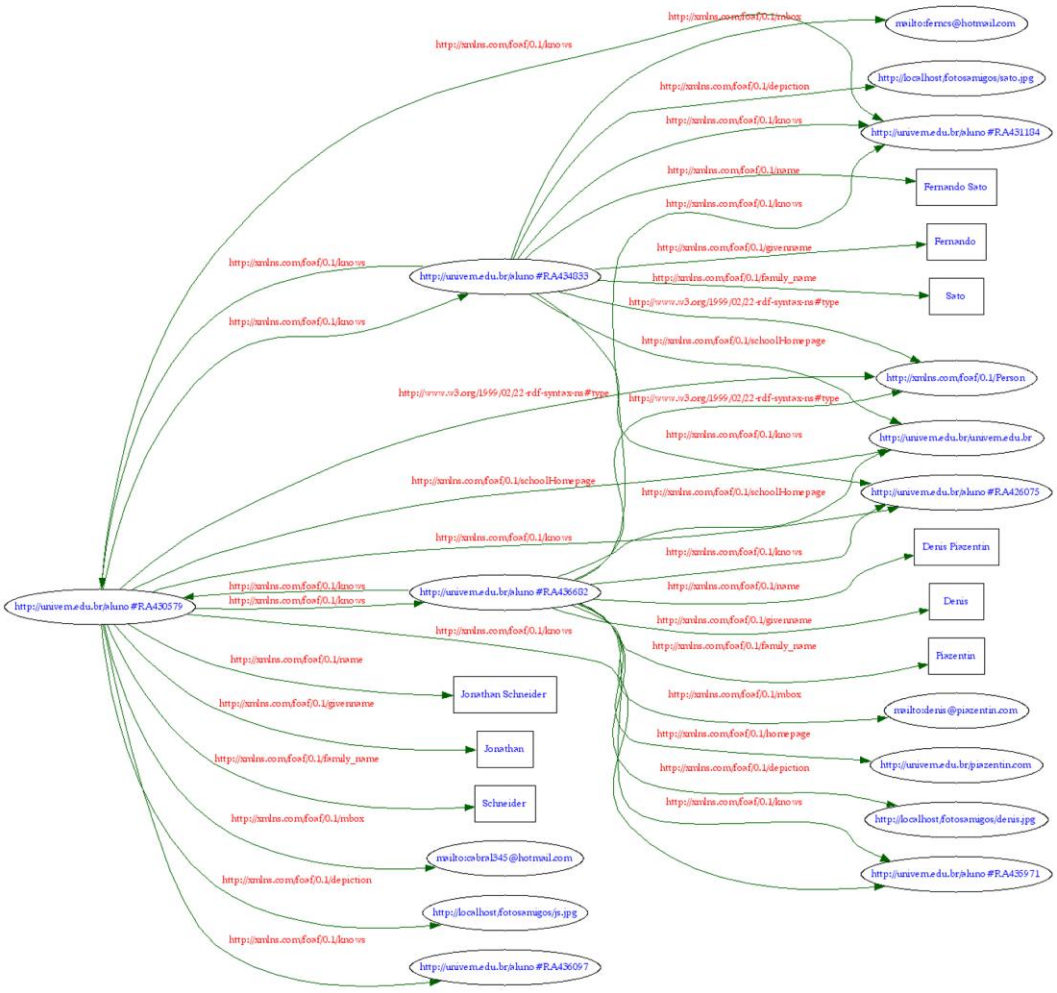
# APÊNDICE E – Tabelas De Funções Primitivas Jena Rules

| Primitivas | Descrição |
|---|---|
| isLiteral(?x) notLiteral(?x)<br><br>isFunctor(?x) notFunctor(?x)<br><br>isBNode(?x) notBNode(?x) | Testa se o único argumento é ou não é um literal, um literal de valor functor ou de um blank-node, respectivamente. |
| bound(?x...) unbound(?x..) | Testa se todos os argumentos são variáveis bound ou not bound. |
| equal(?x,?y) notEqual(?x,?y) | Testa se x = y (ou x! = Y). O teste de igualdade é a igualdade semântica de forma que, por exemplo, o xsd: int 1 e os xsd: decimal 1 são iguais. |
| lessThan(?x, ?y), greaterThan(?x, ?y)<br><br>le(?x, ?y), ge(?x, ?y) | Testar se x é <,>, <= ou> = y. Só passa se ambos x e y são números ou horários (pode ser inteiro, ponto flutuante ou XSDDateTime). |
| sum(?a, ?b, ?c)<br><br>addOne(?a, ?c)<br><br>difference(?a, ?b, ?c)<br><br>min(?a, ?b, ?c)<br><br>max(?a, ?b, ?c)<br><br>product(?a, ?b, ?c)<br><br>quotient(?a, ?b, ?c) | Atribuí para C o retorno de (a + b), (a +1) , (a-b), min (a, b), max (a, b), (a*b), (a / b). |
| strConcat(?a1, .. ?an, ?t)<br><br>uriConcat(?a1, .. ?an, ?t) | Concatena a forma lexical de todos os argumentos, exceto o último, então se liga o último argumento para um literal simples (strConcat) ou um nó URI (uriConcat) com a forma lexical. Em ambos os casos, se um nó argumento é um nó da URI, URI será usado como a forma lexical. |
| regex(?t, ?p)<br><br>regex(?t, ?p, ?m1, .. ?mn) | Corresponde à forma lexical de um literal (t) contra um padrão de expressão regular dada por outra literal (p). Se o match for bem sucedido, e se existem quaisquer argumentos adicionais, então ele irá ligar os primeiros grupos n capturados para os argumentos? m1 para? mn. A sintaxe padrão de expressão regular é a prevista pela java.util.regex. Note-se que os grupos de captura são numerados de 1 e do grupo de primeira captura será ligado a ?m1, ignoramos o grupo implícito 0 , o que corresponde a toda a seqüência correspondente. Assim, por exemplo:<br>regexp('foo bar', '(.*) (.*)', ?m1, ?m2); ligará m1 = "foo" e m2 = "bar".. |

| | |
|---|---|
| now(?x) | Retorna para x, o xsd: dateTime correspondente a data atual. |
| makeTemp(?x) | Liga ?x para um recem-criado blank node. |
| makeInstance(?x, ?p, ?v)<br><br>makeInstance(?x, ?p, ?t, ?v) | Atribui ?v como sendo um blank node que é afirmado como o valor da propriedade p? no recurso?x, e, opcionalmente, tem o tipo?t. Várias chamadas com os mesmos argumentos retornará o mesmo blank node de cada vez - permitindo assim que esta chamada seja usada em regras ser usado em regras *Backward Chaining.* |
| makeSkolem(?x, ?v1, ... ?vn) | Atribui ?x um blank node, este é gerado com base nos valores dos argumentos ?vi. |
| noValue(?x, ?p)<br><br>noValue(?x ?p ?v) | Verdadeira se não houver a tripla (x, p, *) ou (x, p, v) no modelo ou as deduções explícitas Forward Chaining até o momento. |
| remove(n, ...)<br><br>drop(n, ...) | Remova a instrução (tripla) que causou o enésimo termo desta regra (forward-only) . Remover irá propagar a alteração de outras regras conseqüente incluindo a regra de disparo (que deve assim ser guardada por algumas outras cláusulas). Queda silenciosamente remover a tripla (s) a partir do gráfico, mas não todas as regras de fogo como conseqüência. |
| isDType(?l, ?t) notDType(?l, ?t) | Testa se o literal ?l é (ou não) uma instância do tipo de dados definidos pelo recurso ?t. |
| print(?x, ...) | Impressão (para a saída padrão) uma representação de cada argumento. Isso é útil para depuração em vez de trabalhar com IO. |

Adaptado de < http://jena.sourceforge.net/inference/index.html> (tradução nossa)

# APÊNDICE F – Bases RDF redesocial1.rdf e redesocial2.rdf




Grafo da base redesocial1.rdf

```
# redesocial1 ################################################
        <?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xml:base="http://univem.edu.br/aluno">
    <foaf:Person rdf:ID="RA430579">
        <foaf:name>Jonathan Schneider</foaf:name>
        <foaf:givenname>Jonathan</foaf:givenname>
        <foaf:family_name>Schneider</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:cabral345@hotmail.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/js.jpg"/>
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA436682"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA431184"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA435971"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA426075"/>
           <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA434833"/>
              <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA436097"/>
    </foaf:Person>

    <foaf:Person rdf:ID="RA436682">
```

```
        <foaf:name>Denis Piazentin</foaf:name>
        <foaf:givenname>Denis</foaf:givenname>
        <foaf:family_name>Piazentin</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:denis@piazentin.com"/>
        <foaf:homepage rdf:resource="piazentin.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/denis.jpg"/>
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA430579"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA431184"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA435971"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA426075"/>
    </foaf:Person>

      <foaf:Person rdf:ID="RA434833">
        <foaf:name>Fernando Sato</foaf:name>
        <foaf:givenname>Fernando</foaf:givenname>
        <foaf:family_name>Sato</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:ferncs@hotmail.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/sato.jpg"/>
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA430579"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA431184"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA426075"/>
     </foaf:Person>
</rdf:RDF>
```


Grafo da base redesocial2.rdf

```
# redesocial2 ################################################
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xml:base="http://univem.edu.br/aluno">
    <foaf:Person rdf:ID="RA436097">
        <foaf:name>Helena Cabrini</foaf:name>
        <foaf:givenname>Helena</foaf:givenname>
        <foaf:family_name>Cabrini</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:helena_cabrini@hotmail.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/helena.jpg"/>
```

```
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA436682"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA431184"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA430579"/>
    </foaf:Person>

    <foaf:Person rdf:ID="RA431184">
        <foaf:name>Raphael Negrisoli</foaf:name>
        <foaf:givenname>Raphael</foaf:givenname>
        <foaf:family_name>Negrisoli</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:raphaelnegrisoli@gamail.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/negrisoli.jpg"/>
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA430579"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA426075"/>
    </foaf:Person>

      <foaf:Person rdf:ID="RA435971">
        <foaf:name>Luis Fernando Martins</foaf:name>
        <foaf:givenname>Luis</foaf:givenname>
        <foaf:family_name>Martins</foaf:family_name>
        <foaf:mbox rdf:resource="mailto:luisfernando@lfmartins.com"/>
        <foaf:depiction rdf:resource="http://localhost/fotosamigos/luis.jpg"/>
        <foaf:schoolHomepage rdf:resource="univem.edu.br"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA430579"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA431184"/>
        <foaf:knows rdf:resource="http://univem.edu.br/aluno#RA426075"/>
     </foaf:Person>
</rdf:RDF>
```

# APÊNDICE G – Base RDF avaliacao1.rdf


Grafo da base avaliacao1.rdf

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:tp="http://voca.org/temproduto#"
  xml:base="http://siteavaliacao1.com/opnioes">

  <tp:TemProduto rdf:ID="av1">
       <tp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
       <tp:pessoa rdf:resource="http://univem.edu.br/aluno#RA436682" />
       <tp:nota rdf:datatype="http://www.w3.org/2001/XMLSchema#int">9</tp:nota>
       <tp:opniao>Recomendo 100%, agora vejo Ursinhos Carinhosos em 3D.</tp:opniao>
  </tp:TemProduto>

  <tp:TemProduto rdf:ID="av2">
       <tp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
       <tp:pessoa rdf:resource="http://univem.edu.br/aluno#RA436097" />
       <tp:nota rdf:datatype="http://www.w3.org/2001/XMLSchema#int">10</tp:nota>
       <tp:opniao>Maravilhosa, tem entrada VGA assim consigo assistir aos clipes dos
Menudos que baixei!! .</tp:opniao>
  </tp:TemProduto>

  <tp:TemProduto rdf:ID="av3">
       <tp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
       <tp:pessoa rdf:resource="http://univem.edu.br/aluno#RA434833" />
       <tp:nota rdf:datatype="http://www.w3.org/2001/XMLSchema#int">10</tp:nota>
       <tp:opniao>O 3D é fantástico, agora é esperar o lançamento de Maria do Bairro
em 3D .</tp:opniao>
  </tp:TemProduto>

</rdf:RDF>
```

# APÊNDICE H – Vocabulário temproduto


Grafo do vocabulário temproduto

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://voca.org/temproduto">

<rdfs:Class rdf:ID="TemProduto">
                  <rdfs:label>Tem o produto</rdfs:label>
                  <rdfs:label>Tem</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="produto">
   <rdfs:label>Produto</rdfs:label>
   <rdfs:domain rdf:resource="#TemProduto"/>
   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="pessoa">
   <rdfs:label>Pessoa</rdfs:label>
   <rdfs:domain rdf:resource="#TemProduto"/>
   <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="opniao">
                  <rdfs:label>Opnião sobre o produto</rdfs:label>
   <rdfs:domain rdf:resource="#TemProduto"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="nota">
            <rdfs:label>Nota sobre o produto</rdfs:label>
   <rdfs:domain rdf:resource="#TemProduto"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
</rdf:RDF>
```

# APÊNDICE I – Bases RDF loja1.rdf, loja2.rdf e loja3.rdf


Grafo da base loja1.rdf

```
#loja1##################################################################
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:lp="http://voca.org/lojaproduto#"
  xml:base="http://loja1.com.br">

  <lp:Loja rdf:ID="sobre">
      <rdfs:label>Loja1 a primeira em preço baixo!!</rdfs:label>
      <lp:siteLoja rdf:resource="http://loja1.com.br" />
      <lp:logoLoja rdf:resource="http://localhost/fotop/logoloja1.jpg" />
      <lp:nomeLoja>Loja 1</lp:nomeLoja>
  </lp:Loja>
```

```
  <lp:LojaProduto rdf:ID="p1">
      <lp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
      <lp:preco       rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2499.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p1" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p2">
      <lp:produto rdf:resource="http://samsung.com/tv#PL51D491" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">5499.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p2" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p3">
      <lp:produto rdf:resource="http://samsung.com/tv#UN32D4000" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1299.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p3" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>
  <lp:LojaProduto rdf:ID="p4">
      <lp:produto rdf:resource="http://lg.com/tv#42LW4500" />
      <lp:preco       rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2299.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p4" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p5">
      <lp:produto rdf:resource="http://lg.com/tv#52LG50FD" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2599.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p5" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p6">
      <lp:produto rdf:resource="http://lg.com/tv#32LD650" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1199.00</lp:preco>
      <lp:link rdf:resource="http://loja1.com.br/produto.htm?id=p6" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

</rdf:RDF>
```


Grafo da base loja2.rdf

```
#loja2#################################################################
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:lp="http://voca.org/lojaproduto#"
  xml:base="http://loja2.com.br">

  <lp:Loja rdf:ID="sobre">
      <rdfs:label>Loja2, duas vezes mais promoções para você!</rdfs:label>
      <lp:siteLoja rdf:resource="http://loja1.com.br" />
      <lp:logoLoja rdf:resource="http://localhost/fotop/logoloja2.jpg" />
      <lp:nomeLoja>Loja 2</lp:nomeLoja>
  </lp:Loja>

  <lp:LojaProduto rdf:ID="p1">
      <lp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
      <lp:preco       rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2439.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p1" />
```

```
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p2">
      <lp:produto rdf:resource="http://samsung.com/tv#PL51D491" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">5200.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p2" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p3">
      <lp:produto rdf:resource="http://samsung.com/tv#UN32D4000" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1315.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p3" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>
  <lp:LojaProduto rdf:ID="p4">
      <lp:produto rdf:resource="http://lg.com/tv#42LW4500" />
      <lp:preco      rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2200.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p4" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p5">
      <lp:produto rdf:resource="http://lg.com/tv#52LG50FD" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2899.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p5" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p6">
      <lp:produto rdf:resource="http://lg.com/tv#32LD650" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">999.00</lp:preco>
      <lp:link rdf:resource="http://loja2.com.br/produto.htm?id=p6" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

</rdf:RDF>
```


Grafo da base loja3.rdf

```
#loja3#################################################################
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:lp="http://voca.org/lojaproduto#"
  xml:base="http://loja3.com.br">

  <lp:Loja rdf:ID="sobre">
      <rdfs:label>Loja3, 1 é pouco, 2 é bom, 3 é demais!!</rdfs:label>
      <lp:siteLoja rdf:resource="http://loja1.com.br" />
      <lp:logoLoja rdf:resource="http://localhost/fotop/logoloja3.jpg" />
      <lp:nomeLoja>Loja 3</lp:nomeLoja>
  </lp:Loja>

  <lp:LojaProduto rdf:ID="p1">
      <lp:produto rdf:resource="http://samsung.com/tv#UN40D6400" />
```

```
      <lp:preco       rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2239.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p1" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p2">
      <lp:produto rdf:resource="http://samsung.com/tv#PL51D491" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">4999.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p2" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p3">
      <lp:produto rdf:resource="http://samsung.com/tv#UN32D4000" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1299.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p3" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>
  <lp:LojaProduto rdf:ID="p4">
      <lp:produto rdf:resource="http://lg.com/tv#42LW4500" />
      <lp:preco       rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
2009.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p4" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p5">
      <lp:produto rdf:resource="http://lg.com/tv#52LG50FD" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2500.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p5" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

  <lp:LojaProduto rdf:ID="p6">
      <lp:produto rdf:resource="http://lg.com/tv#32LD650" />
      <lp:preco
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">989.00</lp:preco>
      <lp:link rdf:resource="http://loja3.com.br/produto.htm?id=p6" />
      <lp:loja rdf:resource="#sobre" />
  </lp:LojaProduto>

</rdf:RDF>
```

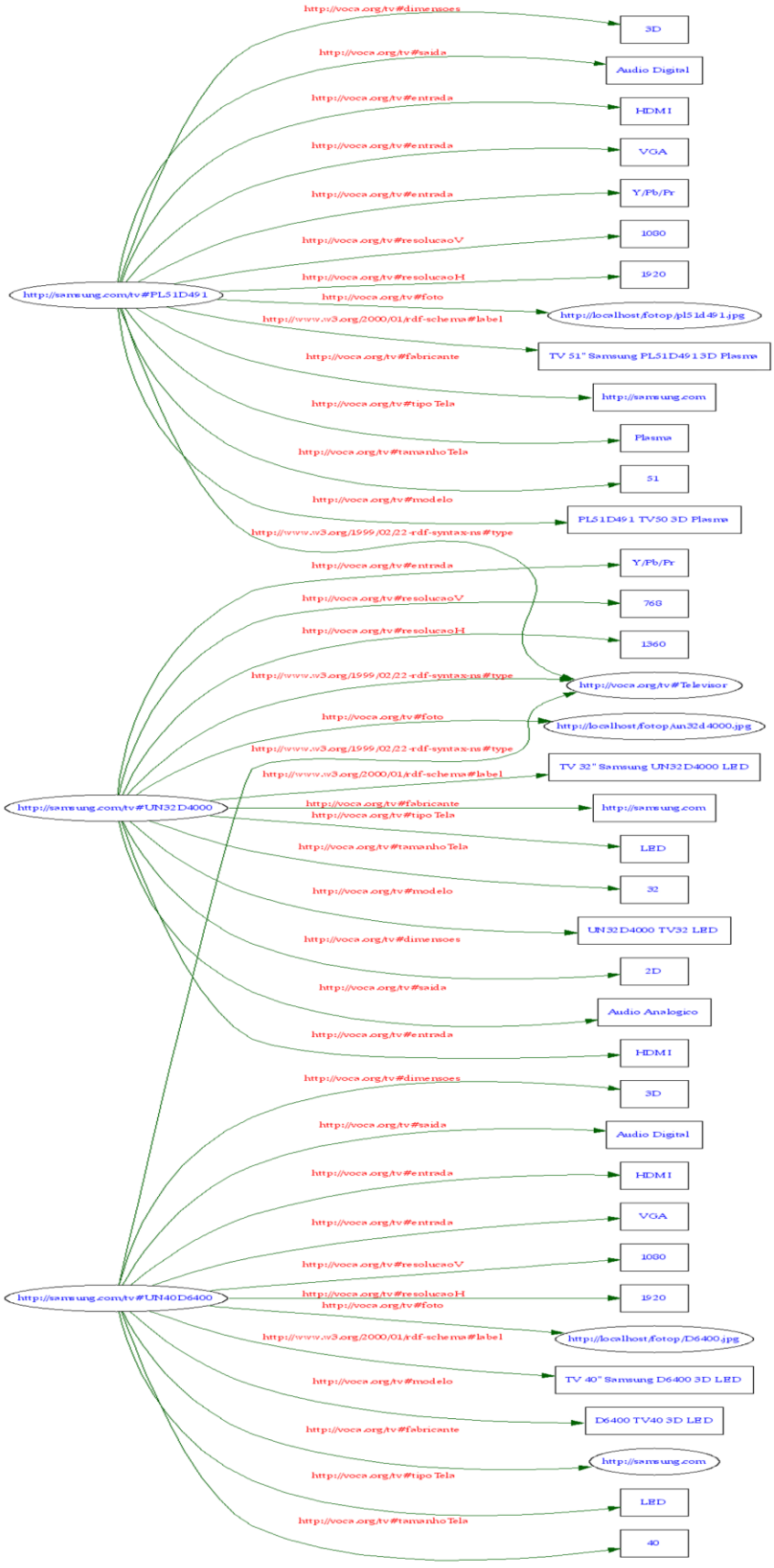
# APÊNDICE J – Vocabulário lojaproduto


Grafo do vocabulário lojaproduto

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://voca.org/lojaproduto">

<rdfs:Class rdf:ID="Loja">
   <rdfs:label>Loja</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="nomeLoja">
   <rdfs:label>Nome da Loja</rdfs:label>
   <rdfs:domain rdf:resource="#Loja"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="siteLoja">
   <rdfs:label>Site da Loja</rdfs:label>
   <rdfs:domain rdf:resource="#Loja"/>
   <rdfs:range                    rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
```

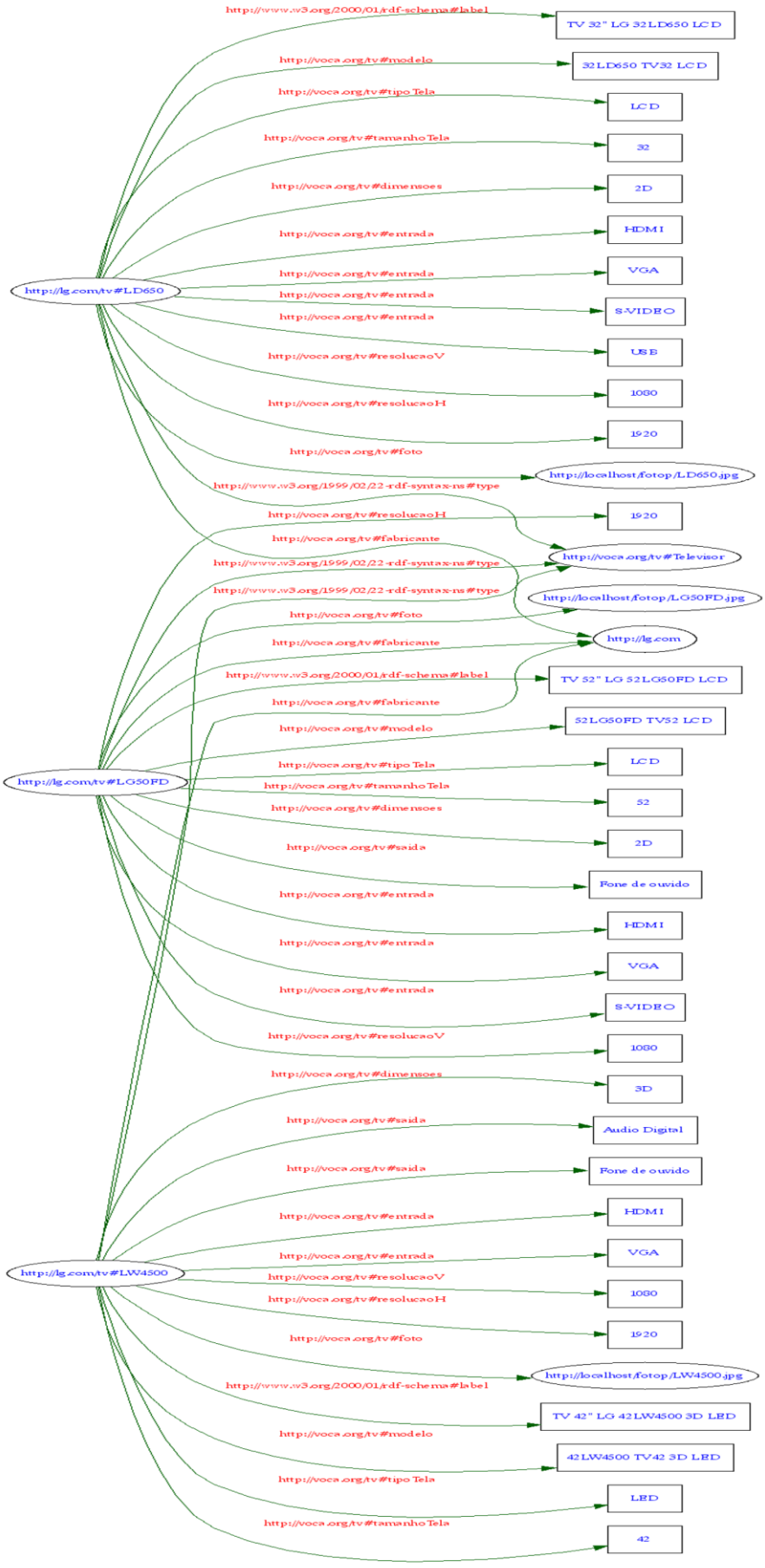
```
</rdf:Property>

<rdf:Property rdf:ID="logoLoja">
   <rdfs:label>Logo da Loja</rdfs:label>
   <rdfs:domain rdf:resource="#Loja"/>
   <rdfs:range                  rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="produto">
   <rdfs:label>URI do Produto</rdfs:label>
   <rdfs:domain rdf:resource="#LojaProduto"/>
   <rdfs:range                  rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="loja">
   <rdfs:label>Loja</rdfs:label>
   <rdfs:domain rdf:resource="#LojaProduto"/>
   <rdfs:range                  rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="preco">
   <rdfs:label>Preço</rdfs:label>
   <rdfs:domain rdf:resource="#LojaProduto"/>
   <rdfs:range rdf:resource="&xsd;float"/>
</rdf:Property>

<rdf:Property rdf:ID="link">
   <rdfs:label>Link do produto na loja</rdfs:label>
   <rdfs:domain rdf:resource="#LojaProduto"/>
   <rdfs:range                  rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

</rdf:RDF>
```

# APÊNDICE K – Bases RDF samsungtv.rdf e lgtv.rdf




Grafo da base samsungtv.rdf

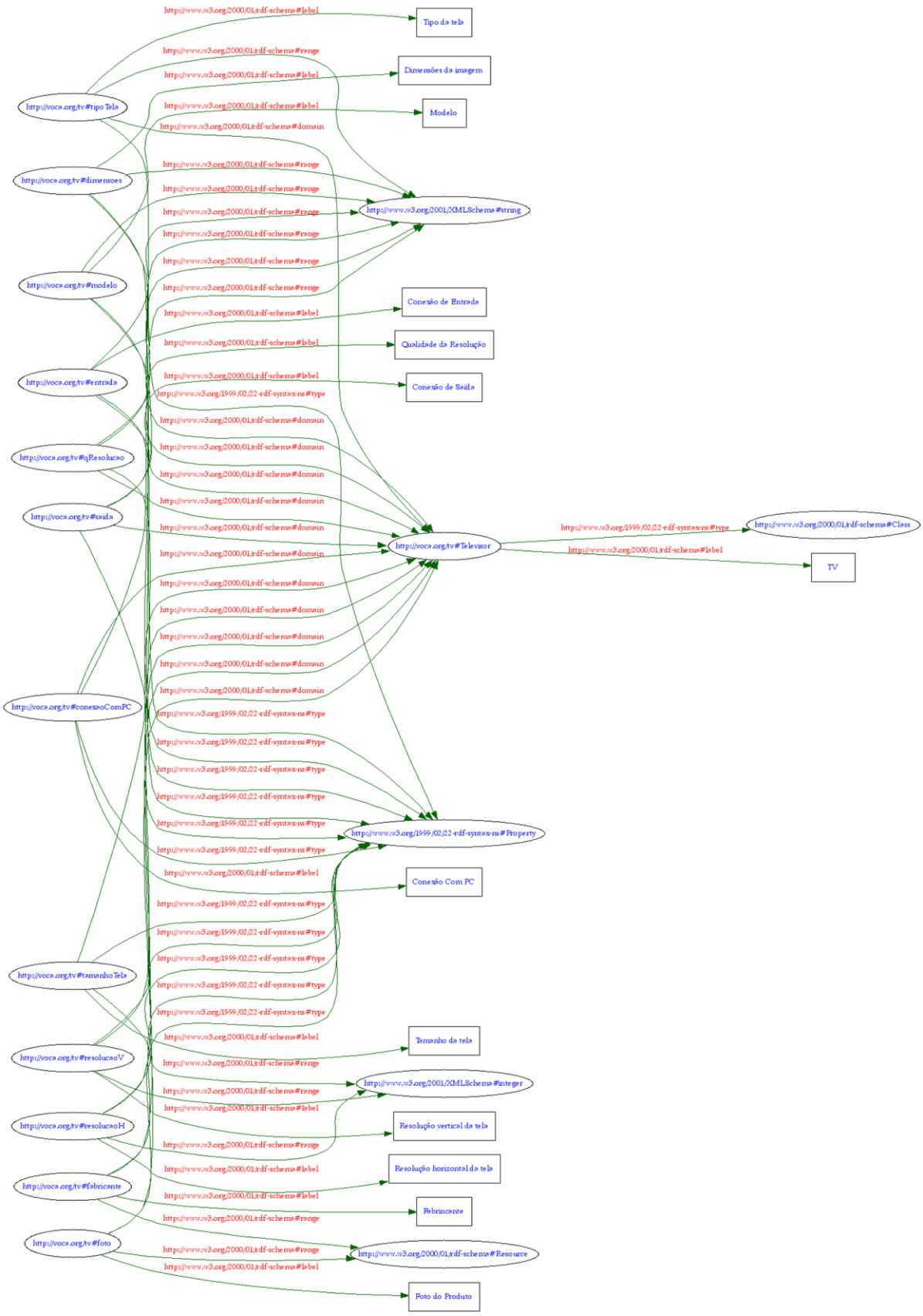
```
#samsungtv.rdf##############################################################
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:tv="http://voca.org/tv#"
  xml:base="http://samsung.com/tv">

  <tv:Televisor rdf:ID="UN40D6400">
      <rdfs:label>TV 40" Samsung D6400 3D LED</rdfs:label>
      <tv:modelo>D6400 TV40 3D LED</tv:modelo>
      <tv:fabricante rdf:resource="http://samsung.com" />
      <tv:tipoTela>LED</tv:tipoTela>
      <tv:tamanhoTela>40</tv:tamanhoTela>
      <tv:dimensoes>3D</tv:dimensoes>
      <tv:saida>Audio Digital</tv:saida>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>VGA</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1080</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1920</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/D6400.jpg" />
  </tv:Televisor>

  <tv:Televisor rdf:ID="PL51D491">
      <rdfs:label>TV 51" Samsung PL51D491 3D Plasma</rdfs:label>
      <tv:fabricante>http://samsung.com</tv:fabricante>
      <tv:tipoTela>Plasma</tv:tipoTela>
      <tv:tamanhoTela>51</tv:tamanhoTela>
      <tv:modelo>PL51D491 TV50 3D Plasma</tv:modelo>
      <tv:dimensoes>3D</tv:dimensoes>
      <tv:saida>Audio Digital</tv:saida>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>VGA</tv:entrada>
      <tv:entrada>Y/Pb/Pr</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1080</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1920</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/pl51d491.jpg" />
  </tv:Televisor>

  <tv:Televisor rdf:ID="UN32D4000">
      <rdfs:label>TV 32" Samsung UN32D4000 LED</rdfs:label>
      <tv:fabricante>http://samsung.com</tv:fabricante>
      <tv:tipoTela>LED</tv:tipoTela>
      <tv:tamanhoTela>32</tv:tamanhoTela>
      <tv:modelo>UN32D4000 TV32 LED</tv:modelo>
      <tv:dimensoes>2D</tv:dimensoes>
      <tv:saida>Audio Analogico</tv:saida>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>Y/Pb/Pr</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">768</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1360</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/un32d4000.jpg" />
  </tv:Televisor>
</rdf:RDF>
```




Grafo da base lgtv.rdf

```
#lgtv.rdf#########################################################################
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:tv="http://voca.org/tv#"
  xml:base="http://lg.com/tv">

  <tv:Televisor rdf:ID="LW4500">
      <rdfs:label>TV 42" LG 42LW4500 3D LED</rdfs:label>
      <tv:modelo>42LW4500 TV42 3D LED</tv:modelo>
      <tv:fabricante rdf:resource="http://lg.com" />
      <tv:tipoTela>LED</tv:tipoTela>
      <tv:tamanhoTela>42</tv:tamanhoTela>
      <tv:dimensoes>3D</tv:dimensoes>
      <tv:saida>Audio Digital</tv:saida>
      <tv:saida>Fone de ouvido</tv:saida>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>VGA</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1080</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1920</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/LW4500.jpg" />
  </tv:Televisor>

  <tv:Televisor rdf:ID="LG50FD">
      <rdfs:label>TV 52" LG 52LG50FD LCD</rdfs:label>
      <tv:modelo>52LG50FD TV52 LCD</tv:modelo>
      <tv:fabricante rdf:resource="http://lg.com" />
      <tv:tipoTela>LCD</tv:tipoTela>
      <tv:tamanhoTela>52</tv:tamanhoTela>
      <tv:dimensoes>2D</tv:dimensoes>
      <tv:saida>Fone de ouvido</tv:saida>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>VGA</tv:entrada>
      <tv:entrada>S-VIDEO</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1080</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1920</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/LG50FD.jpg" />
  </tv:Televisor>

   <tv:Televisor rdf:ID="LD650">
      <rdfs:label>TV 32" LG 32LD650 LCD</rdfs:label>
      <tv:modelo>32LD650 TV32 LCD</tv:modelo>
      <tv:fabricante rdf:resource="http://lg.com" />
      <tv:tipoTela>LCD</tv:tipoTela>
      <tv:tamanhoTela>32</tv:tamanhoTela>
      <tv:dimensoes>2D</tv:dimensoes>
      <tv:entrada>HDMI</tv:entrada>
      <tv:entrada>VGA</tv:entrada>
      <tv:entrada>S-VIDEO</tv:entrada>
      <tv:entrada>USB</tv:entrada>
      <tv:resolucaoV
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1080</tv:resolucaoV>
      <tv:resolucaoH
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1920</tv:resolucaoH>
      <tv:foto rdf:resource="http://localhost/fotop/LD650.jpg" />
  </tv:Televisor>
</rdf:RDF>
```

# APÊNDICE L – Vocabulário tv




Grafo do vocabulário tv

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://voca.org/tv">

<rdfs:Class rdf:ID="Televisor">
            <rdfs:label>TV</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="fabricante">
   <rdfs:label>Fabrincante</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range                  rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="tipoTela">
   <rdfs:label>Tipo da tela</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="tamanhoTela">
   <rdfs:label>Tamanho da tela</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="resolucaoV">
   <rdfs:label>Resolução vertical da tela</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="resolucaoH">
   <rdfs:label>Resolução horizontal da tela</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="modelo">
   <rdfs:label>Modelo</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="dimensoes">
   <rdfs:label>Dimensões da imagem</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="saida">
   <rdfs:label>Conexão de Saída</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="entrada">
   <rdfs:label>Conexão de Entrada</rdfs:label>
```

```
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="qResolucao">
   <rdfs:label>Qualidade da Resolução</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="conexaoComPC">
   <rdfs:label>Conexão Com PC</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="foto">
   <rdfs:label>Foto do Produto</rdfs:label>
   <rdfs:domain rdf:resource="#Televisor"/>
   <rdfs:range                rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Resource"/>
</rdf:Property>

</rdf:RDF>
```

# APÊNDICE M – Regras regrastv.txt

```
#regrastv.txt###################################################################
@prefix tv: <http://voca.org/tv#>.
[fullHD:
( ?uritv tv:resolucaoV ?resv )
ge(?resv, 1080)
 ->
( ?uritv tv:qResolucao "Full HD" )
]

[HD:
( ?uritv tv:resolucaoV ?resv )
ge(?resv, 720)
le(?resv, 1079)
 ->
( ?uritv tv:qResolucao "HD" )
]

[PCHDMI:
( ?uritv tv:entrada ?entrada )
equal(?entrada,"HDMI")
 ->
( ?uritv tv:conexaoComPC "Sim através da entrada HDMI" )
]

[PCVGA:
( ?uritv tv:entrada ?entrada )
equal(?entrada,"VGA")
 ->
( ?uritv tv:conexaoComPC "Sim através da entrada VGA" )
]
```

# APÊNDICE N – Caso de teste APC


Então você esta procurando por um(a) **TV**, mais especificamente o modelo **TV 40" Samsung D6400 3D LED**

Sim — Deseja que eu procure mais informações sobre este modelo?

Sim — Ou prefere que eu procure mais produtos iguais a este?

| | |
|---|---|
| Conexão Com PC | **Sim através da entrada HDMI** |
| Conexão Com PC | **Sim através da entrada VGA** |
| Conexão de Entrada | **HDMI** |
| Conexão de Entrada | **VGA** |
| Conexão de Saída | **Audio Digital** |
| Dimensões da imagem | **3D** |
| Fabricante | **http://samsung.com** |
| Foto do Produto | **http://localhost/fotop/D6400.jpg** |
| Modelo | **D6400 TV40 3D LED** |
| Qualidade da Resolução | **Full HD** |
| Resolução horizontal da tela | **1920** |
| Resolução vertical da tela | **1080** |
| Tamanho da tela | **40** |
| Tipo da tela | **LED** |

Sim — Deseja que eu procure amigos que tenham este modelo?

Nome **Fernando Sato**
Nota **10**
O 3D é fantástico, agora é esperar o lançamento de Maria do Bairro em 3D .

Nome **Denis Piazentin**
Nota **9**
Recomendo 100%, agora vejo Ursinhos Carinhosos em 3D.

Nome **Helena Cabrini**
Nota **10**
Maravilhosa, tem entrada VGA assim consigo assistir aos clipes dos Menudos que baixei!! .

Sim — Deseja que eu procure este modelo nas lojas?

Preço **R$2239.0**
Link do modelo: **Visitar**
Loja3, 1 é pouco, 2 é bom, 3 é demais!!

Preço **R$2439.0**
Link do modelo: **Visitar**
Loja2, duas vezes mais promoções para você!

Preço **R$2499.0**
Link do modelo: **Visitar**
Loja1 a primeira em preço baixo!!

Desenvolvido com: Jena W3C RDF W3C SPARQL