

Intercâmbio de conhecimento na Web

JOÃO GUILHERME DE SOUZA LIMA

Instituto de Computação, Universidade de Campinas - Caixa Postal 6176, 13081-970, Campinas, SP, Brasil
Embrapa Informática Agropecuária - Av. Dr. André Tosello, nº 209, 13083-886, Campinas, SP, Brasil
joao.guilherme@ic.unicamp.br

Abstract. O conhecimento disponível na World Wide Web não está sendo totalmente aproveitado. Sistemas não são capazes de utilizar informações de outros sistemas correlatos devido a limitações impostas pela maneira como os dados são modelados e armazenados. Introduzimos neste texto as técnicas de intercâmbio de conhecimento através da utilização de metadados e da modelagem de conceitos na forma de ontologias. Apresentamos também as principais linguagens utilizadas hoje para estes propósitos.

1 Introdução

É enorme a quantidade de informação disponível na World Wide Web. Bilhões de documentos se encontram disponíveis para milhões de usuários. Porém o sucesso da Internet acabou criando dificuldades para a sua própria utilização: sistemas de gerenciamento de dados e de documentos, ao lidar com tais quantidades de informação, não conseguem oferecer aos usuários mecanismos realmente eficientes de consulta. A informação que se deseja, muitas vezes disponível, acaba não sendo atingida pelo sistema de busca. Faz-se necessária uma melhor indexação do conhecimento.

O armazenamento de descrições dos dados juntamente com os próprios dados pode aprimorar os resultados de buscas por informação na Internet. Tais dados descritivos são chamados de *metadados*. Eles possibilitam que sejam fornecidas informações sobre a informação desejada, facilitando a localização da mesma.

Um campo emergente de pesquisa, que visa aumentar o compartilhamento das informações disponíveis na Internet, é o desenvolvimento da *semantic Web*. Nela procura-se alcançar uma troca não só coerente mas também automatizada de *conhecimento*, e não simplesmente de dados, entre os sistemas conectados pela Internet. Para isto é necessário aumentar a consciência semântica que os sistemas têm sobre os dados armazenados.

A comunicação entre pessoas e organizações é dificultada pela falta de terminologias e vocabulários comuns. Muitas vezes a mesma idéia ou conceito pode ser expressada de maneiras diferentes. Isto frequentemente ocorre entre grupos distintos de usuários, às vezes dentro de uma mesma organização.

Uma tentativa de solucionar estes dois últimos problemas é a utilização de *ontologias* - representações de um domínio de conhecimento de maneira sistemática, consensual e, no caso da computação, também inteligível para computadores. Em ontologias dedica-se um cuidado es-

pecial ao modo como a semântica dos dados é definida. Conceitos são definidos e organizados de maneira que seja possível inferir as relações entre eles.

Neste texto são apresentadas duas das abordagens utilizadas hoje para aprimorar o intercâmbio de conhecimento na Web: associação de metadados aos dados armazenados e modelagem de domínios de conhecimento na forma de ontologias. Na seção 2 definimos metadados e descrevemos brevemente duas das principais linguagens de modelagem de metadados. Na seção 3 caracterizamos ontologias e discutimos o problema de atingir consenso na criação de uma ontologia a ser utilizada por grupos de usuários diferentes. Também apresentamos três difundidas linguagens para construção de ontologias.

2 Metadados

Em diversas situações da vida cotidiana utilizamos informação para encontrar informação. Por exemplo, em grandes livrarias utilizamos sistemas de busca para descobrir a localização de um determinado livro, fornecendo dados como autor, título ou editora. Estes dados associados a outros dados a fim de descrevê-los são o que chamamos de **metadados**. No exemplo anterior, *autor* é um metadado associado ao livro que procuramos. Metadados são, então, dados sobre dados, ou informação sobre informação, dependendo do ponto de vista.

Há quantidades enormes de informação disponível na Internet. O problema a ser resolvido é como localizar a informação que desejamos. Existem sites de busca que, através de robôs, varrem a Internet e processam consultas levando em consideração o texto integral de todas as páginas analisadas. Este tipo de técnica via força bruta não tem uma boa taxa de acerto; muitas vezes os resultados obtidos não são os desejados. A associação de metadados aos conteúdos disponíveis é um interessante recurso para aprimorar a busca por informação na Internet. Mesmo com os sistemas não tendo consciência semântica do conteúdo com

o qual estão trabalhando, a descrição dos dados armazenados através de metadados auxilia também a integração e intercâmbio de dados.

No momento de compartilhar e localizar dados na Internet através de metadados um problema surge quando organizações utilizam terminologias diferentes para descrever os mesmos tipos de entidades. Isto é, se duas organizações desejam compartilhar seus dados mas utilizam termos descritivos diferentes nos seus metadados, o casamento das informações não é possível.

Uma iniciativa para tentar amenizar este problema é o *Dublin Core Metadata Initiative* [2]. Ela consiste de um fórum aberto com o objetivo de desenvolver padrões de metadados que suportem uma larga faixa de propósitos e modelos de negócios. Os padrões de metadados já criados por esta iniciativa são disponibilizados gratuitamente no seu site.

2.1 XML

Em fevereiro de 1998 o World Wide Web Consortium (W3C) divulgou a primeira versão da linguagem *Extensible Markup Language* (XML), voltada para a construção de documentos Web. O objetivo era permitir que fossem construídos documentos cuja informação fosse estruturada, isto é, documentos que contivessem não só conteúdo, mas também indicações a respeito deste. Em outras palavras XML passou a permitir a elaboração de documentos que contenham não só dados mas também metadados. Desde então XML passou a ser largamente utilizada na construção de documentos Web.

Em XML os metadados são especificados através de etiquetas. XML provê facilidades para a definição destas etiquetas e das relações estruturais entre elas. Porém XML não especifica semânticas nem um conjunto pré-definido de etiquetas, diferentemente de HTML. A semântica de um documento XML é definida pelas aplicações que o processam.

O W3C idealiza aplicações de XML onde um documento possa conter elementos e atributos utilizados por vários módulos de software. Isto é, se existe um vocabulário que seja bem entendido e útil para diversos sistemas diferentes, considera-se preferível reusá-lo do que reinventá-lo.

Porém, esta liberdade de criação de elementos e atributos pode criar problemas de vocabulário, como colisões e dificuldade de reconhecimento. Isto pode ocorrer se documentos de um mesmo domínio de aplicação utilizarem termos diferentes ou se documentos de domínios diferentes utilizarem os mesmos termos.

Para amenizar este problema foi criado o mecanismo *XML namespaces*. Um XML namespace é uma coleção de nomes, identificada por uma URI (Uniform Resource Iden-

tifier), para ser utilizada em documentos XML como tipos de elementos e nomes de atributos. Desta maneira, utilizando namespaces documentos semelhantes podem utilizar nomes universais.

Não é objetivo deste texto discutir aspectos da sintaxe XML. Referências a conferências, livros e artigos sobre XML podem ser encontradas em [21].

2.1.1 XML Schema

XML Schema Definition Language, ou, simplesmente, XML Schema, é uma linguagem para definir a estrutura e conteúdo de documentos XML. XML Schema foi aprovada como uma recomendação do W3C em maio de 2001.

XML Schema provê meios para a definição de *esquemas* para documentos XML. Ela permite definir, entre outros:

- O tipo de dados de cada elemento.
- O domínio de valores de cada elemento.
- Regras para etiquetas aninhadas.
- A cardinalidade de cada elemento.

XML Schema também se aproveita do mecanismo de namespaces, como XML. De fato, definições em XML Schema são elas próprias documentos XML. Desta maneira, aplicações desenvolvidas para XML, como ferramentas de validação, podem ser aplicadas a definições de esquema em XML Schema.

A Figura 1 exibe um exemplo simples de uma definição de esquema em XML Schema. Vejamos alguns dos seus aspectos básicos. É definido o elemento de nome *livro*, como sendo de tipo complexo, uma vez que ele não terá apenas um valor mas sim subelementos. Seus subelementos são definidos utilizando a restrição *sequence*, que indica que a sua ordem é relevante. Alguns dos subelementos, como *autor* e *titulo*, são definidos como do tipo simples, isto é, sem subelementos, e o seu tipo de dados (string, date, etc.) também é definido. O subelemento *personagem* é definido como do tipo complexo, e sua cardinalidade é definida de maneira que um livro possa ter nenhum ou ilimitados personagens.

Uma explicação mais detalhada e facilmente entendível de XML Schema pode ser obtida em [8].

2.2 RDF

Visando possibilitar uma utilização eficiente de metadados na Internet, o W3C desenvolveu a linguagem *Resource Description Framework* (RDF) [15]. RDF é um framework para descrever e intercambiar metadados. Ele possibilita interoperabilidade entre aplicações que compartilham metadados, e é voltado para diversas áreas de aplicação, como

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="livro">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titulo" type="xs:string" />
        <xs:element name="autor" type="xs:string" />
        <xs:element name="personagem"
          minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nome" type="xs:string" />
              <xs:element name="amigo-de" type="xs:string"
                minOccurs="0" maxOccurs="unbounded" />
              <xs:element name="desde" type="xs:date" />
              <xs:element name="qualificacao" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Figura 1: uma definição de esquema em XML Schema

descrição de recursos, classificação de conteúdo, comércio eletrônico, serviços colaborativos e preferências de privacidade. RDF é o resultado de um consenso entre membros destas comunidades sobre suas necessidades sintáticas

Bray [4] lista as 4 regras baseadas nas quais RDF foi definido:

1. Um **Recurso** é qualquer coisa que possa ter um URI, o que inclui páginas Web e elementos individuais de documentos XML.
2. Uma **Propriedade** é um Recurso que possui um nome e que pode ser usado como uma propriedade. Por exemplo, autor ou titulo. Na maioria dos casos o que realmente importa é o seu nome, mas uma Propriedade precisa ser um recurso para que ela possa ter suas próprias propriedades.
3. Uma **Declaração** consiste da combinação de um Recurso, uma Propriedade e um valor. Assim, uma declaração associa um valor a uma Propriedade de um Recurso. Por exemplo, poderia-se afirmar que para o recurso `http://www.books.com/Hamlet.html` o valor da propriedade autor é "Shakespeare". Um valor também pode ser um outro recurso.

4. Há um método direto para expressar estas propriedades abstratas em XML, o que facilita o intercâmbio.

RDF utiliza a facilidade de XML namespaces. Um XML namespace permite a um documento RDF definir o escopo e identificar unicamente um conjunto de propriedades. Este conjunto de propriedades, chamado de *esquema*, pode ser acessado na URI correspondente ao namespace.

A Figura 2, retirada de [13], ilustra as características básicas de um documento RDF. A primeira linha indica dois namespaces, RDF e DC, sendo RDF o namespace padrão. Assim, todas as propriedades serão definidas em um destes namespaces. A seção entre as etiquetas `<Description>` define quatro propriedades para o recurso `http://dstc.com.au/report.html`, apontado pela URI no atributo `about`.

A linguagem RDF possui diversos outros meios de descrição de metadados que não serão cobertos neste texto. A intenção aqui é apenas mostrar a aparência e a sintaxe básica de um documento RDF. Documentação mais completa de RDF pode ser obtida em [15].

Uma característica importante de RDF é a *independência*. Como uma Propriedade pode ser um Recurso, qualquer organização ou pessoa pode inventar propriedades, como Autor e Diretor, por exemplo. Porém, as comunidades que desejarem trocar informações entre si de-

```

<RDF xmlns = "http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
  xmlns:DC = "http://purl.org/DC#">
  <Description about = "http://dstc.com.au/report.html">
    <DC:Title> The Future of Metadata </DC:Title>
    <DC:Creator> Jacky Crystal </DC:Creator>
    <DC>Date> 1998-01-01 </DC>Date>
    <DC:Subject> Metadata, RDF, Dublin Core </DC:Subject>
  </Description>
</RDF>

```

Figura 2: um exemplo simples de um documento RDF

vem se organizar para utilizar as mesmas terminologias nos metadados.

Poderia-se perguntar por que simplesmente não utilizar XML no lugar de RDF. XML permite a invenção de etiquetas, que podem conter tanto dados em texto ou outras etiquetas. E ainda possui correspondentes a Propriedades e Declarações de RDF, como em ``. Assim, XML realmente pode ser utilizado para intercâmbio de metadados na Internet. Porém ele falha em um dos objetivos alcançados por RDF: escalabilidade. Isto ocorre por duas razões. Primeiro, porque a ordem na qual elementos aparecem em um documento XML é significativa. No contexto de metadados isto não deveria ter importância. Segundo porque quando se representa documentos XML na memória do computador pode obter-se estruturas de dados complexas que misturam árvores, grafos e cadeias de caracteres. É difícil lidar com estes tipos de estruturas, mesmo em quantidades moderadas.

Vale ressaltar mais uma vez que, ao utilizar metadados para descrever outros dados da maneira como é feito em XML e RDF, não se inclui semântica nos dados. Isto é, com RDF não se ajuda o computador a entender a semântica do que ele está manipulando.

2.2.1 RDF Schema

RDF permite definir declarações sobre Recursos, usando Propriedades e valores. Entretanto, como mencionado, é interessante que as diversas comunidades tenham a capacidade de indicar que estão descrevendo tipos específicos de recursos, e que irão usar propriedades também específicas para descrever estes recursos. Por exemplo, uma companhia de equipamentos de camping poderia querer definir uma *classe* Barraca, e utilizar propriedades como modelo, pesoEmKg, e tamanhoEmbalada para descrevê-la.

RDF por si só não provê um vocabulário para especificar estes termos. Tais classes e propriedades devem ser descritas em um *vocabulário* RDF a ser criado para cada contexto de aplicação. Com o objetivo de descrever vocabulários RDF foi criada a linguagem *RDF Schema* [5].

RDF Schema não provê vocabulários de classes orientadas a aplicação, como Barraca, Livro ou Pessoa, e nem de propriedades como pesoEmKg, autor ou telefone. O que RDF Schema provê são os mecanismos necessários para *especificar* tais classes e propriedades como parte de um vocabulário, e para indicar quais classes e propriedades espera-se que sejam usadas em conjunto. Desta maneira RDF Schema provê um sistema de tipos para RDF. Este sistema de tipos lembra em alguns aspectos sistemas de tipos de linguagens de programação orientadas a objeto. Por exemplo, Recursos podem ser definidos como instâncias de uma ou mais classes, e as classes podem ser definidas de maneira hierárquica, com classes inferiores herdando propriedades de classes superiores. Porém, em outros aspectos classes e propriedades RDF são bem diferentes de tipos de linguagens de programação. Não é objetivo deste texto se aprofundar nestas questões; uma descrição detalhada de RDF Schema, juntamente com exemplos de sua utilização, pode ser obtida em [5].

O W3C provê na Internet um vocabulário RDF de recursos e propriedades, juntamente com seus significados, que podem ser utilizados para descrever classes e propriedades específicas de usuário. Este vocabulário RDF Schema é definido em um namespace identificado pela URI `http://www.w3.org/2000/01/rdf-schema#`.

3 Ontologias

Acredita-se que a representação formal do conhecimento tenha começado na Índia, no primeiro milênio a.C., com o estudo da gramática sânscrita. Da maneira como veremos neste texto, porém, esta disciplina está mais ligada a trabalhos realizados em filosofia na Grécia antiga, principalmente por Aristóteles (384-322 a.C.). Na computação, ontologias foram utilizadas inicialmente no campo da Inteligência Artificial, visando facilitar o compartilhamento de conhecimento e o reuso.

Um dos objetivos de utilizar ontologias é permitir que as idéias de um domínio de conhecimento sejam compartilhadas de maneira consensual entre indivíduos e sistemas de computador. Isto exige certo formalismo na definição da

semântica dos conceitos do domínio.

Qualquer que seja a maneira escolhida para construir a ontologia, deve-se armazenar, juntamente com as definições de conceitos, as relações existentes entre os mesmos. Assim, os conceitos são definidos e ligações entre eles indicam seus relacionamentos semânticos. Estas ligações também precisam ser cuidadosamente definidas.

Muitas vezes a ontologia toma a forma de uma árvore hierárquica. São definidas *classes*, de maneira que cada classe herde as características da classe imediatamente superior. Cada classe representa um conceito do domínio sendo modelado. As características de classes irmãs podem ou não ser mutuamente exclusivas, dependendo do caso. Explicitando-se as similaridades e diferenças entre cada classe, sua classe superior e suas classes irmãs, tem-se como resultado uma taxonomia de noções onde o significado de um nó é dado por todas as propriedades, similaridades e diferenças encontradas no caminho do conceito raiz (o mais genérico) até o nó [3]. Também pode-se ter herança múltipla, quando um conceito possui todas as características de dois ou mais conceitos mais genéricos.

Além da herança, outros relacionamentos importantes entre conceitos também podem ser incluídos na ontologia. Por exemplo, os conceitos *empresaArea* e *turismo*, mesmo que não pertençam ao mesmo ramo de uma árvore, podem ter entre eles o relacionamento *instrumento*. O armazenamento deste relacionamento permitiria inferir que empresas aéreas tornam possível a realização da atividade turismo.

No processo de criação de uma ontologia para um domínio de conhecimento pode-se tentar aproveitar partes de outras ontologias já existentes. Isto é, se já existe uma ontologia que define o conceito de alguns termos a serem utilizados na ontologia sendo desenvolvida, esta ontologia pode ser reutilizada.

Neste sentido, um interessante sistema de referência online e gratuito é o WordNet [6]. Ele consiste de um banco de dados léxico da língua inglesa, e seu design foi inspirado por teorias psicolinguísticas da memória léxica humana. Substantivos, verbos, adjetivos e advérbios em inglês são organizados em conjuntos de sinônimos, cada um representando um conceito léxico. As palavras podem ter mais de um significado, e cada significado de cada palavra é definido de duas maneiras: com um conjunto de termos chamado de *synset* e com uma definição textual chamada de *gloss*. No *synset* são armazenados termos relacionados ao significado em questão. Por exemplo, em um dos significados do verbo *ship*, o *synset* consiste dos termos *transport*, *send* e *ship*. Desta maneira conceitos correlatos podem ser relacionados através de termos em comum nos seus *synsets*.

Ao mesmo tempo que é necessário que os relacionamentos entre os conceitos sejam definidos de maneira clara e sem ambiguidade para permitir seu correto processamento

pelos sistemas computacionais, também é importante que os usuários possam visualizar e entender a ontologia. Por isso algumas abordagens suportam a modelagem de ontologias em várias camadas. A camada mais superior geralmente corresponde ao que um ser humano consegue entender, e tenta-se utilizar nela linguagem natural. Desta maneira o usuário pode varrer a ontologia, seja para consultá-la, modificá-la ou validá-la manualmente. Já a camada mais inferior deve ser definida mais formalmente, de maneira que possa ser compreendida sem erros pelo computador. As camadas intermediárias se constituem de mapeamentos entre as camadas superiores, menos formais, e as camadas inferiores, mais formais.

Além do significado de conceitos e suas relações, outra primitiva de modelagem que uma ontologia pode conter são *axiomas*. Axiomas definem regras sobre os relacionamentos entre os conceitos. Por exemplo, um axioma pode definir se um relacionamento entre dois conceitos é simétrico ou não. Um outro tipo de axioma é a composição de relacionamentos. O relacionamento *AvODe*, por exemplo, poderia ser definido pela composição de dois relacionamentos *PaiDe*. Staab e Maedche [17] propõem uma abordagem de modelagem de axiomas onde busca-se tornar a especificação dos mesmos independente da linguagem a ser utilizada.

Algumas abordagens não se contentam com apenas fornecer meios para modelagem e armazenamento de ontologias, mas também tentam automatizar pelo menos parcialmente este processo. É o caso das ferramentas que realizam *aprendizado automatizado de conceitos*. Geralmente, o que se faz é analisar documentos e Web sites relacionados ao domínio da aplicação a fim de se extrair uma terminologia para o domínio. Então, *processadores de linguagem natural* filtram as informações obtidas e detectam relações taxonômicas entre os conceitos aprendidos.

Este tipo de análise não é trivial, uma vez que é essencial haver interpretação semântica para capturar relacionamentos semânticos importantes entre os conceitos aprendidos. Um destes relacionamentos, por exemplo, é a generalização *tipo-de*. Ela poderia ocorrer entre os conceitos *metrô* e *transporte público*. Um exemplo de utilização de técnicas de aprendizado automatizado é o ambiente desenvolvido com base na ferramenta OntoLearn [16], que suporta não só aprendizado mas também validação automatizada de conceitos.

Para tentar aprimorar a troca de informações na Web através da utilização de ontologias, um consórcio de parceiros comerciais e acadêmicos, principalmente europeus, criou o projeto *On-to-Knowledge* [10]. Atualmente está sendo desenvolvida neste projeto uma ferramenta que visa processar documentos heterogêneos, distribuídos e semi-estruturados, encontrados tanto em intranets de grandes empresas quanto na própria Internet. Para isso são utilizadas

técnicas como tecnologia de busca da semantic Web, aprendizado automatizado de conhecimento, intercâmbio de documentos através de operadores de transformação e suporte a visões específicas de usuário.

Uma introdução detalhada a ontologias pode ser obtida em [19].

3.1 Falta de terminologia comum

Em grandes organizações, principalmente nas geograficamente dispersas, é comum surgirem diferentes vocabulários entre diferentes grupos de pessoas. Estes vocabulários podem ser definidos apenas no uso do dia-a-dia ou podem ser explicitamente controlados. De qualquer maneira, a existência de múltiplos vocabulários faz com que diferentes grupos utilizem termos diferentes para se referir ao mesmo significado. Também pode ocorrer de um mesmo termo ser utilizado com significados diferentes.

A falta de uma terminologia consistente traz vários tipos de problemas, como desentendimentos entre grupos, impossibilidade de interoperabilidade entre sistemas de computador, dificuldades quando se procura por um recurso em outra área da organização (um caso de uso de ontologias tem sido a indexação baseada em conhecimento de documentos e arquivos). Neste último caso, se uma pessoa procura por documentos classificados sob o termo *requisitos* em um grupo, a aplicação de recuperação deve ter consciência de que em um outro grupo este documento pode estar classificado como *requerimentos*.

Uma solução natural para este tipo de problema seria a adoção de um único vocabulário, a ser largamente utilizado por todos na organização. Esta medida, porém, não é viável na prática, mesmo sabendo-se das vantagens que ela traria. Além disto, mesmo se um acordo inicial fosse atingido, há questões de *manutenção* a serem consideradas, devido ao fato da natureza da informação ser volátil. Esta poderia mudar mais rapidamente em algumas partes da organização que em outras.

Se uma organização que deseja organizar seu banco de conhecimentos na forma de ontologias enfrenta este tipo de problema, ela não pode criar apenas uma ontologia global a ser utilizada por todos. Uma possibilidade a ser considerada, então, é a criação de várias ontologias locais, uma para cada vocabulário encontrado.

3.1.1 Múltiplas ontologias locais

Se opta-se pela abordagem de se criar várias ontologias locais para resolver o problema da falta de terminologia comum, enfrenta-se outras questões. A primeira é a decisão de se criar ou não, além das ontologias locais, uma ontologia global. Isto depende das vantagens que a existência de uma ontologia global traria à organização.

Outro tipo de problema ocorreria, por exemplo, no

caso citado anteriormente, quando um documento ou recurso fosse classificado de uma maneira em uma ontologia local e de outra maneira em uma outra ontologia local. Surgiria a questão de quando e como grupos poderiam utilizar ontologias de outros grupos. Para que isto fosse possível seria necessário construir mapeamentos que indicassem o correspondente de um termo em cada ontologia local. Segundo Uschold [18], há duas maneiras de fazer isto:

- *mapeamento direto ponto a ponto* - nesta abordagem são definidas regras de mapeamento entre os termos de cada par de ontologias.
- *mapeamento através de uma ontologia de referência global* - nesta abordagem são definidas regras de mapeamento de termos das ontologias locais para a ontologia global, e da ontologia global para cada ontologia local. Assim, para descobrir o correspondente a um termo de uma ontologia local em outra ontologia local seriam necessários dois passos.

A escolha de um dos dois métodos depende de vários fatores, entre eles [18]: quanto mapeamento é requerido, quanta intersecção há entre as ontologias locais, e se há mesmo uma ontologia global de referência.

A criação e manutenção de uma ontologia global também não é tarefa trivial. Deve-se decidir, por exemplo, se a ontologia global deve consistir da *união* ou da *intersecção* dos conjuntos de termos das ontologias locais.

3.2 Linguagens para modelagem de ontologias

Como já foi observado, a representação de ontologias de uma maneira inteligível para sistemas de computador exige uma linguagem capaz de expressar conhecimento de maneira clara e sem ambigüidades. Desde os anos 90 algumas linguagens têm sido propostas com este objetivo.

Modelos de esquemas de bases de dados, assim como ontologias, também são utilizados para definir a estrutura e parte da semântica de dados. Um exemplo bem conhecido é o Modelo Relacional [7], utilizado em grande parte dos bancos de dados atuais. Assim, pode ser natural comparar estes dois tipos de modelagem de dados. Fensel [9] lista as seguintes diferenças entre ontologias e definições de esquemas:

- Uma linguagem para definição de ontologias é sintática e semanticamente mais rica que abordagens comuns para bancos de dados.
- A informação que é descrita por uma ontologia consiste de textos de linguagem natural semi-estruturados e não de informação tabular.
- Uma ontologia precisa ser uma terminologia consensual e compartilhada, uma vez que ela deve ser

utilizada para intercâmbio e compartilhamento de informação.

- Uma ontologia provê uma teoria de domínio e não a estrutura de um contâiner de dados.

Nas subseções seguintes apresentamos e descrevemos brevemente algumas das linguagens utilizadas hoje para modelagem de ontologias.

3.2.1 OIL

Ontology Inference Layer (OIL) [12] é uma linguagem criada para representar semântica de uma maneira acessível por máquinas, modelando domínios de conhecimento na forma de ontologias. Desenvolvida para ser compatível com padrões do W3C, incluindo XML e RDF, OIL explora as primitivas de modelagem de RDF Schema. Desta maneira, aplicações que suportam apenas RDF podem entender pelo menos parcialmente um documento OIL.

Assumindo-se que uma única linguagem de ontologias não pode se ajustar a todas as aplicações e usuários da Internet, OIL se apresenta em uma abordagem de camadas, cada uma adicionando funcionalidades e complexidade à camada inferior [10]. Desta maneira pessoas ou sistemas familiarizados apenas com as camadas mais inferiores podem entender parcialmente ontologias expressas em uma das camadas mais altas.

A camada mais inferior é chamada de *Core OIL*, e coincide bastante com RDF Schema. A camada *Standard OIL* tenta prover um maior poder de expressão, de maneira bem entendível, permitindo especificar precisamente a semântica e possibilitando a realização de inferências. A terceira camada, *Instance OIL*, oferece integração individual completa, incluindo compatibilidade total com bancos de dados. Ela possui o mesmo esquema de Standard OIL mas suas instâncias são descritas diretamente em RDF. A última camada, *Heavy OIL*, cuja sintaxe e esquema ainda não estão definidos, irá acrescentar novas capacidades de representação e raciocínio.

Uma ontologia em Standard OIL é formada por duas seções: *ontology container* e *ontology definitions*. As Figuras 3 e 4 exibem exemplos, adaptados de [14], que ilustram cada uma das duas seções de uma ontologia em Standard OIL. Com a intenção de dar uma atenção maior a como uma ontologia é expressa em OIL, a seguir descrevemos vários dos elementos desta linguagem.

Na seção *ontology container* são especificadas algumas características gerais da ontologia, como nome, autor, assunto, linguagem, versão, etc. Para descrever estes metadados da ontologia usa-se o padrão Dublin Core [2]. Apesar de todo elemento neste padrão ser opcional, em OIL alguns são obrigatórios ou têm um valor pré-definido. Não entraremos em detalhes sobre esta seção.

Na segunda seção de uma ontologia OIL, *ontology definitions*, a ontologia propriamente dita é definida, sendo definido um vocabulário ontológico particular. Ela consiste de um conjunto dos seguintes elementos [14]:

- **import** - uma lista de uma ou mais referências a módulos OIL a serem incluídos na ontologia. Cada referência deve consistir de um URI especificando de onde o módulo deve ser importado. Por exemplo, `http://www.ontosrus.com/animals/jungle.onto`. Assume-se que nomes de diferentes especificações são diferentes, via prefixos diferentes.
- **class-def** e **slot-def** - zero ou mais definições de slots (**slot-def**), e de classes (**class-def**), cujas estruturas descreveremos a seguir.
- **rule-base** - uma lista de regras que se aplicam à ontologia. Estas regras também são chamadas de axiomas ou restrições globais. No momento a estrutura destas regras não está definida, e elas não possuem semântica formal em OIL, mas isto pode ser adicionado no futuro.

Antes de falarmos sobre as declarações de classes e slots, é preciso definir o que é uma **class-expression** na sintaxe OIL. Uma class-expression pode ser uma de três opções: o nome de uma classe, um **slot-constraint** (um slot-constraint, como veremos, também define uma classe), ou uma expressão booleana de class expressions, podendo-se utilizar os operadores **AND**, **OR** ou **NOT**. Desta maneira class expressions podem ser definidas recursivamente.

Um *slot* é uma relação binária, isto é, suas instâncias são pares de indivíduos. Nas definições de classes podem ser definidas restrições sobre estas relações binárias para caracterizar cada classe. Por exemplo, se tiver sido definido o slot *come*, na definição da classe *carnivoro* pode-se restringir, que, se um membro desta classe se relaciona com um outro indivíduo através do slot *come*, este indivíduo deve ser da classe *animal*.

Para se definir um slot deve ser utilizada a palavra reservada **slot-def**, seguida do nome do slot. Logo a seguir podem ser definidas algumas regras globais ao slot, como por exemplo se a relação binária definida pelo slot é transitiva ou não. As regras que podem ser definidas em um **slot-def** são as seguintes:

- **subslot-of** - uma lista de um ou mais slots, de maneira que o slot sendo definido neste slot-def deve ser um sub-slot de cada um dos slots da lista. Por exemplo, a declaração:

```
slot-def avo
      subslot-of parente
```

ontology-container

```
title "Animais africanos"
creator "Ian Horrocks"
subject "Animais, comida e vegetarianos"
description "Um exemplo didático de ontologia que
  descreve animais africanos."
description.release "1.01"
publisher "Ian Horrocks"
type "ontology"
format "pseudo-xml"
identifier "http://www.exemplos.org/oil/exemplo.onto"
source "http://www.africa.com/animais.html"
language "OIL"
language "pt-br"
relation.hasPart
  "http://www.ontosRus.com/animais/selva.onto"
```

Figura 3: seção *ontology-container* de uma ontologia OIL

define que, para um par de indivíduos se relacionar através do slot *avo*, eles também devem se relacionar pelo slot *parente*.

- **domain** - uma lista de uma ou mais class-expressions. O primeiro indivíduo de cada par que se relaciona através do slot deve ser uma instância de cada uma das class-expressions da lista. Isto é, para o par (A; B) se relacionar através do slot, o indivíduo A deve ser uma instância de cada uma das class-expressions listadas.
- **range** - semelhante à restrição domain, mas se aplica sobre o segundo indivíduo do par. Ou seja, para o par (A; B) se relacionar através do slot, o indivíduo B deve ser uma instância de cada uma das class-expressions listadas.
- **inverse** - um slot S que seja o inverso do slot sendo definido. Isto é, se o par (A; B) se relaciona através do slot sendo definido, o par (B; A) deve se relacionar através do slot S.
- **properties** - uma ou mais propriedades do slot. As propriedades válidas são **transitive**, que indica que a relação binária do slot é transitiva, e **symmetric**, que indica que a relação binária é simétrica.

Passemos agora à definição de classes. Para definir uma classe utiliza-se a palavra reservada **class-def** seguida de um nome para a classe. As seguintes declarações podem ser utilizadas para caracterizar a classe:

- **type** - o tipo da classe, que pode ser **primitive** (primitiva) ou **defined** (definida), e deve ser escrito entre a palavra **class-def** e o nome da classe. Se omitido, o valor padrão é primitivo. Quando uma classe

é primitiva, sua definição, isto é, a combinação dos seus componentes subclass-of e slot-constraint, é tida como uma condição necessária mas não suficiente para participação na classe.

- **subclass-of** - uma lista de uma ou mais class-expressions. Com esta declaração determina-se que a classe sendo definida neste class-def é uma subclasse de cada uma das classes da lista.
- **slot-constraints** - zero ou mais slot-constraints, cuja estrutura será descrita mais adiante. Com isto determina-se que a classe sendo definida neste class-def é uma subclasse de cada um dos slot-constraints da lista. Em outras palavras, para um indivíduo ser uma instância da classe sendo definida, ele deve ser uma instância do slot-constraint.

Um **slot-constraint** define uma ou mais restrições sobre um slot. Como vimos, um slot é uma relação binária entre dois indivíduos. Um slot-constraint também define uma classe: suas instâncias são os indivíduos que, além de se relacionarem com algum outro indivíduo através do slot original, também satisfazem as restrições definidas neste slot-constraint. Os exemplos a seguir esclarecerão como isto funciona.

Após a palavra reservada **slot-constraint**, deve-se seguir o nome do slot original. Este slot pode ou não ter sido definido na ontologia. Se ele não tiver sido definido na ontologia, simplesmente assume-se que este slot não possui propriedades ou restrições globais.

Os seguintes componentes podem ser utilizados na definição de um slot-constraint:

- **has-value** - uma lista de uma ou mais class-expressions. Neste caso, toda instância da classe sendo

ontology-definitions	value-type animal
slot-def come	class-def defined herbivoro
inverse e_comido_por	subclass-of animal
slot-def tem_como_parte	slot-constraint come
inverse e_parte_de	value-type planta OR
properties transitive	slot-constraint e_parte_de
class-def animal	has-value planta
class-def planta	class-def girafa
subclass-of NOT animal	subclass-of animal
class-def arvore	slot-constraint come
subclass-of planta	value-type folha
class-def ramo	class-def leao
slot-constraint e_parte_de	subclass-of animal
has-value arvore	slot-constraint come
class-def folha	value-type herbivoro
slot-constraint e_parte_de	class-def planta-gostosa
has-value ramo	subclass-of planta
class-def defined carnivoro	slot-constraint comido_por
subclass-of animal	has-value herbivoro, carnivoro
slot-constraint come	

Figura 4: seção *ontology-definitions* de uma ontologia OIL

definida por este slot-constraint precisa estar relacionada através do slot original com uma instância de *cada uma* das class-expressions da lista. Por exemplo, a declaração:

```
slot-constraint constroi
has-value ponte, estrada
```

define uma classe cujos membros são os indivíduos que se relacionam com instâncias das classes *ponte* e *estrada* através do slot *constroi*. Porém isto não define que as instâncias desta classe podem se relacionar através do slot *constroi* *somente* com instâncias destas duas classes.

- **value-type** - uma lista de uma ou mais class-expressions. Esta restrição determina que, se uma instância da classe sendo definida por este slot-constraint se relaciona com algum outro indivíduo *I* através do slot original, então o indivíduo *I* deve ser uma instância de *cada uma* das class-expressions da lista.
- **max-cardinality** - um número não-negativo *n* seguido de uma class-expression. Isto determina que uma instância da classe definida pelo slot-constraint pode estar relacionada com no máximo *n* instâncias distintas da class-expression através do slot original. Por exemplo, a declaração:

```
slot-constraint casado-com
max-cardinality 1 mulher
```

define uma classe cujos membros são aqueles que se

relacionam com no máximo uma instância da classe *mulher* através do slot *casado-com*.

- **min-cardinality** e **cardinality** - análogos a **max-cardinality**. Definem a cardinalidade mínima e a cardinalidade exata, respectivamente, com a qual instâncias da classe sendo definida podem estar relacionadas através do slot original.

Vale lembrar que uma classe deve ser vista como uma sub-classe de todas as classes definidas por slot-constraints dentro da sua declaração. Por exemplo, o slot-constraint existente na declaração da classe *girafa*, na Figura 4, implica que para um indivíduo ser uma instância desta classe ele não só precisa ser uma instância da classe *animal*, mas também só pode se relacionar através do slot *come* com indivíduos do tipo *folha*.

Descrevemos apenas algumas das palavras-chave da camada Standard OIL. Documentação mais completa sobre esta sintaxe pode ser encontrada em [11].

3.2.2 DAML+OIL

Uma outra iniciativa a fim de criar uma linguagem para representação de conhecimento na Web é o *DARPA Agent Markup Language Program* [1], de responsabilidade do comitê ad hoc Agent Markup Language, da junta Estados Unidos/União Européia. Este programa desenvolveu, como seu próprio nome diz, a linguagem DARPA Agente Markup Language (DAML).

A motivação inicial para a criação desta linguagem foi

a incapacidade de XML de expressar as relações existentes entre os conceitos representados nos seus documentos. Assim, DAML foi desenvolvida como uma extensão a XML e a RDF, a fim de aumentar o poder de expressão e modelagem do conhecimento em documentos Web.

A versão da linguagem liberada em janeiro de 2001, **DAML+OIL**, provê meios para modelar domínios de conhecimento através de ontologias. DAML+OIL incorpora aspectos tanto da linguagem DAML quanto da linguagem OIL, e pode ser vista como um subdialecto desta.

Existem várias diferenças entre as linguagens OIL e DAML+OIL. Elas existem principalmente devido ao fato de que DAML+OIL foi baseada em RDF. Assim, algumas construções em RDF são possíveis em DAML+OIL mas não em OIL. Há outras diferenças, que não serão listadas neste texto. Referências para esta questão podem ser encontradas em [1].

3.2.3 OWL

O grupo de trabalho Web Ontology Working Group do W3C tem desenvolvido a linguagem **OWL** *Web Ontology Language*, com o objetivo de prover meios para as aplicações entenderem o conteúdo das informações com as quais trabalham. Em OWL, termos de vocabulários podem ser representados explicitamente, bem como as relações entre entidades nestes vocabulários. Segundo o W3C, neste sentido OWL vai além de XML, RDF e RDF Schema, ao permitir maior compreensão do conteúdo da Web pelas máquinas.

A linguagem OWL foi criada a partir de DAML+OIL e é uma revisão desta, incorporando o que se aprendeu no desenvolvimento e aplicação de DAML+OIL. Porém, consciente de que uma linguagem tão expressiva quanto DAML+OIL pode dificultar a criação de ferramentas de edição para a linguagem completa, o W3C definiu um subconjunto de OWL, chamado de *OWL Lite*. O objetivo de OWL Lite é prover uma linguagem que seja vista por desenvolvedores de ferramentas como suficientemente fácil e útil de suportar. OWL Lite tenta cobrir várias das características comumente utilizadas em OWL e DAML+OIL. Com isto espera-se facilitar a adoção de OWL na Web.

Documentação detalhada de OWL e OWL Lite pode ser encontrada em [20].

Referências

- [1] *The DARPA Agent Markup Language*. <http://www.daml.org>.
- [2] *Dublin Core Metadata Initiative*. <http://purl.org/DC>.
- [3] B. Bachimont, A. Isaac, and R. Troncy. Semantic commitment for designing ontologies: a proposal. *EKAW 2002, Lecture Notes in Artificial Intelligence (LNAI)*, (2473):114–121, 2002.
- [4] Tim Bray. *What is RDF?* xml.com, <http://www.xml.com/pub/a/2001/01/24/rdf.html>, 2001.
- [5] D. Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. <http://www.w3.org/TR/rdf-schema/>.
- [6] Cognitive Science Laboratory, Princeton University. *WordNet - a lexical database for the English language*. <http://www.cogsci.princeton.edu/wn>.
- [7] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 3rd edition, 2000.
- [8] D. Fallside. *XML-Schema Part 0: Primer*. <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>.
- [9] D. Fensel. Ontologies: Silver bullet for knowledge management and electronic commerce. *Springer-Verlag, to appear 2000*.
- [10] D. Fensel. Ontology-Based Knowledge Management. *IEEE Computer*, 35(11):56–59, 2002.
- [11] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. Van Harmelen, M. Klein, S. Staab, and R. Studer. *OIL: The Ontology Inference Layer*. <http://www.ontoknowledge.com/oil>.
- [12] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. *OIL: The ontology inference layer*. Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences, setembro 2000.
- [13] Renato Iannella. An idiots guide to the Resource Description Framework. *The New Review of Information Networking*, 4, 1998.
- [14] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and schema-languages: Translating OIL-specifications in XML-Schema. In *Proceedings of the ECAI Workshop on Applications of Ontologies and Problem-Solving Methods*, Amsterdam, Netherlands, 2000. IOS Press.
- [15] E. Miller, R. Swick, and D. Brickley. *Resource Description Framework (RDF)*. <http://www.w3.org/RDF/>.
- [16] M. Missikoff, R. Navigli, and P. Velardi. Integrated approach to web ontology learning and engineering. *IEEE Computer*, 35(11):60–63, 2002.

- [17] S. Staab and A. Maedche. Ontology engineering beyond the modeling of concepts and relations. In *Proceedings of the ECAI Workshop on Application of Ontologies and Problem-Solving Methods.*, Amsterdam, Netherlands, 2000. IOS Press.
- [18] M. Uschold. Creating, integrating and maintaining local and global ontologies. *Workshop on Applications of Ontologies and Problem-Solving Methods, 14th European Conference on Artificial Intelligence*, 2000.
- [19] M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.
- [20] World Wide Web Consortium. *Feature Synopsis for OWL Lite and OWL*. <http://www.w3.org/TR/owl-features/>.
- [21] World Wide Web Consortium. *Extensible Markup Language (XML)*. <http://www.w3.org/XML>.