2
Conceitos

Neste capítulo iremos abordar, de forma introdutória, os principais conceitos teóricos utilizados nesta dissertação e na construção do HyperDE.

2.1. Web Semântica e Ontologias

> Na declaração de atividades do comitê da W3C para Web Semântica temos: “A Web Semântica é uma visão: a idéia de ter dados na Web descritos e conectados de maneira que estes possam ser utilizados por máquinas, não apenas com intuito de exibição, mas também para automação, integração e reuso desses dados através de diversas aplicações”.

A Web, como a conhecemos atualmente, constitui um magnífico repositório de dados, informações e conhecimento. No entanto, as máquinas, nesse contexto, basicamente possuem as funções de armazenamento e recuperação de dados, dependendo totalmente da cognição humana para processamento e transformação desses dados em informações e conhecimento útil e relevante. A Web Semântica tem como objetivo principal descrever semanticamente os dados na Web, de forma que agentes de software [Nwana, 1996] possam, além de armazenar, recuperar e exibir tais dados, processar, aprender, inferir e tomar decisões baseadas na semântica das informações. Quando a Web Semântica for uma realidade, poderemos pedir para um agente de software algo como: “entre nas 10 melhores lojas de livros na Internet e procure os livros da editora XYZ, com preço entre X e Y, do assunto ABC, recomende-me os 5 melhores baseados nas críticas de outros compradores”. Esse agente de software será capaz de computar semanticamente, e não apenas como cadeias de caracteres, conceitos como “loja”, “livro”, “editora”, “preço”, “assunto”, “crítica”, “comprador”.

Para que esses agentes de software entendam esses conceitos, eles se utilizarão de ontologias. Ontologias são especificações formais de um determinado domínio de conhecimento que incluem a descrição de termos e objetos que formam esse domínio, assim como a descrição de como eles se

relacionam uns com os outros. Ontologias fornecem um vocabulário para representar e comunicar sobre determinada área de conhecimento, e promovem o reuso e compartilhamento desse conhecimento.

Através de uma ontologia é possível definir o que é um “livro”, que este possui um “criador”, também conhecido como “autor” que nesse contexto é equivalente a “escritor”, que este livro é publicado por uma “editora”, etc. Na Web Semântica, essas definições serão representadas através de metadados que poderão estar anexados a uma imagem da capa de um livro, por exemplo, e que poderão ser utilizados por máquinas e pessoas.

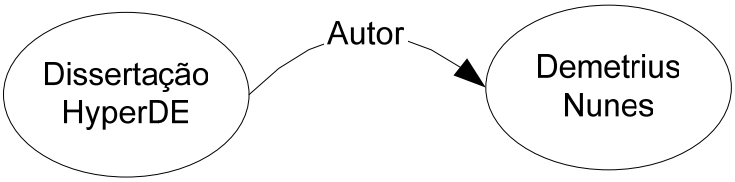
2.2. RDF, RDF(s) e OWL

Para que a Web Semântica possa existir, é necessário que haja uma maneira universal de descrever, através de metadados, qualquer recurso existente na WWW. Para isso, o W3C (World Wide Web Consortium - http://www.w3c.org) vem, desde 1998, promovendo um padrão básico para formato de dados na Web chamado RDF (Resource Description Framework).

RDF utiliza uma estrutura básica bastante simples e uniforme, chamada de “tripla”, que é sempre composta de sujeito, predicado e objeto. Cada tripla é chamada de declaração. Usamos declarações em formato RDF para descrever as coisas que formam determinado universo e como essas coisas se relacionam entre si.

Por exemplo, a declaração “Esta dissertação foi escrita por Demetrius Nunes” pode ser facilmente representada através de uma declaração RDF. No caso, “dissertação” é o sujeito sendo descrito, “foi escrita” é o predicado e “Demetrius Nunes” é o objeto da declaração. E assim poderia se continuar descrevendo muitas outras propriedades do mesmo sujeito, assim como poderia se descrever melhor as próprias propriedades desse sujeito usando a mesma sintaxe.

A melhor forma de representar declarações RDF é, visualmente, através de grafos orientados. No entanto, para que declarações RDF possam ser processadas e interpretadas por máquinas, elas devem ser representadas utilizando um formato textual interoperável como XML. Além disso, para que seja útil na Web, todo sujeito descrito em declarações RDF deve ser representado através de uma URI (*Uniform Resource Identifier*) [Berners-Lee, Fielding & Masinter, 1998].




```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dn="http://inf.puc-rio.br/~demetrius/rdf-demo-ns#">
    <rdf:Description rdf:about="http://inf.puc-rio.br/~demetrius/TeseHyperDE.doc">
        <dn:Autor>Demetrius Nunes</dn:Autor>
    </rdf:Description>
</rdf:RDF>
```

Figura 2 - Exemplo de declaração RDF na forma de grafo orientado e utilizando XML

RDF em formato textual serializável é o primeiro nível de infra-estrutura necessário para a construção de vocabulários de metadados para a Web Semântica. No entanto, RDF é apenas um formato de dados que apresenta uma maneira genérica para representação de vocabulários, mas não indica como esses vocabulários devem ser descritos.

RDFS (RDF *Schema*) é uma linguagem para descrição de vocabulários em RDF. RDFS define primitivas como classes e propriedades que podem ser utilizadas para descrever classes, propriedades e outros recursos. Sendo assim, RDFS é uma extensão semântica do formato RDF e provê mecanismos para descrever grupos de recursos relacionados e as relações entre esses recursos. Como RDFS é descrita utilizando o próprio formato RDF, essa linguagem também possui representação em XML (entre outros formatos) e pode ser processada e interpretada por máquinas.

RDFS pode ser encarada como uma linguagem genérica para definição de sistema de tipos, comumente vista em linguagens orientadas a objeto. Mas para que possamos descrever domínios de conhecimento específicos (ontologias) necessitamos de um poder maior de expressividade. Para isso temos a linguagem OWL (*Web Ontology Language*).

OWL é uma extensão semântica do formato RDF e da linguagem RDFS. OWL fornece primitivas mais ricas para descrição de classes, propriedades e relacionamentos entre grupos de recursos. Com OWL é possível definir semanticamente termos de determinado vocabulário e seus inter-relacionamentos. Além de classes e propriedades, OWL permite descrever conceitos lógicos tais

como cardinalidade, equivalência, disjunção, simetria, entre outros. OWL também pode ser descrito através do formato RDF e, portanto, possui uma representação e sintaxe em linguagem XML que pode ser processada e interpretada por máquinas.

2.3. Arquitetura Orientada a Modelos

A arquitetura orientada a modelos ou arquitetura dirigida por modelos (MDA - *Model Driven Architecture*), é um estilo arquitetônico que recentemente tem recebido bastante atenção de instituições e organizações interessadas em novas formas de projetar e implementar aplicações. MDA encoraja o uso intenso de modelos como meio para gerar de forma automatizada a implementação de um software.

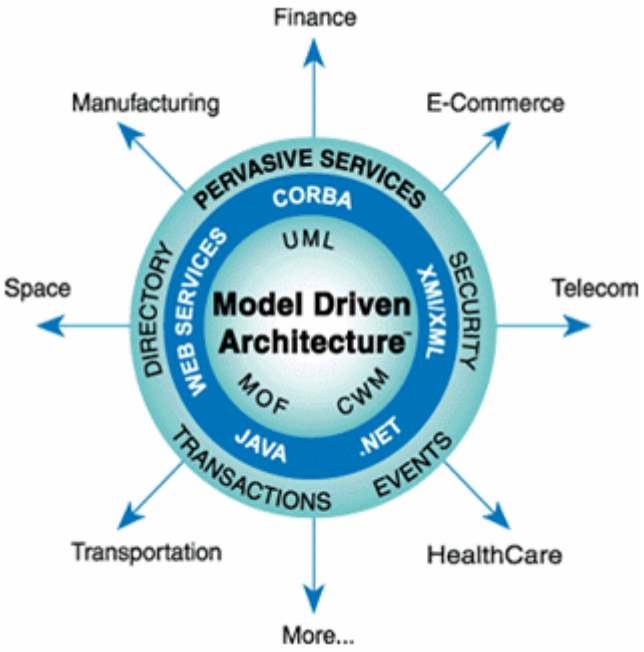


Figura 3 - Arquitetura dirigida por modelos (MDA), segundo a visão do OMG

Em 2003, o OMG (*Object Management Group*) publicou uma especificação oficial da MDA [OMG, 2003], definindo padrões, capacidades, ferramentas e plataformas. Os modelos de aplicação, como definidos pela MDA, devem ser independentes de plataformas e capazes de serem transformados através de ferramentas autônomas gerando uma aplicação implementada. Um dos objetivos da MDA é minimizar ou eliminar a necessidade de codificação manual em linguagem de programação na etapa de implementação, viabilizando o conceito de

fábricas de software. A MDA está sendo difundida pelo OMG e por empresas e instituições ligadas ao uso de UML e ferramentas CASE, como Rational, Sun, IBM, Microsoft, entre outras.

2.4. Linguagens de Programação Dinâmicas e Interpretadas

Em uma linguagem de programação estática, como C++, Java e C#, os tipos dos dados do programa são pré-determinados e necessitam, portanto, de uma fase de compilação antes da execução do programa onde é verificada, entre outros, a existência de erros de incompatibilidade entre os tipos nas operações de atribuição de valores. Linguagens estáticas e compiladas possuem a qualidade de reduzir ou eliminar estes erros de incompatibilidade de tipos, um tipo de erro bastante freqüente em programação de computadores, e são percebidas portanto como linguagens mais robustas e seguras. Ainda, um dos objetivos principais da fase de compilação, além da verificação de erros, é produzir um código de baixo nível, otimizado para execução sobre uma plataforma de hardware específica. No entanto, a fase de compilação e a pré-determinação dos tipos dos dados impõem um custo adicional em termos de esforço de implementação e rigidez, dificultando, por exemplo, técnicas mais avançadas como a metaprogramação e exigindo a produção de mais código-fonte.

Com o aumento da potência e velocidade dos processadores e a crescente popularidade de práticas de resguardo como testes de unidade (“*unit testing*”), que entre outros benefícios, ajudam a reduzir erros de incompatibilidade de tipos, a justificativa por uma fase de compilação vem se mostrando cada vez menos necessária.

Linguagens de programação dinâmicas, tais como Lua (http://www.lua.org), Python (http://www.python.org), Smalltalk (http://www.smalltalk.org) e Ruby (http://www.ruby-lang.org), entre muitas outras, são linguagens em que o sistema de tipos dos dados, representados através de variáveis e referências a objetos, são determinados apenas em tempo de execução, sendo desnecessária uma fase de compilação do programa. Dessa forma, estas linguagens podem ser executadas de forma interpretada, independente da plataforma de hardware e por isso, em muitos casos, apresentam maior flexibilidade e maior expressividade, gerando uma

redução do esforço de implementação, tendo como resultado programas mais eficientes, concisos e portáveis.

2.5. Linguagens Específicas de Domínio e Ontologias

Linguagens específicas de domínio (DSLs - Domain Specific Languages) são linguagens de programação que possuem primitivas e construções criadas exclusivamente para um domínio de problema específico. Com isto, é possível a construção de programas mais simples e concisos dentro do contexto alvo em que eles devem ser executados.

Para ilustrar este aspecto de forma mais didática, vejamos um exemplo de código-fonte em linguagem de propósito geral Java, para a manipulação de dados em uma ontologia referente a informações pessoais, através do framework Jena.

```
DAMLModel model = … // code that loads the VCARD ontology
                    // and some data based on that ontology

DAMLClass vcardClass =
    (DAMLClass) model.getDAMLValue(vcardBaseURI+"#VCARD");

DAMLProperty fnProp =
    (DAMLProperty) model.getDAMLValue(vcardBaseURI+"#FN");

DAMLProperty emailProp =
    (DAMLProperty) model.getDAMLValue(vcardBaseURI+"#EMAIL");

Iterator i = vcardClass.getInstances();
while (i.hasNext()) {
    DAMLInstance vcard = (DAMLInstance) i.next();

    Iterator i2 =
        vcard.accessProperty(emailProp).getAll(true);
    while (i2.hasNext()) {
        DAMLInstance email = (DAMLInstance) i2.next();

        if (email.getProperty(RDF.value).getString().equals(
            "amanda_cartwright@example.org" ) ) {
            DAMLDataInstance fullname =
                (DAMLDataInstance) vcard.accessProperty(fnProp).getDAMLValue();

            if ( fullname != null )
                System.out.println("Name: "+ fullname.getValue().getString());
        }
    }
}
```

O objetivo do programa apresentado é simplesmente iterar sobre uma coleção de cartões de visita, representados pela classe vCard, e imprimir o nome da pessoa do cartão caso um de seus emails seja de determinado valor. No entanto, para esta simples operação é necessário o conhecimento da linguagem DAML e das primitivas do framework Jena para acesso aos objetos e propriedades do modelo específico da aplicação.

Idealmente, se tivéssemos a nossa disposição uma linguagem específica de domínio para este modelo da aplicação, poderíamos re-escrever o programa da seguinte forma, utilizando ainda a sintaxe da linguagem Java:

```
VCardList vcList = Vcards.getAll();
while (vcList.next()) {
    vCard vc = vcList.getCurrent();
    emailList emails = vc.getEmails();
    while (emails.next()) {
        if (emails.getCurrent() == “amanda_cartwright@example.org”) {
            if (vc.getFullName() != null) {
                System.out.println(vc.getFullName());
            }

        }

    }
}
```

Como se pode ver, uma versão bem mais concisa e simples de entender. A manipulação de ontologias através de linguagens específicas de domínio, como discutido em [Goldman, 2003], é uma forma de permitir um modelo de programação mais natural e eficiente como alternativa a modelos de programação genéricos.

2.6. Padrão de Projeto e Frameworks MVC

O padrão de projeto MVC (*model-view-controller*) é um dos padrões mais tradicionais e difundidos em engenharia de software para a construção de interfaces (em sua maioria gráficas) com o usuário. Foi idealizado por Trygve Reenskaug, no final dos anos 70, para apoiar o desenvolvimento da plataforma Smalltalk-80.

Este padrão de projeto promove a estrita separação de responsabilidades entre componentes de uma interface gráfica onde temos componentes responsáveis pela manutenção do estado da aplicação, denominado de “modelo”, pela exibição de parte deste modelo para o usuário, ao que chamamos de “visão” e pela coordenação entre atualizações no modelo e interações com o usuário, feita através do “controlador”.

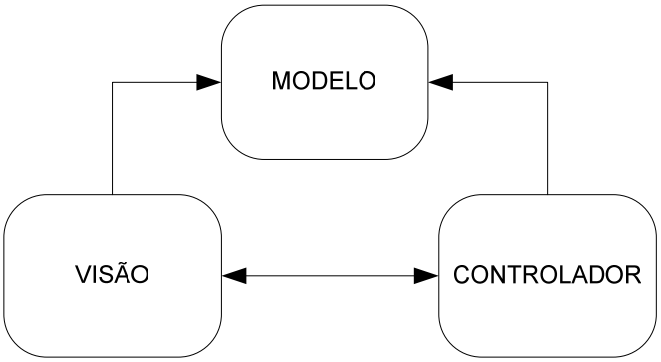


Figura 4 - Componentes do padrão de projeto MVC

O componente de modelo não possui acoplamento com os outros componentes do padrão e encapsula o estado da aplicação, fornecendo uma interface para manipulação e consulta deste estado. O componente de visão acessa o modelo para recuperação do estado da aplicação e exibe estes dados para o usuário. O componente controlador recebe as interações do usuário com a interface, podendo atualizar o estado da aplicação através do componente de modelo e/ou selecionar a visão adequada a ser exibida para o usuário.

Uma arquitetura comumente adotada para o desenvolvimento de aplicações para Web separa a implementação do padrão MVC em 3 camadas correspondentes a cada tipo de componente. Esta arquitetura serviu de base para o surgimento de *frameworks* em diferentes plataformas que fornecem uma infraestrutura de componentes de modelo, visão e controle, reutilizáveis e extensíveis. Exemplos de *frameworks* MVC incluem Struts (http://struts.apache.org/) para plataforma Java, ASP.NET (http://www.asp.net) para plataforma .NET e Rails (http://www.rubyonrails.com) para plataforma Ruby.

Apesar de algumas discussões e críticas interessantes existirem em torno do padrão MVC como visto em [John et al, 2004] e de alternativas existirem, como o padrão PAC (Presentation Abstraction Control) [Buschmann et al, 1996], o padrão MVC é atualmente o padrão de projeto mais popular e mais utilizado na construção de sistemas interativos, seja para plataforma Web ou outras.

2.7. Web Services, Protocolo SOAP e Arquitetura REST

Web Services são componentes reutilizáveis de software que se fazem disponíveis através de interfaces de aplicação na Web. Para isso eles utilizam uma infra-estrutura ubíqua já existente baseada em protocolos de rede onipresentes como HTTP (*Hypertext Transfer Protocol*) e SMTP (*Simple Mail Transfer Protocol*). A descrição das interfaces de aplicação de Web Services, assim como o formato de suas mensagens estão baseadas em linguagem XML. Assim, a adoção e uso de Web Services pode ser feita sem nenhum custo adicional de infraestruturas.

Web Services podem ser considerados, portanto, uma peça fundamental de uma arquitetura para computação distribuída verdadeiramente interoperável, onde o software funciona como um provedor de serviços, independente de plataforma de implementação. O objetivo maior de tal arquitetura é promover a integração entre aplicações, desenvolvidas e implantadas nas mais diversas plataformas de hardware e software, utilizando uma especificação de protocolos universalmente aceita pela indústria, possibilitando a cooperação entre máquinas e pessoas, a qualquer tempo e lugar. A infra-estrutura clássica para Web Services está baseada em três peças fundamentais:

- WSDL (*Web Services Description Language*): WSDL é a linguagem de metadados baseada em XML para descrição dos Web Services, uma espécie de “manual do usuário” para Web Services, definindo como os provedores de serviços e aplicações clientes devem se comunicar. WSDL basicamente descreve os formatos dos dados a serem trocados, quais operações estão disponíveis para execução e quais parâmetros e tipos de retorno serão utilizados nessas operações.
- SOAP (*Simple Object Access Protocol*): SOAP é um protocolo baseado em linguagem XML para troca de mensagens entre aplicações distribuídas. Ele define um mecanismo para a passagem de comandos e parâmetros entre clientes e Web Services, independente de plataforma de implementação ou linguagem de programação.

- UDDI (*Universal Description, Discovery, and Integration*): UDDI pode ser visto como as “páginas amarelas” para Web Services. Um local onde Web Services são catalogados e disponibilizados para busca e utilização. Um registro UDDI de um Web Service é também baseado em XML e contém a descrição do fornecedor e dos serviços sendo oferecidos, a categorização desses serviços, bem como especificações técnicas dos mesmos.

Mais recentemente, esta infraestrutura, fortemente apoiada pela indústria através de grandes empresas tais quais Sun, IBM, Microsoft, Oracle entre muitas outras, e também pelo W3C, vem sendo contestada pela crescente complexidade de linguagens, protocolos, padrões e camadas sendo adicionados a ela, a tal ponto que o acrônimo SOAP já deixou de ser relacionado ao título de “Simple Object Access Protocol” (Protocolo Simples de Acesso a Objetos) e atualmente é utilizado de forma independente. Outros motivos apontam também para inconformidades nas decisões de projeto e arquitetura tomadas para a confecção desses padrões e protocolos, em que foi utilizada uma visão incompatível, orientada a processos, para construir essa infraestrutura de comunicação na Web, sendo que a Web intrinsecamente utiliza uma arquitetura orientada a recursos. Uma discussão sobre estes pontos pode ser vista em [REST-WIKI].

Por conta desses problemas, em 2000 Roy Fielding [Fielding, 2000, 2002] idealizou uma arquitetura alternativa, orientada a recursos, para a implementação de Web Services, utilizando um protocolo de comunicação mais simples, denominado REST (Representational State Transfer). Esta arquitetura promove os conceitos de que todo recurso disponível na Web deve possuir uma representação descrita através de uma URI (Uniform Resource Identifier), podendo ser acessado diretamente através das 4 operações básicas implementadas no protocolo HTTP: GET, POST, PUT e DELETE.

Estes conceitos diferem em relação aos promovidos pelo protocolo SOAP, que determina o uso de pontos de acesso, onde os recursos são acessados de forma indireta através de operações declaradas em mensagens no formato XML. Este acesso indireto a recursos, imposto pelo protocolo SOAP, muitas vezes dificulta ou até mesmo impossibilita a implementação de alguns tipos de serviços. Uma analogia interessante para entender este tipo de restrição: imaginando que uma

pessoa está hospedada em um hotel e deseja contratar um serviço de despertador automático por telefone. Este serviço funciona de forma totalmente autônoma, registrando o número que deve receber a ligação (um identificador de recurso) em horário estabelecido pelo usuário. No entanto, o hotel em que esta pessoa está hospedada não oferece um número para acesso direto aos quartos, permitindo apenas ligações transferidas por um operador central através de um protocolo manual. Neste caso, podemos ver que o operador é um ponto de acesso, que deve receber uma mensagem específica para poder fornecer acesso a um recurso específico, adicionando dessa forma um nível de indireção e inviabilizando o uso do serviço de despertador por seus hóspedes. Uma discussão mais aprofundada sobre a questão SOAP versus REST pode ser vista em [Prescod, 2002].

2.8. Arquitetura Orientada a Serviços

Uma arquitetura orientada a serviços, mais comumente referenciada pelo acrônimo SOA (Service Oriented Architecture), consiste em um projeto de software cujo objetivo maior é obter interoperabilidade entre componentes de software utilizando acoplamentos fracos (loose-coupling). SOA não é um conceito novo, no entanto, com o surgimento de Web Services, essa arquitetura encontrou seu mecanismo ideal de implementação. Desde então o interesse pela adoção de SOA foi grandemente renovado, pois os maiores obstáculos para implementação desse estilo de arquitetura residiam justamente nas tecnologias e infraestruturas disponíveis para tal.

Resumidamente, em um arquitetura SOA, uma camada de serviços agrega componentes e funcionalidades relacionadas a um ou mais processos. Essa camada é publicada em rede e pode ser invocada de forma remota por aplicações clientes ou outros serviços de software. A composição dos serviços oferecidos por essa camada caracteriza uma aplicação SOA. Os principais benefícios de uma arquitetura orientada a serviços implementada através de Web Services são:

- Interoperabilidade: as aplicações clientes de uma aplicação SOA podem estar implementadas em qualquer plataforma de software e hardware, distinta da plataforma onde a camada de serviços foi implementada;

- Reusabilidade: os serviços e funcionalidades oferecidos por uma aplicação SOA são altamente reutilizáveis a baixo esforço, pois suas interfaces discretas podem ser invocadas de forma independente;
- Mobilidade: uma aplicação SOA é acessada remotamente e sua localização deve ser descoberta de forma dinâmica e transparente por aplicações clientes, dessa forma é possível realocar uma aplicação SOA sem comprometer a disponibilidade da aplicação;
- Integrabilidade: as funcionalidades oferecidas em uma aplicação SOA podem ser utilizadas de forma integral ou parcial, permitindo que outras aplicações integrem os serviços desejados de forma granular;