RestMDD: Ferramenta colaborativa para o apoio no desenvolvimento de serviços Web RESTFul

Robson Vinícius Vieira Sanchez

RestMDD: Ferramenta colaborativa para o apoio no desenvolvimento de serviços Web RESTFul

Robson Vinícius Vieira Sanchez

Orientador: ***Profa. Dra. Renata Pontin de Mattos Fortes***

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, para o Exame de Qualificação, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Março/2011

Resumo

Na última década o paradigma de computação orientada a serviços (SOC - *Service Oriented Computing*) tem ganhado cada vez mais espaço na indústria e na academia, a fim de solucionar o problema da falta de comunicação entre os diversos sistemas de informação presentes dentro de um ambiente corporativo. Graças aos recentes avanços da chamada "Web 2.0" um novo estilo arquitetural chamado de Arquitetura orientada a Web (WOA - *Web Oriented Computing*) foi proposto a fim de garantir uma forma simples de conectar os componentes de TI de forma dinâmica. Esse estilo tem como um dos princípios o uso de serviços Web RESTful, a fim de conseguir uma interface funcional simples e uniforme. Este trabalho apresenta uma ferramenta de apoio ao desenvolvimento de serviços Web RESTFul dentro de um ambiente colaborativo utilizando o paradigma de desenvolvimento orientado a modelos (MDD - *Model Driven Development*). Pretende-se mostrar os benefícios do MDD aplicado a Engenharia Web e também as vantagens apresentadas pela colaboração nesse cenário. Além disso, essa ferramenta deve ser capaz de modelar e executar processos de negócio por meio da integração desses serviços. Será realizado ainda um estudo de caso a fim de comprovar a eficiência da ferramenta e benefícios alcançados por ela.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

API *Application Programming Interface* / Interface de Programação de Aplicações

ATL *Atlas Transformation Language* / Linguagem de Transformação Atlas

BPR *Business Process Reengineering* / Reengenharia de Processos de Negócio

CASE *Computer-Aided Software Engineering* / Engenharia de Software assistada por computador

CDE *Collaborative Development Environment* / Ambiente de Desenvolvimento Colaborativo

CIM *Computation Independent Model* / Modelo Independente de Computação

CSCW *Computer Supported Cooperative Work* / Trabalho Cooperativo suportado por computador

CVS *Concurrent Version System* / Sistema de Versões Concorrentes

DSL *Domain Specific Language* / Linguagem Específica de Domínio

DTD *Document Type Definition* / Definição de Tipo de Documento

FAQ *Frequently Asked Questions* / Perguntas Frequentes

HTML *Hypertext Markup Language* / Linguagem de Marcação de Hipertexto

HTTP *Hypertext Transfer Protocol* / Protocolo de Transferência de Hipertexto

JSON *JavaScript Object Notation* / Notação de objetos JavaScript

MDA *Model Driven Architeture* / Arquitetura Orientada a Modelos

MDD *Model Driven Development* / Desenvolvimento Orientado a Modelos

MIME *Multipurpose Internet Mail Extensions* / Extensões Multipropósito para Mensagens de Internet

MOF *Meta-Object Facility* / Infra-estrutura de MetaObjetos

OMG *Object Management Group* / Grupo de Gerenciamento de Objetos

PIM *Plataform Independent Model* / Modelo Independente de Plataforma

PSM *Plataform Specific Model* / Modelo Específico de Plataforma

QVT *Queries/Views/Transformations* / Consultas/Visões/Transformações

REST *Representational State Transfer* / Transferência de Estado Representacional

RPC *Remote Procedure Call* / Chamada remota de procedimento

SaaS *Software as a Service* / Software como um Serviço

SCM *Software Configuration Management* / Gerenciamento de Configuração de Software

SOAP *Simple Object Access Protocol* / Protocolo Simples de Acesso a Objetos

SOA *Service Oriented Architecture* / Arquitetura Orientada a Serviços

SOC *Service Oriented Computing* / Computação Orientada a Serviços

UDDI *Universal Description Discovery and Integration* / Descrição Descoberta e Integração Universal

UML *Unified Modeling Language* / Linguagem de Modelagem Unificada

URI *Uniform Resource Identifier* / Identificador Uniforme de Recursos

UWE *UML-based Web Engineering* / Engenharia Web baseada em UML

WebML *Web Modeling Language* / Linguagem de Modelagem Web

WOA *Web Oriented Architecture* / Arquitetura Orientada a Web

WS-BPEL *Web Services Business Process Execution Language* / Linguagem de Execução de Processos de Negócio

WSCI *Web Service Choreography Interface* / Interface para Coreografia de Serviços Web

WSDL *Web Services Description Language* / Linguagem para descrição de Serviços Web

XMI *XML Metadata Interchange* / Intercâmbio de Metadados em XML

XML *eXtensible Markup Language* / Linguagem de Marcação Extensível

XSLT *eXtensible Stylesheet Language for Transformation* / Linguagem extensível para folhas de estilo de transformações

Capítulo

1

Introdução

Atualmente a maioria das empresas preferem comprar um software ao invés de criá-lo (Aalders, 2001; Cullen e Willcocks, 2003), pois dessa forma é possível escolherem os sistemas mais adequados para atender às suas necessidades únicas (Hohpe e Woolf, 2003; Linthicum, 1999). Por essa razão, o modelo conceitual de negócio é separado em fragmentos que são realizados com a ajuda de diferentes sistemas de informação individuais, o que resulta em um ambiente com múltiplas aplicações de diferentes fornecedores, tendo interfaces e formatos de dados incompatíveis e com isso dificultando a comunicação entre si (Daniel et al., 2010).

Na última década, foi possível testemunhar uma mudança significativa na computação distribuída migrando para um paradigma de computação orientada a serviços (SOC - *Service Oriented Computing*), que está ganhando destaque como uma abordagem eficiente para integrar aplicações em ambientes distribuídos heterogêneos (Mayerl et al., 2005), a fim de solucionar o problema da falta de comunicação entre os diversos sistemas de informação presentes nos ambientes corporativos.

A mais popular área de pesquisa dentro de SOC é dedicada aos avanços da Arquitetura Orientada a Serviços (SOA - *Service Oriented Architeture*) que tem sido discutida intensamente na indústria, ciência e literatura desde sua primeira aparição em meados dos anos 90. As propriedades que caracterizam um conjunto de componentes como uma SOA incluem serviços que englobam as funcionalidades de vários sistemas terceiros de forma atômica e sem estado, além das interfaces bem definidas para comunicação e composição

desses serviços. Graças a essas propriedades, os principais objetivos a serem alcançados com SOA incluem reutilização de código, redução de complexidade e custos, e alta flexibilidade (Thies e Vossen, 2008).

Os diversos padrões existentes para facilitar a criação desse tipo de arquitetura (por exemplo, Serviços Web, UDDI, SOAP) tornam sua construção mais complicada devido ao "excesso de padronização" (Thies e Vossen, 2009). Por outro lado, os recentes avanços na chamada "Web 2.0" nos mostram uma maneira mais fácil de conectar os componentes de TI de forma dinâmica, de modo que os objetivos originais do SOA como flexibilidade, reutilização e a redução da complexidade possam ser atingidos de uma maneira mais simples (Thies e Vossen, 2008). A essa forma alternativa de construção de ambientes orientados a serviços foi denominado por Nicholas Gall (Gall, 2008) o termo Arquitetura Orientada a Web (WOA - *Web Oriented Architecture*).

A arquitetura orientada a Web é considerada um sub-estilo arquitetural baseado em SOA com foco no uso de tecnologias e padrões da Web 2.0 (Thies e Vossen, 2009). Sua principal vantagem é a simplicidade pois não faz uso de novos padrões, mas sim emprega os já existentes na Web, tais como HTTP, XML e JSON como ilustrado na Figura 1.1. O objetivo mais notável de WOA é permitir o desenvolvimento de sistemas descentralizados ligados completamente e que tenham a mesma flexibilidade da Web, e além disso ofereçam um processamento do estado da aplicação via mensagens auto-descritivas (Daniel et al., 2010). Portanto, para alcançar estes objetivos, WOA faz o uso de serviços Web RESTful, a fim de conseguir uma interface funcional simples e uniforme.

Figura 1.1: Web Oriented Architeture (Thies e Vossen, 2008)

O REST é o acrônimo de *Representational State Transfer* e não é um padrão, mas sim um estilo arquitetural para construção de sistemas hipermídia distribuídos que foi definido por Roy Thomas Fielding (Fielding, 2000), um dos fundadores da Apache Software Foundation e um dos autores da especificação do protocolo HTTP. Esse estilo é comumente usado com o padrão CRUD (Kimball e Caserta, 2004) que define o conjunto básico de 4

operações: Criar (Create), Recuperar (Retrieve), Atualizar (Update) e Excluir (Delete). Essas operações são mapeadas para os métodos do protocolo HTTP e podem ser aplicadas a qualquer recurso, permitindo aos consumidores manipularem todos os recursos de forma consistente (Daniel et al., 2010). Dessa forma, cada uma dessas operações pode ser considerada um Serviço Web RESTFul.

Existem 4 princípios que definem o estilo REST, que serão abordados na perspectiva dos serviços Web RESTFul (Pautasso et al., 2008):

- **Identificação de recursos através da URI:** um serviço Web RESTful expõe um conjunto de recursos que identificam os alvos da interação com seus clientes. Os recursos são identificados por URIs, que proporcionam um espaço de endereçamento global para os recursos e a descoberta de serviços.

- **Interface Uniforme:** os recursos são manipulados usando o padrão CRUD, no qual as 4 operações básicas são mapeadas para 4 métodos do protocolo HTTP. O método PUT cria um novo recurso, que pode ser excluído por meio do método DELETE. Para recuperar o estado atual do recurso, é utilizado o método GET. Já para modificar esse estado é utilizado o método POST.

- **Mensagens auto-descritivas:** os recursos são desacoplados de suas representações, o que permite que seu conteúdo seja acessado em uma variedade de formatos (por exemplo HTML, XML, texto simples, JSON, etc.). Os metadados sobre o recurso estão disponíveis e são utilizados para controlar o *cache*, detectar erros de transmissão, informar o formato da representação e realizar a autenticação e controle de acesso.

- **Interações com estado através de hyperlinks:** cada interação feita com um recurso é realizada sem estado, ou seja, a requisição possui toda informação necessária para aquela interação. Dessa forma, interações com estado são baseadas no conceito de transferência explicíta de estado, ou seja, na mensagem de resposta são apontados os futuros estados válidos da interação, por meio de hiperlinks.

Os serviços Web RESTFul são considerados escaláveis, pois permitem suporte a *cache*, clusterização e balanceamento de carga. Além disso, graças a possibilidade de escolher os formatos das representações dos recursos, os desenvolvedores são capazes de otimizar o desempenho dos serviços por meio da adoção de formatos leves como o JSON (Crockford, 2006) ou mesmo texto simples. Mas a principal das vantagens desse tipo de serviço é a simplicidade, pois aproveita os padrões já existentes e bem conhecidos W3C/IETF (HTTP, XML, URI, MIME) e a infra-estrutura necessária já encontra-se pervasiva, com

clientes e servidores HTTP disponíveis para as principais plataformas de hardware e software (Pautasso et al., 2008).

Ao mesmo tempo que esforços vêm sendo dedicados para simplificar a integração de sistemas de informação usando as tecnologias da Web, novas abordagens de modelagem para as aplicações Web vêm sendo propostas, tais como a WebML(Ceri et al., 2002) e UWE(Koch e Kraus, 2003), a fim de sistemizar e tornar independente de plataforma o desenvolvimento dessas aplicações (Schauerhuber et al., 2006). Para que isso seja possível essas abordagens utilizam o paradigma de desenvolvimento orientado a modelos (MDD - *Model Driven Development*), o qual tem como principal característica o uso de modelos como artefatos primários no processo de desenvolvimento de software, ou seja, o desenvolvedor tem seu foco voltado para a modelagem do domínio do problema e não uma implementação específica da solução. Dessa forma, graças a essa abstração por meio de modelos, é possível definir transformações a serem aplicadas nos modelos que permitam gerar diferentes implementações específicas de plataforma.

O MDD é uma abordagem da Engenharia de Software que tem recebido uma atenção especial nos últimos anos pois se apresenta como uma boa opção para melhorar a produtividade, interoperabilidade, portabilidade, corretude, entre outras características importantes dentro de um projeto de software. Um dos principais benefícios alcançados com o MDD é a reutilização, pois os modelos resumem o conhecimento produzido durante as atividades de análise, projeto e implementação de uma forma independente de plataforma, o que torna possível a reutilização desses modelos em outros contextos reaproveitando o conhecimento adquirido (Sherif et al., 2006).

Como a atividade de reutilização de software é bastante colaborativa, pelo fato de que o reutilizador de um artefato na maioria das vezes não ser a mesma pessoa que o criou, é importante a utilização de um ambiente de desenvolvimento colaborativo para que os desenvolvedores possam interagir e colaborar no projeto. Dessa forma, argumentamos que o MDD obtenha benefícios quando utilizado de forma colaborativa.

Objetivo

O projeto ora proposto tem por finalidade projetar a ferramenta **RestMDD - RestFul Model-Driven Development**, de apoio ao desenvolvimento de Serviços Web RESTFul utilizando o MDD dentro de um ambiente colaborativo. Além disso, essa ferramenta deve ser capaz de modelar e executar processos de negócio por meio da integração desses serviços. Espera-se que, com este projeto, seja possível levantar quais os benefícios alcançados com o desenvolvimento dessa ferramenta e que esta possa contribuir diretamente com a área de pesquisa de Engenharia Web.

Estrutura da Monografia

Este documento encontra-se organizado da seguinte forma: no Capítulo 2 é apresentado um estudo sobre a Arquitetura REST e as principais características dos Serviços Web RESTful. No Capítulo 3 é apresentada a fundamentação teórica sobre MDD e suas abordagens dentro da Engenharia Web. No Capítulo 4 são apresentados os conceitos relativos a ambientes colaborativos e os trabalhos relacionados à colaboração em MDD. No Capítulo 5 são mostrados os objetivos do projeto, a metodologia, o cronograma das atividades a serem realizadas e os resultados esperados.

Capítulo

2

Serviços Web RESTFul

O interesse pela computação orientada a serviços tem aumentado nos últimos anos devido à necessidade de integração entre as diferentes aplicações dentro de ambientes distribuídos heterogêneos, o que fez crescer os estudos relacionados a Serviços Web. Devido ao excesso de padronização proposto para os Serviços Web SOAP e graças aos recentes avanços da Web 2.0, uma nova abordagem chamada de Serviços Web RESTFul vem ganhando destaque como uma maneira de simplificar a criação de Serviços Web. Neste capítulo será apresentada a definição da Arquitetura REST, os principais conceitos da Web 2.0 e Serviços Web e finalmente a definição de Serviços Web RESTFul, quais suas vantagens e desvantagens, bem como as possíveis abordagens utilizadas para a combinação destes serviços.

2.1 Arquitetura REST

O REST é um estilo arquitetural proposto por Roy Thomas Fielding (Fielding, 2000) em sua tese de doutorado, que tem como principal objetivo a construção de sistemas hipermídia distribuídos. Segundo Roy, há duas maneiras para o *design* de uma arquitetura. Na primeira delas, o *designer* começa sem nada e constrói uma arquitetura de componentes familiares até satisfazer as necessidades do sistema. Na segunda, o *designer* começa com as necessidades do sistema como um todo, sem restrições, e depois gradualmente identifica e aplica restrições para os elementos do sistema. O estilo arquitetural REST utiliza esse

último processo, no qual o ponto de partida é chamado de Null Style que descreve um sistema onde não há limites distingüíveis entre os componentes.

A primeira restrição aplicada ao REST é o estilo arquitetural cliente-servidor, separando as questões de interface das preocupações com o armazenamento dos dados como ilustrado na Figura 2.1. O componente à esquerda da figura representa o cliente, responsável pela interface, que se comunica com o componente à direita que representa o servidor, responsável pelo processamento e armazenamento dos dados. Assim, devido a essa separação de responsabilidades, os componentes podem evoluir de maneira independente, sendo possível melhorar a portabilidade da interface do usuário através de múltiplas plataformas e aumentar a escabilidade, simplificando os componentes do servidor.

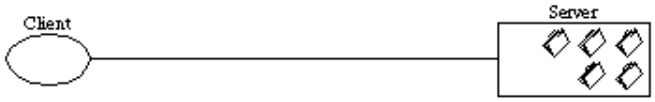


Figura 2.1: Restrição Arquitetura Cliente-Servidor (Fielding, 2000)

A próxima restrição a ser incluída se refere à comunicação entre cliente e servidor, que deve ser realizada sem estado, de tal forma que a requisição do cliente deve possuir toda a informação necessária para que o servidor possa atender o pedido, sem precisar consultar nenhum dado guardado no servidor. Assim, o estado é mantido todo no cliente. Entre as vantagens dessa restrição estão:

1. **Visibilidade:** a visibilidade é melhorada pois precisamos olhar apenas uma requisição para entender qual a natureza da funcionalidade;

2. **Confiabilidade:** facilita a recuperação de falhas parciais;

3. **Escalabilidade:** melhora a escabilidade pois não necessita que o servidor armazene dados entre as requisições, deixando os recursos livres.

Entre as desvantages podemos citar:

a) Diminui o desempenho da rede pois há um aumento de dados repetidos enviados, uma vez que não guardamos o estado no servidor;

b) Diminui o controle do servidor, pois dependemos da implementação correta dos clientes.

Para suprir uma das desvantagens apresentadas pela comunicação sem estado, a fim de melhorar a eficiência da rede, é adicionada mais uma restrição ao estilo REST, a restrição de *cache*. Essa restrição requer que os dados de resposta do servidor sejam

implicitamente ou explicitamente rotulados como *cache* ou não-*cache*. Se a resposta for colocada em *cache*, o cliente poderá reutilizar esses dados, sem a necessidade de uma nova requisição, evitando a utilização da rede. Uma desvantagem observada nesse caso é que há uma diminuição na confiabilidade, pois os dados podem estar desatualizados no *cache*.

Outra restrição que diferencia o estilo REST dos demais estilos para sistemas em rede é a sua ênfase em uma interface uniforme entre os componentes. Dessa forma, a arquitetura geral do sistema é simplificada e a visibilidade das interações é melhorada.

A fim de obter uma interface uniforme, várias restrições arquiteturais são necessárias para orientar o comportamento dos componentes. Para o estilo REST são definidas quatro restrições de interface: identificação de recursos, manipulação de recursos por meio de representações, mensagens auto-descritivas e a hipermídia como o motor do estado do aplicativo.

Por fim, é adicionada a restrição de sistema em camadas como mostra a Figura 2.2, na qual é possível observar componentes intermediários entre o cliente e o servidor.

O estilo do sistema em camadas permite uma arquitetura a ser composta de camadas hierárquicas, restringindo o comportamento do componente de tal forma que cada componente não pode "ver" além da camada com as quais estão interagindo. Ao restringir o conhecimento do sistema em uma única camada, é colocado um limite na complexidade geral do sistema promovendo a independência das camadas. Camadas podem ser usadas para encapsular serviços legados e para proteger os novos serviços a partir de clientes legados. Os intermediários também podem ser usados para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga entre os serviços.

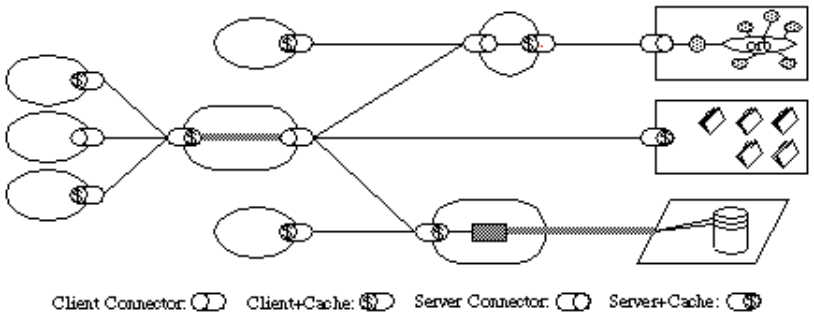


Figura 2.2: Restrição Sistema em Camadas (Fielding, 2000)

2.2 Elementos da Arquitetura REST

O elemento chave na abstração da arquitetura REST é o *recurso*. Um recurso é qualquer conceito que possa ser alvo de referência de um autor de um hipertexto (Fielding, 2000), como por exemplo: um documento, uma imagem, uma coleção de outros recursos ou

até mesmo objetos não-virtuais como uma pessoa. Alguns recursos são considerados estáticos, pois seus valores não se alteram durante o tempo. Já outros têm o seu valor alterado constantemente ao longo do tempo e são considerados dinâmicos. Para localizar os recursos envolvidos em uma interação entre os componentes da arquitetura REST é utilizado o chamado *Identificador de Recurso*.

Uma representação é uma sequência de bytes que representam os dados de um recurso, além de possuir os seus metadados (Fielding, 2000). Os componentes REST que executam ações sobre um recurso utilizam representações para capturar o estado atual do recurso ou para atualizar seu estado. Assim, os recursos ficam desacoplados de suas representações de forma que é possível representar um recurso por meio de diferentes formatos, como por exemplo HTML, XML, JSON, entre outros.

É possível destacar três componentes principais na arquitetura REST:

1. **User Agent:** utiliza um conector cliente a fim de realizar uma requisição ao servidor e recebe a resposta para este pedido. O exemplo mais comum de User Agent é o navegador Web.

2. **Origin Server:** é a fonte das representações dos recursos e responsável por receber e processar as requisições recebidas. Como exemplo, temos os servidores Web.

3. **Componentes intermediários:** entre os componentes intermediários, temos os *proxies* e *gateways*. Ambos são componentes intermediários capazes de fornecer encapsulamento para interfaces de outros serviços, tradução de dados, melhoria de desempenho ou mesmo mecanismos de segurança. A diferença entre eles é que um *proxy* é um intermediário que pode ser escolhido pelo cliente, já um *gateway* é imposto pela rede ou pelo Origin Server.

Observa-se que os desenvolvedores possuem poucas ferramentas que auxiliam a modelagem de suas soluções usando os elementos da arquitetura REST. Assim, é importante o estudo de mecanismos para representar essas soluções de acordo com os elementos propostos nessa arquitetura.

2.3 Web 2.0 e Serviços Web

A Web 2.0 é um conjunto de tendências sociais, econômicas e tecnológicas que coletivamente formam a base para a próxima geração da Internet – mais madura caracterizada pela participação do usuário e por ser aberta (Musser, 2007). O termo surgiu no ano de 2004, como nome de uma série de conferências realizadas pelas empresas O'Reilly Media e a MediaLive International sobre uma nova geração de comunidades e serviços na Web

(O'Reilly, 2005). O principal conceito abordado nessas conferências era a idéia da "Web como plataforma", ou seja, os softwares funcionariam através da Internet, não apenas quando instalados na máquina ou dispositivo móvel do usuário; assim, os mais diversos softwares poderiam se integrar formando uma grande plataforma.

Outro conceito importante que surgiu com a Web 2.0 como a própria definição nos indica é a participação dos usuários na publicação de conteúdo na Web, como é possível observar nos blogs, wikis, redes sociais e outros. Dessa forma, o usuário deixou de ser um mero consumidor de informações e passou a ter um papel de co-desenvolvedor do conteúdo.

Para entender melhor o que vem a ser a Web 2.0 é importante saber quais são os padrões que fundamentam o seu conceito. Entre eles estão (Musser, 2007):

1. **Aproveitamento de Inteligência Coletiva**
   Criar uma arquitetura de participação de forma que o software se torne melhor a medida que as pessoas o utilizem.

2. **Os dados são o próximo "Intel Inside"**
   A mais importante entre as futuras fontes de vantagem competitiva serão os dados.

3. **Inovação na montagem**
   Construir plataformas para promover a inovação, na qual a mistura de serviços e dados crie novas oportunidades e mercados.

4. **Beta perpétuo**
   Afaste-se de antigos modelos de desenvolvimento de software, adotando modelos de Software como um Serviço (SaaS - *Software as a Service*) constantemente atualizados.

5. **Modelos leves e escalabilidade econômica**
   Utilize modelos de desenvolvimento de software mais leves para criar produtos de forma rápida e econômica

Para alcançar o objetivo de criar uma plataforma integrando as diferentes aplicações, são utilizados os chamados Serviços Web. Um Serviço Web é um componente de software com baixo acoplamento que expõe funcionalidades para um cliente na Internet (ou intranet) usando padrões da Web como HTTP, XML, SOAP, WSDL e UDDI (Ribaric et al., 2008). Dessa forma, suas interfaces podem ser definidas, descritas e descobertas por meio de artefatos XML (Yu et al., 2008), permitindo que as aplicações realizem interações com os serviços por troca de mensagens XML.

Os três principais padrões definidos para suportar o desenvolvimento dos serviços Web são:

1. SOAP[1] (*Simple Object Access Protocol*): protocolo baseado em XML para troca de informações entre o serviço Web e seus consumidores;

2. WSDL[2] (*Web Services Description Language*): linguagem para descrever de forma abstrata as funcionalidades do serviço, como as operações da interface e mensagens trocadas entre o serviço e as outras partes envolvidas, bem como os detalhes concretos da implementação como o *endpoint*;

3. UDDI[3] (*Universal Description Discovery and Integration*): oferece um serviço de registro que permite a publicação e descoberta de serviços Web.

Ao longo do tempo, novos padrões foram surgindo a fim de facilitar o desenvolvimento de Serviços Web. Porém, hoje em dia os serviços Web contam com mais de 70 especificações distintas que cobrem diferentes aspectos (Thies e Vossen, 2008), o que acabou tornando esta atividade mais complexa. Para tentar solucionar este problema surgiram os chamados Serviços Web RESTFul.

2.4 Serviços Web RESTFul

Os Serviços Web RESTFul surgiram como uma forma de simplificar o desenvolvimento de serviços Web, seguindo os princípios da arquitetura REST, de modo a garantir a mesma portabilidade alcançada pela Web e tornando mais fácil a publicação e utilização desses serviços (Pautasso et al., 2008). Assim, os padrões já existentes e amplamente usados na Web como HTTP, XML, URI, entre outros foram empregados para o desenvolvimento dos serviços RESTFul, evitando assim o excesso de padronização presente nos serviços Web SOAP.

2.4.1 Propriedades dos Serviços Web RESTFul

Por seguir os princípios da arquitetura REST, os serviços RESTFul são orientados a recursos e possuem quatro propriedades importantes: endereçamento, comunicação sem estado, conectividade e interface uniforme (Richardson e Ruby, 2007). Nas seções seguintes será mostrado como as tecnologias Web tornam possíveis as implementações dessas propriedades.

[1] http://www.w3.org/TR/soap/
[2] http://www.w3.org/TR/wsdl/
[3] http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

2.4.1.1 Endereçamento

Uma aplicação é considerada endereçável se expõe os aspectos interessantes do seu conjunto de dados como recursos. Os recursos nos serviços Web RESTFul são expostos por meio de URIs, portanto um serviço endereçável deve expor uma URI para cada parte da informação que possa vir a servir. Devido a essa propriedade, é possível acessar os recursos diretamente através da URI.

2.4.1.2 Comunicação sem estado

Uma comunicação sem estado significa que cada requisição deve acontecer em completo isolamento, ou seja, quando o cliente realiza a solicitação HTTP, ela deve incluir todas as informações necessárias para que o servidor possa responder o pedido, sem a necessidade de consultar requisições anteriores. Se alguma informação é importante para o processamento da solicitação, esta deve ser enviada juntamente com a requisição.

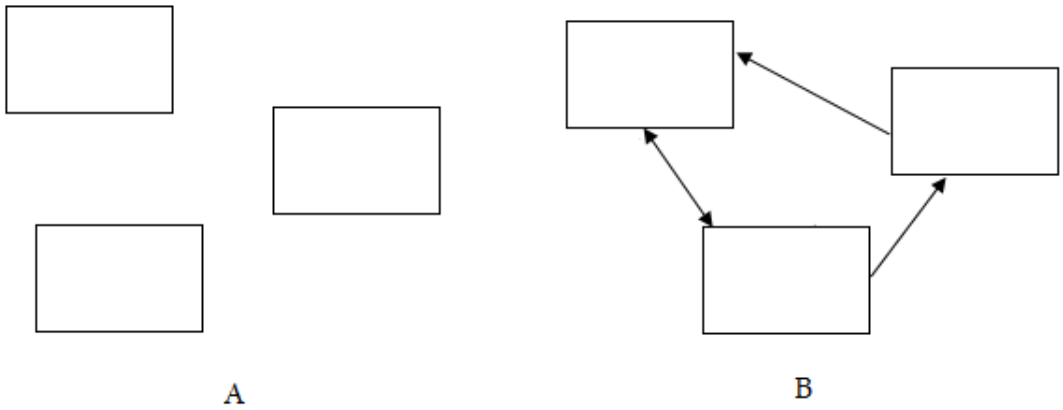
Da mesma forma que a propriedade de endereçamento considera que cada recurso deve possuir sua própria URI, a propriedade de comunicação sem estado afirma que cada possível estado do servidor é um recurso e portanto deve possuir sua própria URI, tornando assim cada requisição totalmente desconectada das outras.

Eliminando a necessidade de armazenamento de estado entre as requisições é possível acrescentar algumas funcionalidades, como por exemplo realizar o *bookmark* da URI com o estado desejado para acesso posterior, evitando a necessidade de percorrer dezenas de estados até chegar aonde deseja. É possível ainda realizar o balanceamento de carga das requisições entre diferentes servidores, pois as requisições são totalmente independentes.

2.4.1.3 Conectividade

As representações dos recursos na maioria das vezes são mais que apenas dados serializados, pois geralmente são hipermídias: documentos com dados e possivelmente hiperlinks para outros recursos. Dessa forma, esses documentos podem conter URIs de outros possíveis estados da aplicação que de alguma forma estão conectados ao recurso que está sendo acessado, seguindo um dos princípios da arquitetura REST que enuncia a "hipermídia como motor do estado da aplicação" (Fielding, 2000).

Essa qualidade de permitir a conexão por meio de hiperlinks é chamada de conectividade. Na Figura 2.3, os recursos estão representados por retângulos e os hiperlinks por setas, e é possível notar as diferenças entre os serviços que são bem-conectados (Serviço B) e os não-conectados (Serviço A).



Serviço A: endereçável mas não é conectado pois não há indicação de relacionamento entre os recursos.
Serviço B: endereçável e bem-conectado com ligações através de hiperlinks.

Figura 2.3: Conectividade - Adaptado de (Richardson e Ruby, 2007)

2.4.1.4 Interface Uniforme

A interface uniforme requerida pelos serviços RESTFul é alcançada por meio dos 4 métodos básicos do protocolo HTTP para as 4 operações mais comuns sobre um recurso. São elas:

1. Recuperar uma representação de um recurso: HTTP GET
2. Criar um novo recurso: HTTP PUT para uma nova URI ou HTTP POST para uma URI já existente
3. Modificar um recurso existente: HTTP PUT para sobrescrever o recurso ou HTTP POST para anexar um novo conteúdo ao recurso
4. Excluir um recurso existente: HTTP DELETE

O método GET aplicado a uma URI existente retorna uma representação de um recurso no corpo da resposta da solicitação. Para excluir um recurso o método DELETE é utilizado, e a resposta à requisição pode conter uma mensagem de status ou simplesmente nenhuma informação. Para criar um novo recurso ou sobrescrever um já existente é utilizado o método PUT. Em ambos os casos usualmente é necessário incluir a representação do recurso no corpo da requisição (a menos que os dados a serem criados ou sobrescritos sejam simples e portanto incluídos como parâmetros da URI). No caso de um pedido para

sobrescrever os dados, é necessário passar a URI já existente do recurso; no caso de criação de um novo recurso é necessário informar uma nova URI não existente.

O método POST possui um comportamento mais complexo em relação aos demais métodos. Ele é utilizado para criar novos recursos "subordinados", ou seja, que dependam de outros recursos já existentes ou ainda para alterar o conteúdo de um recurso sem sobrescrevê-lo. Em geral, quando usado para a criação, o método POST é aplicado sobre o recurso "pai" ou também chamado de "fábrica", que é responsável por criar o novo recurso e informar a URI gerada através do cabeçalho *Location* contido na resposta HTTP. Quando usado para alteração, este método apenas anexa o conteúdo da representação transcrito no corpo da requisição ao recurso.

Em uma arquitetura baseada em recursos é fundamental que todos os métodos HTTP sejam utilizados de acordo com esta interface uniforme, pois dessa forma é possível prever o comportamento de cada requisição. Quando essa interface não é seguida conforme especificado, podem ocorrer consequências desastrosas, como por exemplo, utilizando um método GET para excluir um recurso é possível que os clientes do serviço queiram realizar uma busca do recurso e ao invés disso ele seja removido, consequentemente perdendo os dados.

2.4.2 Vantagens e Desvantagens no uso de Serviços Web RESTFul

Entre as vantagens apresentadas pela arquitetura REST em relação ao estilo RPC estão (Pautasso et al., 2008; Richardson e Ruby, 2007):

- **Simplicidade no desenvolvimento:** utiliza os padrões da Web, evitando o "excesso de padronização";
- **Escalabilidade:** permite suporte a *cache*, *clusterização* e balanceamento de carga;
- **Possibilidade de múltiplas representações para um mesmo recurso:** permite a utilização de formatos leves para melhorar o desempenho;
- **Possui comunicação sem estado:** torna as requisições independentes, permitindo o *bookmark* de URIs e melhoria da escalabilidade;
- **Utiliza a hipermídia com motor do estado da aplicação:** permite conectar os recursos por meio de hiperlinks;
- **Interface Uniforme:** utiliza os métodos do HTTP para realizar as operações sobre os recursos de maneira uniforme.

Como desvantagens estão:

- **Dependência do protocolo HTTP:** é permitido apenas a utilização do protocolo HTTP;

- **Não possui um padrão para descrição dos serviços:** não possui uma linguagem para descrição dos serviços, como o WSDL. Os provedores de serviços oferecem geralmente apenas uma descrição textual da API;

- **Não possui repositório de serviços:** não possui um repositório central de serviços, como o UDDI, portanto a busca de serviços deve ser realizada de forma manual;

- **Alguns *proxies* "barram" requisições com o método PUT:** restrições de infra-estrutura, como por exemplo *firewalls*, impedem o envio de requisições com o método HTTP PUT.

2.5 Composição de Serviços Web RESTFul

A composição de serviços Web é um dos principais fundamentos da computação orientada a serviços (Erl, 2005) e consiste na combinação de diferentes Serviços Web já existentes a fim de promover novas funcionalidades. A abordagem mais comum para a composição de serviços é a chamada Orquestação de Serviços, na qual os serviços estão cientes de sua participação dentro de um processo de composição e um processo central controla e coordena a execução de todos os serviços envolvidos (Rauf et al., 2010).

Para os serviços Web no estilo RPC baseado em operações, como por exemplo o SOAP, existem diferentes linguagens para a composição de serviços tais como WS-BPEL[4] e WSCI[5], que definem o fluxo de controle e dos dados determinando quais operações deverão ser executadas. No entanto, como os serviços Web RESTFul seguem um estilo arquitetural distinto, é necessário que novas tecnologias sejam desenvolvidas para compor esses serviços.

A composição de serviços RESTFul consiste em construir um novo recurso com as funcionalidades fornecidas pelo conjunto de recursos existentes (Pautasso, 2009) como ilustrado na Figura 2.4. O recurso C é um recurso composto, no qual engloba as funcionalidades dos recursos R e S. O recurso composto C pode então manter o seu próprio estado independente e caso ocorra transição neste estado é possível provocar a interação com os seus recursos componentes.

Atualmente já existem algumas abordagens para a composição de serviços RESTFul como o BPEL for REST (Pautasso, 2008) que propõe uma extensão à linguagem BPEL

[4]http://www.oasis-open.org/committees/wsbpel
[5]http://www.w3.org/TR/wsci/

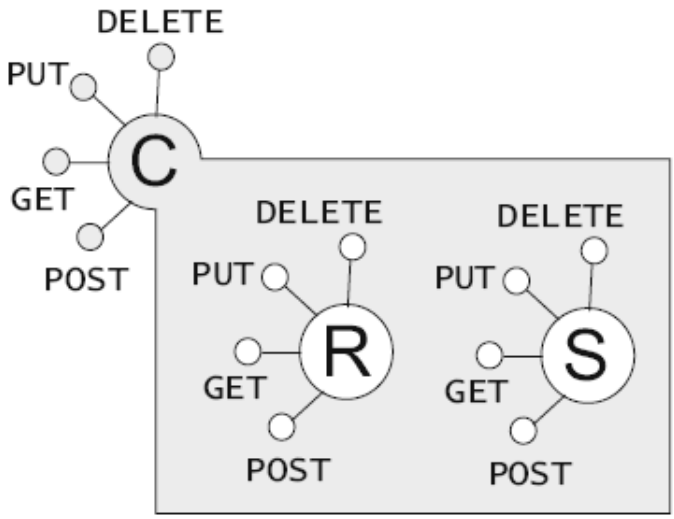


Figura 2.4: Composição de Serviços RESTFul (Pautasso, 2009)

permitindo o seu uso com uma arquitetura orientada a recursos. Há também uma abordagem sugerida por Rauf et al. (Rauf et al., 2010) que propõe a utilização da modelagem UML para a criação de serviços RESTFul compostos.

2.6 Considerações Finais

Neste capítulo foram apresentados os conceitos da arquitetura REST, da Web 2.0 e Serviços Web, bem como as características principais dos Serviços Web RESTFul. Foi apresentada também a importância da composição de serviços Web e quais as novas abordagens dessa prática aplicadas aos serviços RESTFul, que servirão como referenciais para os estudos necessários desse projeto de mestrado a fim de desenvolver a ferramenta proposta.

Capítulo

3

MDD

O Desenvolvimento Orientado a Modelos é uma área da Engenharia de Software que vem crescendo nos últimos anos graças aos consideráveis benefícios que pode proporcionar em termos de produtividade, qualidade e reúso. Neste capítulo serão apresentadas a definição de MDD, suas vantagens e desvantagens, os seus principais elementos, bem como as duas principais abordagens para o desenvolvimento de aplicações Web existentes dentro da Engenharia Web: a WebML e a UWE.

3.1 Definição do MDD

O Desenvolvimento Orientado a Modelos (MDD - *Model Driven Development*) é uma metodologia de desenvolvimento de software que foca na construção de modelos como artefatos de primeira classe e na definição de transformações a serem aplicadas nestes para a produção de código-fonte (Teppola et al., 2009). Dessa forma, elevando o nível de abstração na criação do software. Assim, sua intenção é tornar mais simples e padronizadas as diversas atividades que formam o ciclo de desenvolvimento de software (Hailpern e Tarr, 2006).

O MDD considera que os modelos são artefatos imprescindíveis no processo de desenvolvimento de software e é a partir deles que a aplicação será construída, enquanto que no desenvolvimento de software convencional, a modelagem é utilizada apenas para facilitar a análise do problema, tornando-se portanto um apoio aos desenvolvedores nas fases pos-

teriores do projeto. Dessa forma, os programadores implementam o código manualmente, o que muitas vezes torna os modelos criados anteriormente inconsistentes com a realidade expressa pelo software. Já no MDD, como os modelos são os responsáveis pela geração do código, estão sempre atualizados facilitando a manutenção e documentação do software.

A construção de modelos como uma abstração do software a ser implementado permite que os desenvolvedores foquem os seus esforços na modelagem do domínio do problema e não em uma solução ao nível de programação (Schauerhuber et al., 2006), escondendo possíves detalhes de implementação, o que aumenta a portabilidade e o reúso.

Porém, apesar de todos os benefícios mostrados, o MDD não é a solução definitiva para todos os problemas inerentes ao processo de desenvolvimento de software. Na próxima seção serão listadas algumas das vantagens e desvantagens vindas com a adoção dessa metodologia.

3.1.1 Vantagens e Desvantagens no uso do MDD

As vantagens presentes no desenvolvimento orientado a modelos são alcançadas graças à capacidade de automação no processo de geração de código-fonte, por meio de transformações que são aplicadas aos modelos, evitando que os desenvolvedores precisem executar tarefas repetitivas para a geração do código executável. Entre as vantagens apontadas por Kleppe et al. e Lucrédio (Kleppe et al., 2003; Lucrédio, 2009) podemos citar:

- **Produtividade:** o aumento da produtividade no processo de desenvolvimento de software deve-se principalmente a redução das tarefas repetitivas por parte dos desenvolvedores e também a possibilidade de reúso;

- **Corretude:** as transformações utilizadas pelo MDD para a geração de código-fonte evitam a execução de tarefas repetitivas e manuais por parte dos programadores, evitando erros acidentais como por exemplo erros de digitação, garantindo dessa forma um código mais correto e livre de erros manuais;

- **Portabilidade:** diferentes transformações aplicadas a um modelo são capazes de gerar código para diferentes plataformas, favorecendo dessa forma a portabilidade;

- **Manutenção:** no desenvolvimento convencional à manutenção ou evolução do software é realizada diretamente no código-fonte o que pode despender um esforço muito grande, quase comparado ao produzido durante o desenvolvimento. Já no MDD a manutenção é realizada nos modelos e o código é então re-gerado a partir desses;

- **Documentação:** no MDD os modelos são os artefatos principais do processo de desenvolvimento de software e por essa razão não se desatualizam, pois o código é

gerado a partir desses. Dessa forma, a documentação do sistema é mantida sempre atualizada;

- **Comunicação:** os modelos são mais simples de compreender do que o código-fonte, o que torna mais fácil a comunicação entre os desenvolvedores e os demais membros da equipe;

- **Reutilização:** no MDD a reutilização de modelos faz com que o código possa ser re-gerado automaticamente para um contexto diferente, sem a necessidade da realização de tarefas manuais por parte dos desenvolvedores, o que não ocorre no desenvolvimento convencional.

Segundo Lucrédio (Lucrédio, 2009) algumas das principais desvantagens trazidas pelo MDD são:

- **Rigidez:** o software criado através do MDD possui uma maior rigidez, pois grande parte do código é gerado automaticamente e permanece além do alcance do desenvolvedor;

- **Complexidade:** as ferramentas utilizadas para o desenvolvimento orientado a modelos, como ferramentas de modelagem, transformação e geradores de código, são mais difíceis de construir e manter, o que acaba adicionando complexidade ao processo de desenvolvimento de software;

- **Desempenho:** os geradores de código na maioria das vezes incluem código desnecessário ao software, o que acaba degradando o seu desempenho em relação aos códigos escritos à mão;

- **Alto investimento inicial:** a prática do MDD exige um investimento inicial maior, pois a construção de uma infra-estrutura de reutilização orientada a modelos exige mais esforço e tempo.

3.1.2 Principais Elementos do MDD

Para automatizar o processo de transformação utilizado pelo MDD é necessária a combinação de alguns elementos. Segundo Lucrédio (Lucrédio, 2009) os principais elementos do MDD são:

- *Ferramenta de modelagem:* essa ferramenta é fundamental para a criação dos modelos. Estes são criados pelo engenheiro de software a fim de descreverem os conceitos do domínio, de acordo com uma linguagem específica de domínio (DSL - *Domain*

Specific Language). Como os modelos serão utilizados como entrada para a geração do código-fonte, é necessário que estes estejam semanticamente completos e corretos de acordo com a DSL, pois serão processados pelo mecanismo de execução de transformações e não por um ser humano.

- *Ferramenta para definir transformações:* para que os modelos sejam transformados em outros modelos ou no próprio código-fonte é preciso definir regras de mapeamento de modelo para modelo, ou de modelo para código. Portanto é imprescindível uma ferramenta que permita a definição dessas regras da forma mais natural possível.

- *Mecanismo para executar transformações:* uma vez que o modelo e as transformações já foram definidas, é necessário um mecanismo que execute essas transformações. Além disso, é importante que esse mecanismo mantenha informações de rastreabilidade, permitindo identificar a origem de cada elemento gerado.

Assim, de acordo com esses elementos é possível definir as tarefas a serem executadas para o desenvolvimento do software na metodologia MDD. Como ilustrado na Figura 3.1, o engenheiro de software é responsável por construir os modelos usando uma ferramenta de modelagem e também por criar as regras de mapeamento na ferramenta para definir transformações. Uma vez construídos, estes artefatos servem de entrada para o mecanismo para executar transformações que efetivamente aplica as regras de mapeamento definidas, a fim de gerar novos modelos ou mesmo código-fonte.

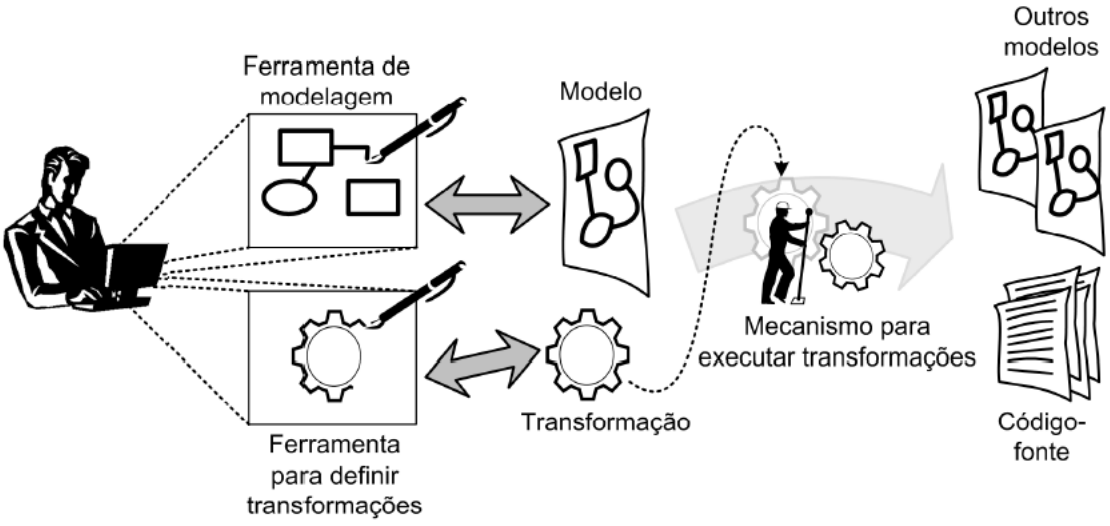


Figura 3.1: Principais Elementos do MDD (Lucrédio, 2009)

Além desses elementos vistos até agora, outros dois conceitos são fundamentais para o entendimento das técnicas de desenvolvimento orientado a modelos: a linguagem específica de domínio (DSL) e a Metamodelagem.

DSL

Uma DSL é uma linguagem processável por máquina cujos termos são derivados de um modelo de domínio específico, ou seja, são usadas para domínios particulares e capturam precisamente a semântica deste domínio (Thomas e Barry, 2003). Em uma definição mais simples, uma DSL pode ser considerada uma linguagem pequena, normalmente declarativa, focada em um domínio de problema em particular (van Deursen et al., 2000). Portanto, essas linguagens são criadas especificamente para resolver problemas neste domínio.

Embora uma DSL não seja capaz de oferecer uma solução geral para diferentes problemas, estas linguagens fornecem soluções melhores para um domínio particular (van Deursen et al., 2000), quando comparadas às linguagens de propósito geral, pois permitem que os *experts* no domínio entendam com mais facilidade a solução proposta.

Ao nível de linguagens de modelagem uma das mais conhecidas é a UML[1], amplamente utilizada para a modelagem de sistemas que usam orientação a objeto. A UML é considerada uma linguagem de modelagem de propósito geral, porém pode ser especializada por meio de mecanismos de extensão, conhecidos como *Profiles*, tornando-a uma linguagem específica de domínio. Como exemplo temos o UML *Profile For Enterprise Application Integration* (OMG, 2004b), especializada para a integração de aplicações, ou ainda o UML *Profile for Modeling and Analysis of Real-Time and Embedded Systems* (OMG, 2009), específica para o desenvolvimento de sistemas embarcados e de tempo real.

Metamodelagem

Um metamodelo descreve a estrutura dos modelos e portanto é uma das principais características do MDD. O metamodelo é capaz de definir de forma abstrata os construtores de uma linguagem de modelagem e seus relacionamentos, além das regras de modelagem. Dessa forma, o metamodelo e o modelo possuem uma relação de classe e instância, ou seja, cada modelo é uma instância do metamodelo.

Graças a essas características o uso de um metamodelo se faz necessário para contemplar atividades fundamentais para o MDD como: validação de modelos, realizar transformações de modelo, construção de uma DSL, geração de código e integração de ferramentas de modelagem a um domínio (Stahl e Voelter, 2006).

3.2 Principais Abordagens do MDD na Engenharia Web

A Engenharia Web foca nas metodologias, técnicas e ferramentas que são a base para o desenvolvimento de aplicações web e portanto é considerado um domínio no qual o

[1]UML - Unified Modeling Language

MDD pode ser útil, especialmente no tratamento de problemas referentes à evolução e à adaptação de sistemas web em relação às mudanças tecnológicas (Koch, 2006). Nos últimos anos, a comunidade tem proposto diferentes abordagens para a modelagem de aplicações web com foco na construção de modelos para navegação e adaptação. Nas seções a seguir serão apresentadas duas abordagens que vêm ganhando destaque: a WebML e a UWE.

3.2.1 WebML

WebML é o acrônimo de *Web Modelling Language* que foi definida no ano de 1998 como um modelo conceitual para a especificação de aplicações Web com grande capacidade de dados (Ceri et al., 2009), ou seja, seu principal objetivo é publicar ou manipular conteúdo armazenado em uma ou mais fontes de dados (Manolescu et al., 2005).

Enquanto outros modelos conceituais focam mais nas fases iniciais do processo de desenvolvimento, como por exemplo levantamento de requisitos, a WebML se concentra nas fases posteriores, como o projeto e em seguida a implementação. Além disso, a WebML tem como princípio a separação de responsabilidades: conteúdo, interfaces lógicas e a apresentação são definidas em modelos separados. Esses modelos podem ser classificados em: modelos de dados, hipertextos e apresentação.

Os modelos de dados são uma extensão do modelo Entidade-Relacionamento tradicional, usado para representar dados e não apresentam nenhuma novidade em relação aos MERs. O modelo de apresentação permite definir o layout desejado para as páginas Web.

Já o modelo de hipertexto consiste de uma ou mais visões do *site*, cada uma delas direcionada a um papel de usuário específico ou dispositivo do cliente. Uma visão do *site* é uma coleção de páginas, possivelmente agrupadas em áreas, na qual o conteúdo dessas páginas é expresso por meio de componentes de publicação de dados chamados de *unidades de conteúdo (content unit)*. A lógica de negócio desencadeada com as interações do usuário é representada por uma sequência de *unidades de operação (operation unit)* que são componentes responsáveis por modificar os dados ou mesmo executar outras ações importantes como por exemplo envio de email.

Unidades de operação e de conteúdo são conectados por meio de links, que especificam o fluxo de dados entre eles e o processo para publicação de conteúdo nas páginas. O cenário completo de um modelo hipertexto pode ser visto na Figura 3.2, a qual mostra um exemplo de um site de empréstimos, em que o usuário pode navegar por um conjunto de 3 páginas partindo da Home Page, além da possibilidade de inclusão de uma nova proposta.

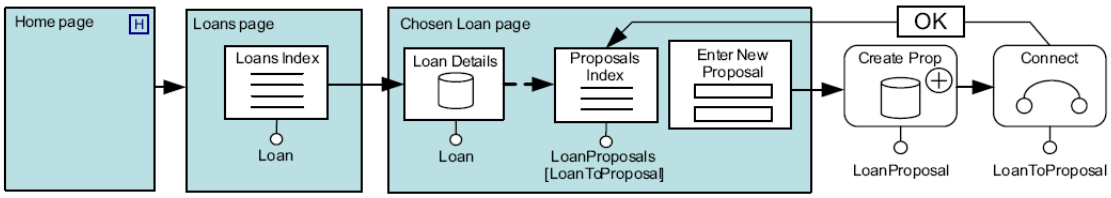


Figura 3.2: Modelo hipertexto WebML (Ceri et al., 2009)

O modelo hipertexto é desenhado em uma notação visual simples e intuitiva, mas com uma semântica rigorosa, o que permite a transformação automática dos diagramas no código-fonte da aplicação.

A linguagem WebML é bastante extensível e permite a definição de unidades customizáveis, o que pode ser usado para incluir novas operações como é o caso das operações relacionadas aos Serviços Web. Para esse fim, foram criadas novas unidades para representar a interação com as operações dos serviços web e sua conversação. Na Figura 3.3 temos um exemplo de como publicar um serviço Web, que busca todos os produtos no banco de dados e envia os dados recuperados como resposta para a aplicação que invocou este serviço. O componente mais à esquerda é o *Solicit Unit*, que representa o ponto de entrada da operação a ser invocada. O segundo componente ilustrado na figura é o *XML Out Unit*, que extrai os dados do banco de dados e gera um documento XML. O componente mais à direita é o *Response Unit*, que recebe o XML criado pelo componente anterior e envia para o cliente do serviço. Mais abaixo na figura temos o componente *Error Response Unit*, que intercepta todos os erros que possam ser lançados pelos demais componentes e notifica o cliente com uma mensagem de falha.

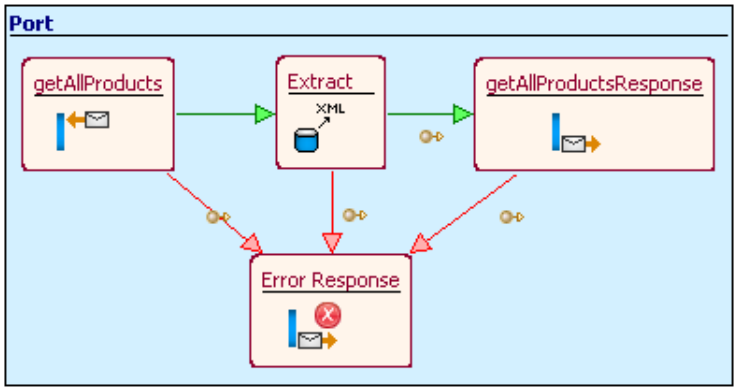


Figura 3.3: Exemplo de Serviço Web modelado com os componentes da WebML (WebRatio, 2009)

Para dar suporte ao desenvolvimento dessas aplicações usando a WebML é disponibilizada uma ferramenta CASE chamada WebRatio capaz de realizar a prototipação rápida desses modelos propostos e a geração do código-fonte a partir deles.

A seguir são apresentadas as principais vantagens e desvantagens no uso da WebML segundo Ceri et al. e Schauerhuber et al. (Ceri et al., 2009; Schauerhuber et al., 2006).

Vantagens

- Disponibilidade de modelos conceituais bem definidos;
- Linguagem bastante extensível, permitindo a criação de novas unidades customizáveis;
- Notação visual simples e intuitiva;
- Pesquisa contínua a mais de uma década permitindo a incorporação de novas tecnologias da Web.

Desvantagens

- Não possui um metamodelo explícito. Os modelos são definidos através de arquivos XML que são estruturados de acordo com alguns DTD's, os quais não possuem a mesma expressividade de um metamodelo;
- Por não utilizar um metamodelo, não é possível o uso de linguagens para transformações de modelo como o QVT(OMG, 2011) e ATL(ECLIPSE, 2010). A transformação de modelos é realizada através de transformações XSLT;
- A prototipação dos modelos é realizada através da ferramenta WebRatio, que trabalha de forma *standalone* dificultando o desenvolvimento colaborativo.

3.2.2 UWE

A UML-based Web Engineering (UWE) é uma abordagem proposta para modelagem de sistemas web, que da mesma forma que a WebML possui como característica a separação de responsabilidades; existem diferentes modelos para apresentação, conteúdo, estrutura hipertexto e processos (Kroib e Koch, 2008). Essa abordagem possui um metamodelo baseado na UML, ou seja, o metamodelo da UWE é definido como uma extensão conservativa da UML 2.0.

Essa extensão é considerada conservativa pois mantém todos os elementos existentes na UML, apenas incluindo novos elementos, por meio de herança, que são necessários para a modelagem dos novos conceitos das aplicações Web. Para a inclusão desses novos elementos foi criado um *Profile* UML. Dessa forma, o metamodelo da UWE continua sendo compatível com o meta-metamodelo usado para a definição da UML chamado de MOF (*Meta-Object Facility*), que é definido pela OMG e portanto segue a abordagem

do desenvolvimento orientado a modelos chamada de *Model Driven Architeture* (MDA) (Koch, 2006).

MDA

O *Object Management Group*[2] (OMG) tem como objetivo ajudar na redução de complexidade e diminuir custos de desenvolvimento e também acelerar a introdução de novas aplicações de software. Para alcançar esses objetivos o grupo introduziu o *framework* arquitetural *Model Driven Architeture* (MDA), que conta com o apoio de diversas especificações (MOF, UML, XMI) para aumentar a interoperabilidade, reusabilidade e portabilidade de componentes de software (OMG, 2003).

A base do MDA é o MOF (*Meta-Object Facility*)(OMG, 2004a), o meta-metamodelo que serve como base para a definição de todas as linguagens de modelagem. O MOF consiste em um padrão orientado a objetos que permite a definição de classes com atributos e relacionamentos, além de oferecer uma interface padrão de acesso aos dados do modelo, oferecendo métodos para manipulação dos dados e dos metadados (Lucrédio, 2009).

A arquitetura de modelagem da MDA pode ser dividida em quatro camadas como podemos ver na Figura 3.4. O primeiro nível (M0) representa os dados propriamente ditos. O segundo nível (M1) corresponde aos modelos. O terceiro nível (M2) é composto pelos metamodelos, utilizados para a definição dos modelos. No caso da MDA o principal metamodelo utilizado é a UML. Já o quarto e último nível (M3) é utilizado para definir os metamodelos do M2, ou seja, um meta-metamodelo que define as linguagens de modelagem. O meta-metamodelo usado pelo MDA é o MOF (Lucrédio, 2009).

Além dessa arquitetura, o MDA define também o XMI (*XML Metadata Interchange*)(OMG, 2007), um formato para representar modelos em XML capaz de fazer o intercâmbio de UML e outros modelos baseados em MOF. Sendo assim, o XMI define um mapeamento de UML/MOF para XML, que é capaz de integrar ferramentas, repositórios e aplicações.

Para realizar consultas e transformções nos modelos o MDA define o padrão QVT (*Queries/Views/Transformations*) (OMG, 2011). O QVT define uma linguagem textual e uma notação para definir consultas e transformações baseadas no MOF, ou seja, é possível definir regras de mapeamento entre modelos que sejam instâncias de MOF (Lucrédio, 2009).

Os principais modelos do MDA

O OMG definiu três modelos principais para o ciclo de vida do desenvolvimento MDA que são: o *Computation Independent Model* (CIM), o *Plataform Independent Model* (PIM) e o *Plataform Specific Model* (PSM) e são descritos pela OMG (OMG, 2003) da seguinte forma:

[2]http://www.omg.org/

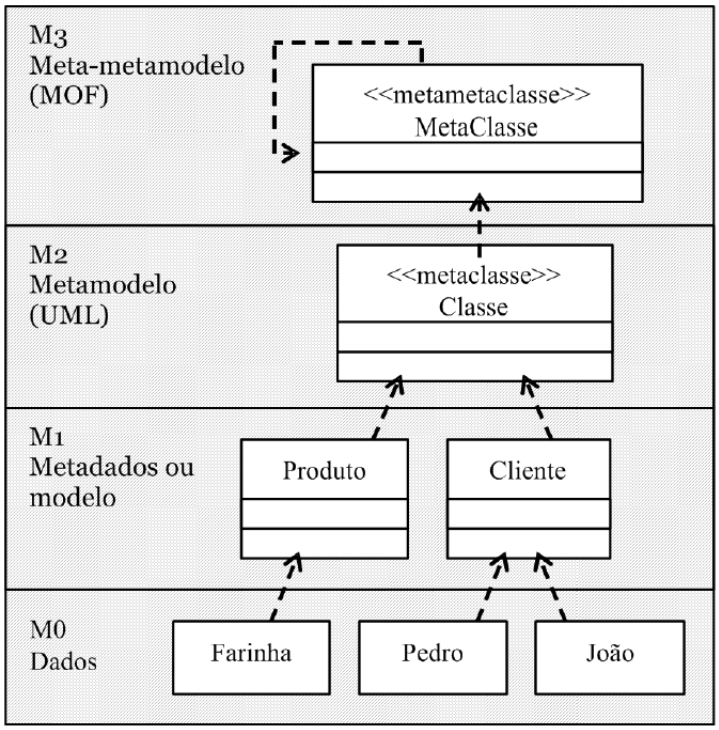


Figura 3.4: Arquitetura de Metamodelagem (Lucrédio, 2009)

- CIM: também chamado de modelo de domínio, não mostra detalhes das estruturas do sistema, apenas um vocabulário familiar aos profissionais do domínio, o qual é usado para realizar as especificações. Dessa forma, esses profissionais não conhecem os modelos e artefatos utilizados para criar as funcionalidades do sistema, conhecendo apenas os seus requisitos expressos pelo CIM. Sendo assim, esse modelo se mostra importante para realizar a ligação entre o trabalho dos especialistas no domínio com os especialistas no *design*, responsáveis pela construção dos artefatos.

- PIM: representa uma visão do sistema independente de plataforma, com foco apenas nas funcionalidades do sistema, escondendo os detalhes específicos de cada plataforma.

- PSM: um PSM combina as especificações do PIM com os detalhes específicos da plataforma.

A Figura 3.5 representa o ciclo de vida do MDA. Inicialmente na fase de análise o especialista do domínio recebe os requisitos para o desenvolvimento da aplicação e gera um CIM a partir desses requisitos. Depois dessa etapa, já na fase de projeto, o especialista no *design* transforma esse CIM em um PIM com os modelos necessários para representar as funcionalidades do sistema propostas anteriormente. Em seguida, esse PIM pode ser transformado em um ou mais PSMs que possuirão as informações específicas de cada

plataforma. A seguir, já na fase de codificação, o PSM sofre uma transformação a fim de gerar o código-fonte da aplicação.

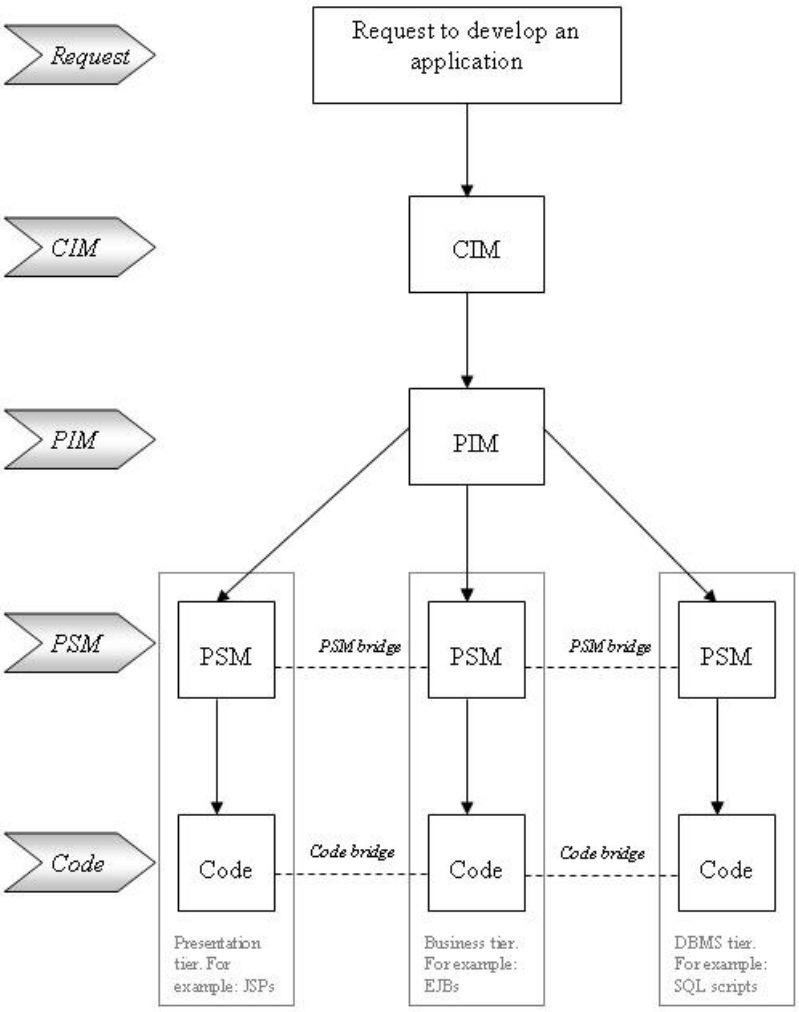


Figura 3.5: Ciclo de vida MDA (Technology, 2006)

A abordagem MDA e a UWE

O metamodelo da UWE em sua estrutura principal conta com cinco pacotes que são separados de acordo com as suas responsabilidades. São eles: *Requirements, Content, Navigation, Presentation e Process.*

- **Requirements:** é utilizado para definir o modelo de requisitos que pode combinar diagramas de casos de uso e diagramas de atividades. Geralmente utilizado para definir um modelo de domínio (CIM).
- **Content:** a modelagem de conteúdo para as aplicações Web não se diferencia das demais aplicações dentro da UWE, ou seja, o conteúdo é modelado através de ele-

mentos padrões da UML como diagramas de classes, associações e pacotes. São utilizados para definir um modelo independente de plataforma (PIM).

- **Navigation:** utilizado para a criação dos modelos de navegação através de diagramas de classes. As principais classes desse pacote são: *Node* e *Link*, que são generalizações de Class(UML) e Association(UML), respectivamente. São utilizados para definir um modelo independente de plataforma (PIM).

- **Presentation:** utilizado para a criação dos modelos de apresentação capazes de fornecer uma visão abstrata da interface do usuário em uma aplicação Web, ou seja, esses modelos descrevem a estrutura básica dos elementos da interface como botões, textos, imagens, entre outros. São utilizados para definir um modelo independente de plataforma (PIM).

- **Process:** utilizado para a integração de processos de negócio com os modelos de navegação e os modelos de apresentação. Dessa forma é possível saber qual processo a ser executado quando ocorrer alguma navegação e também quais as entradas de dados. Além disso, é possível definir o comportamento do processo de negócio através de diagramas de atividades como pode ser visto na Figura 3.6. O diagrama de atividades ilustrado nessa figura mostra qual o fluxo de tarefas a serem executadas quando o processo de negócio *BuyAlbum* for invocado. São utilizados para definir um modelo independente de plataforma (PIM).

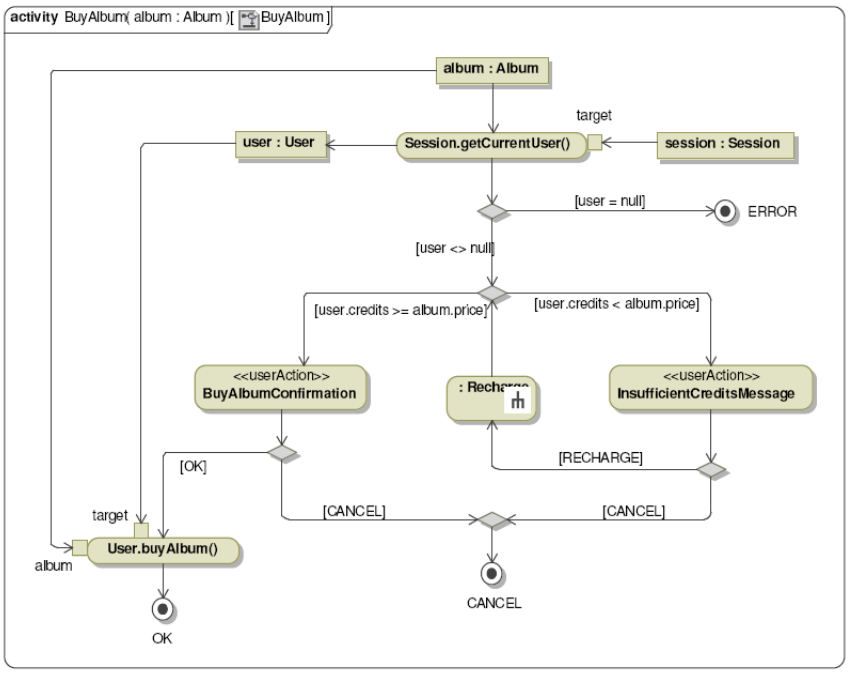


Figura 3.6: Processo de negócio UWE (Kroib e Koch, 2008)

A seguir são apresentadas as principais vantagens e desvantagens no uso da UWE, segundo Kroib e Koch (Kroib e Koch, 2008).

Vantagens

- Possui um metamodelo baseado na UML, o que torna possível o uso da abordagem MDA durante o desenvolvimento;
- Possibilidade de integração com ferramentas de modelagem que trabalhem com *Profiles* UML;
- Uso de diagramas UML modificados, o que torna mais familiar a modelagem para os desenvolvedores acostumados com essa linguagem.

Desvantagens

- Quantidade de componentes específicos para a modelagem de aplicações Web é menor que em relação a WebML;
- A prototipação dos modelos é realizada através de ferramentas de modelagem UML, porém é necessário realizar adaptações nessas ferramentas para que seja possível a geração de código-fonte;
- As ferramentas de modelagem trabalham de forma *standalone* dificultando o desenvolvimento colaborativo.

3.3 Considerações Finais

Neste capítulo foram apresentados os conceitos relativos ao MDD, destacando as vantagens e desvantagens de seu uso. Foram também descritas as duas principais abordagens da Engenharia Web: a WebML e a UWE que servirão como base para os estudos pertinentes ao desenvolvimento da ferramenta proposta neste projeto de mestrado.

Capítulo

4

Colaboração

No projeto em questão pretende-se utilizar um ambiente colaborativo a fim de melhorar os resultados obtidos com a prática do MDD no domínio da Engenharia Web, mais especificamente na modelagem de Serviços Web RESTFul. Dessa forma, para que seja possível obter maior proveito da prática de colaboração é fundamental compreender a complexidade de um ambiente colaborativo e como utilizá-lo da melhor maneira. Assim, neste capítulo serão apresentados conceitos referentes ao tema de colaboração tais como: CSCW, *Groupwares*, CDE e Wikis. Será realizado também um levantamento dos requisitos que um ambiente colaborativo deve possuir e alguns trabalhos relacionados a colaboração no desenvolvimento orientado a modelos.

4.1 Fundamentos de Colaboração

Graças ao aumento na complexidade do desenvolvimento de software, a colaboração vem surgindo como uma atividade cada vez mais importante, pois para o desenvolvimento de um software complexo e eficiente, geralmente se faz necessário o envolvimento de uma equipe de desenvolvimento. Assim, para a implementação e manutenção de softwares complexos é preciso a colaboração de diversos desenvolvedores (Sriplakich et al., 2006).

O termo colaboração foi definido por McQuay (McQuay, 2004) como sendo duas ou mais pessoas trabalhando juntas para compartilhar e trocar dados, informações, conhecimento, entre outros. Esse cenário é bastante comum, impulsionado principalmente pelo

outsourcing, em que os engenheiros de software exercem suas tarefas de *design*, implementação, implantação e manutenção de software de forma distribuída geograficamente e utilizam a Internet como um meio para a sua interação. Além disso, o trabalho colaborativo de forma eficaz pode ser um fator diferencial para o sucesso de uma organização (Booch e Brown, 2003).

Devido a importância da colaboração nas organizações, surgiu o termo *Computer Supported Cooperative Work (CSCW)* que foi definido por Rama e Bishop (Rama e Bishop, 2006) como o estudo de como as pessoas usam a tecnologia em relação ao hardware e software para trabalhar em conjunto, com tempo e espaço compartilhados. Assim, essa área de pesquisa tem como objetivo estudar impactos organizacionais, sociais e psicológicos do trabalho colaborativo bem como o estudo de ferramentas e técnicas para softwares colaborativos, chamados de *groupwares*.

Groupware permite que muitos usuários trabalhem no mesmo projeto, facilitando a comunicação entre os membros da equipe de forma rápida e clara (Rama e Bishop, 2006). Entre os principais exemplos de *groupwares* usados atualmente estão o email, as salas de bate-papo (*chats*), as wikis, os mensageiros instantâneos, os *blogs*, entre outros. Apesar do grande benefício proposto por esse tipo de software, eles são mais complexos para se desenvolver e manter do que um sistema de usuário único.

A fim de dar suporte ao desenvolvimento de software de forma colaborativa e distribuída surgiram os chamados Ambientes de Desenvolvimento Colaborativo (CDE - *Collaborative Development Environment*). Um CDE é um espaço virtual no qual todas as partes interessadas de um projeto podem discutir, realizar *brainstorms*, compartilhar conhecimentos e trabalhar em conjunto para a realização de uma tarefa (Booch e Brown, 2003). Para facilitar as tarefas pertinentes ao processo de desenvolvimento de software, esse ambiente colaborativo fornece um espaço de trabalho aos membros da equipe associado a um conjunto de ferramentas específicas utilizadas no desenvolvimento de software (Lanubile, 2009).

A motivação principal dos CDEs é integrar *groupwares*, usados para comunicação, com as soluções específicas voltadas para a criação de um software. Entre as principais categorias de ferramentas estão as descritas por Lanubile (Lanubile, 2009):

- **Gerenciamento de Configuração de Software:** as ferramentas para Gerenciamento de Configuração de Software (SCM - *Software Configuration Management*) são capazes de gerir mudanças de uma forma mais controlada, por meio do armazenamento de múltiplas versões de componentes dentro de repositórios. Dessa forma, elas são capazes de gerenciar a evolução do software inclusive adicionando comentários a cada revisão, compartilhando artefatos, gerenciando permissões e controlando

o processo de armazenamento e recuperação das revisões. Entre os exemplos mais conhecidos estão o CVS[1] e o Subversion[2].

- **Monitoramento de mudanças e defeitos:** sua função é registrar e tornar visível aos membros da equipe os defeitos encontrados durante o processo de desenvolvimento do software, por meio de uma interface web. Dessa forma, é possível definir um ciclo de vida para a correção dos defeitos, tornando possível o monitoramento das mudanças e o tempo gasto em cada atividade de correção. Um exemplo bastante utilizado desse tipo de ferramenta é o JIRA[3].

- **Gerenciamento de *builds* e *releases*:** essas ferramentas permitem criar e agendar *workflows* que executem scripts de *build*, compilação de binários, invocação de frameworks de teste, implantação de sistemas em produção e envio de notificações por email aos desenvolvedores, automatizando o processo de geração das *releases*. Deve ser possível também visualizar o status de cada *build* em um painel na interface web como podemos ver, por exemplo, na Figura 4.1. Essa figura mostra um painel com a data e hora em que ocorreu o *build* e as modificações realizadas nos artefatos nessa versão. Além disso, no menu a direita é mostrado o histórico dos últimos *builds* gerados pela ferramenta.

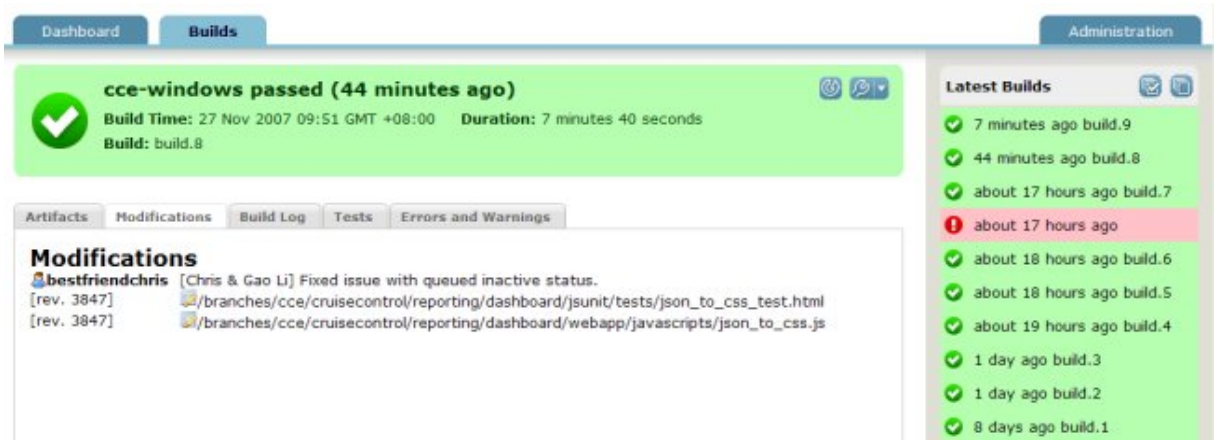


Figura 4.1: Painel com detalhes do build (CruiseControl, 2010)

- **Modelagem de produtos e processos:** essa função engloba os principais recursos disponíveis nas chamadas ferramentas CASE, capazes de modelar visualmente os artefatos de software e também processos.

- **Centro de conhecimento:** um centro de conhecimento permite aos membros da equipe compartilhar explicitamente o conhecimento sobre um determinado pro-

[1]http://savannah.nongnu.org/projects/cvs/
[2]http://subversion.tigris.org/
[3]http://www.atlassian.com/software/jira/

jeto ou unidade de trabalho, incluindo referências técnicas, padrões, FAQs e boas práticas. Atualmente as Wikis tem emergido como uma ferramenta colaborativa importante para a publicação de conhecimento.

- **Ferramentas para comunicação:** servem para realizar a comunicação entre os membros da equipe de forma assíncrona como email, listas de discussão, fóruns e blogs; e de forma síncrona como *chats*, mensageiros instantâneos e videoconferências. A forma mais utilizada atualmente é o email, porém os mensageiros instantâneos vem ganhando cada vez mais espaço nos ambientes de trabalho, pois permitem verificar a disponibilidade dos membros da equipe e contatá-los em tempo hábil.

Portanto, neste projeto o objetivo é desenvolver uma ferramenta de modelagem que seja capaz de se integrar a um ambiente colaborativo. A descrição da ferramenta alvo é apresentada no Capítulo 5.

4.2 Wikis

As wikis são ferramentas web colaborativas capazes de promover uma rápida publicação de informações, pois permite que os usuários adicionem, editem ou revisem conteúdo por meio de um navegador Web, ou seja, uma Wiki pode ser considerada um conjunto de páginas web editáveis (Jang e Green, 2006).

Uma das principais funcionalidades que uma Wiki possui é o suporte à edição colaborativa (Tetard et al., 2009), que permite que todos os usuários habilitados possam editar e revisar seu conteúdo.

Entre os principais recursos existentes em uma Wiki estão os citados por Mehta (Mehta, 2009):

- Histórico completo das revisões, com a possibilidade de reverter a qualquer momento;
- Simples sintaxe de marcação;
- Criação de links de forma simples;
- Algumas wikis não precisam de banco de dados, o que as tornam mais portáveis;
- É possível categorizar páginas para melhorar a organização;
- Controle de acesso de usuários e páginas;
- Recursos de busca.

Devido à simplicidade e facilidade na publicação de conteúdo, uma Wiki será utilizada nesse projeto a fim de atuar como ambiente colaborativo capaz de promover a comunicação e também permitir a troca de experiências entre os integrantes da equipe de desenvolvimento. Além dessas funcionalidades, será incorporada à Wiki uma ferramenta de modelagem e geração de código-fonte conforme a proposta descrita no Capítulo 5, a fim de auxiliar no processo de criação e composição de Serviços Web RESTFul.

4.3 Trabalhos Relacionados de Colaboração no MDD

Na literatura há diversos trabalhos relacionados com o tema de colaboração dentro do desenvolvimento orientado a modelos, o que demonstra a sua importância nessa área de pesquisa. Nessa seção serão descritos trabalhos relacionados de diferentes perspectivas do uso da colaboração aliada ao MDD.

O trabalho de Sriplakich et al. (Sriplakich et al., 2006) é focado em técnicas de comparação e versionamento de modelos, propondo um mecanismo para realizar o *merge* de alterações realizadas em modelos baseados no MOF usando a abordagem de colaboração *copy-modify-merge* que permite diferentes desenvolvedores editarem os modelos concorrentemente. Nessa abordagem, o desenvolvedor realiza uma cópia do artefato que está no repositório para o seu ambiente local, onde serão realizadas as alterações e em seguida estas serão integradas novamente com a cópia existente no repositório.

Para realizar esse processo, é necessário que a ferramenta identifique as mudanças ocorridas, detecte possíveis conflitos causados pela edição concorrente, seja capaz de resolver esses conflitos e integrar o resultado do *merge* ao repositório, sem violar as possíveis restrições de multiplicidade impostas pelo modelo no padrão MOF.

Outro trabalho relacionado com a colaboração no MDD é o de Abeti et al. (Abeti et al., 2009) que está focado no gerenciamento de requisitos para *Business Process Reengineering* (BPR). No seu trabalho Abeti et al. (Abeti et al., 2009) propõem um *framework* BPR para formalizar o conhecimento da organização por meio de modelos ligados aos requisitos do software e que seja capaz de automatizar parcialmente a implementação do sistema. Este *framework* inclui uma Wiki chamada de *Wiki for Requirements*.

Um trabalho importante focado na criação de repositórios para o MDD é o trabalho de Espinazo-Pagán e García-Molina (Espinazo-Pagán e García-Molina, 2010) que propõem um repositório capaz de gerenciar de maneira uniforme os metamodelos e modelos durante todo o processo de desenvolvimento. Os pesquisadores conceberam uma arquitetura para o desenvolvimento incremental e colaborativo dos (meta)modelos por meio de um sistema de controle de versão que registra as mudanças entre as revisões de uma forma alternativa

em relação a maneira tradicional baseada em delta, com o objetivo principal da melhoria do desempenho.

O trabalho de Bendix e Emanuelsson (Bendix e Emanuelsson, 2009) tem o seu foco voltado para os ambientes colaborativos (CDE) para o desenvolvimento utilizando MDD, no qual os pesquisadores identificam uma carência destes ambientes de apoio ao MDD colaborativo. Dessa forma, os autores enumeram os novos desafios trazidos com a adoção da colaboração no desenvolvimento de software e os requisitos para a implementação deste tipo de ambiente, de forma que os desenvolvedores possam trabalhar com o MDD de forma colaborativa e eficiente.

Entre os principais requisitos que um CDE deve possuir estão:

- O projeto deve ser submetido a um controle de versão, onde os artefatos serão identificados e versionados dentro de um repositório;
- É importante garantir que em alguns momentos apenas uma pessoa esteja realizando uma alteração, garantindo o seu trabalho isolado;
- É necessária a integração da alteração realizada localmente com a versão atual do repositório, no processo chamado de *merge*. Em alguns casos, a ferramenta pode realizar o *merge* automático, porém em caso de conflito é necessário a intervenção humana;
- Deve ser possível para o desenvolvedor verificar e validar o resultado de um processo de *merge*;
- O histórico das versões de um artefato deve estar disponível para consulta e também em alguns casos deve ser possível uma comparação entre as diferentes versões.

Especificamente para o MDD, os autores ainda identificaram outros requisitos:

a) A partir dos requisitos do sistema, um modelo de arquitetura deve ser definido para fornecer o contexto do sistema e descrever suas interfaces;

b) Fornecer um detalhamento do sistema por meio de um modelo de *design*;

c) Atualização de modelo sem *merge*: quando não existe nenhuma versão anterior daquele artefato. Dessa maneira o modelo é apenas inserido no repositório;

d) Atualização de modelo com *merge*: quando já existe uma versão do artefato porém é possível realizar o *merge* entre as versões de forma automática;

e) Atualização de modelo com *merge* complexo: quando já existe uma versão do artefato a qual não é possível realizar o *merge* entre as versões de forma automática, necessitando da intervenção humana para resolver os conflitos.

Diante do estudo desses trabalhos relacionados com a colaboração aliada ao MDD, de diferentes perspectivas, foi possível desenvolver a proposta de trabalho a ser discutida no Capítulo 5 contando com a descrição dos objetivos do projeto, bem como os requisitos que a ferramenta proposta deverá atender.

4.4 Considerações Finais

Neste capítulo foram apresentados os principais fundamentos sobre colaboração, os trabalhos relacionados de pesquisas sobre colaboração aliada ao MDD e as características principais de uma Wiki que permitem o trabalho de forma colaborativa. As diversas opções de Wiki existentes atualmente serão analisadas com mais detalhes a fim de verificar qual delas possui o melhor suporte para realizar a integração com a ferramenta proposta no próximo capítulo.

Capítulo

5

Proposta de Trabalho

Neste capítulo são apresentados: o problema de pesquisa do trabalho, os objetivos a serem alcançados, a metodologia para a realização do projeto e o cronograma de atividades previsto.

5.1 Problema de Pesquisa

O paradigma de computação orientada a serviços vem ganhando destaque como uma abordagem eficiente para integrar aplicações em ambientes distribuídos heterogêneos (Mayerl et al., 2005), o que torna a pesquisa de novas tecnologias e ferramentas nessa área muito importante a fim de facilitar esse processo de integração. Entre as novas tecnologias surgidas nesse contexto é possível destacar os Serviços Web RESTFul que graças a sua simplicidade tem evoluído muito nos últimos anos (Thies e Vossen, 2008).

A simplicidade alcançada pelos Serviços Web RESTFul é devido principalmente ao uso de padrões já bem estabelecidos na Web como por exemplo HTTP, XML, URI, entre outros, o que permitiu que o desenvolvimento desses serviços fosse bastante parecido com o desenvolvimento de uma aplicação web (Richardson e Ruby, 2007).

Com o amadurecimento da prática do Desenvolvimento Orientado a Modelos (MDD), surgindo como uma boa opção para melhorar a produtividade, reúso, portabilidade, entre outros aspectos importantes do processo de desenvolvimento de software, novas abordagens têm emergido na Engenharia Web especialmente na modelagem de aplicações Web, a

fim de tratar os problemas referentes à evolução e adaptação de sistemas web em relação às mudanças tecnológicas (Koch, 2006).

Existem abordagens nesse contexto como por exemplo a WebML, que possui uma extensão específica para a modelagem de Serviços Web RESTFul, porém não possui um metamodelo bem definido para tal modelagem. Já outras abordagens não possuem essa funcionalidade e portanto necessitam de uma alteração no seu metamodelo original a fim de implementá-la. Além disso, é necessário que existam ferramentas capazes de suportar esse tipo de desenvolvimento.

Embora existam ferramentas capazes de modelar aplicações Web, como a MagicUWE[1] e até mesmo os serviços Web RESTFul como a WebRatio[2], estas são soluções que não promovem o desenvolvimento colaborativo do MDD, por meio de mecanismos como a edição colaborativa. Além disso, o desenvolvedor é obrigado a instalar um software específico para a modelagem que muitas vezes não propicia a integração desejada com as demais ferramentas de desenvolvimento e comunicação.

A importância na pesquisa dessas ferramentas deve-se ao fato de que a prática da colaboração pode trazer benefícios ao processo de desenvolvimento de software usando MDD como destacado no Capítulo 4, bem como ocorre com o desenvolvimento de software convencional.

5.2 Objetivos

Diante dos problemas apresentados na seção anterior, os quais evidenciam a ausência de um metamodelo específico para a modelagem de Serviços Web RESTFul e também uma carência de ferramentas colaborativas para o desenvolvimento orientado a modelos capazes de modelar esses serviços, este projeto tem como objetivo principal a construção de uma ferramenta para:

a) Modelar os Serviços Web RESTFul e a composição destes em um processo de negócio por meio de um metamodelo bem definido, baseado no MOF;

b) Integrar esta ferramenta ao ambiente colaborativo de uma Wiki.

O uso de um metamodelo baseado no MOF, tem como principal motivação o uso da abordagem MDA para o desenvolvimento orientado a modelos, que possibilitará construir modelos como uma extensão da UML 2 e transformá-los usando linguagens para transformação de modelos como o QVT. Assim, como prova de conceito a ferramenta deverá ser capaz de criar diagramas UML propostos de acordo com o *Profile* UML definido para

[1] http://uwe.pst.ifi.lmu.de/toolMagicUWE.html
[2] http://www.webratio.com/

o metamodelo, além de permitir criar as regras de transformação de modelos utilizando uma notação visual, fácil de compreender pelo projetista.

Com a adoção do mecanismo de colaboração para o processo de desenvolvimento de software, novos requisitos que não existiam no desenvolvimento *standalone* surgiram conforme visto na Seção 4.3. Para este projeto será necessário utilizar um gerenciador de versões que suporte modelos e seja capaz de realizar o *merge* e apresentar a diferença entre as versões dos modelos ao desenvolvedor.

Além desses requisitos, um outro aspecto importante está relacionado às permissões de acesso aos modelos dentro da ferramenta, ou seja, é necessário que apenas usuários autorizados acessem os modelos. Dessa forma, se faz necessário também um mecanismo de autenticação e autorização para criação, consulta e edição de modelos na ferramenta.

Assim é possível definir os seguintes requisitos da ferramenta:

- A ferramenta deve ser capaz de modelar os Serviços Web RESTFul, bem como os processos de negócios por meio da composição destes, usando um *Profile* UML específico definido para o metamodelo;
- Será possível também modelar as transformações de modelo para modelo ou modelo para código-fonte por meio de uma notação visual de simples compreensão aos projetistas;
- Após modelados os serviços e/ou os processos de negócio, será possível testá-los por meio de uma interface simples oferecida pela ferramenta;
- Será gerado o código-fonte dos serviços na própria ferramenta por meio de uma interface na qual o usuário deverá informar o modelo a ser transformado e qual a transformação a ser aplicada. Dessa forma, é possível gerar um código que seja portável para diferentes plataformas em diferentes linguagens de programação;
- Para garantir o trabalho colaborativo de forma eficaz, será utilizado um gerenciador de versões, capaz de identificar e versionar todos os modelos. Será permitido ver o histórico de todas as revisões realizadas nos modelos, juntamente com os comentários sobre cada uma delas;
- A fim de garantir a segurança da ferramenta, será implementado um gerenciador de acessos e permissões, no qual os administradores poderão identificar quais usuários ou grupos de usuários podem acessar determinados modelos. Apenas usuários autorizados poderão realizar operações sobre os modelos.

Uma vez definidos os requisitos necessários para a implementação da ferramenta, na próxima seção é descrita a metodologia que será utilizada para o desenvolvimento de todas as atividades do projeto.

5.3 Metodologia

É previsto o desenvolvimento de oito atividades a fim de construir a ferramenta proposta na Seção 5.2 e integrá-la a um ambiente colaborativo web de tal forma que seja possível trabalhar nessa ferramenta de forma colaborativa e distruibuída. Para realizar o teste da ferramenta, a fim de avaliar sua eficiência e possíveis benefícios alcançados por ela, será realizado um estudo de caso descrito na Subseção 5.3.9.

As atividades a serem realizadas durante o projeto são descritas nas próximas subseções.

5.3.1 Definição do metamodelo

Esta atividade tem por finalidade definir um metamodelo que seja capaz de representar as características estáticas e comportamentais dos Serviços Web RESTFul, a fim de que possam ser criados modelos representativos desses serviços. Esse metamodelo portanto precisa ser suficientemente completo para que os modelos gerados a partir dele possam sofrer transformações automatizadas que gerem o código-fonte. Além disso, o metamodelo deve ser também capaz de modelar um processo de negócio por meio da composição e orquestação de diversos Serviços Web RESTFul.

Para alcançar este objetivo será realizado o estudo das características pertinentes aos Serviços Web RESTFul, bem como o estudo de possíveis extensões a serem aplicadas a linguagem UML a fim de contemplar o metamodelo a ser proposto.

Uma vez realizado estes estudos e de posse das características importantes para a definição do metamodelo, será proposto um *Profile* UML que represente esse metamodelo.

5.3.2 Estudo comparativo das linguagens para transformação de modelos

Depois de definido o metamodelo, é preciso escolher entre as diversas linguagens para transformação de modelos existentes qual delas oferece um melhor suporte para trabalhar com o metamodelo proposto. O ideal é que essa linguagem trabalhe com o padrão MOF e que possua uma linguagem visual simples e não apenas textual. Assim, o próprio projetista poderá modelar as transformações desejadas para as diferentes plataformas escolhidas.

Caso a linguagem escolhida não possua uma linguagem visual ou ela seja muito complexa, deverá ser proposta uma notação visual simples capaz de representar as transformações possíveis de serem realizadas de acordo com o metamodelo definido.

5.3.3 Elaboração das interfaces de modelagem

Uma vez definido o metamodelo e a linguagem de transformação é necessário realizar a criação das interfaces com o usuário para que ele possa modelar os seus serviços, processos de negócio e também as transformações a serem realizadas nos modelos. Assim, a ferramenta contará com 3 interfaces de modelagem distintas: modelagem de recursos, de processos de negócio e de transformações de modelo.

Neste momento deverá ser definido o layout dos componentes de cada interface e também a melhor forma de interação que o usuário poderá ter com esses componentes. A intenção é desenhar interfaces simples e intuitivas, de modo a facilitar o trabalho do projetista no momento da modelagem.

Um grande desafio encontrado nesta atividade é conseguir produzir um layout para uma interface Web que seja familiar às ferramentas CASE já existentes para modelagem.

5.3.4 Elaboração da interface para testes dos serviços e processos de negócio

Além de modelar serviços e processos de negócios, a ferramenta **RestMDD** permitirá também o teste desses serviços, de modo a validar o seu funcionamento correto.

Para isso, da mesma forma que na etapa anterior, será proposta uma interface simples na qual o desenvolvedor poderá realizar o teste dos seus serviços ou processos de negócio, antes mesmo de gerar o código-fonte correspondente.

5.3.5 Componente responsável pela transformação de artefatos

Conforme citado no Capítulo 3, um dos elementos responsáveis pela automatização do processo de transformação de modelos dentro do MDD é o chamado “Mecanismo para executar transformações”. Esse mecanismo é responsável por executar as transformações propostas, além de manter a rastreabilidade entre o modelo e o artefato produzido pela transformação, de modo que seja capaz de identificar a origem de cada elemento gerado.

Assim, esta fase do projeto consiste em elaborar um mecanismo capaz de aplicar as transformações modeladas pelos projetistas sobre os modelos já existentes, gerando novos modelos ou mesmo código-fonte. Será criada também uma interface em que o usuário poderá escolher qual transformação deseja realizar e ainda visualizar o resultado obtido.

5.3.6 Gerenciador de acessos e permissões

Para garantir a segurança no acesso aos modelos será implementado na ferramenta um gerenciador de acessos e permissões, que será responsável por autenticar os usuários e autorizar a criação, consulta e edição de modelos.

Esse gerenciador obrigará todos os usuários, ao acessar a ferramenta, se autenticarem por meio de uma senha. Uma vez que o acesso for liberado e o usuário puder acessar a ferramenta, ao tentar realizar qualquer operação sobre um modelo, o gerenciador irá verificar se este usuário possui a permissão para realizar tal ação, autorizando-o ou não a realizá-la.

Quem definirá quais usuários ou grupos de usuários podem trabalhar em um modelo é o seu criador ou um usuário com perfil de administrador.

5.3.7 Gerenciador de versões

Um repositório de controle de versões é uma ferramenta importante para gerenciar as mudanças no software e manter o histórico durante o desenvolvimento, a fim de facilitar a evolução do sistema. Isso é possível graças ao armazenamento de múltiplas versões dos artefatos dentro desse repositório, inclusive adicionando comentários a cada revisão.

Como mencionado no Capítulo 4, essa é uma ferramenta importante dentro do desenvolvimento colaborativo e portanto será incorporada ao projeto da ferramenta RestMDD. Para isso, será necessário realizar um estudo das ferramentas de versionamento capazes de suportar modelos no padrão MOF e quais as maneiras de incorporá-la à ferramenta RestMDD.

É importante ainda que a ferramenta de controle de versões disponibilize mecanismos para realizar o *merge* de artefatos e comparação de revisões.

5.3.8 Mecanismo de Integração com o ambiente colaborativo

Uma vez a ferramenta construída, ela deverá ser integrada de maneira transparente ao ambiente colaborativo de uma Wiki.

Para que isso seja possível, a ferramenta deve prover mecanismos para que suas interfaces sejam incorporadas dentro das páginas de uma Wiki e seus serviços sejam endereçáveis por meio de URLs e possam ser invocados com uma simples requisição HTTP. Dessa forma, a integração entre a ferramenta RestMDD e a Wiki será realizada da forma menos intrusiva possível.

5.3.9 Estudo de Caso

A fim de avaliar os resultados alcançados com o desenvolvimento da ferramenta, será conduzido um estudo de caso, onde serão realizados experimentos com usuários especialistas em desenvolvimento de software. Esses experimentos consistem na observação do trabalho dos usuários no desenvolvimento de alguns serviços RESTFul com a ferramenta RestMDD em relação a outras ferramentas de modelagem do tipo *standalone* e ainda em relação ao desenvolvimento de software convencional, por meio de codificação. Dessa observação será possível comparar as diferentes formas de interação do usuário com as interfaces, a facilidade no aprendizado, a utilidade das ferramentas em prol da produtividade e o desempenho apresentado quanto ao tempo de resposta de cada uma.

Essa avaliação tem como finalidade mostrar as possíveis vantagens e desvantagens observadas no processo de desenvolvimento de software usando MDD com a adoção dessa ferramenta. Será possível também definir a utilidade da colaboração dentro do MDD, além de investigar quais os pontos de melhorias a serem implementados na ferramenta em trabalhos futuros.

5.4 Cronograma

O cronograma de execução das atividades é apresentado na Tabela 5.1. O projeto tem duração de 24 meses e a tabela apresenta a definição de atividades conforme abaixo:

1. Obtenção de créditos
2. Exame de proficiência língua inglesa
3. Revisão bibliográfica
4. Exame de qualificação
5. Definição do metamodelo
6. Estudo comparativo das linguagens para transformação de modelos
7. Elaboração das interfaces de modelagem
8. Elaboração da interface para testes dos serviços e processos de negócio
9. Componente responsável pela transformação de artefatos
10. Gerenciador de acessos e permissões
11. Gerenciador de versões

12. Mecanismo de Integração com o ambiente colaborativo
13. Estudo de Caso
14. Redação de Artigos
15. Redação da Dissertação
16. Defesa

Atividades	2010					2011						2012		
	Mar/ Abr	Mai/ Jun	Jul/ Ago	Set/ Out	Nov/ Dez	Jan/ Fev	Mar/ Abr	Mai/ Jun	Jul/ Ago	Set/ Out	Nov/ Dez	Jan/ Fev	Mar/ Abr	Mai/ Jun
1	•	•	•	•	•		•	•						
2				•										
3	•	•	•	•	•	•	•	•	•	•	•	•		
4						•	•							
5								•						
6									•					
7									•	•				
8										•				
9											•			
10											•			
11												•		
12												•		
13													•	
14										•	•	•		
15													•	•
16														•

Tabela 5.1: Cronograma

5.5 Resultados Esperados

Os objetivos do projeto e a metodologia aplicada já foram identificados nas seções anteriores. Nesta seção serão apresentados formalmente os resultados que este projeto pretende alcançar:

- Contribuir com a pesquisa do MDD dentro do domínio de Engenharia Web
- Elaborar um metamodelo capaz de representar Serviços Web RESTFul usando o padrão MOF
- Contribuir com a pesquisa do tema colaboração dentro do contexto de MDD
- Identificar os benefícios alcançados dentro do processo de desenvolvimento de software com a adoção de práticas colaborativas associadas ao MDD

- Identificar se ocorre aumento na produtividade com o uso de colaboração aliada ao MDD
- Construir a ferramenta RestMDD e validar seus pontos fortes e fracos, além de propor melhorias futuras

Vislumbra-se ainda a publicação de artigos científicos sobre esses temas abordados no projeto.

Espera-se que com o desenvolvimento da ferramenta RestMDD e a realização do estudo de caso, seja comprovada a eficiência deste projeto de pesquisa.

Referências

Aalders, R. *The IT outsourcing guide.* Wiley, 2001.

Abeti, L.; Ciancarini, P.; Moretti, R. Wiki-based requirements management for business process reengineering. In: *Wikis for Software Engineering, 2009. WIKIS4SE '09. ICSE Workshop on*, 2009, p. 14 –24.

Bendix, L.; Emanuelsson, P. Collaborative work with software models - industrial experience and requirements. In: *Model-Based Systems Engineering, 2009. MBSE '09. International Conference on*, 2009, p. 60 –68.

Booch, G.; Brown, A. W. Collaborative development environments. v. 59 de *Advances in Computers*, Elsevier, p. 1 – 27, 2003.

Ceri, S.; Brambilla, M.; Fraternali, P. The history of webml lessons learned from 10 years of model-driven development of web applications. In: Borgida, A.; Chaudhri, V.; Giorgini, P.; Yu, E., eds. *Conceptual Modeling: Foundations and Applications*, v. 5600 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 273–292, 2009.

Ceri, S.; Fraternali, P.; Bongio, A.; Brambilla, M.; Comai, S.; Matera, M. *Designing data-intensive web applications.* Morgan Kaufmann, 2002.

Crockford, D. Json: The fat-free alternative to xml. In: *Proceedings of XML 2006*, 2006.
Disponível em `http://www.json.org/fatfree.html`

CruiseControl Cruise control web site. 2010.
Disponível em `http://cruisecontrol.sourceforge.net/`

Cullen, S.; Willcocks, L. *Intelligent IT outsourcing: Eight building blocks to success.* Butterworth-Heinemann, 2003.

Daniel, S.; Przemyslaw, T.; Lars, H. Integrating information systems using web oriented integration architecture and restful web services. In: *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, Washington, DC, USA: IEEE Computer Society, 2010, p. 598–605 (*SERVICES '10*,).

van Deursen, A.; Klint, P.; Visser, J. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, v. 35, p. 26–36, 2000.

ECLIPSE Atlas transformation language specification v3.1.0. 2010.
Disponível em `http://www.eclipse.org/atl/`

Erl, T. *Service-oriented architecture: Concepts, technology, and design.* Prentice Hall, 2005.

Espinazo-Pagán, J.; García-Molina, J. A homogeneous repository for collaborative mdd. In: *Proceedings of the 1st International Workshop on Model Comparison in Practice*, IWMCP '10, New York, NY, USA: ACM, 2010, p. 56–65 (*IWMCP '10*,).

Fielding, R. T. *Architectural styles and design of network-based software architectures.* Tese de Doutoramento, UC Irvine, 2000.

Gall, N. Tutorial: Web-oriented architecture: Putting the web back in web services. 2008.
Disponível em `http://www.gartner.com/DisplayDocument?doc_cd=162022`

Hailpern, B.; Tarr, P. Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, v. 45, n. 3, p. 451 –461, 2006.

Hohpe, G.; Woolf, B. *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional, 2003.

Jang, S.; Green, T. M. Best practices on delivering a wiki collaborative solution for enterprise applications. In: *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, 2006, p. 1 –9.

Kimball, R.; Caserta, J. *The data warehouse etl toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data.* Wiley, 2004.

Kleppe, A.; Warmer, J.; Bast, W. *Mda explained: The model driven architecture: Practice and promise.* Addison-Wesley Professional, 2003.

Koch, N. Transformation techniques in the model-driven development process of uwe. In: *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA: ACM, 2006 (*ICWE '06*,).

Koch, N.; Kraus, A. Towards a common metamodel for the development of web applications. In: *Proceedings of the 2003 international conference on Web engineering*, ICWE'03, Berlin, Heidelberg: Springer-Verlag, 2003, p. 497–506 (*ICWE'03*,).

Kroib, C.; Koch, N. Uwe metamodel and profile: User guide and reference. technical report 0802, 2008. 2008.
Disponível em `http://uwe.pst.ifi.lmu.de/download/UWE-Metamodel-Reference.pdf`

Lanubile, F. Software engineering. cáp. Collaboration in Distributed Software Development, Berlin, Heidelberg: Springer-Verlag, p. 174–193, 2009.

Linthicum, D. S. *Enterprise application integration*. Addison-Wesley Professional, 1999.

Lucrédio, D. *Uma abordagem orientada a modelos para reutilização de software*. Tese de Doutoramento, Universidade de São Paulo, 2009.

Manolescu, I.; Brambilla, M.; Ceri, S.; Comai, S.; Fraternali, P. Model-driven design and deployment of service-enabled web applications. *ACM Trans. Internet Technol.*, v. 5, p. 439–479, 2005.

Mayerl, C.; Vogel, T.; Abeck, S. Soa-based integration of it service management applications. In: *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, Washington, DC, USA: IEEE Computer Society, 2005, p. 785–786 (*ICWS '05*,).

McQuay, W. Distributed collaborative environments for systems engineering. In: *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, 2004, p. 9.D.3 – 91–10 Vol.2.

Mehta, N. *Choosing an open source cms: Beginner's guide*. Packt Publishing, 2009.

Musser, J. Web 2.0 principles and best practices. 2007.
Disponível em `http://oreilly.com/catalog/web2report/chapter/web20_report_excerpt.pdf`

OMG Mda guide version 1.0.1. 2003.
Disponível em `http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf`

OMG Meta object facility (mof) 2.0 core specification. 2004a.
Disponível em `http://www.omg.org/cgi-bin/doc?ptc/03-10-04.pdf`

OMG Uml profile for enterprise application integration (eai). 2004b.
Disponível em http://www.omg.org/spec/EAI/1.0/

OMG Mof 2.0/xmi mapping, v2.1.1. 2007.
Disponível em http://www.omg.org/spec/XMI/2.1.1/

OMG Uml profile for modeling and analysis of real-time and embedded systems. 2009.
Disponível em http://www.omg.org/spec/MARTE/1.0/

OMG Query view transformation specification v1.1. 2011.
Disponível em http://www.omg.org/spec/QVT/1.1/

O'Reilly, T. What is web 2.0. 2005.
Disponível em http://oreilly.com/web2/archive/what-is-web-20.html

Pautasso, C. Bpel for rest. In: *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, Berlin, Heidelberg: Springer-Verlag, 2008, p. 278–293 (*BPM '08*,).

Pautasso, C. Composing restful services with jopera. In: *Proceedings of the 8th International Conference on Software Composition*, SC '09, Berlin, Heidelberg: Springer-Verlag, 2009, p. 142–159 (*SC '09*,).
Disponível em http://dx.doi.org/10.1007/978-3-642-02655-3_11

Pautasso, C.; Zimmermann, O.; Leymann, F. Restful web services vs. "big"' web services: making the right architectural decision. In: *Proceeding of the 17th international conference on World Wide Web*, WWW '08, New York, NY, USA: ACM, 2008, p. 805–814 (*WWW '08*,).

Rama, J.; Bishop, J. A survey and comparison of cscw groupware applications. In: *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, SAICSIT '06, , Republic of South Africa: South African Institute for Computer Scientists and Information Technologists, 2006, p. 198–205 (*SAICSIT '06*,).

Rauf, I.; Ruokonen, A.; Systa, T.; Porres, I. Modeling a composite restful web service with uml. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, New York, NY, USA: ACM, 2010, p. 253–260 (*ECSA '10*,).

Ribaric, M.; Gasevic, D.; Milanovic, M.; Giurca, A.; Lukichev, S.; Wagner, G. Model-driven engineering of rules for web services. In: *Generative and Transformational Techniques in Software Engineering II*, v. 5235 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 377–395, 2008.

Richardson, L.; Ruby, S. *Restful web services*. O'Reilly Media, 2007.

Schauerhuber, A.; Wimmer, M.; Kapsammer, E. Bridging existing web modeling languages to model-driven engineering: a metamodel for webml. In: *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA: ACM, 2006 (*ICWE '06*,).

Sherif, K.; Appan, R.; Lin, Z. Resources and incentives for the adoption of systematic software reuse. *International Journal of Information Management*, v. 26, n. 1, p. 70 – 80, 2006.

Sriplakich, P.; Blanc, X.; Gervais, M.-P. Supporting collaborative development in an open mda environment. In: *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, 2006, p. 244 –253.

Stahl, T.; Voelter, M. *Model-driven software development: Technology, engineering, management*. Wiley, 2006.

Technology, A. Model driven architecture (mda). 2006. Disponível em `http://technology.amis.nl/blog/1178/model-driven-architecture-mda`

Teppola, S.; Parviainen, P.; Takalo, J. Challenges in deployment of model driven development. In: *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, 2009, p. 15 –20.

Tetard, F.; Patokorpi, E.; Packalen, K. Using wikis to support constructivist learning: A case study in university education settings. In: *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, 2009, p. 1 –10.

Thies, G.; Vossen, G. Web-oriented architectures: On the impact of web 2.0 on service-oriented architectures. In: *Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, Washington, DC, USA: IEEE Computer Society, 2008, p. 1075–1082.

Thies, G.; Vossen, G. Modelling web-oriented architectures. In: *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling - Volume 96*, APCCM '09, Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2009, p. 97–106 (*APCCM '09*,).

Thomas, D.; Barry, B. M. Model driven development: the case for domain oriented programming. In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, New York, NY, USA: ACM, 2003, p. 2–7 (*OOPSLA '03*,).

WebRatio Getting started with web services. 2009. Disponível em `http://wiki.webratio.com/index.php/Getting_started_with_Web_Services`

Yu, Q.; Liu, X.; Bouguettaya, A.; Medjahed, B. Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, v. 17, p. 537–572, 2008.