2
Fundamentos

A Web Semântica é uma proposta para estender a Web atual de modo que máquinas possam processar e integrar de forma mais inteligente a imensa quantidade de dados existente. Para alcançar este objetivo, diversos esforços vêm sendo realizados por diferentes comunidades. A idéia principal que fundamenta esta nova Web é a capacidade de associar explicitamente semântica a conteúdo.

A linguagem atual HTML [Raggett et al., 1999] não tem este poder de expressividade e nem mesmo a linguagem XML [Bray et al., 2000]. Tornou-se necessário, então, desenvolver novas linguagens mais apropriadas como RDF [Lassila & Swick, 1999], RDF(S) [Brickley & Guha, 2000], DAML+OIL [Connolly et al., 2001].

Estas linguagens permitem que semântica - isto é, significado - seja explicitamente associada ao conteúdo dos dados na Web. Esta semântica pode ser formalmente especificada através de ontologias, compartilhadas pela Internet e possivelmente estendidas para atender a necessidades locais. Atualmente, o consórcio W3C está elaborando uma nova linguagem para ser utilizada como padrão para ontologias: *Web Ontology Language* (OWL) [Smith et al., 2002].

Este capítulo visa fornecer os fundamentos básicos para a compreensão desta tese de doutorado, agrupando os conceitos em seções relacionadas a: Ontologias, Linguagens para Especificação de Ontologias, Linguagens de Consulta, Arquiteturas de Implementação, Lógicas Descritivas, Facetas e Métodos de Projeto de Aplicações Web. Devido à grande diversidade de temas mencionados, serão apresentados resumos e referências para leituras mais aprofundadas.

2.1.
Ontologias

Uma ontologia define os termos usados para descrever e representar uma área de conhecimento [Heflin, 2003). Ontologias são utilizadas por pessoas, bancos de dados e aplicações que necessitam compartilhar informações

pertencentes a um domínio. As ontologias incluem definições que podem ser utilizadas por computadores e dizem respeito a conceitos básicos do domínio e os relacionamentos entre eles. O conhecimento do domínio fica então codificado, podendo ser reusado e assim ultrapassar as “fronteiras” do próprio domínio em questão.

A palavra ontologia tem sido usada para descrever artefatos com diferentes graus de estrutura, variando de simples taxonomias (como a hierarquia do Yahoo[1]) a esquemas de metadados (como Dublin Core[2]) e até mesmo teorias lógicas. A Web Semântica necessita de ontologias com um grau de estrutura significativo e com poder de especificar descrições para conceitos de:

- Classes em diversos domínios de interesse;
- Relacionamentos entre as classes; e
- Propriedades (atributos) que estas classes podem possuir.

Normalmente, as ontologias são expressas em uma linguagem baseada em lógica, para que distinções entre classes, relacionamentos e propriedades possam ser feitas de modo detalhado, preciso, consistente, completo e significativo. Algumas ferramentas podem realizar “raciocínio” (*reasoning*) automático com base nestas ontologias e fornecer serviços sofisticados a aplicações inteligentes.

Na Web Semântica, ontologias são utilizadas como uma forma de representar a semântica de documentos e permitir que esta semântica seja utilizada por aplicações Web e agentes inteligentes. Ontologias são úteis para estruturar e definir o significado de termos de metadados e são críticas para aplicações que precisam realizar buscas em informações de comunidades diversas e/ou integrá-las.

2.2. Linguagens para Especificação de Ontologias

A Web contém uma quantidade imensa de informação que está se expandindo rapidamente. A maior parte destas informações atualmente é representada em HTML, linguagem projetada para permitir que desenvolvedores Web apresentem informações a serem visualizadas por seres humanos através

[1] http://www.yahoo.com

[2] http://dublincore.org/

de *browsers*. Apesar de HTML permitir visualização de informações na Web, não nos fornece capacidade de descrever estas informações de modo a facilitar seu uso por programas, seja para localizá-las ou interpretá-las [DAML, 2003].

Nesta seção abordaremos diversas linguagens criadas para solucionar esta questão e viabilizar a Web Semântica. De forma resumida, podemos antecipar alguns comentários.

A Web Semântica está sendo construída com base na habilidade da linguagem XML definir marcações (*tags*) e na abordagem flexível que RDF possui para representar dados. Além destes elementos é necessário ainda termos uma linguagem capaz de descrever formalmente o significado dos termos utilizados em documentos da Web e também o relacionamento entre estes termos. Para que os computadores possam executar tarefas de inferência a partir destes documentos, a linguagem de definição precisa mais do que a semântica básica oferecida pelo RDF Schema (RDF(S)).

As linguagens da Web Semântica se inter-relacionam da seguinte forma:

- XML fornece uma sintaxe básica para estruturar documentos, sem impor nenhuma restrição semântica neles;
- XML Schema é uma linguagem para restringir a estrutura dos documentos XML e definir tipos de dados;
- RDF é um modelo de dados para objetos (recursos da Web) e seus relacionamentos, fornecendo semântica bem simples para este modelo de dados e permitindo uma representação XML do próprio modelo;
- RDF Schema é um vocabulário para descrever classes e propriedades de recursos RDF, com semântica de hierarquias de generalização/especialização destas classes e propriedades;
- DAML+OIL e OWL fornecem vocabulário adicional para descrever classes e propriedades, como relacionamento de disjunção, cardinalidade, igualdade (de classes e propriedades), classes enumeradas e tipos de dados mais ricos que as linguagens anteriores.

A seguir, abordaremos algumas vantagens e problemas existentes em cada uma destas linguagens.

2.2.1.
XML

A linguagem XML (eXtensible Markup Language)[3] vem sendo utilizada desde a sua recomendação pelo consórcio W3C em 1998, como uma forma de separar conteúdo de apresentação (e lidar somente com conteúdo). XML não tem um conjunto de marcações pré-definido e permite que definamos nossas próprias marcações de acordo com nossos critérios.

As formas mais básicas de verificação de um documento XML definem que ele pode ser bem-formado e/ou válido. Para ser bem-formado, um documento XML deve seguir as regras definidas na recomendação, como ter somente um elemento raiz, e possuir elementos com marcações de abertura e fechamento propriamente aninhadas. Para ser válido, um documento XML deve ter associado a ele uma definição chamada DTD (Document Type Definition) e deve atender às restrições expressas nesta definição.

Entretanto, esta forma de validação de documentos possui alguns problemas. O primeiro e mais importante para o escopo deste trabalho é que as definições expressas em uma DTD não possuem nenhuma semântica. A verificação que esta validação permite é apenas sintática. Além disto, há pouco suporte para restrições sobre o conteúdo dos elementos: restrições de cardinalidade são limitadas a 3 tipos (zero a um, zero a N e um a N) e somente o tipo de dados “cadeia de caracteres” tem suporte para validação. E mais, a ordem dos elementos é fixa, obrigando que um documento XML seja tratado como uma árvore (e não como um grafo). Por último, outro problema é que a linguagem para definição de uma DTD não é XML, e sim uma notação própria.

2.2.2.
XML Schema

A linguagem XML Schema[4] é uma recomendação do consórcio W3C definida em 2001 em dois documentos separados: Estrutura[5] e Tipos de Dados[6]. O primeiro especifica uma linguagem de definição de esquemas, que permite descrever a estrutura de um documento XML e restringir parcialmente seu

---

[3] A especificação formal de XML pode ser encontrada em: http://www.w3.org/TR/2000/REC-xml-20001006

[4] http://www.w3.org/XML/Schema

[5] (http://www.w3.org/TR/xmlschema-1/)

[6] (http://www.w3.org/TR/xmlschema-2/).

conteúdo. A segunda especificação define mais de 20 tipos de dados que podem ser utilizados em elementos e atributos, sendo que estes tipos de dados podem ser primitivos ou derivados. Esta segunda especificação é independente da primeira e pode ser reutilizada em diversos padrões XML.

Os tipos de dados definidos na linguagem XML Schema resolvem a limitação existente nas DTDs. Outra vantagem é o fato de que os esquemas definidos com esta linguagem são codificados em XML. Ou seja, XML Schema é um progresso em relação às DTDs, pelos fatores mencionados e também por permitir restrições na faixa de valores permitidos por tipos de dados: tamanho máximo, precisão, etc.

Esta linguagem é suficiente para troca de dados entre colaboradores que concordam previamente no uso de um vocabulário específico, mas com uma semântica ainda limitada, sem suporte para processamento automatizado de novos vocabulários. Um mesmo termo pode ser usado com diferentes significados. Ou diversos termos poderiam ser usados com um mesmo significado, sem que a linguagem ofereça suporte para essas distinções ou equivalências.

As linguagens RDF e RDF(S) iniciaram uma abordagem para solucionar este problema, ao permitir que semânticas simples fossem associadas a identificadores, conforme descrevemos na próxima sub-seção.

2.2.3. RDF e RDF(S)

Antes de iniciarmos o resumo relacionado a RDF e RDF Schema é necessário comentar dois tópicos usados extensivamente nestas linguagens: *Namespaces* e URIs.

2.2.3.1. Conceitos Básicos de Namespaces e URIs

O *Namespace* de um elemento é o escopo no qual ele está inserido e conseqüentemente o escopo onde ele é válido[7]. Necessitamos de *namespaces*, pois sem eles não temos recursos para lidar com colisão de nomes ao combinar

[7] Pode-se fazer uma analogia com o conceito de escopo de blocos {} em linguagens de programação.

múltiplos documentos da Web. Eles também permitem reusar vocabulários previamente definidos.

Um *Namespace* XML é um nome identificado por uma referência URI (Universal Resource Identifier), que serve para identificar qualquer recurso, seja ele uma página Web ou até mesmo uma pessoa. Documentos que fazem uso de namespaces normalmente apresentam, logo em seu início, uma seqüência de caracteres (que será usada como prefixo no restante do documento) e a URI que identifica o *Namespace* . Este prefixo, ao ser usado ao longo do documento, terá o seguinte formato:

```
<prefixo: tagXML>
```

2.2.3.2. RDF (Resource Description Framework)

De acordo com o consórcio W3C [W3C, 2002], a linguagem RDF (Resource Description Framework)[8] é uma linguagem de propósito geral para representar informações na Web. Ela foi criada para representar particularmente metadados a respeito de recursos Web. Entretanto, generalizando o conceito de "recurso Web", RDF pode ser usada para representar informações a respeito de tudo que possa ser identificado na Web, como informações sobre produtos disponíveis em lojas *on-line*.

RDF diz respeito a expressões processáveis por máquinas, pois fornece um mecanismo para descrever recursos sem fazer nenhuma suposição a respeito do domínio da aplicação. Os recursos são descritos através de metadados (dados a respeito de dados). Vale comentar que a sintaxe XML pode ser usada.

As expressões RDF são divididas em três partes: sujeito, predicado e objeto. O sujeito identifica a que a expressão diz respeito, o predicado identifica uma propriedade, e o objeto é o valor desta propriedade.

Para lidar com as limitações de XML, RDF propõe um modelo de dados simples, que pode ser usado para representar qualquer tipo de dado. Este modelo de dados consiste de nós conectados por arcos rotulados, onde os nós representam recursos Web e os arcos representam propriedades destes recursos. Pode-se observar que este modelo é essencialmente uma rede

[8] A especificação formal de RDF pode ser encontrada em: http://www.w3.org/TR/REC-rdf-syntax/

semântica, porém sem fornecer herança (especialização/generalização) [Heflin, 2001].

A Figura 1 apresenta uma representação gráfica de uma sentença RDF e a Figura 2 apresenta seu equivalente em formato XML serializado.

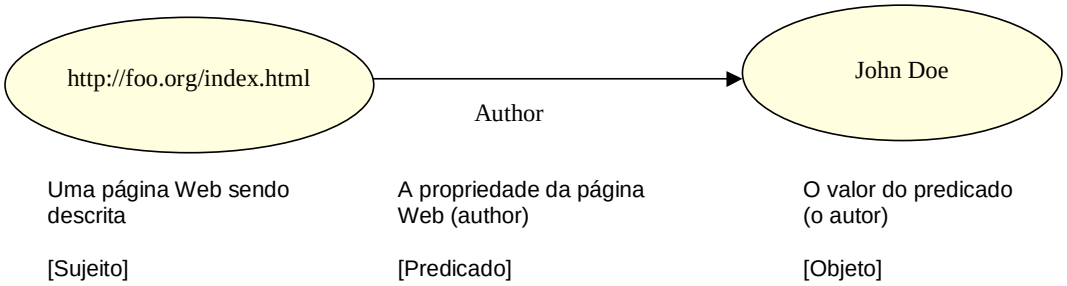

Figura 1 – Exemplo RDF em representação gráfica de grafo

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
     xmlns:test="http://www.teste.org/testschema/">

    <rdf:Description about="http://foo.org/index.html">
        <test:Author>John Doe</test:Author>
    </rdf:Description>
</rdf:RDF>
```

Figura 2 – Exemplo RDF em formato serializado RDF/XML

A sintaxe básica consiste em um elemento `Description` que contém um conjunto de sub-elementos. O atributo `about` identifica o recurso que está sendo descrito. Para evitar conflito de nomes entre diferentes vocabulários, RDF utiliza o conceito de *namespaces*, que aparece neste exemplo como `test`. Enfim, RDF oferece um suporte básico a ontologias, mas ainda é preciso mais.

2.2.3.3. RDF(S) (Resource Description Framework Schema)

Para permitir a criação de vocabulários controlados, compartilháveis e extensíveis, o W3C desenvolveu uma especificação da linguagem de definição de esquemas RDF chamada RDF Schema ou simplesmente RDF(S)[9].

Um esquema é um tipo de ontologia onde são definidos os termos que serão usados nos documentos RDF e o significado específico de cada termo. São justamente os esquemas que expressam a semântica dos documentos RDF.

[9] RDF(S) ainda não alcançou o status de recomendação W3C, mas sua especificação pode ser encontrada em: http://www.w3.org/TR/rdf-schema/

A especificação RDF(S) define algumas classes e propriedades que possuem semântica específica, como as classes `rdfs:Class` e `rdfs:Property`, que permitem atribuir um tipo a um recurso Web: tipo classe ou propriedade, respectivamente. Propriedades podem ser usadas para descrever tanto classes quanto outras propriedades. Por exemplo, a propriedade `rdfs:subClassOf` é usada para declarar que uma classe é subclasse de outra e permite a criação de taxonomias de classes para organizar recursos. RDF(S) também fornece propriedades para descrever propriedades, como `rdfs:subPropertyOf` que permite especialização de propriedades. Já as propriedades `rdfs:domain` e `rdfs:range` definem restrições globais a serem aplicadas em outras propriedades.

Observe as anotações na Figura 3 e note que um esquema RDF(S) é definido em sentenças RDF.

```
<rdf:RDF xml:lang="en"                                   RDF Schema Namespace
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<rdf:Description ID="MotorVehicle">          An "ID" attribute actually defines a
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>   new resource
  <rdfs:subClassOf
   rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>   "Resource" is the top
</rdf:Description>                                                  level class
<rdf:Description ID="PassengerVehicle">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>   PassengerVehicle inherits from
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>                     MotorVehicle
</rdf:Description>
<rdf:Description ID="Truck">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description ID="Van">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
<rdf:Description ID="MiniVan">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>   'Multiple
  <rdfs:subClassOf rdf:resource="#Van"/>                              Inheritance
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>
<rdf:Description ID="registeredTo">
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>          A Domain refers to which Objects
</rdf:Description>                              can use this resource as a property
<rdf:Description ID="rearSeatLegRoom">
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:domain rdf:resource="#Minivan"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes#Number"/>
</rdf:Description>                              The Range refers to the objects
</rdf:RDF>                                      that it can use as a property
```

Figura 3 – Exemplo de um esquema RDF(S) [Gil & Ratnakar, 2001]

Usando RDF(S), esquemas podem ser estendidos ao criarmos novos esquemas referenciando objetos identificados inicialmente como recursos em outro esquema. Devido ao uso de URIs e *Namespaces* é possível garantir que

apenas um objeto está sendo referenciado. No entanto, não é possível exprimir equivalência entre classes ou entre propriedades. Ou seja, não há uma forma de declarar que classes com nomes diferentes em esquemas diferentes são equivalentes.

Ou seja, mesmo com RDF(S), RDF ainda possui uma semântica pouco expressiva. Outras linguagens de maior poder expressivo podem fazer uso de seus fundamentos para intercâmbio de dados, ou seja, a sintaxe RDF pode ser usada, mas com novas classes e propriedades para definir semânticas específicas. É isto que veremos na próxima sub-seção.

2.2.4. DAML+OIL

A linguagem para definição de ontologias DAML+OIL [Connolly et al., 2001] pode ser considerada uma linguagem lógica cuja sintaxe é definida em RDF(S) . A semântica da linguagem é parcialmente definida dentro da semântica de RDF(S). Por isto, costuma-se dizer que DAML+OIL tem como base[10] as linguagens RDF e RDF(S). É importante comentar que DAML+OIL é usado para definir só os esquemas das ontologias, "delegando" para a linguagem RDF a tarefa de definição das instâncias.

DAML+OIL é utilizada para expressar um conjunto mais rico de primitivas de modelagem de ontologias, como restrições (*constraints*), enumerações e tipos de dados. A especificação mais recente de DAML+OIL [van Harmelen et al., 2001], utiliza tipos de dados de XML Schema [Biron & Malhotra, 2001].

De um ponto de vista formal, DAML+OIL pode ser visto como uma linguagem equivalente à lógica descritiva chamada SHIQ [Horrocks et al., 2000], com o acréscimo de classes definidas existencialmente (ou seja, com o construtor `oneOf`) e tipos de dados (freqüentemente chamados de domínios concretos em lógica descritiva) [Bonifácio, 2002]. A lógica descritiva mais próxima de DAML+OIL encontrada na literatura é chamada SHOQ(D) e é descrita em [Horrocks & Sattler, 2001].

O fato de DAML+OIL ser considerada um tipo de lógica descritiva permite o uso das investigações destas lógicas, como algoritmos conhecidos e resultados em relação a tratabilidade. Um sistema de representação de

[10] Esta é uma questão polêmica, pois DAML+OIL ignora algumas definições de RDF e RDF(S), e estende outras, conforme descrito em [Pan & Horrocks, 2001].

conhecimento e inferência baseado de lógica descritiva deve fornecer funções de inferência que podem ser agrupadas em dois tipos: inferências terminológicas e inferências de instâncias [Simov & Jordanov, 2002]. As do primeiro grupo podem ser descritas como quatro tarefas básicas que são implementadas na maioria dos sistemas:

- verificação de subjugamento (*subsumption)*: verifica se uma classe é subclasse de outra;
- verificação de consistência: busca inconsistências nas definições de classes;
- construção de taxonomias: computa todas as relações de subclasse (inclusive as inferidas que não foram explicitamente declaradas, mas sim implicadas por definições fornecidas);
- classificação: determina quais as superclasses ou subclasses imediatas de uma determinada classe.

As inferências de instâncias envolvem três tarefas individuais, que são:

- realização: dada uma descrição parcial de uma instância, encontra a definição de classe mais específica que a descreve;
- verificação de instâncias: dada uma descrição parcial de uma instância e de uma classe, verifica se aquela classe descreve a instância;
- recuperação de instância: encontra todas as instâncias que são descritas por uma classe.

Agora detalhando um pouco mais a linguagem, podemos dizer que tal como em uma lógica descritiva, as classes DAML+OIL podem ser nomes (URIs) ou expressões e uma variedade de construtores é fornecida para a construção de expressões de classes. O poder expressivo da linguagem é determinado por construtores de classes (e propriedades), e tipos de axiomas suportados [Bonifácio, 2002].

DAML+OIL permite expressões de classe para definição de classes que podem ser: uma única classe, uma lista de instâncias que compõem a classe, uma restrição em alguma propriedade, ou uma combinação “booleana” de expressões de classe. As propriedades `daml:intersectionOf`, `daml:unionOf`, e `daml:complementOf` fornecem conjunção, disjunção e negação em expressões de classe.

Uma restrição em propriedade é indicada pela classe `daml:Restriction`, que contém uma propriedade `daml:onProperty` para especificar qual o atributo sendo restringido e também fornecer informações

adicionais a respeito da restrição. A propriedade `daml:toClass`[11] é usada para afirmar que todos os valores do atributo em questão devem ser membros de uma determinada classe. Já `daml:hasClass` define que ao menos um valor do atributo deve ser membro de alguma classe especificada. A propriedade `daml:hasValue` declara que ao menos um valor do atributo tem que ser igual ao valor especificado. Restrições de cardinalidade são chamadas `daml:minCardinality` e `daml:maxCardinality`.

DAML+OIL também fornece primitivas para definir propriedades. Além das básicas disponíveis em RDF, existem a propriedade `daml:inverseOf` e a classe `daml:TransitiveProperty`. As classes `daml:UniqueProperty` e `daml:UnambiguousProperty` servem para definir que uma propriedade só pode ter um valor por instância e que este valor só pode pertencer a uma instância.

Para afirmar que dois recursos são idênticos, a propriedade `daml:equivalentTo` pode ser usada, tanto com classes, quanto com propriedades e até mesmo instâncias.

A Figura 4 apresenta um exemplo comentado.

[11] De acordo com [van Harmelen et al, 2003], várias propriedades e classes de DAML estão sendo renomeadas na definição da linguagem OWL. A tabela abaixo apresenta um resumo da situação atual.

DAML+OIL	OWL
daml:differentIndividualFrom	owl:differentFrom
daml:equivalentTo	owl:sameAs
daml:sameClassAs	owl:equivalentClass
daml:samePropertyAs	owl:equivalentProperty
daml:hasClass	owl:someValuesFrom
daml:toClass	owl:allValuesFrom
daml:UnambiguousProperty	owl:InverseFunctionalProperty
daml:UniqueProperty	owl:FunctionalProperty

```
<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:daml="http://www.daml.org/2000/12/daml+oil#"
  xmlns     ="http://www.daml.org/2000/12/daml+oil-ex#"
>

<daml:Ontology about=””>
<daml:versionInfo>An example ontology</daml:versionInfo>

<daml:Class rdf:ID="Animal">
  <rdfs:label>Animal</rdfs:label>
  <rdfs:comment>
    This class of animals is illustrative of a number of ontological idioms.
  </rdfs:comment>
</daml:Class>

<daml:Class rdf:ID="Male">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>

<daml:Class rdf:ID="Female">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Man">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Male"/>
</daml:Class>

<daml:Class rdf:ID="Woman">
  <rdfs:subClassOf rdf:resource="#Person"/>
  <rdfs:subClassOf rdf:resource="#Female"/>
</daml:Class>

<daml:ObjectProperty rdf:ID="hasParent">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="hasFather">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
  <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>

<daml:DatatypeProperty rdf:ID="age">
    <rdfs:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty

<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:about="#Animal">
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="2">
      <daml:onProperty rdf:resource="#hasParent"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxcardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Start of an ontology (about = “” implies ‘this’ document)

The label is **not** used for logical interpretation

Can explicitly specify the set of Females to be disjointWith the set of Males

The Person class is defined later

To be read conjunctively. A man is a sub-class of ‘Person’ and a ‘Male’

An objectProperty relates objects to objects

Describes the element which encloses this Property

Describes the value of the Property

Note: Contrary to RDF, DAML takes the ‘intersection’ of the domains/ranges if multiple domains/ranges are specified

A datatype property relates an object to a primitive datatype value

The XML Schema datatype is referenced here

The Restriction defines an anonymous class of all things that satisfy the restriction.

Restrictions on the property hasParent (only for the Person class – Local scope, as opposed to rdfs:range).

A Person can have only 1 Father

Addition to the Animal Class without modifying it -- “about”

Restrictions on the property hasParent
An animal can have exactly 2 parents

Restrictions on the property hasSpouse
A person can have only 1 spouse

Figura 4 – Exemplo de DAML+OIL [Gil & Ratnakar, 2001]

2.2.5. Comparação

A Tabela 2 apresenta uma comparação resumida entre a funcionalidade de DAML+OIL e de outras linguagens mencionadas até este momento. Detalhes podem ser encontrados em http://www.daml.org/language/features.html

	XML DTD	XML Schema	RDF(S)	RDF(S) 2002	DAML+ OIL	OWL?
bounded lists				X	X	X
cardinality constraints	X	X			X	X
class expressions					X	X
data types		X		?	X	X
defined classes					X	X
enumerations	X	X			X	X
equivalence					X	X
extensibility			X	X	X	X
formal semantics				X	X	X
inheritance			X	X	X	X
inference					X	X
local restrictions					X	X
qualified constraints					X	
reification			X	X		X

Tabela 2– Comparação resumida da funcionalidade das linguagens XML DTD, XML Schema, RDF(S), DAML+OIL e OWL [12]

Examinando-se a Tabela 2, vê-se que DTDs têm um poder de expressão menor que as outras linguagens, permitindo apenas especificar restrições de cardinalidade e enumerações. A linguagem XML Schema acrescenta uma característica adicional, permitindo utilizar tipos de dados pré-definidos. Já a primeira versão de RDF(S) permitir extensões de classes e propriedades, herança e reificação. A versão mais recente de RDF(S) de 2003, adicionou novas características como: uma semântica formal, a possibilidade de usar tipos de dados de XML Schema e de definir listas completas. Com DAML+OIL, muitas outras funcionalidades foram acrescidas, como: especificação de expressões de classe capazes de definir disjunção, união, interseção e complemento; especificação de novas classes com base em restrições em valores de propriedades de uma classe existente; equivalência entre classes, propriedades

[12] http://www.daml.org/language/features.html

e instâncias de diferentes ontologias para permitir integração; especificação de transitividade, unicidade, disjunção, propriedade inversa que permitem realizar inferências; especificação de restrições locais para permitir adequar o domínio de uma propriedade a diferentes classes; especificação de restrições qualificadas de cardinalidade e classe. A especificação de restrições qualificadas foi considerada redundante pela equipe de definição da linguagem OWL e foi retirada.

A Tabela 3 apresenta uma comparação comentada entre as linguagens XML Schema, RDF(S) e DAML+OIL. O leitor interessado em exemplos desta comparação deve consultar [Gil & Ratnakar, 2002]

Dimensão	**XML Schema**	**RDF Schema ou RDF(S)**	**DAML+OIL**	**OBS**
Escopo de Nomes (Namespace)	Default Namespace: *xmlns* Para declarar outro: *xmlns:<label>* Ex.: XMLSchema Namespace (rotulado de xsd) *xmlns:xsd="www.w3.org/2001/XMLSchema"*	RDF usa XML Namespaces. RDF Syntax Namespace xmlns:rdf="http://www.w3.org/ 1999/02/22-rdf-syntaxns#" RDF Schema Namespace xmlns:rdfs="http://www.w3.org/ 2000/01/rdfschema#" Nota: Em RDF, a URI do namespace identifica a localização do RDF Schema	DAML também usa XML Namespaces. Ele usa elementos RDF & RDF(S) referenciando-se ao seus respectivos Namespaces. O DAML Namespace mais recente é *xmlns:daml= "http://www.daml.org/2001/03/daml+oil#"* Em DAML, deve-se importar as ontologias para ter a capacidade de usar as classes definidas na ontologia	
Classes e Propriedades	Sem conceito de classes e propriedades, somente Elementos de certos tipos. O tipo pode ser *simpleType* ou um *complexType*. (Uma classe pode ser qualquer elemento e sua propriedade pode ser seu elemento filho – não há semântica pré-definida)	Resource é a classe de nível mais alto. (*http://www.w3.org/2001 /01/rdfschema# Resource*) Classes são definidas como *rdfs:Class* e Propriedades são definidas como *rdfs:Property* Nota: Uma propriedade pode ser equivalente a um atributo da UML ou a um relacionamento de associação	DAML também possui Classes e Propriedades. Dois tipos de propriedades são definidos: *ObjectProperties* (Relaciona um objeto a um outro objeto – o valor da propriedade é também um objeto)e *DatatypeProperties* (Relaciona um objeto a um tipo de dado primitivo - o valor da propriedade é um tipo de dado primitivo)	

Dimensão	XML Schema	RDF Schema ou RDF(S)	DAML+OIL	OBS
Herança	Visto que não existe Classe, não há o conceito de herança. Entretanto, Tipos podem ser estendidos ou restritos, definindo-se subtipos	Uma classe pode ser um subclasse de outras classes (*subClassOf*) - herança múltipla é permitida. Propriedades podem também ser subpropriedades de outras propriedades (*subPropertyOf*)	Igual ao RDFS. Além disto, classes podem ser subclasses de uma classe anônima, devido a uma Restrição (*Restriction*) sobre o conjunto de todas as 'Coisas' (*Thing*)	
Contra-domínio de Propriedade (Range Property)	Pode ser especificado global e localmente. Para especificar um contra-domínio localmente, uma restrição deve ser definida em um elemento local	Só pode ser especificado globalmente: *<rdfs:range....>* Declarações múltiplas implicam conjunção (todas devem ser satisfeitas)	Podem ser especificadas globalmente *<rdfs:range...>* quanto localmente *<daml:Restriction>* *<daml:onProperty...>* *<daml:toClass....>* *</daml:Restriction>* Declarações múltiplas implicam conjunção (todas devem ser satisfeitas).	Contra-domínio especifica o tipo de valor que uma propriedade pode ter
Domínio de Propriedade (Domain Property)	Sem declaração explícita do domínio do elemento. O domínio é, implicitamente, o elemento na qual a definição aparece	Pode ser especificado só globalmente *<rdfs:domain....>* Declarações múltiplas de domínio implicam conjunção (todas devem ser satisfeitas)	Pode ser especificado globalmente (*<rdfs:domain...>).* Declarações múltiplas de domínio implicam conjunção (todas devem ser satisfeitas)	Domínio especifica quais classes podem ter uma determinada propriedade
Cardinalidade de Propriedade	Pode ser especificado usando-se *minOccurs* e *maxOccurs*	Não definido em RDF Schema. Por definição, não existem restrições de cardinalidade sobre propriedades. Entretanto, novas restrições como estas podem ser especificadas fazendo com que elas sejam subclasses da classe *ConstraintProperty*	Podem ser especificadas localmente *minCardinality*, *maxCardinality*, *cardinality* Também podem ser especificadas globalmente, embora somente como uma *UniqueProperty* (com valor simples de 1)	A cardinalidade de uma propriedade especifica o número de ocorrências da propriedade para uma certa classe
Tipos de dados básicos	Os tipos de dados suportados pelo XMLSchema são, principalmente, variações de tipos de dados numéricos, de datas e strings	RDF Schema somente inclui 'Literais' como o conjunto de todas strings. (Na verão mais nova, pode-se utilizar os tipos de dados de XMLSchema)	Permite o uso dos tipos de dados do XMLSchema pelo simples referência a URI do XMLSchema	

Dimensão	XML Schema	RDF Schema ou RDF(S)	DAML+OIL	OBS
Enumeração de valores de propriedades	Possível com a tag <*enumeration*>	Não é possível especificar	Enumeração de tipos de propriedades é possível através da tag <*daml:one of rdf:ParseType= "daml:collection"...*> Também é possível utilizar um tipo de dado enumerado de XML Schema	Uma Enumeração restringe o valor de uma propriedade para um conjunto de valores pré-definido
Conjunto de dados ordenados	Data Sets mantêm ordem por definição Nota: Tipos de declarações possuem a tag <*sequence*>, só significa que os dados devem aparecer na ordem especificada	Data Set ordenado com a tag <*rdf:Seq...*>	Possível com a tag <*daml:list*>.	
Listas restritas (Bounded Lists)	Não é possível especificar	Não é possível especificar	Possível com a tag <*daml:collection*>	
Propriedades transitivas	Não é possível especificar	Não é possível especificar	Possível com a tag <*daml:TransitiveProperty*>	
Negação	Não é possível especificar	Não é possível especificar	Possível com a tag <*daml:complementOf*>	Negação implica a ausência de algum elemento (nenhum Carro é uma pessoa)
Classes disjuntas (Disjoint)	Uma união de possíveis tipos para um elemento é possível com a tag <*union*>	Pode usar um Bag para indicar coleções desordenadas (ou uniões de propriedades). Entretanto, não se pode ter uma classe como uma união de 2 classes	Uma classe pode ser uma união de 2 outras classes. Possíveis com a tag <*unionOf*...>. Pode-se, ainda, representar uniões disjoint com a tag <*disjointUnionOf*...>	

Dimensão	XML Schema	RDF Schema ou RDF(S)	DAML+OIL	OBS
Condições necessárias e suficientes para membros de uma classe	Não (embora <*unique*> possa ser interpretado como uma *UnambigousProperty*)	Não é possível especificar	Possível com as tags *sameClassAs*, *equivalent*, e usando combinações booleanas de expressões de classes. *UnambigousProperty* especifica uma propriedade que identifica um recurso (como uma chave primária)	Condições necessárias e suficientes para membros de uma classe (Class Membership) especificam uma definição de classe que pode ser usada: 1) para determinar (ou identificar) se uma instância pertence àquela classe 2) para determinar (ou classificar) se a classe é uma subclasse de uma outra classe

Tabela 3 – Resumo da comparação entre as linguagens XML Schema, RDF(S) e DAML+OIL [Gil&Ratnakar2002] [13]

2.3. Linguagens de Consulta

RDF pode ser representado em formato XML (serializado). Aliás, pode ser representado de diversas formas em formato XML. Justamente esta diversidade de formas sintáticas faz com que seja extremamente difícil consultá-las com uma linguagem de consulta criada para consultar apenas XML.

Consultar XML no nível sintático ou no nível estrutural não extrai informação semântica. Uma consulta sintática nos força a navegar em uma árvore rotulada nos nós, enquanto que RDF é um grafo dirigido rotulado tanto nas arestas quanto nos nós. Uma consulta estrutural permite explorar as definições de classes, mas não auxilia a explorar relacionamentos de subclasse indicados na semântica. As linguagens para consultar RDF devem possuir capacidade de lidar com a semântica da linguagem, inclusive com a habilidade de explorar o esquema RDF.

[13] Traduzido em [Bonifácio 2002].

2.3.1. RQL

A linguagem de consulta RQL [Karvounarakis et al., 2002] é uma linguagem tipada seguindo uma abordagem funcional como ODMG-OQL [Cattell et al., 2000]. RQL oferece suporte a expressões genéricas de caminho com variáveis tanto nos nós (i.e., classes) quanto nas arestas (i.e., propriedades). Esta linguagem tem como base um modelo formal de grafos que captura as primitivas de modelagem RDF e permite a interpretação de descrições superpostas de recursos utilizando um ou mais esquemas.

A novidade de RQL consiste em sua habilidade de combinar consultas a esquemas e instâncias de forma integrada. Além disto, a linguagem oferece suporte a: tipos de dados XML Schema (para filtrar valores literais), agrupamento de primitivas (para construir resultados XML aninhados), operações aritméticas (para converter valores literais), funções de agregação (para extrair estatísticas), facilidades de *namespaces* (para lidar com múltiplos esquemas), consulta a meta-esquemas e navegação recursiva em hierarquia de classes e propriedades.

RQL é definida através de: um conjunto de consultas padrão, um conjunto de filtros básicos, e uma forma de construir novas consultas por composição funcional e iteradores. Outras características que serão úteis para nossos exemplos são funções de agregação, predicados "booleanos" e operadores de conjuntos. A Tabela 4 apresenta três exemplos de consultas RQL. A primeira consulta retorna instâncias de uma classe utilizando apenas o nome da classe. A segunda recupera subclasses através do uso da palavra reservada `subclassof`. Por fim, a terceira consulta utiliza o prefixo `$` para consultar o esquema. Após a cláusula `from`, é usada uma expressão de caminho composta da propriedade "cria" (uma aresta) e de duas variáveis de classe $x e $y (variáveis sobre rótulos dos nós). A notação {} é usada para introduzir variáveis de esquema($x) e dado(x). Esta consulta irá retornar os nomes das classes às quais estas propriedades pertencem e também as instâncias.

Consultas RQL	Descrições
http://www.icom.com/schema.rdf#Pessoa	recupera todas as instâncias da classe "Pessoa"
subclassOf(http://www.icom.com/schema.rdf# Pessoa)	recupera todas as subclasses da classe "Pessoa"
select x, $x, y, $y from {x; $x} #cria {y; $y}	recupera todos os recursos que tenham uma propriedade chamada 'cria'. Recupera também o destino desta propriedade e os nomes das classes as quais origem e destino pertencem. Este é um exemplo de consulta a esquema e dado (instância).

Tabela 4 - Exemplos de Consultas RQL

Devemos ressaltar que até a época do término desta tese, RQL era a única linguagem capaz de consultar definições de esquemas e dados integradamente.

2.3.2. Comparação com outras linguagens de consulta

No Anexo 2 apresentamos uma tabela de comparação entre poder expressivo de diversas linguagens de consulta para RDF(S). Devido a restrições de espaço o leitor interessado em detalhes das outras linguagens e na comparação propriamente dita deve consultar [Magkanaraki et al., 2002].

2.4. Arquiteturas de Implementação

2.4.1. ICS-FORTH

O ambiente de implementação ICS-FORTH RDFSuite[14] [Alexaki et al., 2001], cuja equipe contém autores da linguagem RQL, foi um dos primeiros ambientes a oferecer suporte à linguagem de consulta RQL. No entanto, esta implementação não possui a mesma flexibilidade de Sesame (a seguir), tendo sido criada para os sistemas operacionais Mandrake Linux e Solaris.com armazenamento persistente somente no SGBD PostgreSQL.

[14] http://139.91.183.30:9090/RDF/index.html

2.4.2.
Sesame

A arquitetura Sesame[15], desenvolvida em Java, oferece armazenamento RDF e RDF(S) e suporte à linguagem de consulta RQL. Além disto, o ambiente básico é capaz de realizar inferências nas definições RQL(S) (como transitividade) e persistir as novas tuplas inferidas, tanto em relação ao esquema quanto às instâncias.

Existe um ambiente mais recente com um *middleware* chamado OMM[16] (*Ontology Middleware Module*), desenvolvido para oferecer suporte a três questões importantes: inferências para linguagem mais expressivas que RDF(S) como DAML+OIL, segurança e versionamento. Maiores detalhes podem ser obtidos em [Kiryakov et al., 2002a, b].

2.4.3.
SeBOR

Em novembro de 2002, o centro de pesquisa Sirma[17] disponibilizou um "raciocinador de lógica descritiva" (*description logic (DL) reasoner*) chamado BOR[18] [Simov & Jordanov, 2002]. Mais precisamente, este raciocinador oferece suporte a lógica descritiva SHQ(D), sendo SHQ(D) uma lógica SHOQ(D) sem definições nominais. O procedimentos de decisão de BOR são baseados em algoritmos *tableaux* descritos em [Horrocks & Sattler, 2001]. O suporte oferecido inclui definições DAML+OIL e OWL, entretanto nem todas as definições estavam operacionais até o momento do término desta tese.

Como parte do projeto On-To-Knowledge[19], BOR foi integrado à arquitetura Sesame em uma versão chamada SeBOR (SesameBOR). A integração foi efetuada através de uma camada de inferência adicional, capaz de tratar ontologias definidas em DAML+OIL e obter novas tuplas RDF(S) inferidas. Os processos de inferência ocorrem tanto no nível de esquema quanto de instância, portanto novas tuplas de instâncias RDF também são obtidas e armazenadas em qualquer repositório que estiver em uso. Por exemplo, ao usarmos uma

[15] http://sesame.aidministrator.nl

[16] http://www.ontotext.com/omm

[17] http://www.sirma.bg/OntoText/

[18] http://www.ontotext.com/bor, http://www.sirma.bg/OntoText/bor/index.html

[19] http://www.ontoknowledge.org/

definição `daml:sameClassAs`, o raciocinador BOR gerará as equivalências correspondentes e armazenará as definições e instâncias nas linguagens RDF(S) e RDF no repositório Sesame.

A linguagem de consulta RQL continua habilitada a lidar com o repositório, uma vez que as tuplas armazenadas utilizam RDF e RDF(S). Vale comentar que atualmente o módulo BOR oferece suporte apenas a parte das definições DAML+OIL, conforme detalhado em [Jordanov & Simov, 2002]

2.4.4. Comparação com outras Arquiteturas de Implementação para RDF(S)

No Anexo 1 apresentamos uma tabela de comparação entre ferramentas e ambientes para armazenamento e consulta de ontologias. Devido a restrições de espaço o leitor interessado deve consultar [Magkanaraki et al., 2002] para detalhes de outras ferramentas e ambientes para armazenamento e consulta de ontologias e detalhes da própria comparação.

2.5. Description Logics

De acordo com a página Web oficial da área de lógicas descritivas (*Description Logics* ou DL) [Lutz, 2002], o principal esforço das pesquisas de representação do conhecimento é fornecer teorias e sistemas para: expressar conhecimento estruturado, fazer acesso a este conhecimento e também obter conclusões lógicas a partir dele. A família de lógicas descritivas é considerada como o formalismo mais importante para representação do conhecimento, unificando e fornecendo base lógica para sistemas tradicionais baseados em *frames,* linguagens de redes semânticas e tipo-KL-ONE, representações orientadas a objeto, modelos de dados semânticos e sistemas de tipos.

Para o processamento do conhecimento disponível na Web Semântica o uso de máquinas de inferência se faz necessário, de modo a deduzir novo conhecimento a partir de conhecimento previamente especificado. Duas abordagens diferentes são aplicáveis: o uso de máquinas de inferência baseadas em lógica e o uso de algoritmos especializados (métodos de resolução de problemas). A primeira abordagem engloba diferentes tipos de linguagem de representação e máquinas de inferência como lógicas descritivas.

A área de lógicas descritivas existe como área de pesquisa há mais de duas décadas, entretanto apenas recentemente transformou-se em uma área de

interesse generalizado, devido à sua aplicabilidade na área de Web, conforme comentamos nesta subseção.

Historicamente, as lógicas descritivas, previamente conhecidas como *Terminological Logics* e sistemas *KL-ONE-like*, tiveram como motivação principal fornecer os alicerces formais para as redes semânticas [McGuinness, 2001]. O primeiro sistema DL implementado, KL-ONE [Brachman, 1977], utilizou uma abordagem de conjunto fixo de primitivas epistemológicas independentes de domínio, que poderiam ser usadas para construir descrições complexas e estruturadas de objetos. O foco principal era fornecer os fundamentos para construção de vocabulários de forma semanticamente significativa e não ambígua.

Desde então, diversos sistemas como CLASSIC [Brachman et al, 1989], BACK [Peltason, 1991], LOOM [MacGregor, 1991] e K-REP [Mays et al, 1991] foram projetados e implementados fornecendo diferentes posições nos requisitos de poder expressivo, completude de implicação e tratabilidade de implicação. Porém estes primeiros sistemas tipicamente sacrificavam algo - normalmente poder expressivo, mas em alguns casos, até mesmo completude - para manter a usabilidade.

Outras implementações de lógicas descritivas do final dos anos 90 como DLP [Patel-Schneider, 1998], FACT [Horrocks, 1999] e RACE [Haarslev & Moeller, 1999] são mais expressivas (ao menos no quesito implicação lógica de conceitos) e mantêm argumentadores/raciocinadores (*reasoners*) completos com implementações computacionalmente eficientes. Apesar destes esforços, DL ainda estava restrita a uma parte do mundo acadêmico até recentemente, ao se mostrar útil no ambiente da Web. Tem sido justamente este uso o responsável por promover a divulgação desta lógica no mundo da Web.

Diversas comunidades reconhecem que lógicas descritivas, com suas pesquisas de longa data em fundamentos formais para formalismos de representação de conhecimento estruturado, pode beneficiar as linguagens para a Web, como RDF e RDF(S). Desta nova geração de esforços surgiu OIL [Fensel et al, 2001], resultado da junção dos objetivos de sistemas baseados em *frames* (usabilidade), com linguagens para Web (uso disseminado) e lógicas descritivas (fundamentos formais para sistemas extensíveis e semanticamente compreensíveis). OIL significa *Ontology Inference Layer* e representa uma proposta de camada de representação e inferência para ontologias. Enquanto OIL era desenvolvido na Comunidade Européia, nos Estados Unidos era

desenvolvida a linguagem DAML [Hendler et al, 2000] com a mesma junção de objetivos mencionada acima. DAML significa *DARPA Agent Markup Language.*

A união das duas abordagens resultou na criação da linguagem DAML+OIL. Atualmente, DAML+OIL tem fornecido alicerces nos quais aplicações Web podem ser construídas. Estes alicerces são compatíveis com os padrões da Web: XML e RDF, proporcionando os fundamentos formais para especificação de significado de vocabulário de forma não ambígua.

O consórcio W3C, que é possivelmente a maior força atual na definição de padrões para a Web, já adotou DAML+OIL como a proposta inicial para a definição da linguagem de ontologias para Web OWL (*Web Ontology Language*). É esperado que aplicações futuras requeiram um poder de inferência maior do que as aplicações baseadas em ontologias do passado recente, como é o caso do Yahoo e outras aplicações baseadas em taxonomias. O que se almeja são aplicações “mais inteligentes” capazes de realizar deduções para o usuário. Segundo McGuinness [McGuinness, 2001], a ênfase de lógicas descritivas em fundamentos formais pode ter sido a responsável por sua distância das áreas de destaque, devido à abundância de trabalhos formais que podem ter desencorajado prováveis usuários. Atualmente, entretanto, as necessidades das novas aplicações da Web têm motivado pessoas a buscar alicerces nos quais aplicações extensíveis e duráveis possam ser construídas.

Segundo [Beck & Pinto, 2002] existem vários motivos para não se trabalhar com uma linguagem de representação do conhecimento que contenha todos os possíveis aspectos expressivos. A principal razão para não projetarmos linguagens complexas, de acordo com estudos da complexidade computacional de lógicas descritivas, é que as inferências executadas nestes sistemas são computacionalmente intratáveis. É necessário que seja definida uma linguagem que seja expressiva o bastante para construir ontologias ricas, mas não complexas demais a ponto de ser impossível executar operações computacionais. Aguardemos, portanto, as novas definições de OWL.

2.6. Facetas

Historicamente Ranganathan [Ranganathan, 1963] foi o primeiro a utilizar o termo “faceta” nas áreas de ciência da informação e biblioteconomia, e o primeiro a desenvolver, de forma consistente, uma teoria de “análise de facetas”.

Originalmente facetas foram definidas como aspectos, propriedades ou características de uma classe ou tópico específico definidas claramente, e devem ser mutuamente exclusivas e exaustivas. Ranganathan demonstrou que os processos de análise (divisão de tópicos em conceitos elementares) e síntese (recombinação destes conceitos em cadeias de tópicos) poderiam ser aplicados a todos os tópicos de forma sistemática. Apesar de as idéias apresentadas terem sido consideradas revolucionárias para a época, algumas restrições no método fizeram com que seu uso não fosse disseminado da forma original [Maple, 1995].

Entretanto, atualmente diversos autores têm feito menção a facetas, "análise de facetas" e acesso facetado com diferentes conotações. Segundo [Maple, 1995], análise de facetas é definida como a organização de termos de um domínio de conhecimento em facetas homogêneas, mutuamente exclusivas. Cada faceta é derivada de um "ancestral" através de uma única característica de divisão. Cada categoria lógica deve ser isolada e cada nova característica de divisão deve ser indicada claramente. Portanto, uma faceta consiste em um grupo de termos que representa uma e somente uma característica de divisão de um tópico pertencente a um domínio.

Na essência, a idéia é dividir um tópico (relevante ao domínio) em facetas de alto nível ou categorias fundamentais, o que é equivalente a classes principais em sistemas de classificação enumerativos. Cada faceta de alto nível é subdividida em componentes, as subfacetas, que representam diferentes características.

Facetas representam categorias de alto nível nas quais um determinado domínio está dividido. Hierarquias representam a subdivisão das facetas em subfacetas. Tradicionalmente, hierarquias em classificação sugerem divisões todo-parte ou *genus-species*, mas esta não é a base da divisão utilizada em "análise de facetas".

2.7. Projeto de Aplicações Hipermídia e Aplicações Web

De acordo com [Halasz, 1991], "Hipermídia é um estilo de construção de sistemas para a criação, manipulação, apresentação e representação da informação no qual:

- a informação é armazenada em uma coleção de nós multimídia;
- os nós são organizados explícita ou implicitamente em uma ou mais estruturas - normalmente, uma rede de nós conectados por *links*;

- os usuários podem fazer acesso à informação ao navegar nas estruturas de informação disponíveis ou através delas."

A partir desta definição podemos observar que um sistema hipermídia pode apresentar nós ou outras estruturas que serão acessadas quando o sistema apresentar estes nós aos usuários. E também que os usuários podem navegar através das informações apresentadas pelo sistema.

Atualmente, as aplicações tradicionais baseadas em sistema de informação também incorporam o conceito de navegação entre suas informações (nós navegacionais). Entretanto, os métodos usados durante o desenvolvimento de aplicações tradicionais não são suficientes para modelar a navegação, principalmente quando a quantidade de nós nos quais o usuário navega é grande. Estes métodos não abordam todos os aspectos relacionados com a navegação e geralmente não desassociam a navegação da interface com o usuário.

2.7.1. Métodos para Projeto de Aplicações Hipermídia e Aplicações Web

O reconhecimento da necessidade de métodos específicos para o desenvolvimento de aplicações hipermídia ocorreu no início da década de 90, com o surgimento do HDM (*Hypermedia Design Model*) [Garzotto et al., 1991, 1993]. A partir de então, vários outros métodos específicos para hipermídia foram definidos: RMM [Isakowitz et al., 1995], WebML [Ceri et al., 2000], UWE [Hennicker & Koch, 2000], OO-H Method [Gómez et al., 2000] e OOHDM [Schwabe & Rossi 1998]. De modo geral, além do projeto conceitual e de outras fases conhecidas do ciclo de desenvolvimento, tais métodos precisam incluir uma fase específica para o projeto navegacional [Rossi, 1999].

A navegação permite ao usuário, como parte da tarefa, explorar as informações necessárias para a realização desta. No projeto de navegação, são identificados os objetos navegacionais e a navegação entre estes objetos. A navegação é a operação pela qual se seguem os *links* (elos) de hipertexto que estão associados aos objetos de hipertexto e é, assim, um dos possíveis tipos de operações disparadas por meio da interface com o usuário.

O projeto de navegação fica bastante facilitado quando já existe um modelo que representa a interação entre o usuário e o sistema. Isto porque parte desta interação corresponde à navegação nos nodos de informação, bastando,

neste caso, apenas complementar o modelo com detalhes de projeto, tais como as estruturas de acesso utilizadas na navegação.

2.7.2. O Método OOHDM

O método OOHDM utiliza uma abordagem baseada em modelos para projetar aplicações Web, em um processo composto de 5 etapas: Levantamento de Requisitos, Modelagem Conceitual, Modelagem Navegacional, Projeto da Interface Abstrata e Implementação. A cada etapa, um conjunto de modelos orientados a objetos é construído ou enriquecido, descrevendo detalhes do projeto.

Segundo [Rossi et al., 1999], aplicações Web, bem como aplicações hipermídia, devem fornecer fácil acesso navegacional a seus recursos relacionados, evitando que usuários se percam e fornecendo operações de navegação consistentes. Para alcançar estas premissas, a abordagem OODHM define as seguintes primitivas de navegação [Schwabe & Rossi, 1998]:

- objetos navegacionais: visões de objetos conceituais;
- contextos navegacionais: conjuntos de objetos navegacionais de acordo com regras determinadas pelo projetista da aplicação;
- estruturas de acesso: índices que auxiliam o usuário a alcançar o objeto desejado.

Estas primitivas permitem que projetistas usando OOHDM definam a navegação de aplicações Web, decidindo quais atributos devem ser mostrados, quais objetos devem ser navegados, quais são seus relacionamentos com os objetos conceituais, quais contextos são úteis para a navegação, como o usuário pode ir de um contexto para outro, como é a navegação dentro de um contexto, e como perfis de usuário podem alterar a forma como os objetos são mostrados dentro dos contextos.

2.7.2.1. Modelagem Conceitual OOHDM

No método OOHDM, o Modelo Conceitual descreve um domínio específico através de classes e relacionamentos.

2.7.2.1.1.
Esquema de Classes Conceituais

Classes são representadas como em modelos orientados a objetos da UML, com um detalhe adicional nos atributos: eles podem ser multi-tipados representando diferentes perspectivas de uma mesma entidade do mundo real.

2.7.2.2.
Modelagem Navegacional OOHDM

O Modelo Navegacional OOHDM consiste de dois esquemas: o Esquema de Classes Navegacionais e o Esquema de Contextos de Navegação. O primeiro define todos os objetos navegacionais como visões sobre um domínio de aplicação, usando um conjunto pré-definido de classes: nós e elos. Já o segundo esquema define estruturas de acesso e também a principal primitiva do espaço navegacional: o contexto de navegação. Os contextos podem representar objetos relacionados uns aos outros em algum aspecto (ex.: objetos com atributos em comum ou objetos relacionados a um outro objeto em comum) e também podem organizar estes objetos como conjuntos de nós, definindo como eles serão acessados (ex.: seqüencialmente).

2.7.2.2.1.
Esquema de Classes Navegacionais

O conjunto de nós (classes navegacionais) e elos que fazem parte de uma visão da aplicação conhecida como Esquema de Classes Navegacionais ou Esquema Navegacional. Uma aplicação pode ter um ou mais esquemas navegacionais de acordo com as visões existentes da aplicação.

2.7.2.2.1.1.
Nós Navegacionais

No OOHDM, as informações pelas quais os usuários navegam em uma aplicação hipermídia são conhecidas como nós [Schwabe & Rossi, 1998]. Um nó, também denominado de classe navegacional, é uma visão sobre uma ou mais classes do Esquema Conceitual. Ele é definido de acordo com o conjunto de informações, relacionadas entre si, necessárias para a tarefa, que serão mostradas para o usuário durante a navegação [Vilain, 2002].

2.7.2.2.1.2. Elos

Os elos, também chamados de relacionamentos navegacionais, fazem a conexão entre os objetos navegacionais (nós e estruturas de acesso) e são, geralmente, oriundos dos relacionamentos presentes no esquema conceitual.

Um elo pode conectar duas instâncias de nós diferentes (elo binário) ou de um mesmo nó (elo unário).

No esquema navegacional, somente os relacionamentos conceituais que permitem a navegacão entre objetos são representados como elos. A representação é feita de maneira semelhante à representação dos relacionamentos unários e binários do esquema conceitual. Entretanto, é acrescentada uma seta que representa a navegação permitida a partir do elo.

Os elos unidirecionais representam que a navegação entre os objetos é permitida somente na direção da seta. Os elos bidirecionais representam que a navegação entre os objetos é permitida nas duas direções.

2.7.2.2.2. Esquema de Contextos Navegacionais

A navegação na aplicação se dá sempre entre objetos que fazem parte de um conjunto de objetos relevantes, chamado de contexto de navegação. A navegação entre os contextos é especificada através de um Esquema de Contextos. Este diagrama inclui os nós (classes navegacionais), os contextos nos quais os nós aparecem, as estruturas de acesso aos nós e a navegação permitida entre os contextos e as estruturas de acesso. A diferença entre o Esquema de Contextos e o Esquema de Classes Navegacionais é que o primeiro está relacionado aos conjuntos de objetos relacionados com uma mesma tarefa e o segundo está relacionado às instâncias dos nós.

2.7.2.2.2.1. Contexto Navegacional

Um contexto de navegação, ou somente contexto, é um conjunto dos nós que estão disponíveis ao usuário durante a navegação [Schwabe & Rossi, 1998], e são definidos baseados nas tarefas dos usuários [Vilain, 2002].

Os elementos de um contexto apresentam, em geral, algumas características em comum dentro da mesma tarefa, isto é, satisfazem a um predicado, que pode ser: pertencer à mesma classe, estar relacionado através

de uma mesma associação, possuir o mesmo valor para um determinado atributo, etc. [Güell et al., 2000]. Um contexto também pode ser formado por um conjunto arbitrário (uma enumeração explícita) de elementos.

Os objetos de um contexto de navegação (as instâncias) podem fazer parte de vários contextos. A representação do contexto no diagrama de contextos é feita através de um retângulo colocado dentro da caixa cinza que representa a classe navegacional. O nome do contexto é definido de acordo com os elementos apresentados no contexto. Se o contexto é acessado a partir de um índice, o seu nome também deverá estar de acordo com esse índice.

Para cada contexto de navegação é necessário especificar um cartão com as suas propriedades, que não estão descritas no diagrama. Contextos navegacionais podem ser especificados também como grupos de contextos, já que, em algumas situações, é possível parametrizar a propriedade que os define. Por exemplo, um contexto de `CD por Gênero` é na verdade um conjunto de conjuntos; sendo cada conjunto um contexto, determinado por um valor do atributo `Gênero`. Uma definição análoga pode ser obtida para contextos cuja propriedade seja baseada em relações 1-para-N, como `CD por Artista` [Lima & Schwabe, 2002b].

2.7.2.2.2.2. Estrutura de Acesso

O Esquema de Contextos também apresenta estruturas de acesso (índices) aos contextos, que são representadas através de retângulos tracejados [Vilain, 2002]. Assim como os contextos de navegação, as estruturas de acesso também têm seus detalhes especificadas em cartões.

2.7.2.2.2.3. Classes em Contextos

Os nós navegacionais podem apresentar diferentes informações de acordo com os contextos de navegação nos quais aparecem. Esta abstração é representada através de classes em contexto (que podem ser vistas como decoradores [Gamma et al., 1995]) e só são necessárias quando o nó tem uma aparência diferente ou âncoras distintas em algum contexto especial.

2.7.2.2.2.4. Cartão de Especificação de Contexto

A Figura 5 apresenta um cartão de especificação de contexto ou grupo de contexto. Detalhes a respeito do significado de cada campo deste exemplo de cartão podem ser obtidos no manual da notação do Método OOHDM em [Vilain & Schwabe, 2002].

Contexto: Professor por Área
Parâmetros: a: Área
Elementos: p: Professor where p *associado_a* a
Classe em Contexto:
Ordenação: por p.nome, ascendente
Navegação Interna: por índice (Idx Professores por Área(a))
Operações:
Usuário: alunos **Permissão**: leitura
Comentários:

Figura 5 - Cartão de contexto ou grupo de contexto

2.7.2.2.2.5. Cartão de Especificação de Estruturas de Acesso

Assim como os contextos de navegação, as estruturas de acesso também são detalhadas através da especificação de cartões, como os da Figura 6 e Figura 7, cujos detalhes podem ser obtidos no manual da notação do Método OOHDM em [Vilain & Schwabe, 2002].

Estrutura de Acesso: Professores	
Parâmetros:	
Elementos: p: Professor	
Atributos: p.nome p.grau	**Destino:** Ctx Professor Alfabético (self) Ctx Professor por Grau (self.grau)
Ordenação: por p.nome, ascendente	
Restrições de Uso **Usuário:** alunos	**Permissão**: leitura
Comentários:	
Depende de: Nav_Professor	**Depende de:** Nav_Professor

Figura 6 - Cartão de Estrutura de Acesso

A Figura 7 apresenta o cartão de uma estrutura de acesso hierárquica.

<table>
<tr><td colspan="4">Estrutura de Acesso: Professores : Áreas</td></tr>
<tr><td>Nível</td><td colspan="3"></td></tr>
<tr><td rowspan="3">Professores</td><td colspan="3">Elementos: p: Professor</td></tr>
<tr><td>Atributos
p.nome.....</td><td colspan="2">Destino
Idx Áreas de Pesquisa por Professor (p)</td></tr>
<tr><td colspan="3">Ordenação: por p.nome, Ascendente</td></tr>
<tr><td rowspan="4">Áreas de Pesquisa por Professor</td><td colspan="3">Parâmetros: p: Professor</td></tr>
<tr><td colspan="3">Elementos: a: Área where a pertence_a p</td></tr>
<tr><td>Atributos
a.nome...</td><td colspan="2">Destino
Ctx Áreas de Pesquisa por Professor (p)</td></tr>
<tr><td colspan="3">Ordenação: por a.nome, Ascendente</td></tr>
<tr><td colspan="2">Restrições de Uso
Usuário: alunos</td><td colspan="2">Permissão: leitura</td></tr>
<tr><td colspan="4">Comentários:</td></tr>
<tr><td colspan="2">Depende de:</td><td colspan="2">Influencia em::</td></tr>
</table>

Figura 7 - Cartão de Estrutura de Acesso hierárquica