5
Tecnologias estudadas

Esse capítulo descreve as principais linguagens existentes para o desenvolvimento de ontologias para uso na *Web*. Essas linguagens são vistas como a infra-estrutura básica para o desenvolvimento da *Web* Semântica. Utilizamos algumas das linguagens apresentadas na implementação do estudo de caso nesse trabalho. O objetivo desse capítulo é fornecer uma visão geral das principais linguagens. Para uma visão mais aprofundada de cada detalhe de cada linguagem, deve-se olhar as referências utilizadas.

Iniciamos falando a respeito da linguagem XML (*eXtensible Markup Language*) e do modelo RDF (*Resource Description Framework*). Com a XML consegue-se criar uma estrutura de *tags* específica para uma família de documentos, sendo a sintaxe dessa estrutura explicitada por meio de uma DTD ou XML Schema. Já RDF fornece um modelo para descrição de metadados, sendo possível através desse modelo descrever fatos sobre recursos da *Web*.

Falamos também sobre algumas linguagens existentes para a descrição de ontologias: RDF Schema, SHOE e DAML+OIL. É através dessas linguagens que se proporciona semântica a informação descrita. Após isso, há uma discussão sobre as diferenças e capacidades dessas diferentes linguagens.

A figura 17 apresenta como essas linguagens estão organizadas em camadas. Através dessa podemos ver o papel de cada linguagem.

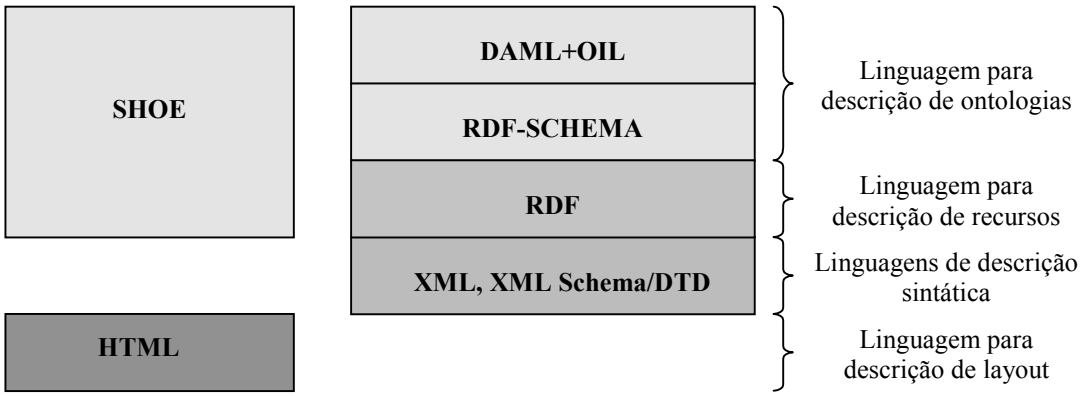


Figura 17 - Organização em camada das linguagens.

5.1. XML

A linguagem XML (*eXtensible Markup Language*) foi desenvolvida pelo XML *Working Group* e pelo consórcio W3C (*World Wide Web Consortium* – (www.w3c.org)) sendo que em 1998 a versão 1.0 da linguagem virou uma recomendação desse consórcio. A XML é um subconjunto da linguagem SGML (*Standard Generalized Markup Language*), que é mais flexível que a XML, porém mais complicada de ser utilizada.

Devido à extensibilidade proposta pela definição dessa linguagem, é possível definir uma estrutura de marcação arbitrária para descrever o conteúdo de uma família de documentos. Essa definição da estrutura está presente em um documento à parte (DTD ou XML Schema), sendo que os documentos XML de uma mesma família devem referenciá-lo. Essa estrutura não informa nada a respeito de como o conteúdo de um documento deve ser apresentado.

Em comparação com a linguagem HTML, a XML possui as seguintes vantagens:

- HTML é uma linguagem que não é extensível, ou seja, HTML fornece um conjunto de marcadores (*tags*) fixos, enquanto os marcadores de XML podem ser definidos para uma determinada família de documentos. Por exemplo, em HTML, documentos sobre carros são descritos utilizando o mesmo conjunto de *tags* utilizadas para descrever documentos sobre livros. Já em XML é possível ter conjuntos de *tags* distintas para descrever documentos sobre carros e livros.
- Enquanto os documentos em HTML são estruturados de forma a descrever como devem ser apresentados, os documentos em XML são estruturados conforme o conteúdo que o documento descreve. Colocando de outra forma, o documento XML separa a estrutura da informação do seu *layout* de apresentação. A apresentação de um documento XML é feita usando o mecanismo de folhas de estilo descrito na linguagem XSL (*eXtensible Stylesheet Language*).
- Devido ao fato de o documento XML possuir uma estrutura adequada ao conteúdo que descreve (estrutura intrínseca do conteúdo

que descreve), torna-se muito mais fácil e precisa a tarefa de busca por informação. Enquanto em documentos HTML só é possível fazer buscas por palavras-chave, em documentos XML é possível fazer buscas por palavras em campos específicos da estrutura do conteúdo do documento. Isso se deve também ao fato de a XML fornecer uma descrição formal da estrutura da informação dos documentos por meio de uma DTD ou XML Schema.

5.1.1. Visão geral de um documento XML (Martin et al., 2001; Holzner, 2001; Bray et al., 1998)

Como XML é uma linguagem de marcação, um documento XML é formado basicamente por marcações e dados de caracteres. A estrutura de um documento XML é dada com o uso de marcação.

As marcações em um documento expõem a sua estrutura, sendo essas utilizadas para interpretar o documento. São marcações de XML:

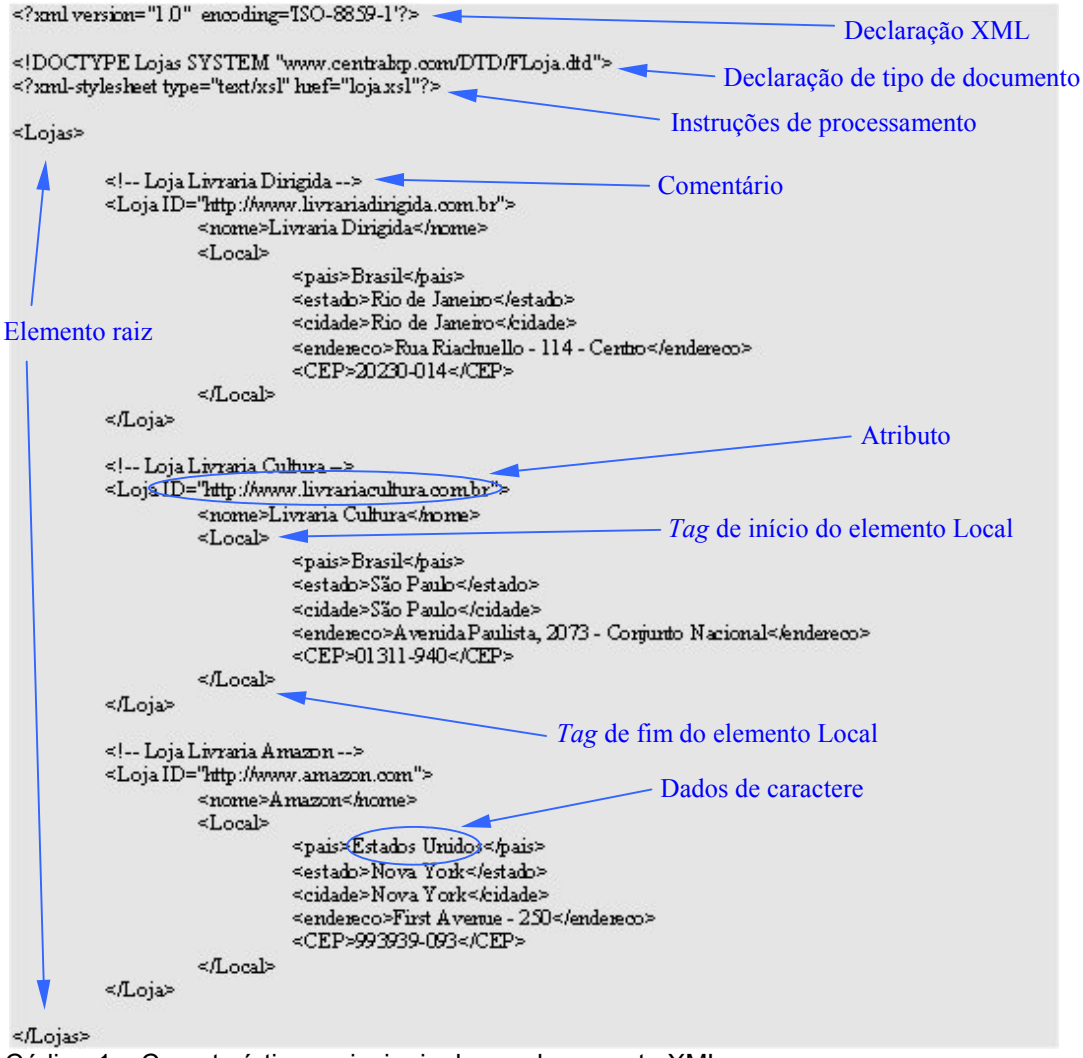
- *Tags* de início – A *tag* de inicial é responsável por delimitar o início de um elemento XML. Ela é formada por um nome do tipo do elemento XML envolto por "<" ">". Muitas vezes, além do nome do tipo do elemento, estão presentes também um ou mais atributos dentro do "<" ">". O atributo de um elemento será visto mais à frente no texto. O nome do tipo do elemento deve seguir as restrições impostas na especificação de XML (Bray et al., 1998).
- *Tags* de fim – A *tag* final é responsável por delimitar o fim de um elemento XML. Ela é formada pelo nome do tipo do elemento XML a ser delimitado envolto por "</" ">". Por XML ser sensível a letras maiúsculas e minúsculas, o nome do tipo do elemento que aparece na *tag* de início e fim devem ser idênticos. Por exemplo, para a *tag* inicial "<*Carro*>" é necessário uma *tag* final "</*Carro*>" e não *tags* como "</*carro*>", "</*CARRO*>" ou "</*CaRro*>".
- *Tags* de elemento vazio – Os elementos vazios possuem apenas uma única *tag* formada pelo nome do elemento envolto por "<" "/>". Assim como nas *tags* de início, também é possível que as *tags* de elemento vazio venham acompanhadas de atributos.

- Referências à entidade – Permitem a inserção de qualquer *string* literal no conteúdo de elementos ou valores de atributos, assim como oferecem alternativas mnemônicas para referências de caracteres. Elas são formadas por um nome legal para XML, precedidas por "*&*" e seguidas de "*;*". As entidades podem ser uma das cinco predefinidas na especificação XML, ou devem ser definidas dentro da declaração de tipo de documento, por meio do subconjunto interno ou externo.
- Referências de caractere – São utilizadas como substitutas para as formas literais de um caractere em locais onde o processamento direto da forma literal resultaria em uma violação das restrições de boa formação da sintaxe XML. Pode ser formada por "&#" seguido pelo número decimal que representa o caractere no Unicode permitido pelo XML, ou por "&#x" seguido pelo número hexadecimal que representa o caractere no Unicode permitido pelo XML.
- Comentários – São utilizados para colocar notas explicativas dentro do documento XML. Essas notas são ignoradas pelo *parser* XML. O comentário é um texto iniciado com "*<!--*" e finalizado com "*-->*".
- Delimitadores de seção CDATA – Através de seções CDATA é possível incluir um texto que normalmente é reconhecido como marcação XML. Ele é uma alternativa à utilização de referências a entidades para textos muito grandes, o que dificultaria a leitura do XML pelo usuário. Nas seções CDATA são mantidos dados de caractere que devem permanecer sem análise pelo *parser* XML. A seção é iniciada por "*<![CDATA*" e é finalizada por "*]]>*".
- Declarações de tipo de documento – Essa declaração é utilizada para validar o documento. É por meio dessa declaração que se especificará as restrições e o formato da estrutura do documento. Ela pode conter, um subconjunto interno, e pode também se referir a um subconjunto externo da Definição de Tipo de Documento (DTD). A declaração de tipo de documento é feita por meio da marcação "*<!DOCTYPE nome_elemento RESTRIÇÃO_ESTRUTURA >*" onde *nome_elemento* é o ponto de ligação para as restrições em que todos

os filhos na árvore de elementos dos documentos XML herdarão as restrições desse nó, *e RESTRIÇÃO_ESTRUTURA* é uma referência a uma DTD externa (subconjunto externo) ou uma descrição interna da estrutura do documento (subconjunto interno).

- Instruções de processamento - São instruções para serem utilizadas pelo processador de XML (*parser*). Essas instruções começam com "<?" e terminam com "?>", sendo que não se pode usar "<?xml?>" por ser reservado para declaração XML. Como as instruções de processamento dependem do processador XML utilizado, elas não se encontram embutidas na recomendação padrão de XML.

Os dados de caracteres de um documento XML são tudo que não é considerado marcação desse documento. O código 1 apresenta um exemplo do que são marcações e o que são dados de caractere de um documento XML.



Código 1 – Características principais de um documento XML.

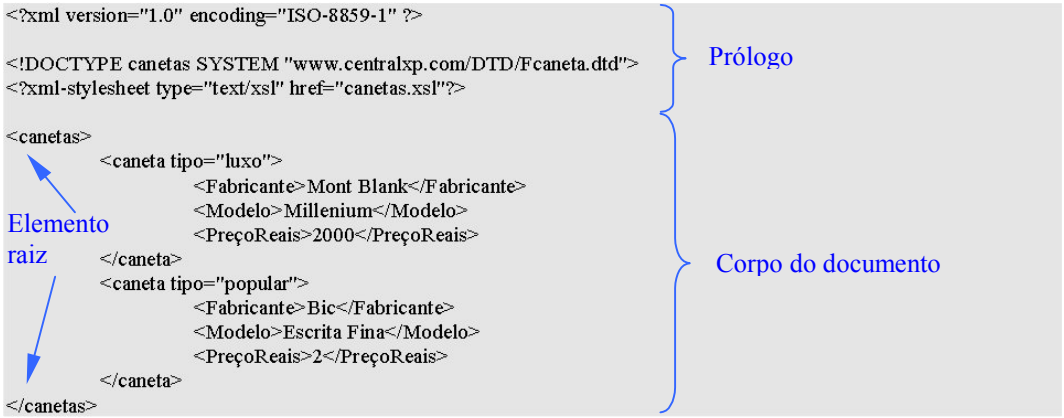
Atributos de elementos são pares de nome e valor que permitem especificar dados adicionais as *tags* de início e as *tags* de elemento vazio. Através desse artifício, é possível anexar informação a um elemento sem que essa esteja contida no elemento. Atributos são expressos das seguintes formas: *nome_atributo=”valor_atributos”* ou *nome_atributo=’valor_atributos’*.

5.1.2. Partes que formam um documento XML

Um dos requisitos para um documento XML ser considerado bem formado é que ele seja composto pelas seguintes partes:

- Um prólogo, que pode ser vazio.
- O corpo do documento, que é formado a partir de um elemento raiz.
- E uma parte miscelânea opcional (epílogo).

O código 2 mostra um documento XML e suas partes.




```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE canetas SYSTEM "www.centralxp.com/DTD/Fcaneta.dtd">
<?xml-stylesheet type="text/xsl" href="canetas.xsl"?>

<canetas>
        <caneta tipo="luxo">
                <Fabricante>Mont Blank</Fabricante>
                <Modelo>Millenium</Modelo>
                <PreçoReais>2000</PreçoReais>
        </caneta>
        <caneta tipo="popular">
                <Fabricante>Bic</Fabricante>
                <Modelo>Escrita Fina</Modelo>
                <PreçoReais>2</PreçoReais>
        </caneta>
</canetas>
```

Código 2 – Partes de um documento XML. Documento possui somente prólogo e corpo.

O prólogo do documento XML é usado para sinalizar o começo dos dados XML, descrever seu método de codificação de caracteres, e apresentar algumas dicas de configuração ao *parser* e à aplicação que se utiliza desse documento. O prólogo do documento pode ser formado pelas seguintes estruturas: declaração XML, instruções de processamento, declaração de tipo documento e comentários.

Apesar de ser opcional, a declaração XML é indicada para todos os documentos XML. A declaração XML utiliza o elemento “*<?xml?>*”. Através dessa declaração são definidas as seguintes informações: versão do XML em que o documento foi codificado, codificação de caractere utilizado no documento e informação se o documento possui alguma referência externa.

O corpo do documento XML consiste de um ou mais elementos, na forma de uma árvore hierárquica. Elementos são os blocos de construção básicos de uma

marcação XML. Um elemento funciona com um *container* dentro do qual podem existir outros elementos, referências a caractere, referência a entidades, instruções de processamento, comentários, seções de CDATA e dados de caracteres. O elemento é delimitado por uma *tag* de início e uma *tag* de fim, exceto no caso de elementos que são definidos para serem vazios, que consistem apenas de uma *tag* de elemento vazio.

Todo o documento XML bem formado precisa conter um elemento único que contém todos os outros elementos do documento. Esse elemento único é conhecido como elemento raiz. Para os outros elementos contidos dentro do elemento raiz, é importante que se preste atenção no aninhamento correto desses elementos (a *tag* inicial e final do elemento filho deve ocorrer entre a *tag* inicial e final do elemento pai) de forma a garantir a boa formação do documento. Seguindo o aninhamento correto, um documento XML forma uma árvore de elementos tendo o elemento raiz como a raiz da árvore.

A parte do epílogo do documento XML pode incluir comentários, instruções de processamento e espaços em branco. Ele é opcional, sendo até mesmo indicada a não utilização do epílogo. Isso se deve ao fato de que, como em XML não há um indicador de fim de documento, a maioria das aplicações usa a *tag* final do elemento raiz do documento como sinalização de fim de arquivo, fazendo com que o epílogo nem seja lido.

Após termos visto as partes que compõem um documento XML, é interessante falarmos a respeito do que é considerado um documento bem formado. Segundo a especificação de XML (Bray et al., 1998) um documento de texto é considerado um documento XML bem formado se ele atende aos seguintes requisitos:

- Tomando como um todo, ele corresponde à produção intitulada *document*. Essa produção na verdade é a estrutura de partes supracitadas.
- Ele atende a todas as restrições de boa formação dadas na especificação de XML (Bray et al., 1998).
- Cada uma das entidades analisadas, referenciada direta ou indiretamente dentro do documento, é bem formada.

Uma confusão muito comum quando se aprende a linguagem XML é a diferença entre documento bem formado e documento válido. Documento bem

formado é o documento que segue a especificação da linguagem XML (Bray et al., 1998), enquanto um documento válido é aquele cuja sintaxe foi verificada com sucesso em relação ao seu DTD ou XML Schema. Dessa forma para um documento ser válido, é necessário que ele possua uma referência a uma definição de tipo de documento (DTD) ou XML Schema associado a ele, e que esse documento esteja de acordo com essa DTD ou esse XML Schema.

Resumidamente, algumas observações práticas são interessantes para a criação de documentos XML bem formados (Holzner, 2001):

- Uma declaração XML deve iniciar o documento.
- Incluir um ou mais elementos no documento.
- Incluir *tags* de início e fim para elementos que não estejam vazios.
- Fechar *tags* vazias com "/>".
- O elemento raiz deve conter todos os outros elementos.
- Aninhar corretamente os elementos.
- Usar nomes de atributos exclusivos.
- Usar apenas as cinco referências de entidades preexistentes.
- Delimitar os valores de atributos com aspas ou apóstrofes.
- Usar < e *&* somente para iniciar *tags* e entidades.

5.1.3. Tecnologias auxiliares a XML

Após a publicação da recomendação da versão 1.0 da linguagem XML padrão na W3C, outras tecnologias apareceram para dar suporte e adicionar valor à linguagem XML. Algumas dessas tecnologias são: XML Schema, Namespaces, XSLT, XLink, etc. Falaremos rapidamente a respeito das duas primeiras tecnologias por serem importantes para o entendimento das linguagens que seguem. Falaremos também sobre a DTD.

5.1.3.1. DTD e XML Schema

Devido ao fato de a XML ser uma linguagem extensível, deve existir algum meio de verificar que um determinado documento XML contenha somente os elementos e atributos que estejam de acordo com a família de documentos a que pertence. Conforme visto acima esse processo se chama validação do documento. Existem duas formas para validação de documentos XML: através de uma DTD

(*Document Type Definition*), que é especificada juntamente com a recomendação da versão 1.0 de XML, e mais recentemente através de XML Schema.

Uma DTD é um documento que especifica que elementos e atributos podem ser incluídos em uma família de documentos XML, e também as relações que esses elementos e atributos podem ter entre si. Por exemplo, através de uma DTD é possível especificar que se pode ter um ou mais livros em uma lista de livros, porém que cada livro tenha somente um nome.

Os principais controles que uma DTD pode exercer sobre um documento XML são:

- Controle em relação à construção de elementos – É possível restringir o conteúdo de um elemento pai, de forma a controlar que elementos filhos podem aparecer dentro desse elemento pai e se pode haver repetição de elementos filhos. Nas DTDs, a *tag* "*ELEMENT*" é utilizada para fazer restrições em relação ao conteúdo de elementos de um documento XML.
- Controle em relação aos atributos dos elementos – É possível controlar que atributos pertencem a cada elemento. Através desse controle é possível restringir: se o atributo é opcional ou não, se o valor do atributo deve ser único no documento (ser um identificador), etc. Nas DTDs, a *tag* "*ATTLIST*" é utilizada para fazer restrições sobre os atributos dos elementos de um documento XML.
- Definir as entidades que podem ser usadas no documento – É possível declarar pedaços de conteúdo de forma a referenciá-los posteriormente no documento XML. Nas DTDs, a *tag* "*ENTITY*" é utilizada para fazer declarações de conteúdo reutilizável em documentos XML.
- Controle em relação a conteúdos externos – É possível fazer declarações de formato para conteúdo externo que não deve ser analisado em termos de sintaxe, e definir que aplicação externa que lida com esse conteúdo (Holzner, 2001). Nas DTDs, a *tag* "*NOTATION*" é utilizada para fazer declarações de formatos de dados externos que se deseja ligar a um documento XML.

Da mesma forma que a DTD, XML Schema fornece um meio de especificar que elementos e atributos podem ser incluídos em um documento XML. XML Schema permite as mesmas construções de uma DTD, porém há algumas diferenças entre as duas abordagens que fazem com que XML Schema seja uma melhor opção. As principais vantagens de XML Schema sobre DTD são:

- XML Schemas são documentos XML, portanto alguém que já saiba XML pode mais facilmente ler e criar XML Schemas.
- Com XML Schema é possível definir o tipo de dado de um elemento ou atributo. Já em uma DTD isso não é possível, pois ele considera que todo o pedaço de dado é somente texto.
- Escrever um XML Schema é mais simples do que escrever uma DTD. Isso se deve principalmente porque a DTD é uma herança da linguagem SGML, que é mais complexa que XML.
- XML Schema pode ser estendido por ser um documento XML enquanto a DTD não pode ser estendida.
- É possível validar um documento XML utilizando mais de um XML Schema através do uso de namespace. Já se utilizada uma DTD, a validação só pode ser executada especificando-se uma única DTD por documento XML.
- A DTD não permite herdar a sintaxe de outros esquemas, já XML Schema permite.

É importante salientar nesse ponto do trabalho que tanto DTD como XML Schema não são linguagens de representação de ontologia. Quando se fala que uma DTD ou um XML Schema define um vocabulário, está se falando a respeito da definição sintática desse vocabulário, ou seja, das construções permitidas dentro desse vocabulário. Tanto DTD como XML Schema não têm a capacidade de definir semântica para uma família de documentos XML.

5.1.3.2. Namespace

Outra tecnologia que visa estender a linguagem padrão de XML é a tecnologia de namespace. Quando existe um documento XML contendo múltiplos vocabulários de marcação, pode aparecer o problema de colisão entre *tags*, caso haja nos vocabulários *tags* com o mesmo nome. Por exemplo, suponha que um

documento XML faça uso de um vocabulário (*XML Schema*) sobre pessoas e um vocabulário (*XML Schema*) sobre casas, e que em ambos apareça à definição da *tag* <*endereço*>. Caso esse documento XML faça referência a essa *tag*, ocorrerá o problema de conflito na hora de validar o documento XML, pois devido ao fato de a *tag* aparecer em ambos os vocabulários o *parser* não saberá qual dos vocabulários seguir para executar a validação.

Visando resolver esse problema de sobreposição de vocabulários, foi desenvolvida a tecnologia de namespace. Utilizando namespace é possível que antes de cada *tag* ou nome de atributo se inclua o prefixo do namespace seguido de ":", visando assim eliminar as ambigüidades.

A forma de declaração de um namespace é através do atributo "*xmlns : prefixo = URI*" onde: "*xmlns"* é uma palavra reservada da recomendação de namespace em XML (Bray et al., 1999), "*prefixo"* é o prefixo a ser usado para referenciar o namespace, e "*URI"* é um URI (*Uniform Resource Identifier*) que deve ser um identificador exclusivo e que pode direcionar o processador XML a um *XML Schema* específico do *namespace* (mas não necessariamente). Esse atributo pode aparecer juntamente com qualquer *tag* inicial do documento, porém é muito comum que as declarações de namespaces sejam feitas na *tag* inicial do elemento raiz do documento.

Uma alternativa de declaração é fazer uma declaração de uma namespace padrão para o escopo de um elemento e todos os seus descendentes. Nesse caso a *tag* do elemento deve possuir o atributo "*xmlns = URI*", onde não há prefixo no atributo.

Depois de feita as declarações dos namespaces, para referenciar um namespace basta adicionar antes de uma *tag* ou nome de atributo o prefixo da *namespace* seguido de ":". O trecho de código 3 apresenta um exemplo do uso de *namespaces*.

```
<Lojas xmlns="www.centralxp.com/Loja"
       xmlns:loc="www.centralxp.com/Local">

    <!-- Loja Livraria Dirigida -->
    <Loja ID="http://www.livrariadirigida.com.br">
        <nome>Livraria Dirigida</nome>
        <loc:Local>
            <loc:pais>Brasil</loc:pais>
            <loc:estado>Rio de Janeiro</loc:estado>
            <loc:cidade>Rio de Janeiro</loc:cidade>
            <loc:endereco>Rua Riachuello - 114 - Centro</loc:endereco>
            <loc:CEP>20230-014</loc:CEP>
        </loc:Local>
    </Loja>
</Lojas>
```

Declaração da namespace padrão

Declaração da namespace "loc"

Uso do namespace

Código 3 – Exemplo de utilização de namespaces.

Conforme visto, a tecnologia de namespace permite a utilização de mais de um vocabulário em um mesmo documento XML. Um ponto importante a ser salientado é que só é possível fazer a validação desse documento XML para esses vocabulários se eles forem expressos utilizando XML Schema, e o *parser* tenha suporte a XML Schema. As DTDs não se encontram preparadas para essa tarefa, uma vez que a recomendação XML 1.0 não permite mais de uma DTD por documento.

Essas são as principais tecnologias associadas a XML que são utilizadas nesse trabalho. Demos aqui uma visão geral dessas tecnologias, pois não faz parte do escopo desse trabalho explicar detalhes dessas, bastando apenas explicar as funcionalidades proporcionadas. Uma discussão mais aprofundada sobre essas e outras tecnologias associadas a XML pode ser visto em (Martin et al., 2001; Holzner, 2001).

5.2. RDF (Martin et al., 2001; Holzner, 2001; Swick & Lassila, 1999)

O RDF (*Resource Description Framework*) é a base para o processamento de metadados. Esse padrão prevê interoperabilidade entre aplicações que trocam informações processáveis por máquinas na *Web*. O RDF provê facilidades em permitir o processamento automatizado de recursos da *Web* (Swick & Lassila, 1999).

É importante salientar que RDF não é uma linguagem, mas sim um modelo para representar dados sobre recursos da *Web*. Esse tipo de dado sobre dados (recursos da *Web*) é que é chamado de metadado.

Apesar de não ser uma linguagem, e sim um modelo, a especificação de RDF (Swick & Lassila, 1999), que é uma recomendação da W3C desde fevereiro de 1999, propõe além do modelo uma sintaxe para sua especificação baseada em XML.

Pode-se ver que RDF e XML são complementares. RDF proporciona um modelo para descrição de metadados, sendo que esse modelo não fixa necessariamente nada sobre a sintaxe que será utilizada para exprimir esse modelo em uma forma codificada. É nessa hora que entra XML fornecendo uma sintaxe de forma a permitir armazenar instâncias do modelo dentro de arquivos processáveis e permitir a troca dessas instâncias entre as aplicações. Apesar de a especificação de RDF propor a sintaxe de XML para exprimir o modelo, ela não é a única sintaxe possível.

Por outro lado, XML não fornece capacidade nenhuma de descrever a semântica do seu conteúdo por meio de um DTD ou XML Schema. Já RDF fornece essa capacidade através de RDF Schema, que é a próxima linguagem a ser vista.

5.2.1. Modelo básico

O modelo de RDF proporciona uma forma de descrição de recursos. Esses recursos são descritos através de relações binárias entre recursos ou literais (*strings*), formando assim declarações.

O modelo de dados básico de RDF consiste de três tipos de objetos:

- Recursos (*Resource*) – Todas as coisas que são descritas por uma expressão RDF são chamadas de recursos. Os recursos são sempre identificados por meio de uma URI, sendo que qualquer coisa pode ter uma URI. Um recurso pode ser por exemplo: uma página específica da *Web*, um conjunto de páginas, ou até mesmo objetos que não são diretamente acessíveis via *Web* (por ex.: CDs).
- Propriedades (*Properties*) – Uma propriedade é um aspecto, característica, atributo, ou relação usada para descrever um recurso. Cada propriedade tem um significado específico, define seus valores permitidos, os tipos de recursos que pode descrever, e sua relação com outras propriedades. As definições dessas características são feitas através de RDF Schema.
- Declarações (*Statements*) – Uma declaração, ou tripla, acontece quando um valor é atribuído a um recurso específico por meio de uma propriedade. Uma declaração é formada de três partes: sujeito, predicado e objeto. O sujeito da declaração é o recurso que está sendo descrito. O predicado da declaração é a propriedade que está sendo valorada. E o objeto é o valor do predicado específico para o sujeito descrito. O sujeito e o predicado da declaração são sempre recursos, porém o objeto da declaração pode ser tanto um recurso como também um literal (*string* ou outro tipo de dado primitivo definido em XML).

Uma forma bem didática para representar declarações RDF é por meio de um diagrama de grafo dirigido. Nesse diagrama os nós ovais representam recursos, os nós retangulares representam literais, e os arcos dirigidos representam propriedades. O arco dirigido sempre parte do sujeito e aponta para o objeto da declaração RDF.

Um exemplo de uma declaração RDF é a seguinte:

"Paulo Coelho é autor do recurso www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=310919"

É importante observar, que por uma questão didática, o predicado da declaração não é representado através de uma URI apesar de o mesmo ser um recurso. Na prática esse predicado deve ser uma URI (formado através do nome da propriedade e o namespace da URI do RDF Schema que define a propriedade).

A figura 18 apresenta a declaração feita acima na forma gráfica.

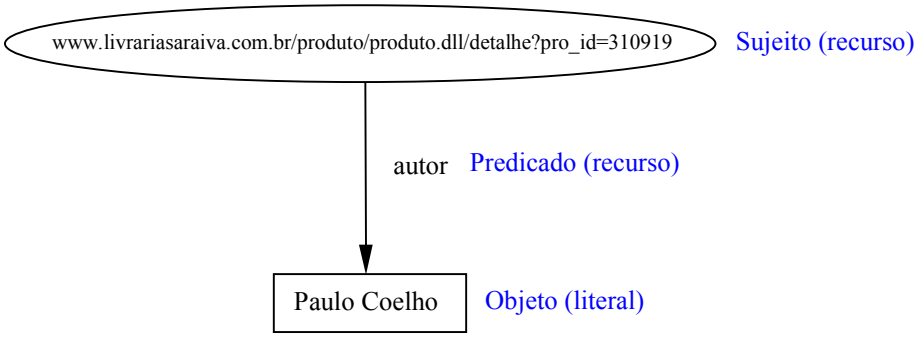


Figura 18 – Representação gráfica de uma declaração RDF.

Na declaração acima é feito um relacionamento entre um recurso e o nome de um autor, que é um literal. Para fazer uma declaração com uma estrutura mais complexa fornecendo mais informações sobre o autor, deve-se criar um recurso que vai receber todas as propriedades do autor. Esse recurso anônimo deve possuir uma URI que o identifique. Um exemplo de expressão com uma estrutura mais complexa é a seguinte:

"O recurso de nome Paulo Coelho que tem uma homepage http://www.paulocoelho.com/ é autor do recurso www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=310919"

A figura 19 apresenta a declaração feita acima na forma gráfica.

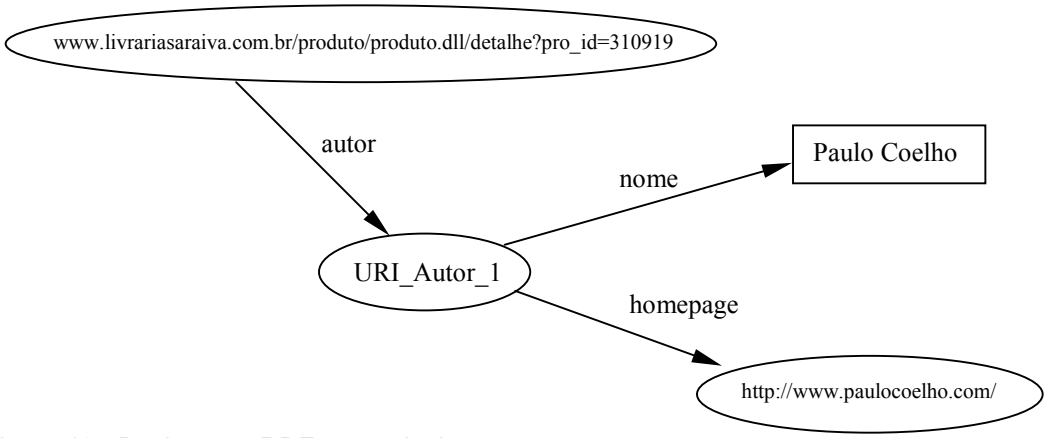


Figura 19 –Declaração RDF com relacionamento entre recursos.

Podemos observar através desses exemplos que o modelo de dados de RDF forma um grafo de declarações com relações entre recursos e literais. Assim sendo, duas expressões RDF são equivalentes se e somente se suas representações no modelo de dados são iguais, ou seja, as duas expressões geram o mesmo grafo com os mesmos elementos.

5.2.2. Sintaxe básica

Conforme falado anteriormente, a especificação RDF (Swick & Lassila, 1999) utiliza XML como meio de codificar o modelo RDF. Adicionalmente a XML padrão, a especificação RDF requer que se utilize também a tecnologia de XML namespace, de forma a associar cada propriedade com o esquema que define essa propriedade.

A especificação propõe duas sintaxes XML para a codificação de uma instância do modelo de dados RDF. A sintaxe serializada expressa a capacidade total do modelo de dados de uma maneira muito regular. A sintaxe abreviada inclui construções adicionais que fornecem uma forma mais compacta para representar um determinado subconjunto do modelo de dados. É esperado que os interpretadores de RDF implementem ambas as sintaxes. Sendo assim, os autores de documentos RDF estão livres para misturar essas duas sintaxes (Swick & Lassila, 1999).

5.2.2.1. Sintaxe serializada básica

Devido ao fato de todo documento RDF ser também um documento XML, a primeira coisa que deve aparecer no prólogo do documento é a declaração XML. O corpo do documento deve ter um elemento raiz específico, <RDF>, que delimitará o que deve ser mapeado dentro do modelo de dados RDF.

Além da linguagem XML padrão, os documentos RDF utilizam a tecnologia de XML namespace para associar uma determinada propriedade com o Schema RDF que a define.

O namespace (http://www.w3.org/1999/02/22-rdf-syntax-ns#) é definido como o namespace padrão para as propriedades e classes definidas na especificação de RDF (Swick & Lassila, 1999). Dessa forma, todas as *tags* dos elementos e atributos definidos na especificação de RDF (Swick & Lassila, 1999) devem ser precedidos pelo prefixo desse namespace, que tipicamente é definido como "*rdf*".

O código 4 apresenta o prólogo e o elemento raiz de um documento RDF típico.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        .
        .
        .
        .
</rdf:RDF>
```

Código 4 – Prólogo e elemento raiz de um documento RDF típico.

Dentro do elemento "*RDF*" do documento é feita a descrição dos diversos recursos. Para fazer essas descrições é utilizado o elemento "*<rdf:Description>*" que pode ser entendido como simplesmente um local que possui a identificação do recurso que está sendo descrito, ou seja, o sujeito de uma declaração. Dentro desse elemento, pode-se fazer mais de uma afirmação sobre o mesmo recurso identificado. Esse elemento pode possuir vários atributos:

- *about* – Através do atributo *about* é possível especificar qual recurso o elemento descreve. Através dele faz-se referência a um recurso já existente.
- *aboutEach* – Através do atributo *aboutEach* é possível à criação de declarações individuais para cada um dos recursos de uma coleção.
- *aboutEachPrefix* – Através do atributo *aboutEachPrefix* é possível identificar um conjunto de recursos que fazem parte de uma coleção por seu prefixo.
- *ID* – Através do atributo *ID* é possível atribuir um identificador a um elemento. Esse atributo sinaliza a criação de um novo recurso dentro do modelo.
- *bagID* – Através do atributo *bagID* é possível especificar o identificador de um recurso *container*, ou seja, um recurso que representa uma coleção.
- *type* – Através do atributo *type* é possível especificar o tipo de descrição.

Todos esses atributos são opcionais, sendo que os atributos *about, aboutEach, aboutEachPrefix* e *ID* são mutuamente exclusivos em uma descrição.

Dentro de um elemento, que descreve um recurso sujeito, pode existir um ou mais elementos que descrevem propriedades desse recurso sujeito. A sintaxe de um elemento, que descreve uma propriedade, pode variar conforme o valor do recurso a ser atribuído ao recurso sujeito.

Caso o objeto seja um literal ou deseje-se criar uma nova descrição dentro da propriedade, a sintaxe de um elemento de propriedade é dada por "< *nome_propriedade* > *valor* </*nome_propriedade*>", onde *nome_propriedade* é o nome da propriedade juntamente com o prefixo do namespace do esquema que a descreve e *valor* é um literal (*string*) que valora a propriedade, ou outra descrição, fazendo assim um aninhamento de descrições.

Caso se queira valorar a propriedade com outro recurso já criado, a sintaxe de um elemento de propriedade é dada por "<*nome_propriedade resource= URI_referencia/>*", onde: *nome_propriedade* é o nome da propriedade juntamente com o prefixo do namespace do esquema que a descreve, *resource* é o atributo para indicar o identificador que a propriedade valora (da mesma forma que o atributo *about* faz o elemento de descrição) e *URI_referencia* é a URI que identifica o recurso que será o objeto da declaração.

Para exemplificar um documento RDF, o código 5 apresenta na sintaxe proposta o modelo RDF descrito pela figura 19. Uma observação a ser feita é que na figura 19 as propriedades não estão prefixadas com o namespace do esquema que as define, já no código 5 isso é corrigido.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:liv="http://www.livraria.com.br/arquivos/ontologiaLivro.xml#">

            <!-- Descrição do recurso de identificador "URI_Autor_1" -->
            <rdf:Description rdf:ID="URI_Autor_1">
                        <liv:nome>Paulo Coelho</liv:nome>
            <liv:homepage resource="http://www.paulocoelho.com/"/>
            </rdf:Description>

            <!-- Descrição do recurso de identificador "http://www.paulocoelho.com/"  -->
            <rdf:Description rdf:ID="http://www.paulocoelho.com/ "/>

            <!-- Descrição do recurso de identificador "www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=
310919"  -->
            <rdf:Description rdf:ID="www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=310919">
            <liv:autor rdf:resource="URI_Autor_1"/>
            </rdf:Description>

</rdf:RDF>
```

Código 5 – Documento RDF da figura 19.

5.2.2.2. Sintaxe abreviada básica

Enquanto a sintaxe serializada apresenta a estrutura do modelo RDF mais claramente, freqüentemente se deseja usar uma forma mais compacta de XML. A sintaxe abreviada de RDF realiza isso (Swick & Lassila, 1999). Três formas de abreviação são definidas nessa seção.

A primeira abreviação é usada para propriedades que não são repetidas dentro de um elemento *Description* e onde os valores dessas propriedades são literais. Nesse caso, as propriedades podem ser escritas como um atributo XML dos elementos *Description* (Swick & Lassila, 1999). O código 6 apresenta um exemplo de um código serializado e o mesmo código tendo esse tipo de abreviação.

Versão serializada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:pes="http://descricao.org/schemaPessoa.xml#">
          <rdf:Description rdf:ID="URI_Autor_1">
                    <pes:nome>Paulo Coelho</pes:nome>
                    <pes:email>pauloc@bol.com.br</pes:email>
          </rdf:Description>
</rdf:RDF>
```

Versão abreviada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:pes="http://descricao.org/schemaPessoa.xml#">
          <rdf:Description rdf:ID="URI_Autor_1"
                    pes:nome = “Paulo Coelho”
                    pes:email = “pauloc@bol.com.br” />
</rdf:RDF>
```

Código 6 – Versão serializada e versão abreviada de um código.

A segunda abreviação é usada para declarações específicas quando o objeto da declaração é outro recurso e o valor de qualquer propriedade desse segundo recurso é *string*. Nesse caso, as propriedades do recurso no elemento *Description* aninhado pode ser escrito como atributos XML do elemento de propriedade do elemento *Description* que o contém (Swick & Lassila, 1999). O código 7 apresenta um exemplo de um código *serializado* e o mesmo código tendo esse tipo de abreviação.

Versão serializada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:liv="http://www.livraria.com.br/arquivos/ontologiaLivro.xml#"
          xmlns:pes="http://descricao.org/schemaPessoa.xml#">
          <rdf:Description rdf:about = "www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=310919">
                    <liv:autor>
                              <rdf:Description rdf:about = "URI_Autor_1">
                                        <pes:nome>Paulo Coelho</pes:nome>
                                        <pes:email>pauloc@bol.com.br</pes:email>
                              </rdf:Description>
                    </liv:autor>
          </rdf:Description>
</rdf:RDF>
```

Versão abreviada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:liv="http://www.livraria.com.br/arquivos/ontologiaLivro.xml#"
          xmlns:pes="http://descricao.org/schemaPessoa.xml#">
          <rdf:Description rdf:about = "www.livrariasaraiva.com.br/produto/produto.dll/detalhe?pro_id=310919">
                    <liv:autor rdf:resource = "URI_Autor_1"
                              pes:nome = “Paulo Coelho”
                    pes:email = “pauloc@bol.com.br” />
          </rdf:Description>
</rdf:RDF>
```

Código 7 – Versão serializada e versão abreviada de um código.

A terceira abreviação é usada para o caso em que o elemento *Description* contenha uma propriedade *type*. Nesse caso, o tipo de recurso definido no esquema correspondendo ao valor da propriedade *type* pode ser usado diretamente como o nome do elemento (Swick & Lassila, 1999). O código 8 apresenta um exemplo de um código *serializado* e o mesmo código tendo esse tipo de abreviação. Esse tipo de abreviação é muito comum na literatura e será muito usada nesse trabalho.

Versão serializada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:pes="http://descricao.org/schemaPessoa.xml#">
             <rdf:Description rdf:ID="URI_Autor_1">
                          <rdf:type rdf:resource="http://descricao.org/schemaPessoa/Pessoa"/>
                          <pes:nome>Paulo Coelho</pes:nome>
                          <pes:email>pauloc@bol.com.br</pes:email>
             </rdf:Description>
</rdf:RDF>
```

Versão abreviada

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:pes="http://descricao.org/schemaPessoa.xml#">
             <pes:Pessoa rdf:ID="URI_Autor_1">
                          <pes:nome>Paulo Coelho</pes:nome>
                          <pes:email>pauloc@bol.com.br</pes:email>
             </pes:Pessoa>
</rdf:RDF>
```

Código 8 – Versão serializada e versão abreviada de um código.

5.2.3. Coleções

Freqüentemente é necessário fazer referência a uma coleção de recursos, ou seja, fazer uma declaração não entre recursos, mas entre uma coleção desses. Por exemplo, caso se deseje afirmar que um projeto tem como integrantes um conjunto de pessoas. Visando satisfazer essa necessidade o modelo RDF introduz o conceito de RDF *containers*. Através dos *containers* é possível fazer referência a um conjunto de recursos ou literais.

O RDF define três tipos de *containers*:

- *Bag* – É uma lista desordenada de recursos ou literais. *Bags* são usadas para declarar que uma propriedade tem múltiplos valores e que não há nenhum significado na ordem em que os valores são dados. Valores duplicados são permitidos (Swick & Lassila, 1999). Um exemplo é o caso de uma lista de alunos onde a ordem de seus nomes não é importante. Devido a um recurso (sujeito) poder ter várias outras propriedades (predicados) iguais fazendo referência a outros recursos ou literais (objetos) em várias declarações, muitas

vezes ocorre uma confusão entre qual é a melhor decisão de projeto: se é utilizada uma propriedade que referencia uma *Bag* com vários recursos, ou se são utilizadas várias propriedades, sendo que cada uma referencia um único recurso. Uma discussão a este respeito pode ser vista na especificação do RDF (Swick & Lassila, 1999).

- *Sequence* – É uma lista ordenada de recursos ou literais. *Sequence* é usada para declarar que uma propriedade tem múltiplos valores e a ordem em que esses valores são dados tem importância. Valores duplicados são permitidos (Swick & Lassila, 1999). Um exemplo de seu uso é para controlar uma fila de processos que desejam partilhar um recurso, onde a ordem é importante.
- *Alternative* – É uma lista de recursos ou literais que representam alternativas para o único valor de uma propriedade. Uma aplicação usando uma propriedade, em que o valor é uma coleção desse tipo, deve estar ciente que ela pode escolher apenas um dos itens na lista (Swick & Lassila, 1999). Um exemplo de uso é para definir uma das várias formas de pagamento em uma operação de compra e venda.

A figura 20 apresenta a representação gráfica da seguinte sentença:

"O recurso www.wrox.com/id_professional_xml tem autores de nomes Didier Martin, Mark Birbeck, Michale Kay e Brian Loesgen"

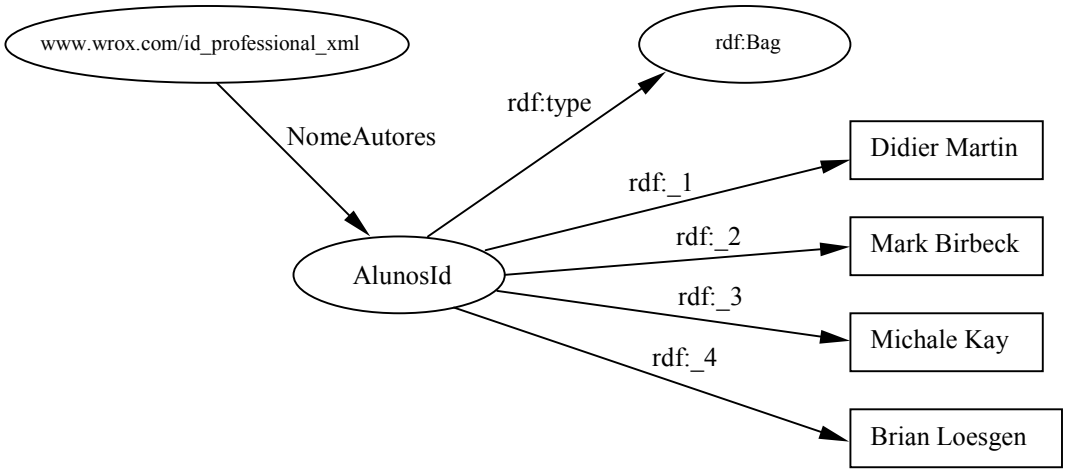


Figura 20 – Declaração RDF que utiliza coleções.

Para representar uma coleção de recursos, RDF usa um recurso adicional para identificar a coleção específica. Esse recurso deve ser declarado para ser uma instância de um dos tipos de *containers* apresentados. A propriedade *rdf:type* é usada para fazer essa declaração. As relações de participação entre o recurso

container e os recursos que pertencem à coleção são definidos através das propriedades "*rdf:_1*", "*rdf:_2*", "*rdf:_3*", etc, que são criadas somente para esse propósito. O recurso *container* pode ter outras propriedades além das definidas pela propriedade de participação e a propriedade *type*. Essas outras propriedades visam dar suporte a referências distributivas sobre os membros do *container*, e também a definição de *containers* por meio de padrões de URI (Swick & Lassila, 1999).

Outras duas variações sobre a coleção de tipo *Bag* é a referência distributiva entre membros do *container* e *containers* definidos por meio de padrões de URI. O primeiro, ao invés de instanciar uma única afirmação entre o sujeito e o recurso *container* (objeto), cria várias afirmações entre o sujeito e cada membro da coleção não existindo assim o recurso *container*. Já o segundo é uma sintaxe abreviada que representa uma instância de *Bag* na qual os seus membros são por definição todos os recursos que possuem o identificador de recurso começando com uma *string* específica.

Segundo a especificação do RDF (Swick & Lassila, 1999), para especificar uma coleção em um documento RDF, deve-se começar criando um elemento em que se declarará todos os recursos e literais pertencentes à coleção. Esse elemento possui o formato "*<tipo_coleção ID=URI_coleção>*" onde: *tipo_coleção* identifica o tipo da coleção (*rdf.Bag* para coleção do tipo *Bag*, *rdf.Seq* para coleção do tipo *Sequence*, e *rdf.Alt* para coleção do tipo *Alternative*), *ID* é o atributo que define o identificador da coleção, e *URI_coleção* é uma URI que identifica a coleção. Dentro desse elemento descrevem-se os recursos ou literais pertencentes à coleção por meio do elemento "*<rdf:li>*". Os códigos 9 e 10 apresentam exemplos da utilização de coleções em um documento RDF.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:liv="http://www.livraria.com.br/arquivos/ontologiaLivro.xml#">

        <rdf:Description rdf:about = "www.wrox.com/id_professional_xml">
        <liv:Genero>
                        <rdf:Bag ID="GenerosLiterarios_1">
                                <rdf:li>Informatica</rdf:li>
                                <rdf:li>XML</rdf:li>
                                <rdf:li>Computação</rdf:li>
                                <rdf:li>Programação</rdf:li>
                        </rdf:Bag>
                </liv:Genero>
        </rdf:Description>

</rdf:RDF>
```

Código 9 – Documento RDF que instância uma *Bag* de literais.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:pec="http://www.mecanica.com.br/arquivos/ontologiaCarro.xml#">

        <rdf:Description rdf:about = "www.taif.com/automovel_Golf_xp">
                <liv:Pecas>
                        <rdf:Bag rdf:ID="PecasAutomovel_Golf_xp">
                                <rdf:li rdf:resource="www.pireli.com/pneu_radial_143"/>
                                <rdf:li rdf:resource="www.cofap.com/suspensao"/>
                                <rdf:li rdf:resource="www.vargas.com/333"/>
                        </rdf:Bag>
                </liv:Pecas>
        </rdf:Description>

        <pec:Pneu rdf:ID="www.pireli.com/pneu_radial_143"/>

        <pec:Suspensão rdf:ID="www.cofap.com/suspensao"/>

        <pec:Freio rdf:ID="www.vargas.com/333"/>

</rdf:RDF>
```

Código 10 – Documento RDF que instância uma *Bag* de recursos.

5.2.4. Declaração sobre declaração RDF

Em RDF é possível, além de fazer declarações sobre recursos, fazer declarações sobre outras declarações RDF.

Para conseguir isso, deve-se criar um recurso de forma a que ele represente uma declaração XML. Esse recurso deve possuir as seguintes propriedades de forma a modelar a declaração:

- *rdf:subject* – Esta propriedade identifica o recurso que é o sujeito da declaração que está sendo modelada.
- *rdf:predicate* – Esta propriedade identifica o recurso que é o predicado da declaração que está sendo modelada.
- *rdf:object* - Esta propriedade identifica o recurso que é o objeto da declaração que está sendo modelada.

- *rdf:type* – O valor da propriedade *type* descreve o tipo de um recurso. Como o recurso modela uma declaração, seu tipo deve ser definido como *rdf:Statement*.

Um exemplo de criação de declaração sobre declaração é quando se faz a seguinte declaração sobre a declaração feita pela figura 18:

"Marcelo discorda que Paulo Coelho é autor do recurso www.livrariasaraiva.com.br/produto/produto.dll /detalhe?pro_id=310919"

Seguindo o modelo de declaração sobre declaração proposto, a figura 21 apresenta o grafo resultante.

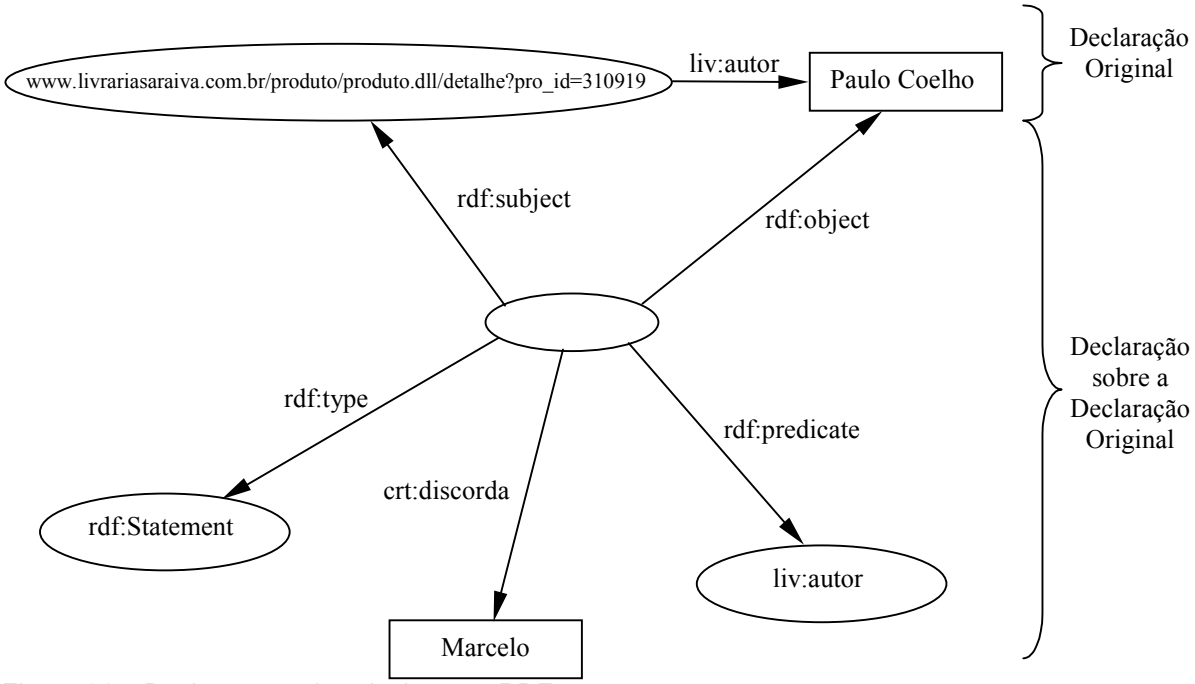


Figura 21 – Declaração sobre declaração RDF.

Nas seções seguintes começamos a falar a respeito das linguagens de descrição de ontologias analisadas para fazer o estudo de caso do trabalho.

5.3. RDF Schema

A partir dessa seção começamos a tratar especificamente de linguagens que são utilizadas para descrever ontologias. Essas linguagens apresentam um vocabulário primitivo que é utilizado para descrever o conhecimento e as restrições existentes em uma ontologia específica.

Através do vocabulário fornecido por cada linguagem é possível, em maior ou menor grau conforme a expressividade da linguagem, conseguir descrever a estrutura de uma ontologia específica. A primeira linguagem que se enquadra nesse tipo de classificação é RDF Schema.

O RDF Schema é o mecanismo através do qual é possível fornecer um sistema básico de tipos que pode ser usado nos modelos RDF. Ele é utilizado para estruturar a informação contida nos modelos RDF, visando assim fornecer a capacidade de validar os dados contidos no modelo. Com o uso de RDF Schema também é possível fazer inferências sobre as informações contidas em um modelo RDF.

A expressividade da estrutura fornecida por um documento RDF Schema é bem diferente da fornecida por um documento XML Schema ou DTD. O DTD ou XML Schema fornecem uma estrutura que visa permitir a validação da sintaxe de um documento XML contra essa estrutura. Já a estrutura fornecida pelo RDF Schema permite não só que se faça validações sobre uma instância do modelo RDF, mas também que se faça algumas inferências em cima das instâncias do modelo, pois a semântica também é fornecida nessa estrutura.

O RDF Schema é uma aplicação RDF, ou seja, RDF Schema é uma instância de um modelo RDF. Da mesma forma que RDF, o RDF Schema não está vinculado a nenhuma sintaxe específica.

Através do vocabulário básico proposto na especificação de RDF Schema é possível descrever as seguintes estruturas de uma ontologia:

- Os conceitos e a taxinomia desses conceitos. Os conceitos são descritos por meio de classes, e a taxinomia através de relacionamentos de herança.
- Os relacionamentos entre os conceitos. Os relacionamentos são descritos por meio das propriedades. É possível dizer em RDF

Schema que uma propriedade pertence a um ou mais tipos de classe, e que a propriedade pode ser valorada com um ou mais tipos de classes.

Conforme se pode observar, o vocabulário proposto por RDF Schema não possui uma grande capacidade para expressar uma ontologia. Isso acontece, pois a idéia de RDF Schema é permitir o desenvolvimento de outros vocabulários de maior expressividade, mas uma vez que esses vocabulários são construídos em uma camada superior a RDF Schema, eles possuem as mesmas noções básicas de classe e propriedade.

Apesar de a especificação de RDF (Swick & Lassila, 1999), que é uma recomendação da W3C desde 1999, fazer menção da existência de RDF Schema como um vocabulário para descrever esquemas do modelo RDF, um consenso quanto ao vocabulário dessa linguagem ainda não foi alcançado pelo grupo de discussão da W3C. O estado atual da especificação de RDF Schema dentro da W3C é de um rascunho em trabalho (*Working Draft*), o que indica que ainda não é uma especificação estabilizada. Porém, comparando as duas ultimas versões da especificação (Brickley & Guha, 2000; Brickley & Guha, 2002) pode-se observar que pouca coisa mudou em seu vocabulário.

O item seguinte proporciona uma visão geral do vocabulário proposto na última versão da especificação de RDF Schema (Brickley & Guha, 2002).

5.3.1. Vocabulário RDF Schema

O vocabulário básico proposto na especificação de RDF Schema (Brickley & Guha, 2002) é construído utilizando o modelo RDF. Através desse vocabulário básico proposto é possível fazer a modelagem de ontologias específicas (por exemplo: ontologia de livros, circuitos eletrônicos, automóveis, etc.). A especificação de RDF Schema (Brickley & Guha, 2002) utiliza XML como meio de codificar o documento RDF Schema, apesar de o vocabulário não estar necessariamente preso a nenhuma sintaxe específica (assim como o modelo RDF).

Como qualquer aplicação RDF pode ter sua estrutura descrita segundo um RDF Schema, o vocabulário básico proposto na especificação de RDF Schema também é descrito desta mesma forma (há recursividade). A descrição do vocabulário de RDF Schema está na URI padrão (http://www.w3.org/2000/01/rdf-

schema) que vem a ser o identificador do namespace do vocabulário de RDF Schema. Esse namespace tem tipicamente como prefixo "*rdfs*". As declarações RDF expressas no documento desse esquema têm o seu significado dado pela especificação de RDF Schema (Brickley & Guha, 2002).

RDF Schema fornece meios de definir classes e propriedade de forma muito similar à encontrada em linguagens orientadas a objeto. É possível, em RDF Schema, definir classes (através de *rdfs:Class*) e taxinomia entre elas (através da propriedade *rdfs:subClassOf*). É possível também definir propriedades (através de *rdf:Property*), porém diferentemente das linguagens orientadas a objeto, em RDF Schema uma propriedade define com que classes podem se relacionar, tanto em se tratando do domínio da propriedade (através da propriedade *rdfs:domain*), como também dos valores que a propriedade pode receber (através da propriedade *rdfs:range*).

Uma vez definidas as classes e as propriedades em um ou mais documentos RDF Schemas, é possível que um documento RDF faça a tipificação de seus recursos (através da propriedade *rdf:type*) baseado nesses RDF Schemas, permitido assim verificar se os relacionamentos entre os recursos estão de acordo com o esquema proposto no documento RDF Schema.

As tabelas 1 e 2 apresentam o vocabulário básico de RDF Schema e também as primitivas de RDF.

Nome da classe	Finalidade
rdfs:Resource	É a classe que representa um recurso.
rdfs:Class	É o conceito de classe. O nome da classe é dado pelos atributos *rdf:about* ou *rdf:ID*.
rdf:Property	É o conceito de propriedade. O nome da propriedade é dada pelos atributos *rdf:about* ou *rdf:ID*.
rdfs:Literal	Essa classe representa o conjunto de valores literais, ou seja, strings.
rdf:Statement	É a classe que representa uma declaração RDF.
rdfs:Container	É a classe que representa o conjunto de containers.
rdf:Bag	É a classe que representa uma coleção desordenada.
rdf:Seq	É a classe que representa uma coleção ordenada.
rdf:Alt	É a classe que representa uma coleção de alternativas.
rdfs:ContainerMembershipProperty	É a classe que indica que um recurso é uma propriedade de participação de uma coleção. Ela serve para indicar que rdf:_1, rdf:_2, ... são propriedades de sociedade de uma coleção.

Tabela 1 – Classes RDF/RDF Schema predefinidas na especificação (Brickley & Guha, 2002).

Nome da propriedade	Finalidade	Classe domínio da propriedade	Classe aceita pela propriedade
rdfs:isDefinedBy	Indica o namespace de um recurso.	rdfs:Resource	rdfs:Resource
rdf:subject	O sujeito de uma declaração RDF.	rdf:Statement	rdfs:Resource
rdf:predicate	O predicado de uma declaração RDF.	rdf:Statement	rdf:Property
rdf:object	O objeto de uma declaração RDF.	rdf:Statement	Não especificado
rdf:type	Indica a que classe um recurso pertence.	rdfs:Resource	rdfs:Class
rdfs:member	Um membro de um container.	rdfs:Container	Não especificado
rdfs:subClassOf	Indica especialização de classes, sendo que é permitido a herança múltipla. A versão atual de RDF Schema (Brickley & Guha, 2002) passou a permitir a existência de ciclos nas heranças de classes.	rdfs:Class	rdfs:Class
rdf:value	Identifica o valor principal (usualmente string) de uma propriedade quando o valor da propriedade é um recurso estruturado.	rdfs:Resource	Não especificado
rdfs:subPropertyOf	Indica especialização de propriedades. Através dessa é possível tanto criar uma nova propriedade baseada em uma propriedade anterior, como também criar uma propriedade mais específica baseada em uma propriedade mais genérica. A versão atual de RDF Schema (Brickley & Guha, 2002) passou a permitir a existência de ciclos nas heranças de propriedades.	rdf:Property	rdf:Property
rdfs:comment	Usado para fazer descrições	rdfs:Resource	rdfs:Literal
rdfs:label	Fornece uma versão de apresentação do nome do recurso.	rdfs:Resource	rdfs:Literal
rdfs:domain	Indica a classe de domínio de uma propriedade, ou seja, indica a que classe deve pertencer o sujeito de uma propriedade. Podem existir zero ou mais domínios para uma mesma propriedade.	rdf:Property	rdfs:Class
rdfs:range	Indica a classe aceita de uma propriedade, ou seja, indica a que classe deve pertencer o objeto de uma propriedade. A versão atual de RDF Schema (Brickley & Guha, 2002) passou a permitir a possibilidade de duas ou mais classes serem aceitas por uma mesma propriedade.	rdf:Property	rdfs:Class
rdfs:seeAlso	Provê informação sobre o assunto do recurso.	rdfs:Resource	rdfs:Resource

Tabela 2 – Propriedades RDF/RDF Schema predefinidas na especificação (Brickley & Guha, 2002).

Para exemplificarmos o uso de RDF Schema, vamos definir uma ontologia simples para vermos como ela é codificada. Considere que temos o conceito *Mamifero* e que o *SerHumano* é subclasse de *Mamifero*. Os conceitos *Homem* e *Mulher* são subclasses do conceito *SerHumano*. Todo *SerHumano* tem a propriedade *TerParentesco*, onde só é valido para essa propriedade receber

valores que sejam pertencentes ao conceito *SerHumano*. Todo *Homem* e *Mulher* têm a propriedade *TerPai*, que é herdada da classe *SerHumano*, e onde só é valido para essa propriedade receber valores que sejam pertencentes ao conceito *Homem*. Todo *Homem* e *Mulher* têm a propriedade *Nome*, onde só é valido para essa propriedade receber valores que sejam pertencentes à classe Literal.

Uma vez colocada a ontologia acima, o código 11 apresenta o documento RDF Schema que modela a ontologia proposta e o código 12 apresenta um documento RDF que utiliza esse esquema na criação de suas instâncias.

```
<?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 = "http://www.seres.com/seres/rdfSchemaSeres.xml#">

            <rdfs:Class rdf:ID="Mamifero">
                        <rdfs:comment>Cria a classe Mamifero</rdfs:comment>
            </rdfs:Class>

            <rdfs:Class rdf:ID="SerHumano">
                        <rdfs:comment>Cria a classe SerHumano</rdfs:comment>
                        <rdfs:subClassOf rdf:resource="#Mamifero"/>
            </rdfs:Class>

            <rdfs:Class rdf:ID="Homem">
                        <rdfs:comment>Cria a classe Homem</rdfs:comment>
                        <rdfs:subClassOf rdf:resource="#SerHumano"/>
            </rdfs:Class>

            <rdfs:Class rdf:ID="Mulher">
                        <rdfs:comment>Cria a classe Mulher</rdfs:comment>
                        <rdfs:subClassOf rdf:resource="#SerHumano"/>
            </rdfs:Class>

            <rdf:Property rdf:ID="TerParentesco">
                        <rdfs:comment>Cria a propriedade TerParentesco</rdfs:comment>
                        <rdfs:domain rdf:resource="#SerHumano"/>
                        <rdfs:range rdf:resource="#SerHumano"/>
            </rdf:Property>

            <rdf:Property rdf:ID="TerPai">
                        <rdfs:comment>Cria a propriedade TerPai</rdfs:comment>
                        <rdfs:subPropertyOf rdf:resource="#TerParentesco"/>
                        <rdfs:domain rdf:resource="Homem"/>
                        <rdfs:domain rdf:resource="Mulher"/>
                        <rdfs:range rdf:resource="Homem"/>
            </rdf:Property>

            <rdf:Property rdf:ID="Nome">
                        <rdfs:comment>Cria a propriedade Nome</rdfs:comment>
                        <rdfs:domain rdf:resource="Homem"/>
                        <rdfs:domain rdf:resource="Mulher"/>
                        <rdfs:range rdf:resource = rdfs:Literal />
            </rdf:Property>
</rdf:RDF>
```

Código 11 – Documento RDF Schema que especifica a ontologia se seres.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:seres="http://www.seres.com/seres/rdfSchemaSeres.xml#">

         <seres:Mamifero rdf:ID = "Vaca_1"/>

         <seres:Mamifero rdf:ID = "Boi_1"/>

         <seres:Homem rdf:ID = "Marcelo_1">
                  <seres:Nome>Marcelo Andrade Santos</seres:Nome>
         </seres:Homem>

         <seres:Homem rdf:ID = "Joao_2">
                  <seres:Nome>Joao Andrade Santos</seres:Nome>
         </seres:Homem>

         <seres:Mulher rdf:ID = "Maria_3">
                  <seres:Nome>Maria Andrade Santos</seres:Nome>
         </seres:Mulher>

</rdf:RDF>
```

Código 12 – Documento RDF que instância conceitos do esquema do código 11.

Na subseção seguinte veremos uma linguagem de descrição de ontologias que adiciona mais expressividade a RDF Schema.

5.4. DAML+OIL

DAML+OIL (*DARPA Agent Markup Language plus Ontology Inference Layer*) é uma linguagem de marcação semântica construída sobre RDF Schema, estendendo assim esse vocabulário, proporcionando-lhe uma maior expressividade.

DAML+OIL foi desenvolvido por um comitê conjunto dos Estados Unidos (DARPA) e da União Européia (*Information Society Technologies Programme* - IST). Ele é resultado da fusão das características de duas linguagens: a linguagem DAML-ONT que foi desenvolvida pelo programa DARPA - DAML, e a linguagem OIL que foi desenvolvida pelo projeto *On-To-Knowledge* da IST. A especificação DAML+OIL (Connolly et al., 2001b) foi submetida em 18 de Dezembro de 2001 ao W3C, especificamente ao grupo de trabalho sobre ontologias (WebONT), como um ponto inicial para o desenvolvimento de uma linguagem padrão para a descrição de ontologias.

Em 29 de Julho de 2002 a W3C publicou a especificação de OWL (*Web Ontology Language*) como um rascunho em trabalho (*Working Draft*) sendo que foi absorvida grande parte da linguagem DAML+OIL.

Assim como a especificação de RDF Schema (Brickley & Guha, 2002) fornece uma semântica específica para o vocabulário básico de RDF Schema expresso no modelo RDF, a especificação de DAML+OIL (Connolly et al., 2001b) fornece uma semântica específica para o vocabulário básico de DAML+OIL expresso no modelo RDF.

O vocabulário básico proposto na especificação de DAML+OIL herda todo o vocabulário básico proposto por RDF Schema, adicionando também novas primitivas ao vocabulário. Através desse novo vocabulário são adicionadas as seguintes capacidades para descrição da estrutura de uma ontologia:

- Capacidade de definir restrições de propriedades sobre classes (*daml:Restriction*).
- Capacidade de definir relação de equivalência entre classes, propriedades, e instâncias (*daml:sameClassAs*, *daml:samePropertyAs*, *daml:equivalentTo, daml:sameIndividualAs*).

- Capacidade de definir coleções com número de elementos finito (*daml:oneOf*).
- Capacidade de indicação da cardinalidade máxima e mínima das propriedades (*daml:maxCardinality, daml:minCardinality*).
- Capacidade de definir classes baseado na combinação de outras classes. Pode-se definir uma classe como sendo a interseção de classes, a união de classes ou o complemento de uma classe (*daml:intersectionOf*, *daml:unionOf*, *daml:complementOf*).
- Capacidade de definir uma propriedade como sendo inversa de uma outra propriedade (*daml:inverseOf*).
- Capacidade de definir que uma propriedade é transitiva (*daml:TransitiveProperty*).
- Capacidade de definir que uma propriedade tem valor distinto para cada objeto (*daml:UnambigousProperty*).
- Capacidade de definir que uma propriedade só tem um valor para cada objeto (*daml:UniqueProperty*).

5.4.1. Documento DAML+OIL

Da mesma forma que em RDF Schema, DAML+OIL fornece meios de definir classes e propriedades com a maioria das primitivas herdadas de RDF Schema, adicionado a essas outras primitivas que visam dar maior expressividade à linguagem.

O esquema do vocabulário de DAML+OIL está na URI padrão (http://www.daml.org/2001/03/daml+oil) que vem a ser o identificador do namespace do vocabulário de DAML+OIL. Esse namespace tem tipicamente como prefixo "*daml*". As declarações RDF expressas no documento desse esquema têm o seu significado dado pela especificação de DAML+OIL (Connolly et al., 2001b).

Uma ontologia descrita por um documento DAML+OIL é formada por zero ou mais cabeçalhos, seguindo de zero ou mais elementos de classes, elementos de propriedade, e instâncias.

5.4.1.1. Cabeçalho do documento DAML+OIL

O cabeçalho do documento DAML+OIL é formado pelo elemento *daml:Ontology* que contém zero ou mais informações de versão e elementos de importação.

A informação de versão é dada pelo elemento *daml:versionInfo*. Esse elemento contém uma string arbitrária que descreve a informação sobre a versão da ontologia.

O elemento de importação é dado por *daml:import*. Cada declaração de *daml:import* referência outro documento DAML+OIL que contém definições que se aplicam aos recursos existentes no documento que faz a importação. Cada declaração possui uma URI que especifica o local do documento DAML+OIL a ser importado. A declaração de importação é transitiva entre os documentos, ou seja, se o documento A importa o documento B e o documento B importa o documento C, então o documento A importa tanto B quanto C.

É importante observar a diferença entre o que o elemento *daml:import* faz e o que a definição de um namespace faz. Ao se definir um namespace cria-se uma notação abreviada para uma URI (que pode fazer referência a um documento), porém não se inclui nenhuma nova definição nesse documento. Com *daml:import* é incluído ao documento que faz o pedido de importação todas as declarações existentes no documento a ser importado. Portanto é muito comum encontrar em documentos DAML+OIL a importação dos documentos referenciados pelos namespaces.

O trecho de código 13 apresenta um cabeçalho de um documento DAML+OIL.

```
<?xml version="1.0"  encoding='ISO-8859-1'?>
<rdf:RDF
        xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

        <daml:Ontology rdf:about="">
                <daml:versionInfo>1.0</daml:versionInfo>
                <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil" />
        </daml:Ontology>
        …
</rdf:RDF>
```

Código 13 – Cabeçalho de um documento DAML+OIL.

5.4.1.2. Elemento de classe

Através do elemento de classe, dado por *daml:Class*, é permitido definir o conceito de classe. O nome da classe é dado pelos atributos *rdf:about* ou *rdf:ID*. O elemento de classe pode conter:

- Zero ou mais combinações lógicas de expressões de classe. A classe que está sendo definida deve ser equivalente à classe definida por cada expressão de classe. Uma expressão de classe é formada por uma das seguintes construções: ou o nome de uma classe (URI); ou uma enumeração, sendo essa cercada pelas *tags* <*daml:Class*> ... </*daml:Class*>; ou uma restrição de propriedade (explicado mais adiante); ou uma combinação lógica de expressões de classe, cercada pelas *tags* <*daml:Class*> ... </*daml:Class*>. As combinações lógicas de expressões de classes existentes são as seguintes: *daml:unionOf*, *daml:intersectionOf*, *daml:complementOf*.
- Zero ou mais propriedades *rdfs:subClassOf*, sendo que cada propriedade contém uma expressão de classe. Essa propriedade indica especialização de classes, sendo que é permitida a herança múltipla.
- Zero ou mais propriedades *daml:disjointWith*, sendo que cada propriedade contém uma expressão de classe. Essa propriedade afirma que a classe que está sendo definida não tem instâncias em comum com a expressão de classe contida no elemento *daml:disjointWith*.
- Zero ou mais propriedades *daml:disjointUnionOf*, sendo que cada propriedade contém uma lista de expressões de classe. Através dessa propriedade é possível afirmar que todas as classes presentes na lista de expressões de classe do elemento *daml:disjointUnionOf* não possuem interseção entre si, e que a classe que está sendo definida é a união dessas classes. Por exemplo, pode-se definir a classe *SerHumano* como sendo *daml:disjointUnionOf* da classe *Homem* e *Mulher*.

- Zero ou mais propriedades *daml:sameClassAs*, sendo que cada propriedade contém uma expressão de classe. Essa propriedade afirma que a classe que está sendo definida é equivalente à expressão de classe contida no elemento *daml:sameClassAs*.
- Zero ou mais propriedades *daml:equivalentTo*, sendo que cada propriedade contém uma expressão de classe. Essa propriedade quando aplicada a uma classe tem a mesma semântica que a propriedade *daml:sameClassAs*. É indicada a utilização de *daml:sameClassAs* em detrimento a *daml:equivalentTo* (Connolly et al., 2001b).
- Zero ou mais propriedades de enumeração dadas pela *tag daml:oneOf*. Cada propriedade de enumeração contém uma lista de objetos que são suas instâncias, permitindo assim definir classes baseado na enumeração exaustiva de seus elementos. A classe que está sendo definida através dessa propriedade contém exatamente os elementos enumerados.

A especificação de DAML+OIL (Connolly et al., 2001b) predefine dois tipos de classe: *daml:Thing* e *daml:Nothing*. Todo o objeto (instância de uma classe) é um membro de *daml:Thing*, e nenhum objeto é membro de *daml:Nothing*. Dessa forma, toda a classe é subclasse de *daml:Thing* e *daml:Nothing* é uma subclasse de toda classe (Connolly et al., 2001b).

O trecho de código 14 apresenta um exemplo com as principais construções de elementos de classe utilizados nesse trabalho.

```
<!-- Define a classe Pessoa -->
<daml:Class rdf:ID="Pessoa">
        <rdfs:label>Pessoa</rdfs:label>
</daml:Class>

<!-- Define a classe CD -->
<daml:Class rdf:ID="CD">
        <daml:subClassOf rdf:resource="http://localhost:8080/arquivos/ontologiaProduto.daml#Produto"/>
        ...
</daml:Class>
```

Código 14 – Elementos de classe.

5.4.1.3. Restrições de propriedade

Uma restrição de propriedade é um tipo especial de restrição de classe. Ela implicitamente define uma classe anônima, isso é, a classe de todos os objetos que satisfazem a restrição da propriedade.

Uma restrição é dada pelo elemento *daml:Restriction* que contém o elemento *daml:onProperty*, sendo que esse se refere ao nome da propriedade (URI) em que se aplicará a restrição, e a uma ou mais das seguintes construções:

- Elementos indicando o tipo de restrição:
  - O elemento *daml:toClass* (que contém uma expressão de classe), define a classe em que todos os objetos desta devem ter para todos os valores da propriedade restringida objetos pertencentes à expressão de classe. Por exemplo, definir que todo objeto da classe *Pessoa* tenha a propriedade *TerParente* com outros objetos da classe *Pessoa*, ou seja, toda a valoração da propriedade *TerParente* deve ser necessariamente da classe *Pessoa*.
  - O elemento *daml:hasValue* (que contém uma referência a um objeto (instância) ou a um valor), define a classe em que todos os objetos que têm como valor da propriedade pelo menos um valor equivalente ao valor especificado. Por exemplo, definir que todo objeto da classe *Homem* tem a propriedade *Sexo* com valor "Masculino".
  - O elemento *daml:hasClass* (que contém uma expressão de classe ou referência a um XML Schema *datatype*), define a classe em que todos os objetos dessa classe devem ter pelo menos um valor da propriedade restringida como membro da expressão de classe ou do XML Schema *datatype* (Connolly et al., 2001b).
- Elementos, contendo inteiros não negativos *N*, indicando restrições de cardinalidade não qualificada, ou seja, só importa a cardinalidade da propriedade sem importar tipo de classe a que pertence o valor da propriedade. Esse tipo de restrição pode ser dada pelos seguintes elementos:
  - O elemento *daml:cardinality* define a classe de todos os objetos que têm exatamente N valores distintos para a propriedade restringida. Por exemplo, definir que todo objeto da classe *Produto* tem uma e somente uma propriedade *Peso*.

- O elemento *daml:maxCardinality* define a classe de todos os objetos que têm no máximo N valores distintos para a propriedade restringida. Por exemplo, para o padrão sócio-cultural brasileiro, definir que todo objeto da classe *Homem* tem no máximo uma propriedade *Esposa*.
- O elemento *daml:minCardinality* define a classe de todos os objetos que tem no mínimo N valores distintos para a propriedade restringida. Por exemplo, definir que todo objeto da classe *CD* tem no mínimo uma propriedade *Musica*.

- Elementos, contendo inteiros não negativos *N*, indicando restrições de cardinalidade qualificada, ou seja, importa a cardinalidade da propriedade como também importa o tipo de classe a que pertence o valor da propriedade, que é indicado pelo elemento *daml:hasClassQ* (que contém uma expressão de classe ou referência a um XML Schema *datatype*). Esse tipo de restrição pode ser dada pelos seguintes elementos:
  - O elemento *daml:cardinalityQ* define a classe de todos os objetos que têm exatamente N valores distintos da propriedade restringida que são instâncias da expressão de classe ou do XML Schema *datatype*. Por exemplo, definir que todo objeto da classe *Homem* tem uma e somente uma propriedade *TerPai* que deve ser valorado necessariamente com um objeto da classe *Homem*.
  - O elemento *daml:maxCardinalityQ* define a classe de todos os objetos que têm no máximo N valores distintos da propriedade restringida que são instâncias da expressão de classe ou do XML Schema *datatype*. Por exemplo, definir que todo objeto da classe *Homem* tem no máximo uma propriedade *Esposa* que deve ser valorada necessariamente com um objeto da classe *Mulher*.
  - O elemento *daml:minCardinalityQ* define a classe de todos os objetos que têm no mínimo N valores distintos da propriedade restringida que são instâncias da expressão de classe ou do XML Schema *datatype*.

O trecho de código 15 apresenta um exemplo com as principais construções de restrições de propriedade utilizadas nesse trabalho.

```
<!-- Define a classe Livro -->
<daml:Class rdf:ID="Livro">

        <daml:subClassOf rdf:resource="http://localhost:8080/arquivos/ontologiaProduto.daml#Produto"/>
        <daml:subClassOf>
                <daml:Restriction daml:cardinality="1">
                        <daml:onProperty rdf:resource="#titulo"/>
                </daml:Restriction>
        </daml:subClassOf>

        <daml:subClassOf>
                <daml:Restriction daml:maxCardinality="1">
                        <daml:onProperty rdf:resource="#peso"/>
                </daml:Restriction>
        </daml:subClassOf>

</daml:Class>

<!-- Define a restricao Homem -->
<daml:Class rdf:ID="Homem">
        <daml:sameClassAs>
                <daml:Restriction>
                        <daml:onProperty rdf:resource="sexoPessoa"/>
                        <daml:hasValue>masculino</daml:hasValue>
                </daml:Restriction>
        </daml:sameClassAs>
</daml:Class>
```

Código 15 – Restrições de propriedade.

5.4.1.4. Elemento de propriedade

Através do elemento de propriedade, dado por *rdf:Property*, é permitido definir o conceito de propriedade. O nome da propriedade é dado pelos atributos *rdf:about* ou *rdf:ID*.

Propriedades que são usadas em restrições de propriedade devem ser tipificadas conforme a sua valoração. Propriedades que relacionam um objeto com outro objeto devem ser do tipo *daml:ObjectProperty*, enquanto que propriedades que relacionam um objeto com um valor dado por um XML Schema *datatype* devem ser do tipo *daml:DatatypeProperty*.

O elemento *rdf:Property* pode conter:

- Zero ou mais *rdfs:subPropertyOf*, que contém o nome de uma propriedade. Esse elemento é utilizado para indicar especialização das propriedades que estão sendo definidas. É permitida a existência de ciclos nas heranças de propriedades.
- Zero ou mais *rdfs:domain*, que contém uma expressão de classe. Cada *rdfs:domain* afirma que a propriedade que está sendo definida só se aplica a instâncias que estão de acordo com a expressão de classe. Múltiplos *rdfs:domain* restringem o domínio da propriedade à

interseção dos múltiplos *rdfs:domain* (que é diferente da semântica desse elemento em RDF Schema).

- Zero ou mais *rdfs:range*, que contém uma expressão de classe. Cada *rdfs:range* afirma que a propriedade que está sendo definida só aceita valores que são instâncias que estão de acordo com a expressão de classe. Múltiplos *rdfs:range* são permitidos, onde o valor da propriedade deve ser a interseção dos múltiplos *rdfs:range* (que é diferente da semântica desse elemento em RDF Schema).
- Zero ou mais *daml:samePropertyAs*, que contém o nome de uma propriedade. Esse elemento é utilizado para afirmar equivalência entre propriedades.
- Zero ou mais *daml:equivalentTo*, sendo que esse elemento contém um nome de uma propriedade. Quando esse elemento é aplicado a uma propriedade (e não uma classe), esse elemento tem a mesma semântica que a propriedade *daml:samePropertyAs*. É indicada a utilização de *daml:samePropertyAs* em detrimento a *daml:equivalentTo* (Connolly et al., 2001b).
- Zero ou mais *daml:inverseOf*, sendo que esse elemento contém um nome de uma propriedade. Através de *daml:inverseOf* é possível afirmar que a propriedade que está sendo definida é a inversa de outra propriedade.

Além da tipificação de uma propriedade como *daml:ObjectProperty* ou *daml:DatatypeProperty*, é possível também usar os seguintes elementos de forma a adicionar informação sobre uma propriedade:

- O elemento *daml:TransitiveProperty* permite afirmar que a propriedade é transitiva.
- O elemento *daml:UniqueProperty* permite afirmar que a propriedade pode ter cardinalidade máxima de 1.
- O elemento *daml:UnambigousProperty* permite afirmar que a propriedade não aceita que haja dois objetos com o mesmo valor de propriedade, ou seja, define a propriedade como um identificador único.

O trecho de código 16 apresenta um exemplo com as principais construções de elementos de propriedade utilizadas nesse trabalho.

```
<!-- Define o Slot (Atributo) peso -->
<daml:DatatypeProperty rdf:ID="peso">
        <rdfs:subPropertyOf rdf:resource="http://localhost:8080/arquivos/ontologiaProduto.daml#peso"/>
        <rdfs:domain>
                <daml:Class rdf:about="Livro"/>
        </rdfs:domain>
</daml:DatatypeProperty>

<!-- Define o Slot (Atributo) editora -->
<daml:DatatypeProperty rdf:ID="editora">
        <rdfs:domain>
                <daml:Class rdf:about="Livro"/>
        </rdfs:domain>
        <rdfs:range>
                <daml:Class rdf:about="http://www.w3.org/2000/10/XMLSchema#string"/>
        </rdfs:range>
</daml:DatatypeProperty>

<!-- Define o Slot (Atributo) ehRelacionado -->
<daml:ObjectProperty rdf:ID="ehRelacionado">
        <rdfs:domain>
                <daml:Class rdf:about="Produto"/>
        </rdfs:domain>
        <rdfs:range>
                <daml:Class rdf:about="Produto"/>
        </rdfs:range>
</daml:ObjectProperty>

<daml:TransitiveProperty rdf:about="ehRelacionado"/>
```

Código 16 – Elementos de propriedade.

5.4.1.5. Instâncias

As instâncias são declarações RDF que fazem referência as classes e propriedades definidas pelo vocabulário DAML+OIL. As definições de instância são, escritas, em sua maior parte, com o vocabulário básico de RDF e RDF Schema, sendo que DAML+OIL adiciona apenas os seguintes elementos: o elemento *daml:sameIndividualAs* para afirmar que dois objetos são iguais, e *daml:differentIndividualFrom* para afirmar que dois objetos são distintos.

5.5. SHOE

SHOE (*Simple HTML Ontology Extension*) é uma linguagem para descrição de ontologias proposta pelo *Parallel Understanding Systems Group* da Universidade de Maryland, sendo a versão mais recente de sua especificação (Luke & Heflin, 2000) (versão 1.01) datada de 28 de abril de 2000.

A proposta inicial de SHOE é ser uma linguagem que adiciona *tags* a linguagem HTML de forma possibilitar adicionar semântica a documentos HTML. Essa proposta inicial não utilizava nenhuma das linguagens anteriormente discutidas, sendo o vocabulário de SHOE colocado logo acima de HTML. Porém posteriormente foi definida uma DTD de SHOE possibilitando assim escrever documentos XML com *tags* de SHOE.

Através do vocabulário básico proposto na especificação de SHOE (Luke & Heflin, 2000) é possível descrever as seguintes estruturas de uma ontologia:

- Os conceitos e a taxinomia desses conceitos. Os conceitos são descritos por meio de categorias, e a taxinomia através de relacionamentos de herança.
- Os relacionamentos entre os conceitos.
- Axiomas dedutivos, usando regras de inferência.

5.5.1. Vocabulário SHOE

As *tags* propostas na especificação de SHOE podem ser dividas em duas partes: *tags* utilizadas para declaração de ontologias, e *tags* utilizadas para a declaração de instâncias de uma ontologia.

As *tags* para declaração de ontologias fornecem meios de declarar classes (chamadas de categorias) e taxinomia entre elas, propriedades (chamadas de relacionamento) e regras de inferência em uma ontologia. Essas *tags* são apresentadas na tabela 3.

As *tags* para a declaração de instâncias de uma ontologia fornecem meios de criar instâncias de conceitos e relacionamentos presentes em uma ou mais ontologias. Essas *tags* são apresentadas na tabela 4.

Tag	Descrição
<ONTOLOGY>...</ONTOLOGY>	Declara uma ontologia, indicado seu início e fim. Tags HTML não podem ocorrer dentro da declaração de uma ontologia.
<USE-ONTOLOGY>	Declara que a ontologia que está sendo definida estendo uma ou mais ontologias.
<DEF-CATEGORY>	Cria uma categoria. Ela deve descender de uma ou mais categorias.
<DEF-RELATION >...</DEF-RELATION>	Define um relacionamento entre instâncias de uma categoria, ou entre instâncias de uma categoria e dados.
<DEF-ARG>	Define os argumentos de um relacionamento.
<DEF-RENAME>	Permite atribuir nomes alternativos a uma categoria, a relacionamento ou a um tipo.
<DEF-INFERENCE>...</DEF-INFERENCE>; <INF-IF>...</INF-IF>; <INF-THEN>...</INF-THEN>; <RELATION>...</RELATION>; <CATEGORY>; <COMPARISON>...</COMPARISON>	Define uma regra de inferência. A regra de inferência é uma declaração do tipo IF (*corpo*) THEN (*resultado*).
<DEF-CONSTANT>	Define uma constante.
<DEF-TYPE >	Define um tipo de dado arbitrário em uma ontologia.

Tabela 3 – *Tags* utilizadas para a declaração de ontologias em SHOE.

Tag	Descrição
<INSTANCE>...</INSTANCE>	Declara uma instância de uma ontologia em um documento HTML. Tags HTML não podem ocorrer dentro da declaração de instância.
<USE-ONTOLOGY>	Declara o uso de uma ontologia dentro de um documento HTML, definindo um prefixo para o mesmo dentro desse documento. A ontologia é usada para que seja possível classificar instâncias ou estabelecer relacionamento entre instâncias.
<CATEGORY>	Indica a que classe pertence uma instância. Uma instância pode pertencer a mais de uma classe.
<RELATION>...</RELATION>	Define um relacionamento entre instâncias, ou entre instâncias e dados. Um relacionamento pode ter um ou mais argumentos.
<ARG>	Define o valor do argumento de um relacionamento.

Tabela 4 – *Tags* utilizadas para a declaração de instâncias em SHOE.

5.6. Comparação entre as linguagens

Nas três seções anteriores foram vistas, isoladamente, as características de algumas linguagens existentes para a descrição de ontologias. Agora faremos uma comparação entre essas três linguagens com o objetivo de escolher a que melhor se adeqüe ao desenvolvimento do estudo de caso. Além das informações presentes nas três últimas seções, utilizamos também (PLUS; Gil & Ratnakar; DAML) para a elaboração dessa comparação.

A comparação será feita levando-se em consideração as funcionalidades fornecidas por cada linguagem para a descrição dos principais componentes de uma ontologia: conceitos (classes) e sua taxinomia, relacionamentos e funções, axiomas, e instâncias. As tabelas 5, 6, 7 e 8 fazem a comparação para cada um desses componentes respectivamente.

	RDF Schema	**SHOE**	**DAML+OIL**
Permite a especialização de conceitos ?	Sim (*rdfs:subClassOf*)	Sim (atributo *ISA*)	Sim (*rdfs:subClassOf*)
Possível adicionar documentação a um conceito ?	Sim (*rdfs:comment*)	Sim (atributo *DESCRIPTION*)	Sim (*rdfs:comment*)
Permitido definir partição entre conceitos ?	Não	Não	Sim (*daml:disjointWith*)
Permite definir uma partição entre conceitos onde todos os conceitos da partição são subclasse de um conceito em comum ?	Não	Não	Sim (*daml: disjointUnionOf*)
Permite definir um conceito como complemento de um outro ?	Não	Não	Sim (*daml:complementOf*)

Tabela 5 – Comparação das funcionalidades para descrição de conceitos.

	RDF Schema	SHOE	DAML+OIL
Permite atributos de instância de conceito ?	Sim	Sim	Sim
Permite atributos de conceitos ?	Não	Não	Sim
Permite definir relacionamentos locais a um conceito ?	Sim (*rdfs:domain*)	Sim (atributo *TYPE* do elemento *DEF-ARG*)	Sim (*rdfs:domain)*
Permite definir relacionamentos de escopo global ?	Sim	Não	Sim
Permite definir o tipo de valor aceito pela relação ?	Sim (*rdfs:range*)	Sim (atributo *TYPE* do elemento *DEF-ARG*)	Sim (*rdfs:range*)
Permite definir restrições de cardinalidade?	Não	Não	Sim (*daml:cardinality*, *daml:maxCardinality*, *daml:minCardinality*)
Possível adicionar documentação a um relacionamento ?	Sim (*rdfs:comment*)	Sim (atributo *DESCRIPTION*)	Sim (*rdfs:comment*)
Permite um relacionamento com n argumentos?	Sim, mas não é uma primitiva, deve ser construido por meio de atributos de conceitos.	Sim	Sim, mas não é uma primitiva, deve ser construido por meio de atributos de conceitos.

Tabela 6 – Comparação das funcionalidades para descrição de relacionamentos e funções.

	RDF Schema	SHOE	DAML+OIL
Permite definir axiomas sobre propriedades de uma relação?	Não	Sim (elemento *DEF-INFERENCE*)	Sim (*daml:inverseOf*, *daml: TransitiveProperty*, *daml:UniqueProperty*, *daml: UnambigousProperty*)
Permite definir axiomas dedutivos arbitrários?	Não	Sim (elemento *DEF-INFERENCE*)	Não

Tabela 7 – Comparação das funcionalidades para descrição de axiomas.

	RDF Schema	SHOE	DAML+OIL
Permite definir instância de um conceito ?	Sim	Sim	Sim
Permite definir relações entre instâncias (declarações) ?	Sim	Sim	Sim
Permite fazer declarações sobre declarações ?	Sim (*rdf:Statement*)	Sim	Sim (*rdf:Statement*)

Tabela 8 – Comparação das funcionalidades para descrição de instâncias.

Por meio das tabelas anteriores, podemos observar que a capacidade de expressão das linguagens para descrever uma ontologia possui a seguinte ordem crescente: RDF Schema, SHOE, DAML+OIL.

RDF Schema puro não é capaz de descrever a ontologia do estudo de caso, pois não possui construções que são importantes para a mesma. RDF Schema não permite definir cardinalidade de um relacionamento e também não trabalha com axiomas básicos. Por esse motivo RDF Schema foi descartada como a linguagem utilizada no estudo de caso.

A linguagem DAML+OIL é a mais apropriada para ser utilizada no desenvolvimento das ontologias do estudo de caso. Apesar de SHOE e DAML+OIL praticamente empatarem em capacidade de descrição de ontologias, DAML+OIL possui outros atrativos comparada a SHOE:

- Originalmente em SHOE, a instância SHOE que descreve um recurso é embutida no documento HTML desse recurso. Já a instância DAML+OIL que descreve um recurso, está em um documento RDF separado do documento HTML desse recurso.
- A W3C, que é um nome de peso em padronização para *Internet*, tem o seu grupo de trabalho sobre ontologias (WebONT) trabalhando em uma linguagem para descrição de ontologias que tem por base a linguagem DAML+OIL. Recentemente, esse grupo propôs a linguagem OWL para a descrição de ontologias, sendo que essa linguagem é muito semelhante a DAML+OIL.
- A quantidade de ferramentas que fornecem suporte para a criação de ontologias, criação de instâncias e inferências sobre ontologias é muito maior para a linguagem DAML+OIL que para SHOE.

A desvantagem de DAML+OIL comparado com SHOE está no fato de a última proporcionar a capacidade de escrever regras de inferência arbitrárias, enquanto DAML+OIL possui apenas alguns padrões axiomáticos (*daml:inverseOf*, *daml:TransitiveProperty*, *daml:UniqueProperty*, *daml:UnambigousProperty*) predefinidos que podem ser aplicados. Apesar disso, esses padrões cobrem a maioria dos axiomas que normalmente são utilizados. Atualmente está sendo discutido pelo mesmo grupo que desenvolveu DAML+OIL (DARPA e IST) a linguagem DAML Rule (http://www.daml.org/rules/) que é

uma camada superior a DAML+OIL que proverá a capacidade de descrever regras de inferência.

Portanto, partimos para o capítulo seguinte com a escolha de DAML+OIL como a linguagem para a descrição das ontologias.