# RestMDD: Ambiente colaborativo para o apoio no desenvolvimento de serviços Web RESTful

***Robson Vinícius Vieira Sanchez***

# RestMDD: Ambiente colaborativo para o apoio no desenvolvimento de serviços Web RESTful

**Robson Vinícius Vieira Sanchez**

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

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

**USP – São Carlos**
**Novembro de 2013**

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

S211r

Sanchez, Robson Vinícius Vieira
RestMDD: Ambiente colaborativo para o apoio no desenvolvimento de serviços Web RESTful / Robson Vinícius Vieira Sanchez; orientadora Renata Pontin de Mattos Fortes. -- São Carlos, 2013.
111 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2013.

1. REST. 2. Serviços Web. 3. Desenvolvimento orientado a modelos. 4. Colaboração. 5. Engenharia Web. I. Fortes, Renata Pontin de Mattos, orient. II. Título.

*"É melhor atirar-se à luta em busca de dias melhores, mesmo correndo o risco de perder tudo, do que permanecer estático, como os pobres de espírito, que não lutam, mas também não vencem, que não conhecem a dor da derrota, nem a glória de ressurgir dos escombros. Esses pobres de espírito, ao final de sua jornada na Terra não agradecem a Deus por terem vivido, mas desculpam-se ante Ele, por terem apenas passado pela vida."*

*Bob Marley*

*Dedico este trabalho à minha esposa Luciane, à minha mãe Beatriz, ao meu pai Adauto e ao meu irmão Otávio que estão ao meu lado em todos os momentos de minha vida e contribuíram de uma forma muito especial para o meu crescimento pessoal e profissional. Amo vocês!*

# Agradecimentos

Agradeço primeiramente a Deus que está sempre do meu lado.

Aos meus pais Adauto e Beatriz e ao meu irmão Otávio, que durante toda minha vida me apoiaram e incentivaram na realização dos meus objetivos. Serei eternamente grato aos seus ensinamentos e ao amor que recebo de vocês.

A minha esposa Luciane que compartilha comigo os momentos difíceis e alegres de nossas vidas sempre com muito amor e companheirismo, não permitindo que eu desista dos meus objetivos nos momentos de dificuldade e orgulhando-se com as minhas conquistas.

A toda minha família que sempre torceu pelo meu sucesso e felicidade.

A todos os meus amigos pelo companheirismo e por ótimos momentos que passamos juntos. Sem vocês essa caminhada não seria possível. Obrigado a todos.

A todos do Laboratório do Intermídia e do grupo de pesquisa, pelo apoio, amizade e a troca de experiências que ajudaram muito no desenvolvimento deste projeto. Em especial, aos meus amigos: Ricardo Ramos de Oliveira, David Fernandes Neto e Willian Massami Watanabe por todo apoio e contribuições que foram fundamentais para o desenvolvimento do trabalho.

A todos meus colegas de trabalho que de alguma maneira influenciaram no meu crescimento pessoal e profissional durante esses anos de convívio.

A minha orientadora, a professora Dr$^{a}$. Renata Pontin de Mattos Fortes, pelo seu carinho, profissionalismo, ética, paciência e dedicação para comigo durante esses anos que tivemos o prazer de trabalhar juntos neste projeto de mestrado.

Por fim, agradeço ao ICMC-USP pela oportunidade de realização deste trabalho.

# 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 Architecture*) foi proposto a fim de garantir uma maneira simples de conectar os componentes de software dinamicamente. 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 um ambiente colaborativo de apoio ao desenvolvimento de serviços Web RESTful 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. Foi realizado ainda um experimento a fim de comprovar a eficiência do ambiente colaborativo e benefícios alcançados por ele.

**Palavras Chaves:** Computação orientada a Serviços, Arquitetura orientada a Web, Serviços Web RESTful, Engenharia Web, Desenvolvimento orientado a modelos, Linguagem específica de domínio, Ambientes colaborativos

# Abstract

*In the last decade the paradigm of Service Oriented Computing has gained more attention in industry and academia in order to solve the problem of lack of communication between different information systems present within a corporate environment. Due to recent advances in Web 2.0 a new architectural style called Web Oriented Architecture is proposed to ensure a simple way to connect software components dynamically. This style has as a principle the use of RESTful Web Services in order to achieve a uniform interface simple and functional. This paper presents a collaborative environment to support the development of RESTful Web Services using the Model Driven Development paradigm. It is intended to show the benefits of MDD applied to Web Engineering and also the advantages presented by collaboration in this scenario. It was also performed a experiment to prove the efficiency of the collaborative environment and benefits achieved by the same.*

***Keywords:*** *Service Oriented Computing, Web Oriented Architecture, RESTful Web Services, Web Engineering, Model Driven Development, Domain Specific Language, Collaborative environments*

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Siglas

**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

**CRUD** *Create, Retrieve, Update and Delete* / Criar, Recuperar, Atualizar e Excluir

**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

**IETF** *Internet Engineering Task Force* / Força-Tarefa de Engenharia na Internet

**IoC** *Inversion of Control* / Inversão de controle

**JavaEE** *Java Enterprise Edition*

**JAX-RS** *Java API for RESTful Web Services* / API Java para Serviços Web RESTful

**JSON** *Javascript Object Notation* / Notação de objetos Javascript

**MDA** *Model Driven Architeture* / Arquitetura Orientada a Modelos

**MDD** *Model Driven Development* / Desenvolvimento Orientado a Modelos

**MER** Modelo Entidade-Relacionamento

**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

**RestMDD** *RESTful Model-Driven Development* / Desenvolvimento Orientado a Modelos RESTful

**RestML** *RESTful Modeling Language* / Linguagem de Modelagem RESTful

**REST** *Representational State Transfer* / Transferência de Estado Representacional

**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

**W3C** *World Wide Web Consortium* / Consórcio *World Wide Web*

**WAR** *Web Archive* / Arquivo Web

**WebML** *Web Modeling Language* / Linguagem de Modelagem Web

**WOA** *Web Oriented Architecture* / Arquitetura Orientada a 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

Desde o início do século, observa-se que 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 (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 área de pesquisa mais popular dentro de SOC é dedicada aos avanços da Arquitetura Orientada a Serviços (SOA - *Service Oriented Architecture*), 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 como por exemplo, *Simple Object Access Protocol* (SOAP), *Web Services Description Language* (WSDL), *Universival Description Discovery and Integration* (UDDI), entre outros tornam sua construção mais complicada devido ao "excesso de padrões" (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 software de modo dinâmico, tal que os objetivos originais da SOA como flexibilidade, reutilização e a redução da complexidade possam ser atingidos de maneira mais simples (Thies e Vossen, 2008). Essa forma alternativa de construção de ambientes orientados a serviços foi denominada por Nicholas Gall (Gall, 2008) "Arquitetura Orientada a Web" (WOA - *Web Oriented Architecture*).

A arquitetura orientada a Web é considerada um 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 *Hypertext Transfer Protocol* (HTTP), *eXtensible Markup Language* (XML) e *Javascript Object Notation* (JSON), como ilustrado na Figura 1.1. O objetivo mais notável de WOA é permitir o desenvolvimento de sistemas descentralizados 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). Para alcançar esses 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 Soft-*

*ware 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 quatro 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).

Existem quatro princípios que definem o estilo arquitetural 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 Identificadores Uniformes de Recursos (URI - *Uniform Resource Identifier*), 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 é utilizado para criar 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, indicar o formato da representação, realizar a autenticação e controle de acesso, entre outros.

- **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*, agrupamento 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 simples para a troca de mensagens 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 (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 (Teppola et al., 2009), 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 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 apresenta benefícios quando utilizado de modo colaborativo.

Embora existam ferramentas capazes de modelar Serviços Web RESTful, estas são soluções que não promovem o desenvolvimento colaborativo do MDD, por meio de mecanismos como o compartilhamento de espaço de trabalho, projetos e modelos. 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.

## Objetivo

O objetivo deste trabalho de mestrado é definir uma linguagem específica de domínio para a construção de Serviços Web RESTful utilizando MDD e implementar o ambiente de

desenvolvimento colaborativo **RestMDD - RESTful Model-Driven Development**, que apoie a construção desses serviços.

## Estrutura da Monografia

Este trabalho está organizado em 8 capítulos. Este capítulo introduziu a área de pesquisa apresentando a motivação necessária para a realização deste trabalho. O Capítulo 2 apresenta um estudo sobre o estilo arquitetural 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 é apresentada a linguagem específica de domínio RestML que foi proposta para o desenvolvimento de Serviços Web Restful usando MDD. No Capítulo 6 é apresentado o ambiente de desenvolvimento colaborativo RestMDD e suas principais ferramentas. No Capítulo 7 é descrito o experimento realizado para validação do ambiente RestMDD. O Capítulo 8 apresenta as conclusões e as contribuições deste trabalho no decorrer deste projeto. Por fim, são listadas as referências bibliográficas utilizadas ao longo deste documento e os documentos apêndices.

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 padrões 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 é apresentada a definição do estilo arquitetural REST, os principais conceitos da Web 2.0 e Serviços Web e finalmente a definição de Serviços Web RESTful e quais suas vantagens e desvantagens.

## 2.1 Estilo Arquitetural 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. 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.

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 é necessário olhar apenas uma requisição para entender qual a natureza da funcionalidade;

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

Entre as desvantagens pode-se citar:

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

*b*) Diminui o controle do servidor, pois depende 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 implicita 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 o uso da hipermídia como o motor para o estado da aplicação.

Por fim, é adicionada a restrição de sistema em camadas, 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 por camadas hierárquicas, restringindo o comportamento do componente de tal forma que cada componente não pode "ver" além da camada com a qual está 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 componentes intermediários também podem ser usados para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga entre os serviços.

## 2.2 Elementos do Estilo Arquitetural REST

O elemento chave na abstração do estilo arquitetural 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, estão os servidores Web.

3. **Componentes intermediários:** entre os componentes intermediários, estão 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 e 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 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**
   Cria-se uma arquitetura de participação, de forma que o software se torne melhor a medida que as pessoas o utilizem.

2. **Inovação na montagem**
   Constrói-se plataformas para promover a inovação, na qual a mistura de serviços e dados crie novas oportunidades e mercados.

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

4. **Modelos leves e escalabilidade econômica**
   Utilize-se de 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 dar suporte ao 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.

[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 Serviços Web RESTful

Os Serviços Web RESTful surgiram como uma forma de simplificar o desenvolvimento de serviços Web, seguindo os princípios do estilo arquitetural 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 padrões presente nos serviços Web SOAP.

### 2.4.1 Propriedades dos Serviços Web RESTful

Por seguir os princípios do estilo arquitetural 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 é descrito como as tecnologias Web tornam possíveis as implementações dessas propriedades.

#### 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 por meio 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 *hyperlinks* para outros recursos. Dessa forma, esses documentos podem conter URIs de outros recursos da aplicação que de alguma forma estão conectados ao recurso que está sendo acessado, seguindo um dos princípios do estilo arquitetural REST que enuncia a "hipermídia como motor do estado da aplicação" (Fielding, 2000).

Essa qualidade de permitir a conexão por meio de *hyperlinks* é chamada de conectividade. Na Figura 2.1, os recursos estão representados por retângulos e os *hyperlinks* por setas, e é possível notar as diferenças entre os serviços que são bem-conectados (Serviço B) e os não-conectados (Serviço A).

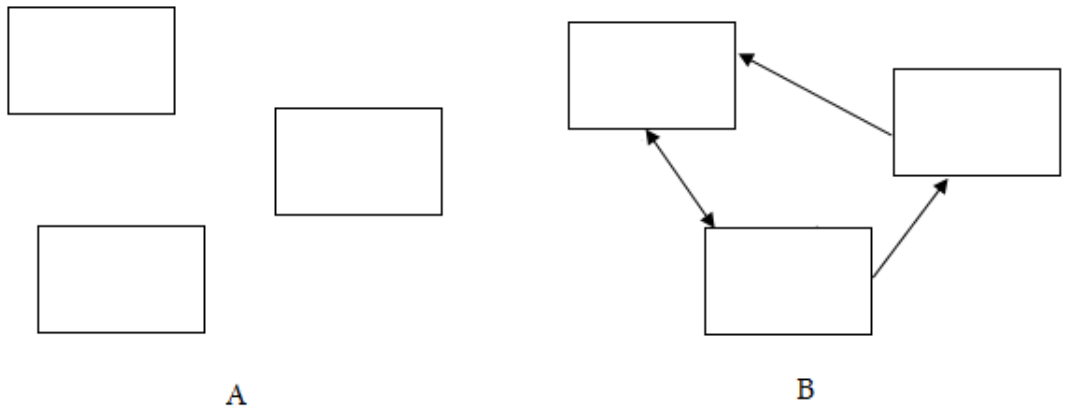


**Figura 2.1:** Conectividade - Adaptado de (Richardson e Ruby, 2007)

#### 2.4.1.4 Interface Uniforme

A interface uniforme requerida pelos serviços RESTful é alcançada por meio dos quatro métodos básicos do protocolo HTTP para as quatro 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, a exclusão de um recurso de forma indesejada. Ao utilizar o 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 pelo estilo arquitetural REST estão (Pautasso et al., 2008; Richardson e Ruby, 2007):

- **Simplicidade no desenvolvimento:** utiliza os padrões da Web, evitando o excesso de padrões;
- **Escalabilidade:** permite suporte a *cache*, *clusterização* e balanceamento de carga;
- **Possibilidade de múltiplas representações para um mesmo recurso:** permite a utilização de diferentes formatos de dados, o que torna mais portável entre diferentes plataformas;
- **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 *hyperlinks*;
- **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 Considerações Finais

Neste capítulo foram apresentados os conceitos do estilo arquitetural REST, da Web 2.0 e Serviços Web, bem como as características principais dos Serviços Web RESTful. Estas características foram essenciais para a proposta da linguagem RestML que será discutida no capítulo 5.

Capítulo

3

# MDD

O Desenvolvimento Orientado a Modelos é uma área da Engenharia de Software que vem crescendo nos últimos anos devido aos consideráveis benefícios que pode proporcionar em termos de produtividade, qualidade e reúso. Neste capítulo são 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), elevando dessa forma 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 são 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) estão:

- **Produtividade:** o aumento da produtividade no processo de desenvolvimento de software deve-se principalmente à 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, a 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, 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, 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, esses modelos e regras de mapeamento 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.

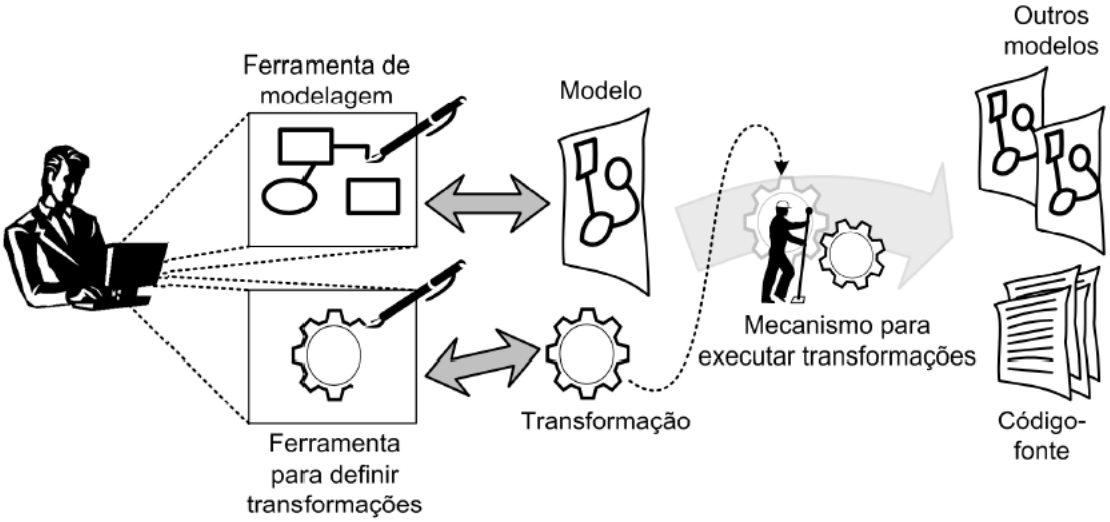


**Figura 3.1:** Principais Elementos do MDD (Lucrédio, 2009)

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, geralmente 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, ela fornece soluções melhores para um domínio particular, quando comparadaa às linguagens de propósito geral, pois permite que os *experts* no domínio entendam com mais facilidade a solução proposta (van Deursen et al., 2000).

Entre as 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 corresponde a 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 Model Driven Architeture

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

[1] UML - Unified Modeling Language
[2] http://www.omg.org/

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 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 é possível observar na Figura 3.2. 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).

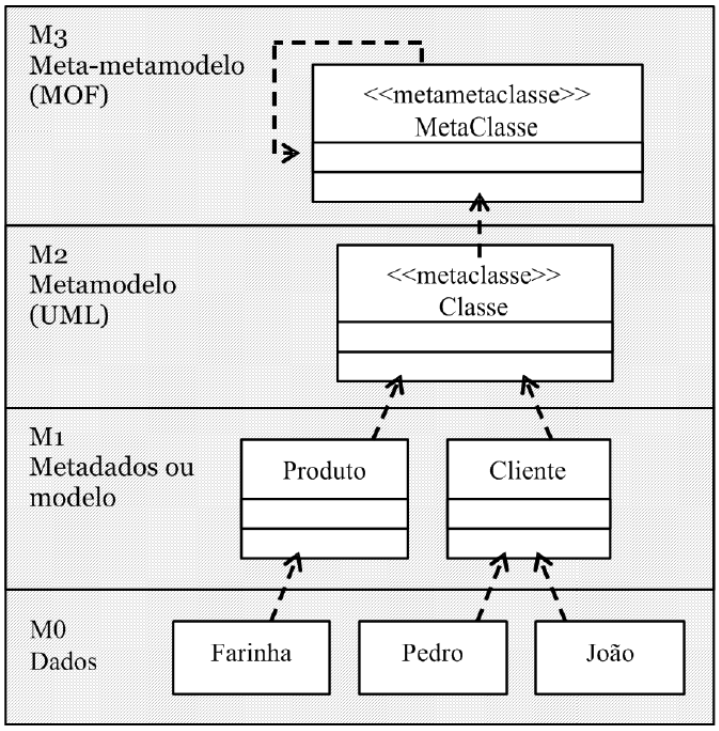


**Figura 3.2:** Arquitetura de Metamodelagem (Lucrédio, 2009)

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 transformações 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:

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

A Figura 3.3 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.

## 3.3 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

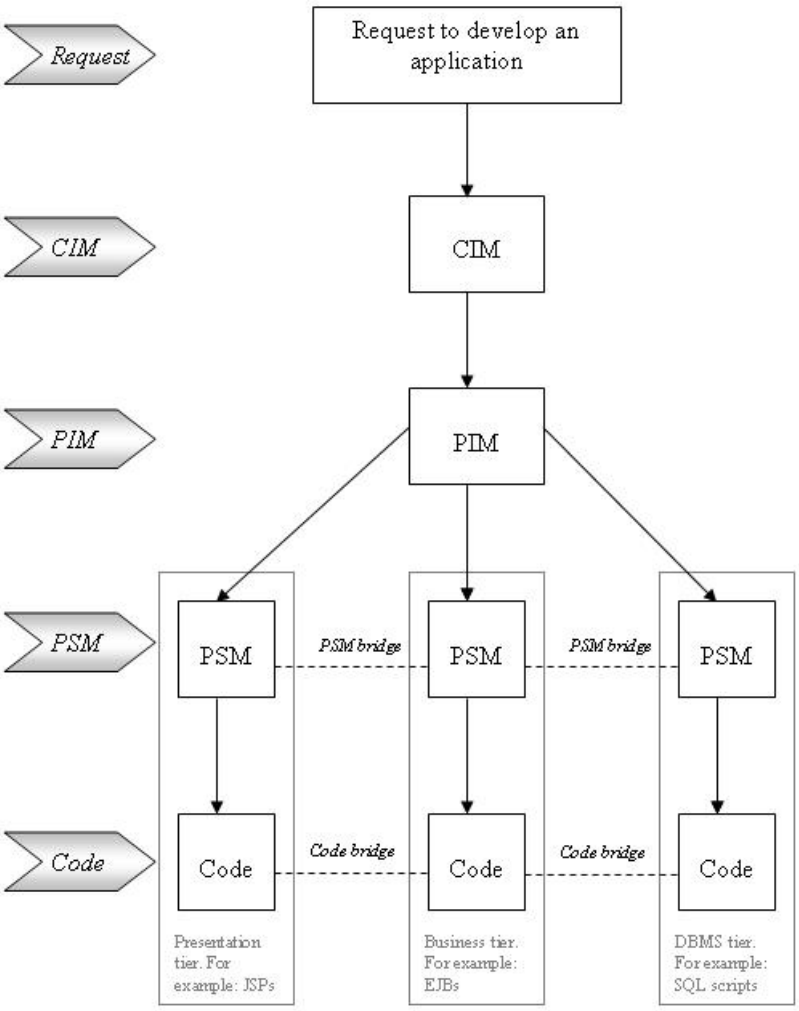


**Figura 3.3:** Ciclo de vida MDA (Technology, 2006)

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 são apresentadas duas abordagens que vêm ganhando destaque: a WebML e a UWE.

### 3.3.1 WebML

WebML é o acrônimo de *Web Modeling 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 (MER) 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 observado na Figura 3.4, a qual mostra um exemplo de um site de empréstimos, em que o usuário pode navegar por um conjunto de três páginas partindo da Home Page, além da possibilidade de inclusão de uma nova proposta.

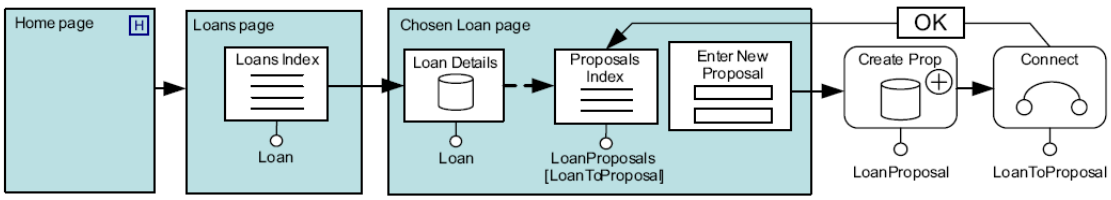


**Figura 3.4:** Modelo hipertexto WebML (Ceri et al., 2009)

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. A Figura 3.5 ilustra 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 existe 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.

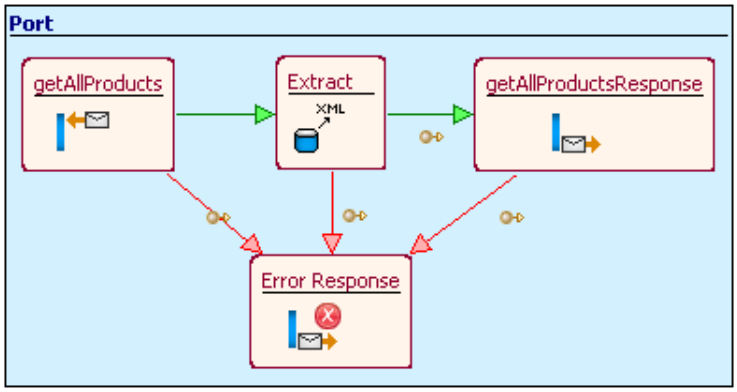


**Figura 3.5:** Exemplo de Serviço Web modelado com os componentes da WebML (WebRatio, 2009)

Para dar suporte ao desenvolvimento dessas aplicações usando a WebML é disponibilizada uma ferramenta CASE (*Computer-Aided Software Engineering*) chamada WebRatio (WebRatio, 2013), 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 há 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 *Document Type Definition* (DTD), 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 (*eXtensible Stylesheet Language for Transformation*);

- A prototipação dos modelos é realizada através da ferramenta WebRatio, que trabalha de forma *standalone* dificultando o desenvolvimento colaborativo.

### 3.3.2 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 MOF (*Meta-Object Facility*), usado para a definição da UML, que é definido pela OMG e portanto segue a abordagem do desenvolvimento orientado a modelos MDA (Koch, 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

elementos 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).

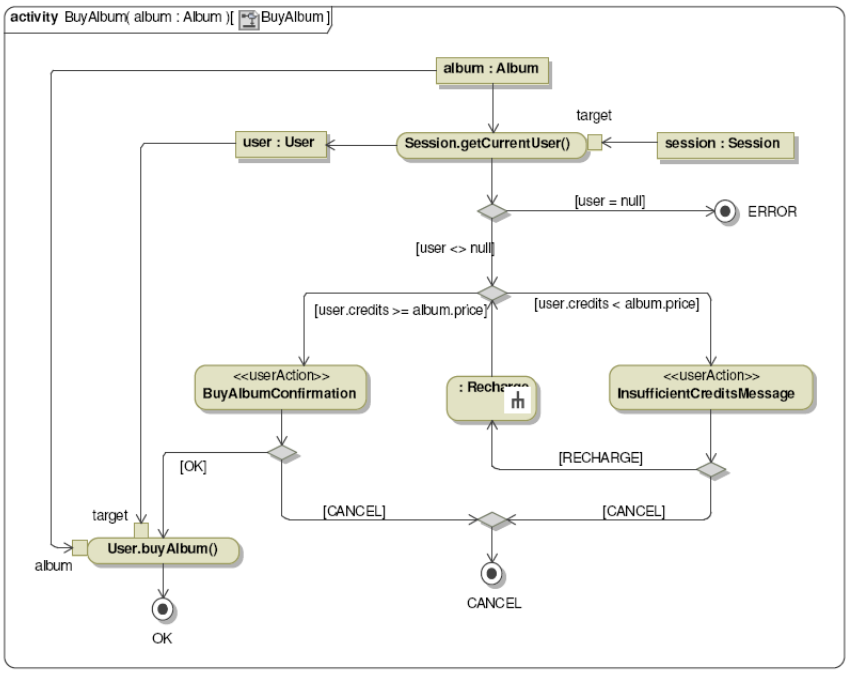


**Figura 3.6:** Processo de negócio UWE (Kroib e Koch, 2008)

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

## 3.4 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 fundamentaram os estudos pertinentes ao desenvolvimento da linguagem RestML proposta nesta dissertação de mestrado.

Capítulo

4

# Colaboração

No projeto em questão foi utilizado 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 e será realizado também um levantamento dos requisitos que um ambiente colaborativo deve possuir, além de 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, 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 *outsour-*

*cing*, 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, 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 (Rama e Bishop, 2006).

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 que devem fazer parte dos CDEs, estão as descritas por (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. Entre os exemplos desse tipo de ferramenta estão o JIRA[3] e o Mantis.

- **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 é possível 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.

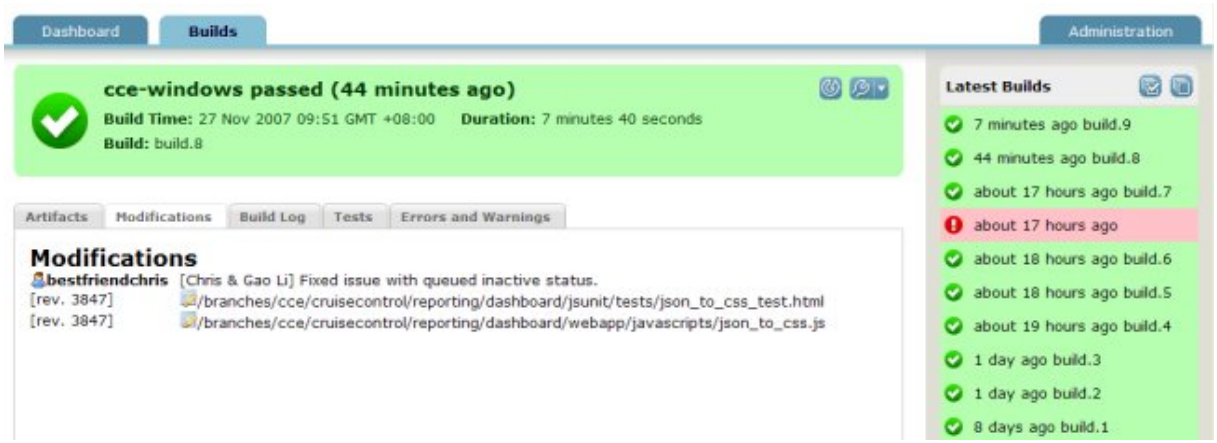


**Figura 4.1:** Painel com detalhes do build (CruiseControl, 2010)

- **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.

## 4.2 Trabalhos Relacionados à 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 são descritos trabalhos relacionados sob diferentes perspectivas do uso da colaboração aliada ao MDD.

O trabalho de (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., 2009), que está focado no gerenciamento de requisitos para *Business Process Reengineering* (BPR). No seu trabalho (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, 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 incremen-

tal 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, 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.

## 4.3 Considerações Finais

Neste capítulo foram apresentados os principais fundamentos sobre colaboração e os trabalhos relacionados de pesquisas sobre colaboração aliada ao MDD. Diante do estudo desses trabalhos relacionados com a colaboração aliada ao MDD, sob diferentes perspectivas, foi possível projetar o ambiente RestMDD que será discutido no Capítulo 6.

Capítulo

5

# RestML

Neste capítulo é apresentada a linguagem específica de domínio RestML, projetada para representar Serviços Web RESTful. São apresentadas as principais características da linguagem, os seus tipos de modelos e a qual contexto se aplicam, bem como as formas de representar as características estáticas e comportamentais dos serviços.

## 5.1 Linguagem específica de domínio

Este trabalho tem como um dos objetivos a definição de uma linguagem específica de domínio (DSL - *Domain Specific Language*) 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. Essa DSL portanto deve ser suficientemente completa para que os modelos gerados a partir dela possam sofrer transformações automatizadas que gerem o código-fonte da aplicação.

Esta DSL batizada de RestML é definida por *Profiles* UML e tem como principal motivação o uso da abordagem MDA (OMG, 2003) para o desenvolvimento orientado a modelos, que possibilitará construir modelos como uma extensão da UML 2 e transformá-los em código-fonte usando linguagens para transformação de modelos como o QVT.

## 5.2 Profile UML

De acordo com (Fuentes-Fernández e Vallecillo-Moreno, 2004) a UML oferece um conjunto de mecanismos de extensão (estereótipos, *tagged values* e restrições) para especializar seus elementos, permitindo extensões customizadas da UML para plataformas específicas ou um domínio em particular. Essas extensões são agrupadas em um *Profile* UML.

Entre os principais motivos que os engenheiros de software desejam especializar os seus metamodelos estão:

- Terminologia adaptada para uma plataforma específica ou domínio;
- Sintaxe para elementos que ainda não possuam notação;
- Notação distinta para símbolos já existentes, mais apropriada ao domínio ou plataforma;
- Adicionar semântica ao metamodelo;
- Adicionar restrições a fim de limitar a maneira que o metamodelo pode ser usado e suas construções;
- Adicionar informação que pode ser utilizada para transformação de um modelo em outro modelo ou ainda em código-fonte.

O *Profile* UML contém o conjunto de mecanismos de extensão do metamodelo agrupados em um pacote UML que possui o estereótipo ≪profile≫. Na linguagem RestML foram definidos dois *profiles* UML, sendo um deles para a modelagem de Serviços Web RESTful e outro para construção desses serviços para a plataforma Java Enterprise Edition (JavaEE) (Oracle, 2012).

### 5.2.1 Modelagem de Serviços Web RESTful

Este *profile*, chamado *RestML Profile for Modeling RESTful Web Services*, foi criado para adicionar informação aos modelos UML de acordo com a proposta do estilo arquitetural REST. Dessa forma, foram criados estereótipos e *tagged values* que possam contemplar as restrições arquiteturais definidas no REST, permitindo assim que os modelos criados por esse *profile* sejam transformados em outros modelos, de forma automatizada.

Assim, esse *profile* visa a modelagem de Serviços Web RESTful seguindo suas restrições arquiteturais, porém sem se importar com detalhes específicos da plataforma, ou seja, os diagramas UML criados com esse *profile* modelam serviços Web RESTful que podem ser especializados para diferentes plataformas.

Com este *profile* será possivel construir diagramas UML para expressar as características e relacionamentos de cada componente de um Serviço Web RESTful. Os tipos de componentes definidos são:

- Recurso (*Resource*): recebe as requisições externas através de uma interface uniforme;
- Aplicação (*Application*): é um recurso especial que contém *hyperlinks* para todos os demais recursos da aplicação;
- Serviço (*Service*): componente responsável pela lógica de negócio da aplicação;
- Representação (*Representation*): formato de dados de um recurso;
- Exceção (*Exception*): identifica um problema ocorrido na execução de um serviço;
- Acesso a dados (*DataAccess*): componente capaz de acessar dados externos a aplicação.

Cada um destes tipos de componente tem um objetivo distinto dentro da arquitetura REST proposta pela linguagem RestML e devem ser distinguidos claramente para que seja possível aplicar diferentes tipos de transformações em cada um destes componentes. Esta distinção é realizada aplicando os estereótipos existentes no *profile* em cada um dos componentes da aplicação.

Os estereótipos são usados para introduzir um novo tipo de elemento de modelo como uma extensão ou uma classificação do elemento base já existente (Aldawud et al., 2003). Os estereótipos, nesse cenário, devem ser aplicados em diagramas de classes e diagramas de sequência. Quando um estereótipo é aplicado a um elemento do modelo, os valores de suas propriedades podem ser referenciadas como *tagged values*, que nada mais são que um par de chave-valor anexado ao elemento do modelo.

Nesse *profile* foram definidos doze tipos de estereótipos como ilustra a Figura 5.1. São eles:

1. *Application*: identifica um recurso que representa a aplicação. Deve ser aplicado apenas na definição das classes;
2. *Resource*: identifica um recurso do sistema. Este estereótipo pode ser aplicado na definição das classes;
3. *Create*: identifica uma criação de um novo recurso. Deve ser aplicado apenas para métodos;

4. *Retrieve*: identifica uma busca de recursos. Deve ser aplicado apenas para métodos;
5. *Update*: identifica uma atualização de um recurso existente. Deve ser aplicado apenas para métodos;
6. *Delete*: identifica uma exclusão de um recurso existente. Deve ser aplicado apenas para métodos;
7. *Service*: classifica como um componente responsável por regra de negócio, portanto deve possuir um comportamento transacional;
8. *Representation*: identifica uma representação de um recurso, ou seja, é o formato de dados que representará um recurso da aplicação;
9. *Optional*: identifica uma propriedade da representação que não é obrigatória;
10. *Range*: identifica o valor mínimo e máximo de uma propriedade numérica da representação ou o tamanho mínimo e máximo de uma propriedade textual da representação;
11. *Exception*: identifica um componente que será responsável por informar uma condição de erro no sistema;
12. *Data Access*: referencia um componente responsável por acesso a dados externos da aplicação, como por exemplo banco de dados, arquivos, RMI, Serviços Web, entre outros.

Com o uso desse *profile* é possível modelar os serviços Web RESTful de maneira independente da plataforma. Na próxima seção será mostrado outro *profile* porém este voltado para a modelagem desses serviços específica para uma plataforma.

### 5.2.2 Serviços Web RESTful na plataforma JavaEE

O *RestML Profile for Modeling RESTful Web Services* tem por objetivo modelar as características comuns de serviços Web RESTful de maneira independente de plataforma, porém é importante que os modelos criados com esse *profile* sejam transformados em modelos específicos de uma plataforma para que assim seja possível gerar código-fonte de maneira mais direta.

Para atender essa abordagem, nesta dissertação de mestrado propõe-se um outro *Profile* UML chamado *RestML JavaEE Profile for Modeling RESTful Web Services* que tem por objetivo tornar específicos para a plataforma Java Enterprise Edition os modelos criados com o *RestML Profile for Modeling RESTful Web Services*.

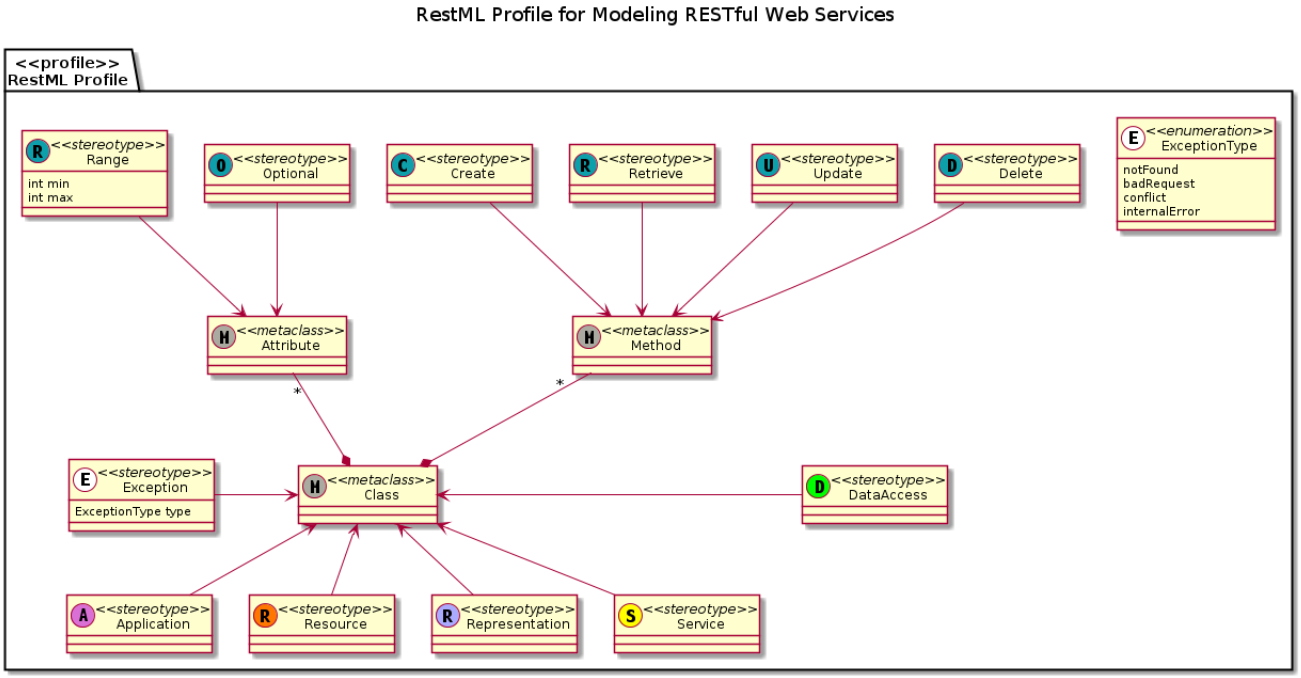


**Figura 5.1:** RestML Profile for Modeling RESTful Web Services

Na Figura 5.2 é apresentada a definição do *RestML JavaEE Profile for Modeling RESTful Web Services*.

Ao observar este *profile* é possível visualizar que este possui um número maior de estereótipos e contém também *tagged values* para especificar detalhes da plataforma que a aplicação será implementada.

A transformação entre os dois *profiles* descritos nesse capítulo será abordada com maiores detalhes na Seção 5.2.6.

### 5.2.3 Abordagem MDA

Com o uso da abordagem MDA três tipos de modelos podem ser construídos: Modelos de domínio (CIM), Modelos independentes de plataforma (PIM) e Modelos específicos de plataforma (PSM). Nas próximas seções são apresentados cada um desses tipos de modelos descrevendo qual o seu propósito geral, como devem ser modelados usando a RestML, as entradas necessárias para sua construção e o resultado final obtido por meio de um exemplo.

### 5.2.4 Modelos de domínio (CIM)

A primeira etapa para o desenvolvimento de uma aplicação usando a abordagem MDA é a construção dos modelos de domínio que tem como principal objetivo representar os requisitos da aplicação por meio de um vocabulário comum aos especialistas de domínio

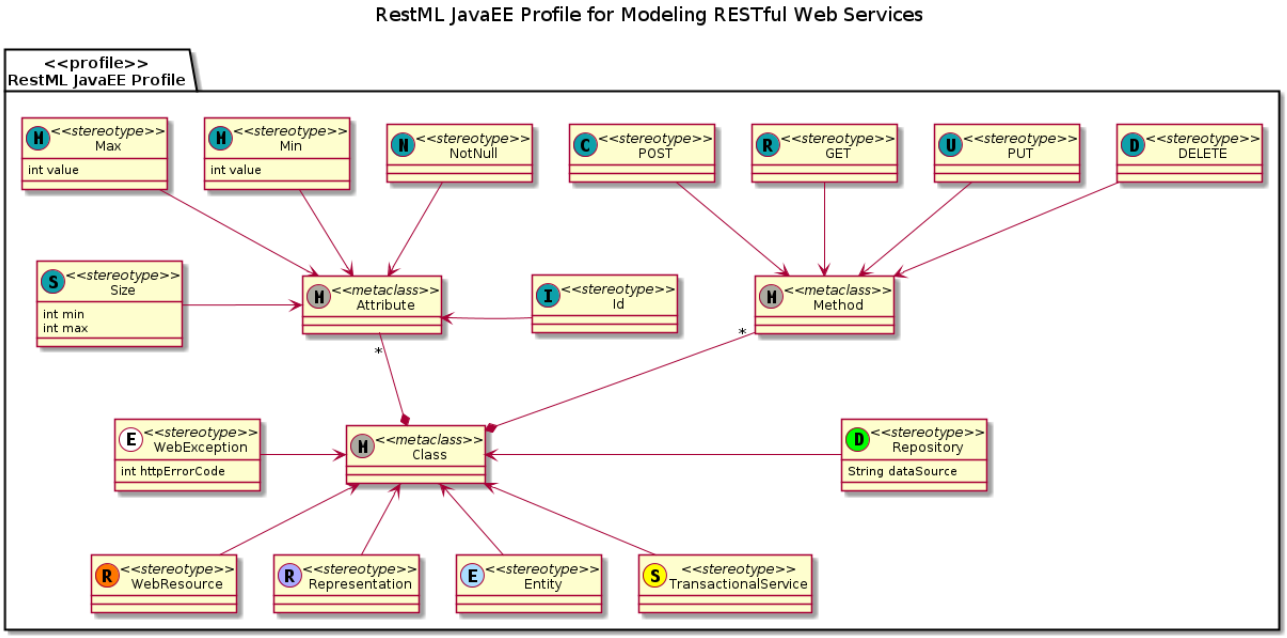


**Figura 5.2:** RestML JavaEE Profile for Modeling RESTful Web Services

e especialistas em *design*. Assim, é possivel modelar o comportamento do sistema sem a necessidade de conhecer os artefatos necessários para a implementação das funcionalidades.

Na linguagem RestML foram definidos três tipos de modelos de domínio:

1. Casos de uso
2. Diagrama de caso de uso
3. Diagrama de atividades

Estes modelos possuem propósitos distintos dentro da modelagem dos serviços RESTful e são geralmente definidos por especialistas no domínio com base em um documento de requisitos do sistema servindo de entrada para a construção dos modelos independentes de plataforma.

#### 5.2.4.1 Casos de uso

Um caso de uso é uma descrição das possíveis sequências de interações entre o sistema em discussão e seus atores externos, relacionados a um objetivo particular (Cockburn, 2000). Em uma definição mais simples, um caso de uso descreve a forma como o usuário interage com o sistema e a forma como o sistema responderá (Rosenberg e Stephens, 2007).

Por meio de modelagem usando casos de uso é possivel identificar os eventos que devem acontecer para que um requisito seja atendido e a interação que os atores realizam com

o sistema. Assim, os casos de uso podem ser considerados o alicerce do comportamento dinâmico do sistema.

Na linguagem RestML o desenvolvimento dos Serviços Web RESTful dependem da modelagem de um caso de uso para cada tarefa que possa ser realizada pelos usuários. Nesta abordagem, usando MDA, os casos de uso são os artefatos de primeira classe e através deles que os demais modelos são construídos.

O conteúdo de um modelo de caso de uso neste contexto deve conter os seguintes componentes:

- Nome: nome do caso de uso, utilizado para identificá-lo;
- Descrição: resumo breve do objetivo do caso de uso. Na descrição pode referenciar o requisito ao qual este caso de uso atende;
- Ator: nome do ator que realiza este caso de uso;
- Tipo: identifica o tipo do caso de uso. Os valores permitidos são: Cadastro, Alteração, Exclusão e Consulta. Este tipo será fundamental para identificar qual método da interface uniforme dos serviços RESTful deverá ser utilizado;
- Recurso: nome do recurso que será a porta de entrada para a execução do caso de uso;
- Pré-condição: estado inicial do sistema para que o caso de uso possa ser executado;
- Fluxo principal: contêm as tarefas que o ator deve executar, bem como as respostas produzidas pelo sistema;
- Fluxo alternativo: contempla as exceções do fluxo principal;
- Pós-condição: resultado esperado após a execução do caso de uso.

O estilo arquitetural REST prevê que toda comunicação entre um cliente e um recurso seja realizada utilizando uma interface uniforme. Por essa razão, a modelagem dos casos de uso exige que seja informado o tipo do caso de uso e o seu recurso para que seja possível relacionar um caso de uso a interface uniforme do recurso.

Uma vez que os casos de uso tenham sido modelados é possivel criar um diagrama de caso de uso, obtendo uma representação simples e visual dos casos de uso existentes e também seus relacionamentos com os atores.

#### 5.2.4.2 Diagrama de caso de uso

Um diagrama de caso de uso é capaz de mostrar as relações entre os atores e casos de uso dentro de um sistema (Ambler, 2005). Em geral são utilizados para:

- Proporcionar uma visão geral dos requisitos de um sistema;
- Comunicar o escopo de um projeto;
- Modelar a análise sobre os requisitos na forma de um modelo de caso de uso sistêmico.

No escopo da linguagem RestML o diagrama de caso de uso será construido a partir das definições de caso de uso e os seus respectivos atores, de forma automatizada pelo ambiente colaborativo RestMDD.

É possivel criar diferentes diagramas de caso de uso representando partes independentes do sistema. A Figura 5.3 ilustra um exemplo de um diagrama de caso de uso de manutenção de universidades realizados na aplicação AgendaWS, que será vista em detalhes no Capítulo 7.

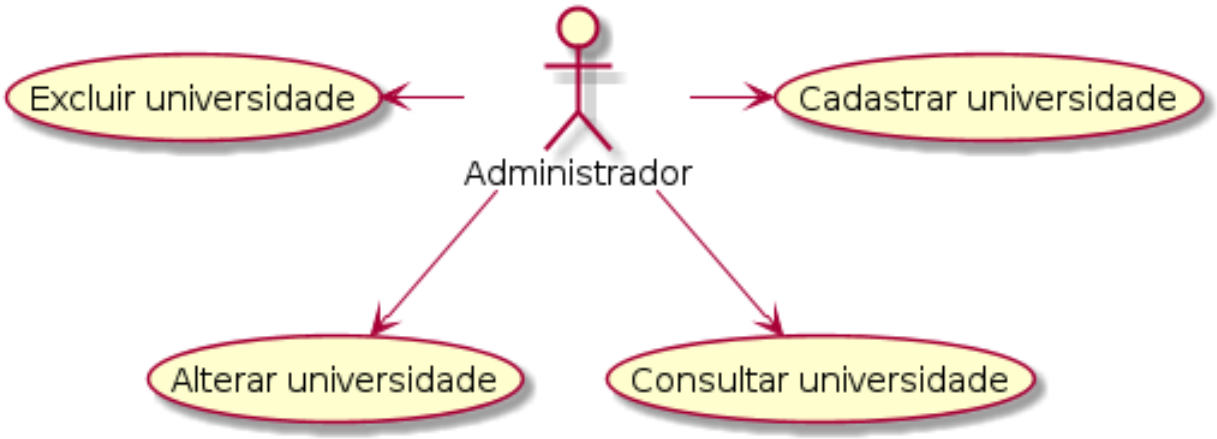


**Figura 5.3:** Diagrama de caso de uso

O diagrama de caso de uso não servirá diretamente para a criação dos demais artefatos e modelos da aplicação, mas ele tem um papel fundamental na documentação do sistema e também na comunicação entre os membros da equipe.

#### 5.2.4.3 Diagrama de atividades

Os diagramas de atividades são geralmente utilizados para a modelagem de processos de negócio, lógica capturada por um único caso de uso ou ainda para modelar de forma detalhada a lógica de uma regra de negócio (Ambler, 2012).

Estes diagramas podem ser considerados modelos de domínio, pois não trazem nenhuma informação sobre como a atividade será implementada, escodendo detalhes técnicos pertinentes ao *design* da solução. Os diagramas de atividades apenas identificam quais atividades devem ser executadas pelo caso de uso para que este se concretize, sem informar o funcionamento interno de cada um destas atividades.

Para representar um diagrama de atividades existe uma notação gráfica básica que contempla os seguintes elementos:

- *Initial node*: é o ponto de partida do diagrama, representado por um círculo preenchido. Apesar de não ser obrigatório, facilita o entendimento de onde o fluxo de atividades se inicia;

- *Final node*: é onde o diagrama se encerra, representado por um círculo preenchido com uma borda envolvendo-o. Tambem não é obrigatório, mas ajuda na percepção de término do processo;

- *Activity*: comportamento a ser executado. É representado por um retângulo com bordas arredondadas;

- *Flow*: são as setas que interligam os demais componentes do diagrama. Como o próprio nome diz, determina o fluxo das atividades;

- *Fork*: representa o início de atividades em paralelo, sendo representado por uma barra preta com um único fluxo de entrada e vários fluxos de saída;

- *Join*: representa o fim do processamento das atividades em paralelo, sendo representado por uma barra preta com vários fluxos de entrada e um único fluxo de saída;

- *Condition*: texto colocado sobre um fluxo que deve ser avaliado como uma condição verdadeira para que este fluxo possa ser continuado;

- *Decision*: representado por um diamante com um único fluxo de entrada e vários fluxo de saida, cada uma deles com um condição associada;

- *Merge*: representado por um diamante com vários fluxos de entrada e um único fluxo de saída. Para que o fluxo de atividades continue é necessario que todos os fluxos de entrada contemplem condições verdadeiras;

- *Partition*: serve para indicar quem está executando determinada atividade.

No cenário previsto pela RestML, será construído um diagrama de atividades para cada caso de uso cadastrado anteriormente. Estes diagramas de atividades devem ser gerados de forma automática através de transformações aplicadas aos modelos de caso de uso, de tal forma que cada evento do fluxo básico e também dos fluxos alternativos se transformem em atividades dentro do diagrama.

A representação da estrutura destes diagramas é definida no formato XMI (XML Metadata Interchange) (OMG, 2007), um formato para representar modelos em XML capaz de fazer o intercâmbio de UML e outros modelos baseados no MOF. Assim, o XMI define um mapeamento de UML/MOF para XML, que é capaz de integrar ferramentas, aplicações e repositórios.

A Figura 5.4 ilustra um diagrama de atividades gerado a partir do modelo de caso de uso definido na seção 5.2.4.1:

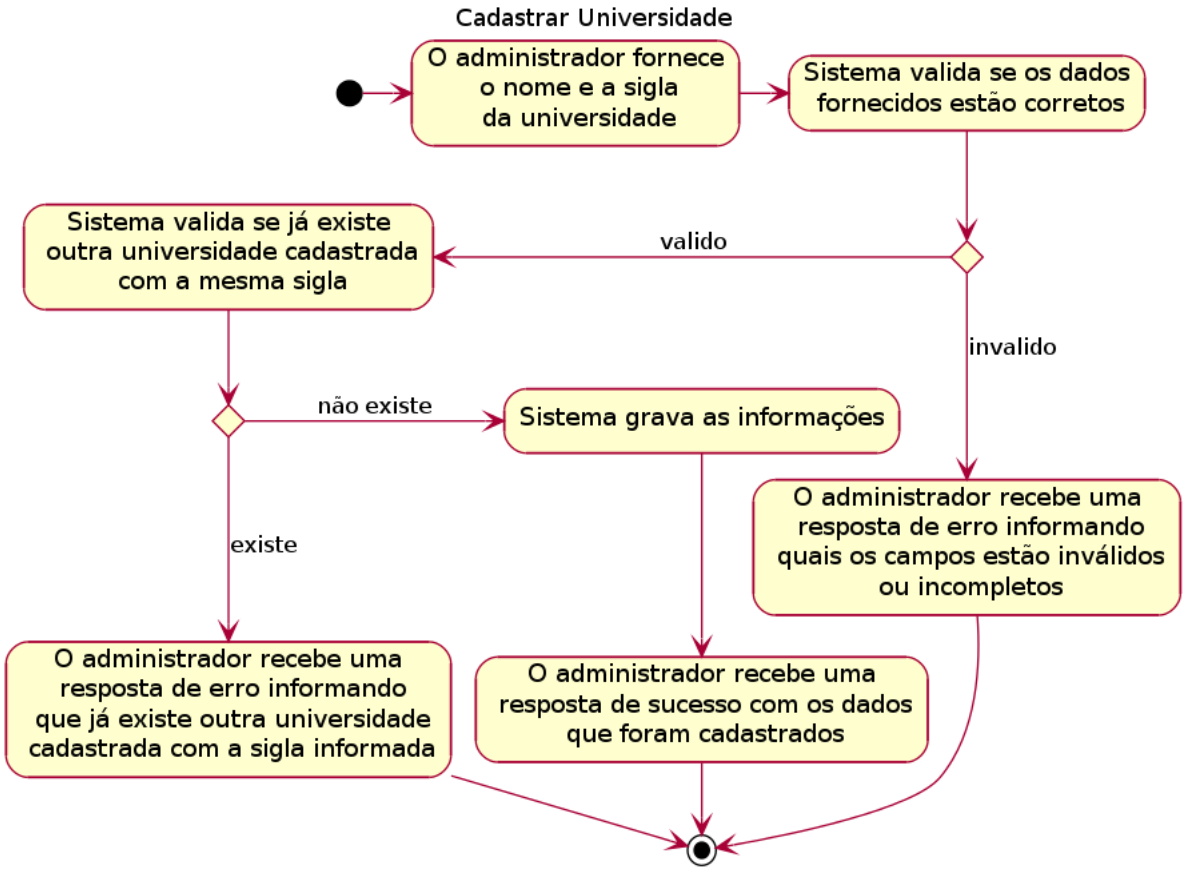


**Figura 5.4:** Diagrama de atividade

Assim, é possivel modelar o comportamento dinâmico da aplicação em seus diferentes cenários de uso e, por meio deste comportamento, os *designers* da solução conseguem projetar os modelos independentes de plataforma que documentarão o comportamento interno de cada componente de software. Neste momento o desenvolvimento do software passa da fase de análise para a sua fase de projeto.

### 5.2.5 Modelos independentes de plataforma (PIM)

Em resumo, um modelo PIM é uma visão formal da funcionalidade e da estrutura de um sistema de software sem referência a qualquer plataforma computacional específica (Merriman, 2003). Estes tipos de modelos são criados a partir de transformações aplicadas aos modelos de domínios e modelam claramente como será realizada a implementação das funcionalidades do sistema, porém sem conter aspectos específicos de uma plataforma ou linguagem de programação. Portanto, é possivel obter diferentes modelos do projeto da solução de software portável para diferentes ambientes.

No contexto da linguagem RestML foram definidos dois tipos de modelos independentes de plataforma:

1. Diagrama de classes
2. Diagrama de sequência

Cada um destes modelos possuem objetivos distintos na modelagem, sendo os diagramas de classes responsáveis pela definição das estruturas estáticas da aplicação como por exemplo recursos, serviços, representações, entre outros e os diagramas de sequência definem o comportamento dinâmico do sistema mostrando a troca de mensagens entre os diversos componentes da arquitetura.

#### 5.2.5.1 Diagrama de classes

Os diagramas de classes são utilizados para uma variedade de propósitos, desde modelagem conceitual até a modelagem detalhada do projeto. Estes diagramas mostram as classes do sistema, suas inter-relações (incluindo agregação, herança e associação) e as operações e atributos existentes em cada classe (Ambler, 2004).

Entre os principais conceitos existentes em um diagrama de classes podemos destacar:

- Classe: elemento abstrato que representa um conjunto de objetos. A classe contêm a especificação do objeto com suas características e comportamentos;
- Atributo: define as características do objeto;
- Operação: ações executadas pelo objeto;
- Associação: relacionamento entre as classes;
- Agregação: representa um todo que é composto de várias partes. Não há um relacionamento de contenção;

- Composição: é um relação de contenção, ou seja, um objeto (*container*) contêm outros objetos (elementos). Os elementos dependem do *container* para existir;

- Herança: representa as hierarquias existentes entre as classes.

No cadastro dos casos de uso o analista informa o recurso ao qual o caso de uso está associado e o seu tipo. Com estas informações, é possível gerar um diagrama de classes definindo as classes para os recursos e seus respectivos métodos. Para estas classes de recursos deve ser aplicado o estereótipo ≪Resource≫ e para os métodos pode ser aplicado um dos seguintes estereótipos: ≪Create≫, ≪Retrieve≫, ≪Update≫ ou ≪Delete≫.

Para cada recurso definido nos casos de uso, uma classe de recurso deve ser criada. Além disso, todo caso de uso possui um tipo e este servirá para incluir um método na classe de recurso. Para os casos de uso do tipo Cadastro deve ser criado um método com o estereotipo ≪Create≫; para o tipo Alteração o método deve ter o estereótipo ≪Update≫; para o tipo Exclusão o estereótipo utilizado deve ser o ≪Delete≫; e por fim o tipo Consulta deve conter o estereótipo ≪Retrieve≫.

A classe de recurso ficará responsável por receber a requisição dos clientes, validar os parâmetros de entrada, invocar um método da classe de serviço que cuidará de toda a lógica de negócio e por último construir a resposta a ser enviada ao usuário. Portanto, para toda classe de recurso deverá ser criada também uma classe de serviço que será responsável pela lógica do processo de negócio e deve conter o estereótipo ≪Service≫. O nome dos métodos devem coincidir nas duas classes.

Para cada recurso se faz necessário a criação de uma classe que definirá a representação contendo todos os atributos que representem aquele recurso. A classe da representação deve ter o estereótipo ≪Representation≫, os atributos não obrigatórios dessa classe devem conter o estereótipo ≪Optional≫ e os atributos que tenham algum tipo de validação quanto a quantidades de caracteres (para atributos textuais) ou valores mínimo e máximo (para atributos numéricos) devem incluir o estereótipo ≪Range≫ e informar os *tagged values min* e *max*.

Para acessar os dados externos da aplicação, é necessário que seja definida uma classe de acesso a dados com o estereótipo ≪Data Access≫.

Por fim, na execução dos casos de uso podem ocorrer falhas que devem ser reportadas com diferentes códigos e mensagens de erro. Portanto, devem ser criadas classes que representem essas falhas e possuam o estereótipo ≪Exception≫.

Na Figura 5.5 é possível ver um diagrama de classes contendo os recursos da aplicação.

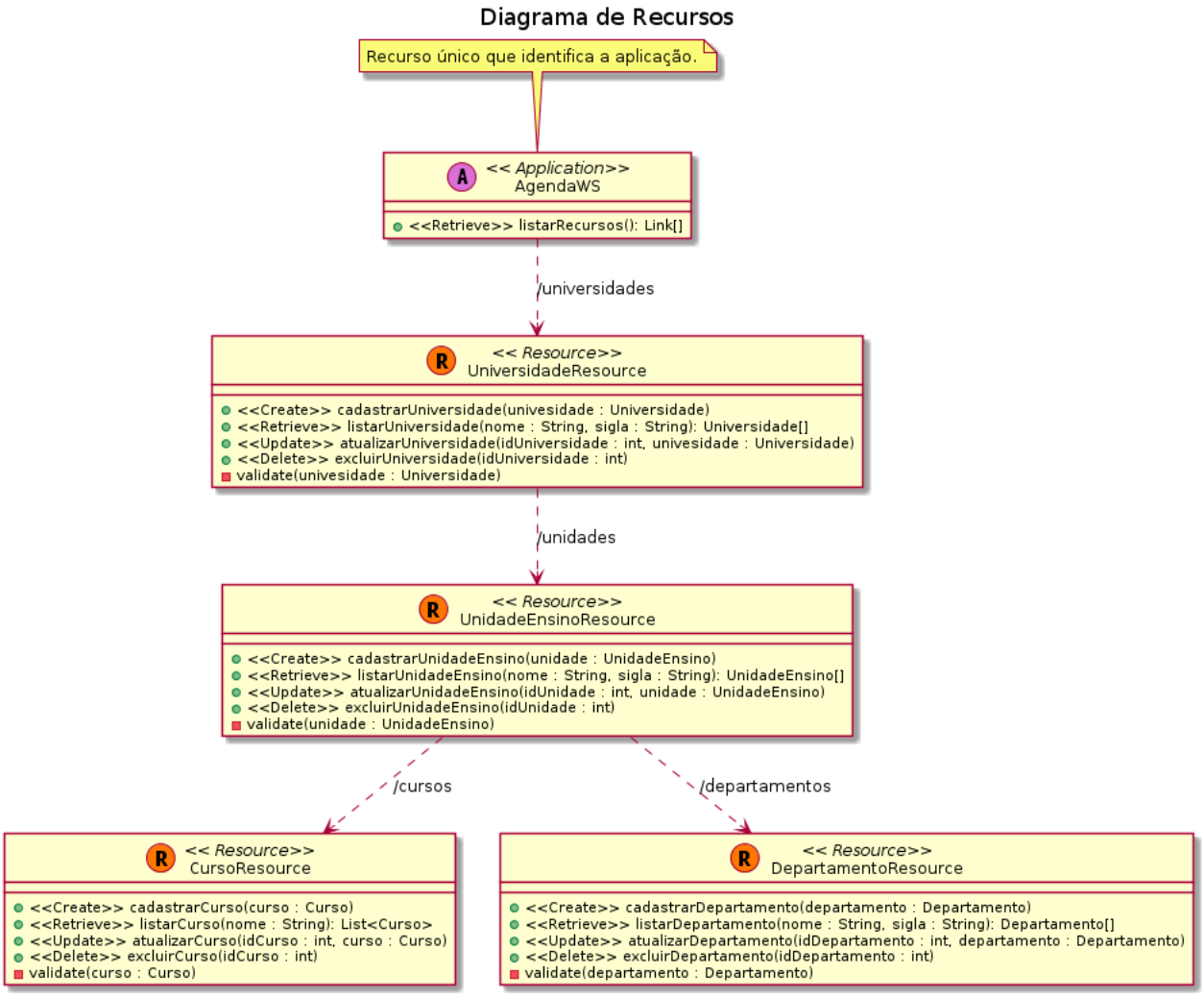


**Figura 5.5:** Diagrama de classes para recursos

#### 5.2.5.2 Diagrama de sequência

Os diagramas de sequência modelam o fluxo de lógica dentro do seu sistema de uma forma visual, permitindo tanto documentar quanto validar sua lógica. Estes diagramas são artefatos muito populares para a modelagem dinâmica com o foco de identificar o comportamento do sistema (Ambler, 2012).

Em síntese, o diagrama de sequência é um dos diagramas UML usados para representar interações entre objetos de um cenário, realizadas através de métodos, dando ênfase na ordenação temporal em que as mensagens são trocadas entre os objetos.

A construção do diagrama de sequência utilizando a linguagem RestML será realizada de forma semi-automatizada com informações vindas do diagrama de atividades criado na fase de análise.

Para cada atividade do diagrama de atividades, se faz necessária a definição de quais componentes irão realizar esta atividade e quais as mensagens trocadas entre eles. Essa

troca de mensagens deve ser definida pelo desenvolvedor, utilizando apenas as classes identificadas como *Resource*, *Service*, *Exception* e *Data Access*.

Na Figura 5.6 há um exemplo de diagrama de sequência baseado no diagrama de atividades referente a Figura 5.4.

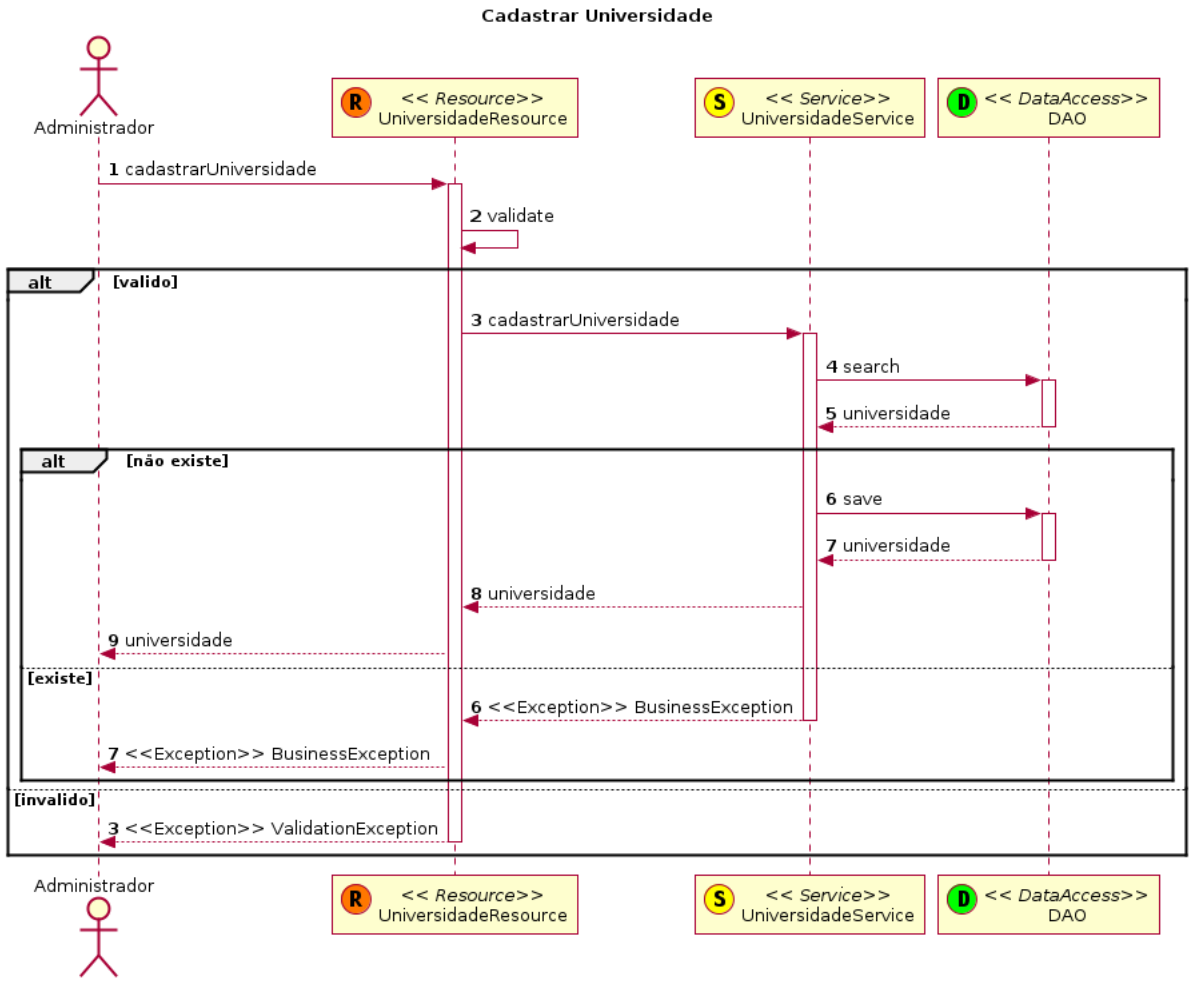


**Figura 5.6:** Diagrama de sequência

### 5.2.6 Modelos específicos de plataforma (PSM)

O próximo passo no processo de desenvolvimento de software utilizando a linguagem RestML é a transformação do modelo independente de plataforma em um modelo específico de plataforma. Um PSM combina as especificações obtidas no PIM com os detalhes específicos da plataforma, ou seja, este modelo representa um visão do sistema inserido no contexto de uma plataforma específica.

Assim, na linguagem RestML inicialmente foi proposta apenas uma transformação para a plataforma a Java Enterprise Edition (Oracle, 2012) que contempla diversas API's para o desenvolvimento Web juntamente com os frameworks Spring (SpringSource, 2012),

que provê um *container* capaz de realizar inversão de controle (IoC) e injeção de dependência, e JAX-RS que provê API's para o desenvolvimento de Serviços Web RESTful (JCP, 2012).

Essa transformação consiste basicamente em aplicar o *RestML JavaEE Profile for Modeling RESTful Web Services* às classes existentes no PIM. Porém, é necessário que algumas etapas sejam seguidas:

- As classes que contêm os estereótipos ≪Application≫ e ≪Resource≫ devem receber o estereótipo ≪WebResource≫, identificando que estes serão recursos da aplicação;

- As classes que contêm o estereótipo ≪Representation≫ passarão a ter dois estereótipos ≪Representation≫ e ≪Entity≫. O primeiro deles identifica que esta classe será uma representação do recurso e o segundo estereótipo indica que esta classe será uma entidade do modelo entidade-relacionamento do banco de dados acessado pela aplicação. Os atributos numéricos que contêm o estereótipo ≪Range≫ deverão ter dois estereótipos ≪Min≫ e ≪Max≫, enquanto os atributos textuais passam a ter o estereótipo ≪Size≫. Os atributos que não possuíam o estereótipo ≪Optional≫ agora devem ter o estereótipo ≪NotNull≫;

- As classes com o estereótipo ≪Service≫ receberão o estereótipo ≪TransactionalService≫, identificando um componente de negócio transacional;

- As exceções com o estereótipo ≪Exception≫ agora serão ≪WebException≫. O *tagged value httpErrorCode* deve conter o código de erro HTTP de acordo com o seu tipo. As exceções do tipo "*notFound*" possuem código 404; o tipo "*badRequest*" deve ter o código 400; já o tipo "*conflict*" deve ter o código 409; e por último exceções do tipo "*internalError*" com o código 500;

- A classe de acesso a dados com o estereótipo ≪DataAccess≫ deve agora ter o estereótipo ≪Repository≫. O *tagged value dataSource* deverá conter o nome da fonte de dados configurada no servidor;

- Os métodos com estereótipos ≪Create≫, ≪Retrieve≫, ≪Update≫ e ≪Delete≫ agora se transformarão em ≪POST≫, ≪GET≫, ≪PUT≫ e ≪DELETE≫, respectivamente.

Para o diagrama de sequência não há mudanças significativas na troca de mensagens entre os componentes da aplicação. As classes transformadas do PIM para o PSM continuam trocando mensagens na mesma ordem temporal definida no PIM.

## 5.3 Considerações Finais

Neste capítulo foi descrita uma abordagem de Desenvolvimento Orientado a Modelos para a construção de Serviços Web RESTFul através de uma linguagem especifica de domínio chamada RestML que é definida por meio de *Profiles* UML e segue os princípios da arquitetura MDA.

Foram descritos também todos os modelos criados em cada fase do desenvolvimento de um serviço Web RESTful, bem como todas as características que estes modelos devem possuir. Além disso, foi mostrada a relação entre os modelos de domínio, modelos independentes de plataforma e modelos especificos de plataforma.

Capítulo

6

# RestMDD

Neste capítulo será apresentado o ambiente colaborativo RestMDD que se utiliza da linguagem RestML para produção de Serviços Web RESTful utilizando o paradigma de desenvolvimento orientado a modelos. São descritas suas principais funcionalidades e as etapas a serem seguidas para a modelagem dos serviços Web RESTful usando o paradigma MDD.

## 6.1 Considerações Iniciais

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).

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 o compartilhamento de espaço de trabalho, projetos e modelos. Além disso, o desenvolvedor é

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

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.

Diante dessa carência de ferramentas colaborativas para o desenvolvimento orientado a modelos capazes de modelar Serviços Web RESTful, este projeto realizou o desenvolvimento do ambiente colaborativo RestMDD.

## 6.2 Ambiente colaborativo

O ambiente RestMDD têm como objetivo prover suporte ao desenvolvimento de Serviços Web RESTful utilizando técnicas de desenvolvimento orientado a modelos (MDD). Neste ambiente é possivel desenvolver projetos do começo ao fim de maneira colaborativa entre os desenvolvedores utilizando a linguagem RestML, criada especialmente para a definição de Serviços Web RESTful.

Entre as principais funcionalidades que o ambiente provê estão:

1. Espaço de trabalho compartilhado

   O ambiente RestMDD possui o conceito de espaço de trabalho, que é o local onde os usuários criam os seus projetos que contêm todos os modelos da aplicação, ou seja, para acessar este ambiente o usuário deve possuir o acesso a um espaço de trabalho. Ao criar um novo espaço de trabalho, o usuário pode decidir compartilhá-lo com os demais desenvolvedores que irão colaborar com ele em seus projetos. Dessa forma, é possível ter um espaço de trabalho compartilhado entre diversos desenvolvedores, o que permite a colaboração entre eles.

   Dentro do espaço de trabalho residem os diferentes projetos criados, que inicialmente são acessíveis apenas ao seus criadores. Para que outros desenvolvedores possam acessá-los é necessário que o criador do projeto o compartilhe através da ferramenta. Assim, os desenvolvedores podem ter diferentes visões de projetos que existem dentro do espaço de trabalho.

   Em um projeto compartilhado todos os desenvolvedores que o compartilham podem criar, alterar, visualizar e excluir os modelos presentes neste projeto. Porém, não é permitido que dois usuários editem simultaneamente um mesmo modelo, ou ainda que executem alterações em um modelo que têm dependência com outro modelo que está sendo editado por outro usuário.

   Existe ainda uma barra de status na interface gráfica que indica quantos usuários estão conectados ao espaço de trabalho naquele momento.

2. Capacidade de trabalhar de forma isolada

Ao acessar um modelo existente no projeto, o usuário tem a opção de entrar no modo "Somente Leitura" que apenas exibe os dados do modelo, ou no modo "Editar" que permite que o usuário realize alterações no modelo. Uma vez, que o usuário acessar o modo "Editar" o sistema automaticamente bloqueia os demais usuários para edição deste modelo e seus dependentes, fazendo que o usuário consiga trabalhar de forma isolada, garantindo que nenhuma alteração externa possa influenciar em seu desenvolvimento naquele instante. Os demais desenvolvedores do projeto conseguem apenas acessar este modelo e seus dependentes no modo "Somente Leitura".

3. Desenvolvimento dos projetos de forma simultânea

   Com o compartilhamento de espaços de trabalho e projetos é possível que os desenvolvedores consigam construir modelos de forma simultânea, produzindo artefatos que poderão ser reutilizados pelos membros do projeto, porém sem interferir ou até mesmo destruir as alterações realizadas pelos demais desenvolvedores do projeto. Isso é possível graças ao isolamento previsto pelo ambiente colaborativo, garantindo a integridade dos modelos produzidos.

4. Controle de acesso

   Para o acesso ao ambiente colaborativo é necessário que o usuário possua um cadastro que determina qual seu perfil de acesso. Em todo acesso realizado, o usuário deve informar suas credenciais através de um *login* e senha pessoal. Além disso, o usuário consegue acessar apenas os espaços de trabalho e projetos ao qual possui acesso.

   Apenas o criador do espaço de trabalho pode excluí-lo, renomeá-lo ou compartilhá-lo com outros membros da equipe. A mesma regra se aplica aos projetos. Quantos aos modelos, todos os usuários que possuem acesso ao projeto tem acesso total aos modelos. A criação de novos espaços de trabalho e projetos são restritos a alguns perfis de acesso, ou seja, apenas alguns usuários que tenham privilégios específicos podem criar os espaços de trabalho e projetos.

5. Criação dos modelos para as diferentes fases do projeto

   Seguindo a arquitetura MDA proposta pela RestML, os desenvolvedores podem criar diferentes tipos de modelos para cada fase do desenvolvimento do software. Além disso, como os modelos são criados a partir de transformações de outros modelos, há um nível de dependência entre eles, ou seja, a maioria dos modelos dependem da criação de outros modelos anteriormente. Como exemplo é possível citar os diagramas de atividades que são gerados a partir dos modelos de casos de uso, ou seja, é imprescindível que um caso de uso tenha sido criado para que este se transforme

e gere um diagrama de atividades. Assim, não é permitido criar diretamente um diagrama de atividades.

6. Geração de código-fonte executável de forma automatizada

   Depois de criado todos os modelos da fase de Análise e Projeto é possível gerar o código-fonte para a plataforma Java Enterprise Edition (JavaEE) (Oracle, 2012) com o uso dos frameworks Spring (SpringSource, 2012), que prove um container capaz de realizar inversão de controle (IoC) e injeção de dependência, e JAX-RS que prove API's para o desenvolvimento de Serviços Web RESTful de forma simples (JCP, 2012).

7. Empacotamento da aplicação

   Depois de gerado o código-fonte da aplicação é possível empacotá-lo em um *Web Archive* (WAR) de acordo com a especificação JavaEE, de modo a tornar mais fácil a sua distribuição e implantação em um Servidor Web Java.

Essas funcionalidades permitem aos desenvolvedores uma maior colaboração durante o projeto como um todo, porém sempre garantindo integridade dos modelos. Na próxima seção será apresentado o processo de desenvolvimento de software que deve ser seguido pelos desenvolvedores para a criação dos Serviços Web RESTful de forma produtiva e confiável.

## 6.3 Modelagem dos serviços

Segundo (Richardson e Ruby, 2007) existem algumas etapas básicas a serem seguidas para se projetar os serviços Web RESTful de acordo com uma arquitetura orientada a recursos. As etapas a serem aplicadas são:

- Identificar o conjunto de dados;
- Dividir o conjunto de dados em recursos;
- Nomear os recursos com identificadores únicos;
- Expor um subconjunto da interface uniforme;
- *Design* das representações aceitas do cliente;
- *Design* das representações servidas pelo servidor;
- Integrar o recurso com outros existentes através de *hyperlinks*;

- Considerar condições de erros.

Ainda seguindo a arquitetura orientada a recursos proposta por (Richardson e Ruby, 2007) é possível identificar dois tipos recursos:

1. Recursos únicos pré-definidos;
2. Itens individuais de dados.

Um recurso único pré-definido atua como uma *homepage* da aplicação, ou seja, retorna uma representação contendo uma lista de *hyperlinks* para acesso aos demais recursos da aplicação.

Já os itens individuais de dados representam os dados da aplicação, e por essa razão cada um destes deve ser identificado por um identificador único (URI).

Para conseguir modelar os serviços conforme as etapas previstas pela arquitetura orientada a recursos o usuário do ambiente RestMDD deve seguir o seguinte procedimento:

1. Cadastrar espaço de trabalho;
2. Compartilhar espaço de trabalho;
3. Cadastrar projetos;
4. Compartilhar projetos;
5. Cadastrar recursos;
6. Cadastrar atores;
7. Cadastrar casos de uso;
8. Construir diagramas de casos de uso;
9. Construir diagramas de atividades;
10. Cadastrar representações;
11. Cadastrar exceções;
12. Construir diagramas de classes para recursos;
13. Construir diagramas de classes para as representações;
14. Construir diagramas de classes para exceções e serviços;
15. Construir diagramas de sequências;

16. Transformação de modelos PIM para modelos PSM;

17. Geração de código-fonte;

18. Empacotamento.

Cada um desses procedimentos será exibido em detalhes nas próximas subseções.

### 6.3.1 Cadastrar espaço de trabalho

Inicialmente para que todos os membros de uma equipe de desenvolvimento possam colaborar nos projetos de uma organização é necessário que seja criado um espaço de trabalho que será compartilhado entre todos estes.

Apenas usuários com perfil COORDENADOR podem realizar a manutenção de espaços de trabalho, ou seja, apenas os coordenadores das equipes poderão criar, renomear, compartilhar ou excluir espaços de trabalho. Assim, o coordenador da equipe inicialmente deve criar um espaço de trabalho comum aos membros de sua equipe.

Ao acessar o ambiente colaborativo RestMDD uma tela para *login* é exibida, solicitando usuário e senha. Ao inserir de forma correta estas credenciais, o usuário é encaminhado a tela de escolha de espaço de trabalho. No primeiro acesso, nenhum espaço de trabalho existirá e portanto o coordenador deverá prosseguir com a criação deste por meio do *hyperlink* "Criar workspace".

Ao clicar nesse *hyperlink*, será exibida uma janela na qual o coordenador deverá informar o nome do espaço de trabalho e uma identificação única para este, conforme Figura 6.1.

Ao concluir o cadastro, é possível acessar o novo espaço de trabalho. Caso o coordenador precise renomear este espaço de trabalho, basta acessar o menu "Workspace" e clicar na opção "Renomear". Caso o espaço de trabalho tenha sido criado desnecessariamente é possível excluí-lo acessando o menu "Workspace" e clicando na opção "Excluir".

### 6.3.2 Compartilhar espaço de trabalho

Depois de criado o espaço de trabalho, o coordenador pode acessá-lo e compartilhar o mesmo com os demais desenvolvedores de sua equipe. Assim, é possível que estes desenvolvedores acessem esse espaço de trabalho recém-criado e colaborem no desenvolvimento dos projetos da equipe.

Para compartilhar um espaço de trabalho, o coordenador deve acessar o menu "Workspace" e clicar na opção "Compartilhar". Feito isso, uma janela será exibida para busca dos usuários do sistema. É possível realizar a busca por nome, login ou ainda buscar todos os

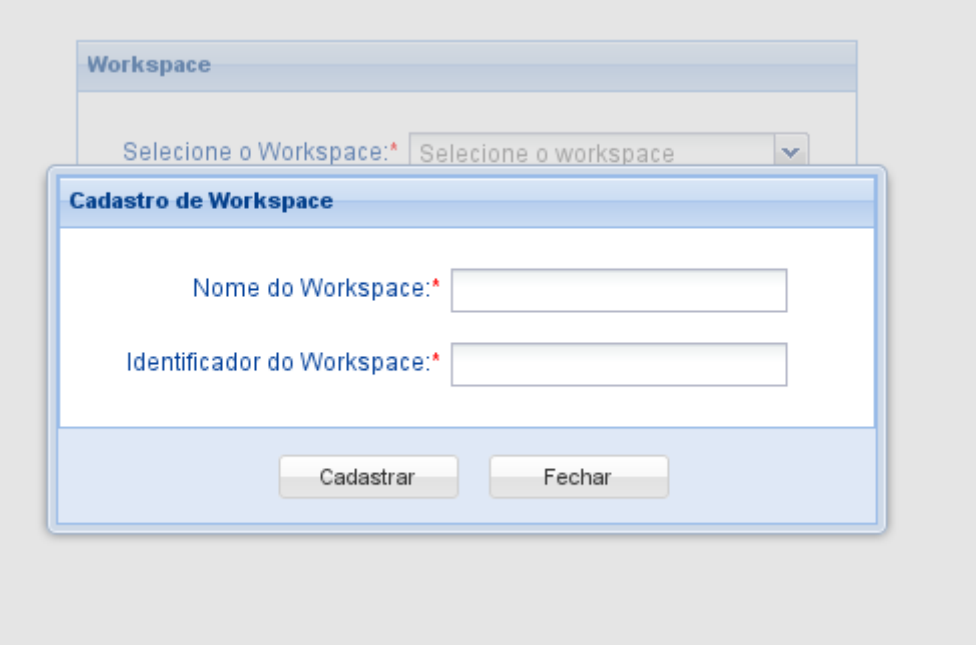


**Figura 6.1:** Cadastro de espaço de trabalho

usuários. Realizada a busca será exibida uma tabela com os usuários conforme a Figura 6.2.

Nessa tabela será exibido o nome, login e status de compartilhamento do espaço de trabalho com aquele usuário. Para incluir um compartilhamento ou excluí-lo basta clicar no ícone da coluna "Compartilhamento".

Concluída esta etapa, os desenvolvedores poderão compartilhar o mesmo espaço de trabalho para desenvolvimento dos projetos.

### 6.3.3 Cadastrar projetos

Dentro dos espaços de trabalho devem ser criados projetos que irão conter os modelos que serão parte da aplicação, ou seja, os modelos de uma aplicação devem estar contidos dentro de um projeto. Portanto, a criação de um projeto é uma etapa fundamental para o desenvolvimento de uma nova aplicação.

Os projetos podem ser criados por usuários com perfil COORDENADOR ou GERENTE. Um usuário poderá acessar apenas os projetos que forem compartilhados com ele. Dessa forma, evita-se que usuários que compartilham o mesmo espaço de trabalho acessem projetos aos quais não estão relacionados.

Para realizar o cadastro de um projeto é necessário acessar o menu "Arquivo", submenu "Novo" e clicar na opção "Projeto", ou ainda através de um menu de contexto localizado na região esquerda da interface gráfica onde se encontra a árvore de projetos. Ao entrar no cadastro de projeto o usuário deverá fornecer um nome ao projeto conforme a Figura

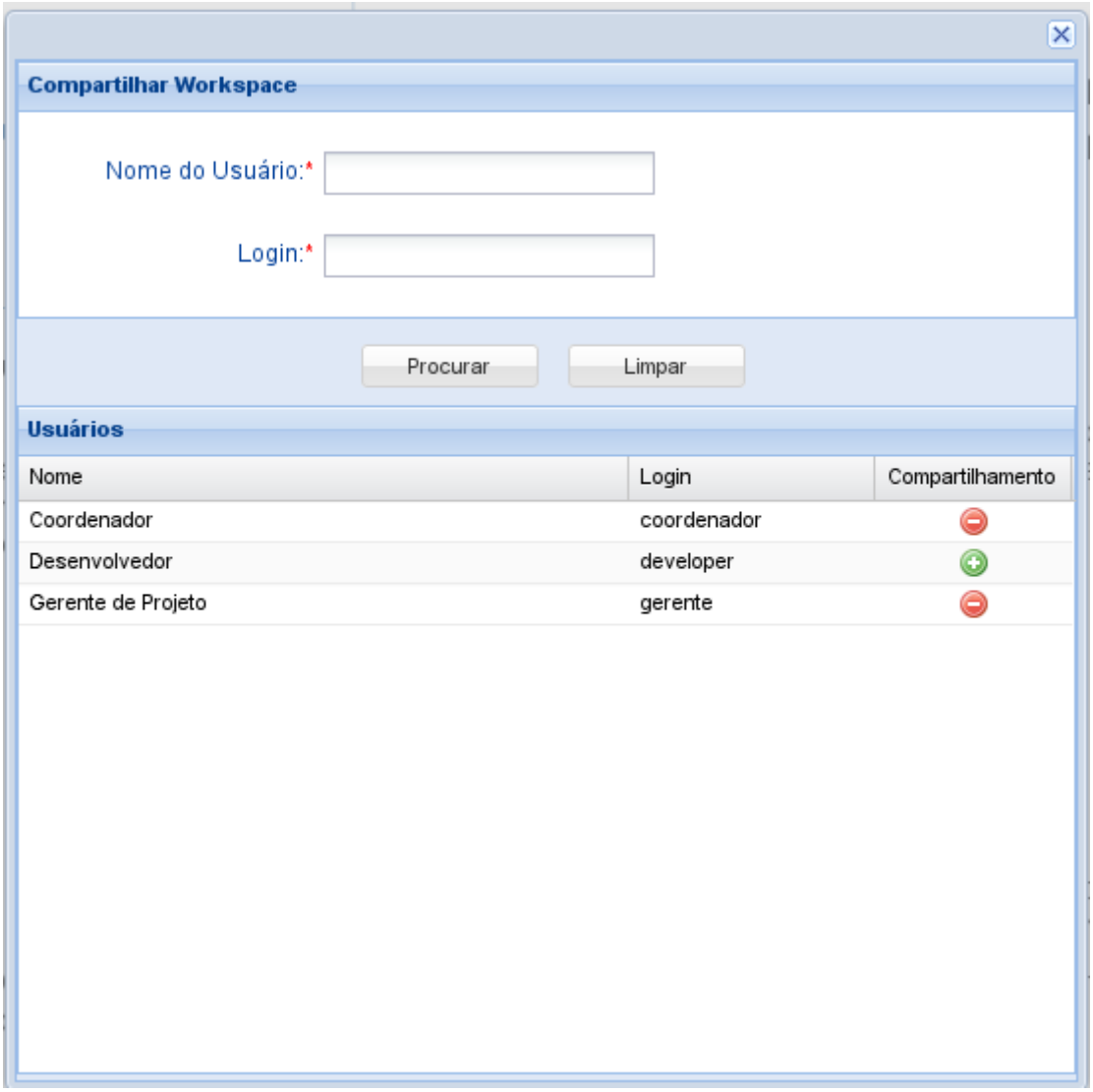


**Figura 6.2:** Compartilhar espaço de trabalho

6.3. Não é permitido cadastrar mais de um projeto com o mesmo nome dentro do mesmo espaço de trabalho.

Caso seja necessário renomear algum projeto basta selecionar o projeto na árvore de projetos e acessar o menu "Projeto" e clicar na opção "Renomear". Para excluir um projeto é necessário acessar o menu "Projeto" e clicar na opção "Excluir", lembrando que apenas os criadores do projeto podem renomeá-lo ou excluí-lo.

### 6.3.4 Compartilhar projetos

Depois de cadastrado o projeto, o criador pode compartilhá-lo com os demais desenvolvedores da equipe, assim permitindo que estes desenvolvedores acessem o projeto e colaborem no desenvolvimento dos modelos.

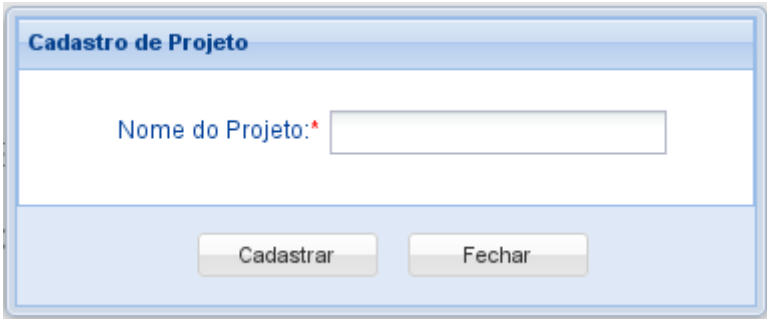


**Figura 6.3:** Criar projeto

Para compartilhar o projeto é preciso acessar o menu "Projeto" e clicar na opção "Compartilhar". Este processo é semelhante ao compartilhamento de workspace.

Realizado o compartilhamento, os desenvolvedores poderão trabalhar de forma simultânea na criação de modelos para o projeto.

### 6.3.5 Cadastrar recursos

Conforme previsto no estilo arquitetural REST os recursos são a parte fundamental na modelagem de um Serviço Web RESTful. Assim, estes devem ser cadastrados antes da criação dos modelos de domínio.

Portanto, para realizar o cadastro de um recurso o usuário deve acessar o menu "Gerenciar", submenu "Recursos" e clicar na opção "Cadastrar". Para o cadastro é necessário informar o nome do recurso e um identificador, que devem ser únicos dentro de um projeto, e caso seja necessário informar qual o recurso pai deste, ou seja, se este recurso a ser criado depende de um recurso criado anteriormente. Assim, é possível definir uma hierarquia de recursos dentro da aplicação.

Além disso, é fundamental que no cadastro de recurso seja descrito o relacionamento deste com os demais recursos já existentes. Caso o recurso a ser cadastrado possua um recurso pai é obrigatório informar este relacionamento durante o cadastro.

A Figura 6.4 ilustra a interface gráfica para cadastro de recursos.

Caso queira alterar ou excluir um recurso, o usuário deve acessar o menu "Gerenciar", submenu "Recursos" e clicar na opção "Alterar/Excluir". Na janela exibida o usuário deve primeiro realizar uma consulta através do botão "Consultar", em seguida clicar no ícone da ação desejada. É possível excluir apenas os recursos que não estão sendo utilizados em nenhum modelo e que não tenham nenhum filho na hierarquia de recursos.

### 6.3.6 Cadastrar atores

Ainda antes de cadastrar os modelos de domínio é necessário definir quais serão os atores do sistema. Esta etapa é importante para realizar o cadastro de casos de uso.

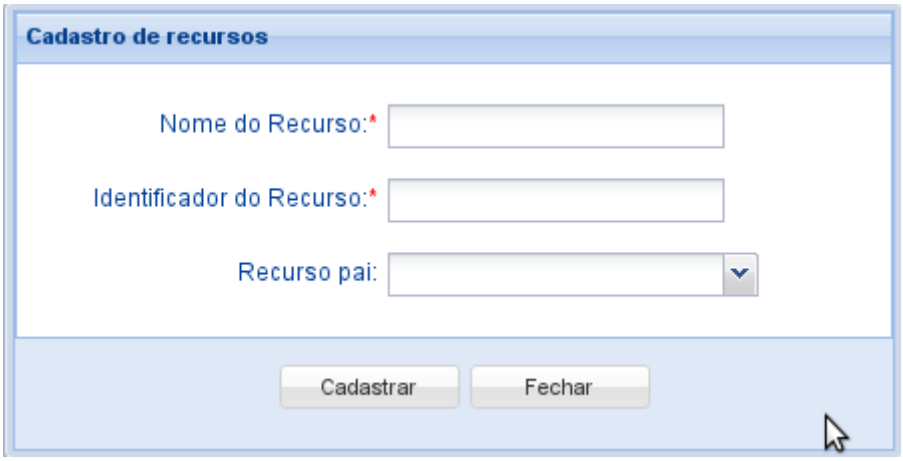


**Figura 6.4:** Cadastrar recurso

O cadastro de atores é bem simples, e para realizá-lo o usuário deve acessar o menu "Gerenciar", submenu "Atores" e clicar na opção "Cadastrar". Os atores são identificados apenas por um nome único dentro do projeto. A exclusão e alteração de atores funciona da mesma forma que os recursos. Não é permitido excluir atores que façam parte de algum caso de uso.

### 6.3.7 Cadastrar casos de uso

O primeiro tipo de modelo de domínio que deve ser criado no ambiente colaborativo RestMDD são os casos de uso. Nesse cadastro é possível identificar qual o subconjunto da interface uniforme deve ser exposto para um determinado recurso e as atividades que devem ser executadas para a realização deste caso de uso.

Para o cadastro do caso de uso o usuário deve acessar o menu "Arquivo", submenu "Novo", submenu "Modelo" e clicar na opção "Caso de uso". Será exibida uma janela onde inicialmente o usuário deve fornecer um nome, uma descrição, qual será o ator, o tipo e o recurso que este caso de uso está relacionado conforme Figura 6.5.

Ao proceder com este cadastro o modelo de caso de uso recém-criado deverá aparecer na árvore de projetos, dentro da fase de Análise e uma nova aba será aberta para o usuário já no modo "Editar", permitindo que o desenvolvedor cadastre a pré-condição, a pós-condição e as atividades do fluxos principal e alternativo, conforme é apresentado na Figura 6.6.

O modelo de caso de uso será utilizado como referência para a criação dos demais modelos de domínio da aplicação. É permitido criar apenas um tipo de caso de uso para cada recurso, permitindo assim que seja mantida a interface uniforme prevista no estilo arquitetural REST.

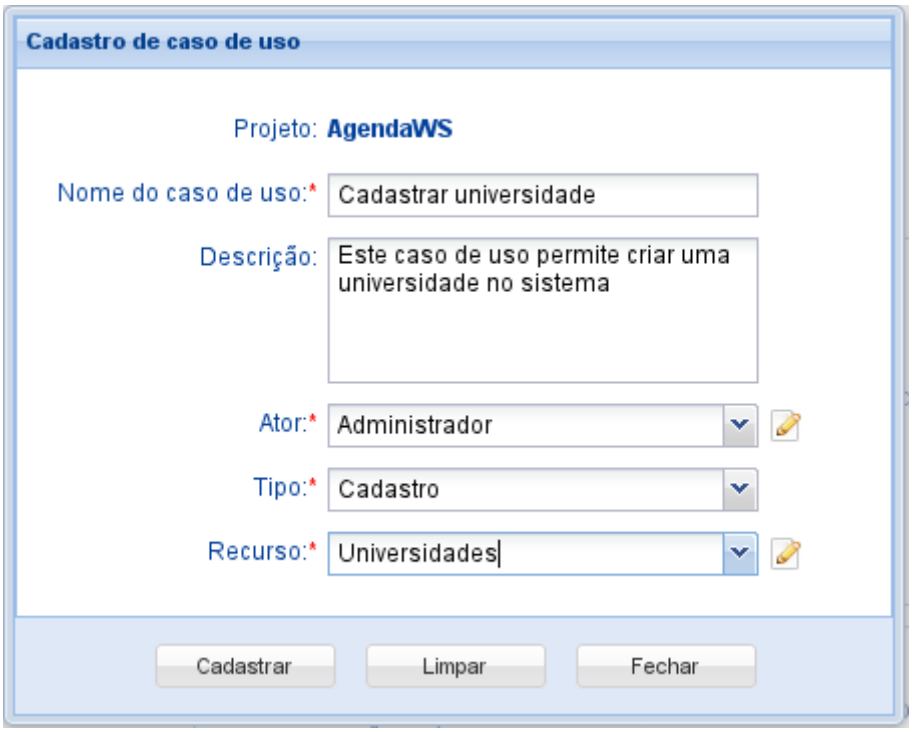


**Figura 6.5:** Cadastro de caso de uso

### 6.3.8 Construir diagramas de casos de uso

Depois de cadastrado um conjunto de casos de uso, o desenvolvedor pode construir uma representação gráfica destes modelos por meio de um diagrama de casos de uso.

Um diagrama de casos de uso é capaz de mostrar as relações entre os atores e casos de uso dentro de um sistema, e ainda é capaz de proporcionar uma visão geral dos requisitos expondo o escopo do projeto (Ambler, 2005). Este diagrama faz parte da UML e sua estrutura será representada no formato XMI (XML Metadata Interchange) (OMG, 2007), permitindo o intercâmbio com outras ferramentas de modelagem UML.

Para construir um diagrama de casos de uso o usuário deve acessar o menu “Arquivo”, submenu “Novo”, submenu “Modelo” e clicar na opção “Diagrama de caso de uso”. Na janela que será aberta o usuário deve escolher quais casos de uso farão parte do diagrama e informar um nome para este modelo. O ambiente colaborativo RestMDD irá analisar os atores envolvidos na construção deste diagrama e construirá a representação gráfica do mesmo. Após construído, os usuários poderão acessá-lo e modificá-lo incluindo novos casos de uso ou ainda removendo os já existentes.

### 6.3.9 Construir diagramas de atividades

Os diagramas de casos de uso são capazes de mostrar o escopo do projeto como um todo, identificando cada um dos cenários de uso da aplicação, porém não são capazes de mostrar

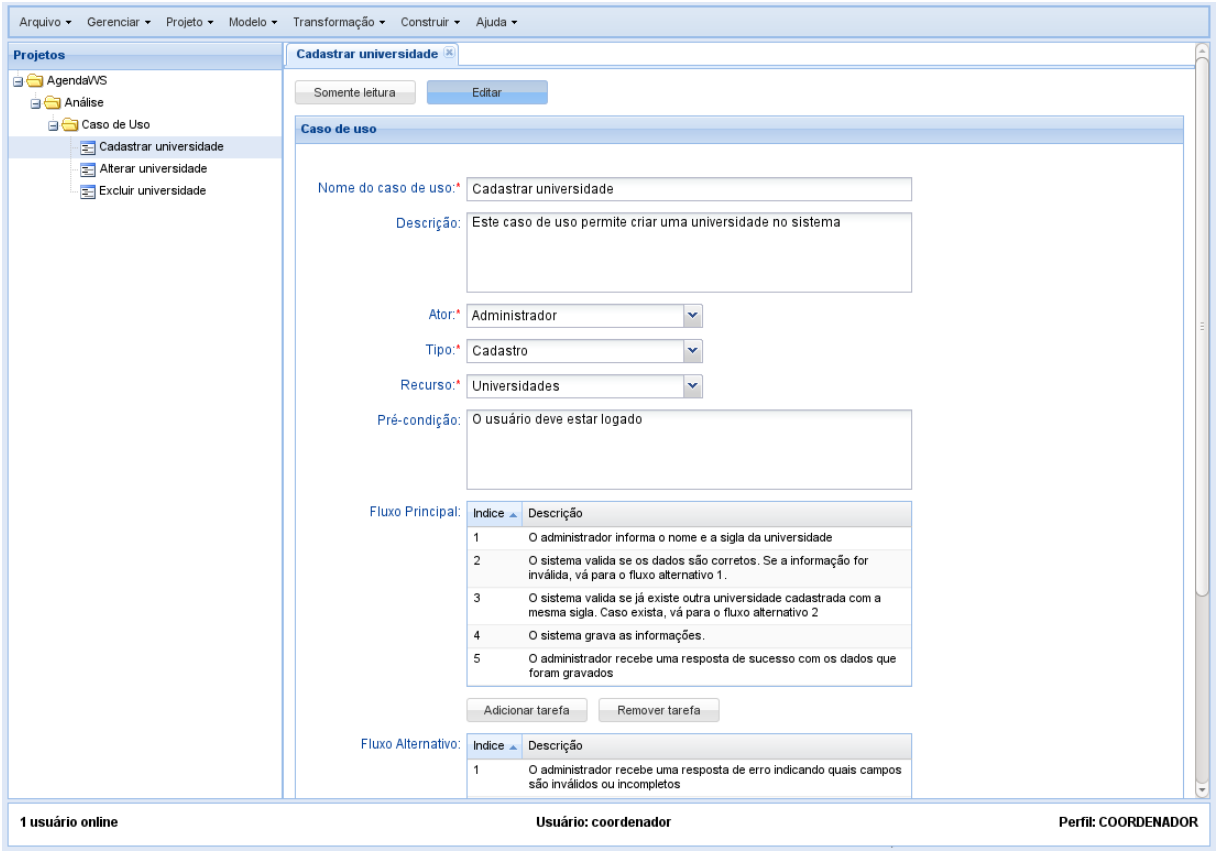


**Figura 6.6:** Editar caso de uso

em detalhes como acontece o fluxo de atividades dentro de cada um desses cenários. Para essa finalidade, são utilizados os diagramas de atividades.

Os diagramas de atividades têm a capacidade de apresentar cada tarefa que deve ser executada para que um caso de uso se concretize, porém sem expôr detalhes de implementação, por isso este também é considerado um modelo de domínio. Os diagramas de atividades também são modelos da UML e a representação da estrutura destes diagramas são definidas no formato XMI.

O principal objetivo destes diagramas é modelar a lógica de negócio envolvida na execução de um caso de uso. Portanto, para a linguagem RestML será construído um diagrama de atividades para cada caso de uso registrado anteriormente.

Para construir um diagrama de atividades o usuário deve acessar o menu “Arquivo”, submenu “Novo”, submenu “Modelo” e clicar na opção “Diagrama de atividades”. Na janela apresentada o usuário deve escolher qual caso de uso será usado como entrada para o diagrama de atividades.

Estes diagramas são construídos de forma automatizada, sendo que cada tarefa cadastrada nos fluxos principal e alternativo do caso de uso são transformadas em atividades. Durante o cadastro de caso de uso, sempre que seja deseja realizar uma validação para

um propor um fluxo alternativo é necessário definir uma expressão de controle, que será transformada em um elemento de condição dentro do diagrama de atividades.

É possível ainda atualizar um diagrama de atividades quando o caso de uso correspondente também tiver sido atualizado. Para isso, o ambiente RestMDD irá verificar todas as vezes que um diagrama de atividades for acessado se o seu caso de uso correspondente foi atualizado e caso isso ocorra questionará o usuário se deseja atualizar o respectivo diagrama de atividades. É possível ainda atualizar manualmente o diagrama de atividades clicando no botão "Atualizar".

### 6.3.10 Cadastrar representações

Conforme descrito no Capítulo 2 o recurso é o elemento chave na abstração da arquitetura REST e deve ser representado por uma sequência de bytes com os dados deste recurso, que recebe o nome de representação. Os componentes do estilo arquitetural REST que executam ações sobre um recurso utilizam representações para capturar o estado atual do recurso ou para atualizar seu estado.

Em geral, as representações são mais que apenas dados isolados, na maioria das vezes estas representações são hiperdocumentos, que contêm *hyperlinks* para os recursos associados. Assim, é fundamental modelar quais dados uma representação deve conter para que expresse de maneira clara o conteúdo do recurso.

Conforme previsto na linguagem RestML, para cada recurso cadastrado no ambiente colaborativo RestMDD é necessário modelar sua representação. Portanto, para realizar o cadastro de uma representação os usuários devem acessar o menu "Gerenciar", submenu "Representações" e clicar na opção "Cadastrar". Nessa modelagem, o usuário deverá incluir todos os atributos que o recurso deve conter, informando o tipo do dado (inteiro, booleano, ponto flutuante ou texto), o nome do atributo, bem como se o valor é obrigatório e possíveis validações, como por exemplo valor mínimo e máximo para atributos do tipo inteiro e quantidade de caracteres mínima e máxima para atributo textuais.

### 6.3.11 Cadastrar exceções

Conforme previsto na arquitetura orientada a recursos é importante considerar as condições de erro que um recurso pode retornar. Portanto, para atender esse requisito o usuário deve cadastrar todos os tipos de falhas que podem acontecer durante a execução de um caso de uso.

Esse cadastro deve ser realizado no menu "Gerenciar" clicando na opção "Exceções". Uma exceção deve identificada por um nome e um tipo que devem ser únicos. Entre

os tipos de exceção estão: Recurso não encontrado, Requisição mal-formada, Falha de negócio e Erro desconhecido.

### 6.3.12 Construir diagramas de classes para os recursos

Neste momento todas as informações necessárias para a modelagem de classes da aplicação já foram definidas e portanto o ambiente RestMDD consegue gerar um diagrama de classes para os recursos cadastrados anteriormente. O diagrama de classes também é parte da UML e portanto sua estrutura será definida no formato XMI. Esses diagramas são apresentados na árvore de projetos na seção Projeto.

Com este diagrama contendo os recursos é possível identificar claramente a hierarquia de recursos presente na aplicação. Esta hierarquia é montada da seguinte forma:

- Uma classe representando a aplicação será criada com o nome do projeto. Esta classe deve receber o estereótipo *Application* e contêm apenas um método chamado "listarRecursos". Este método retornará uma lista de *hyperlinks* para todos os recursos da aplicação e deve conter o estereótipo *Retrieve*;

- Para cada recurso será criada uma classe com o nome do identificador do recurso concatenado com a palavra *Resource*. Estas classes devem receber o estereótipo *Resource*;

- Os métodos das classes de recursos são definidos de acordo com os tipos de casos de uso cadastrados para aquele recurso. Para o tipo Cadastro é criado um método com o estereótipo *Create*. Já o tipo Consulta o estereótipo a ser utilizado deve ser o *Retrieve*. Para o tipo Alteração deve ser utilizado o estereótipo *Update*. Por fim, para os casos de uso de exclusão deve ser aplicado o estereótipo *Delete*;

- Para toda classe de recurso será criado também um método chamado *validate*, que será responsável por validar os dados de entrada recebidos pelo recurso;

- O relacionamento das classes é realizado de acordo com a hierarquia proposta no cadastro de recursos, ou seja, os recursos que não possuírem recurso pai definido estarão associados a classe que representa a aplicação. Já os demais recursos serão associados ao seu respectivo recurso pai. Os relacionamentos declarados durante o cadastro do recurso também devem ser considerados para a construção do diagrama.

Para construir esse diagrama o usuário deve acessar o menu "Arquivo", submenu "Novo", submenu "Modelo" e clicar na opção "Diagrama de classes para recursos". Nesse momento será construído um diagrama de classes conforme descrito anteriormente. Caso

já exista um diagrama de classes para os recursos da aplicação, o usuário será questionado se deseja atualizar o diagrama já existente.

### 6.3.13 Construir diagramas de classes para as representações

Depois de modelada as representações, o ambiente RestMDD é capaz de gerar um diagrama de classes com o relacionamento entre as representações conforme mostra a Figura 6.7. Nessa figura é possível observar que as classes contêm o estereótipo *Representation*. Os atributos que não foram identificados como obrigatórios devem conter o estereótipo *Optional* e os que atributos que necessitarem de validação devem conter o estereótipo *Range*.

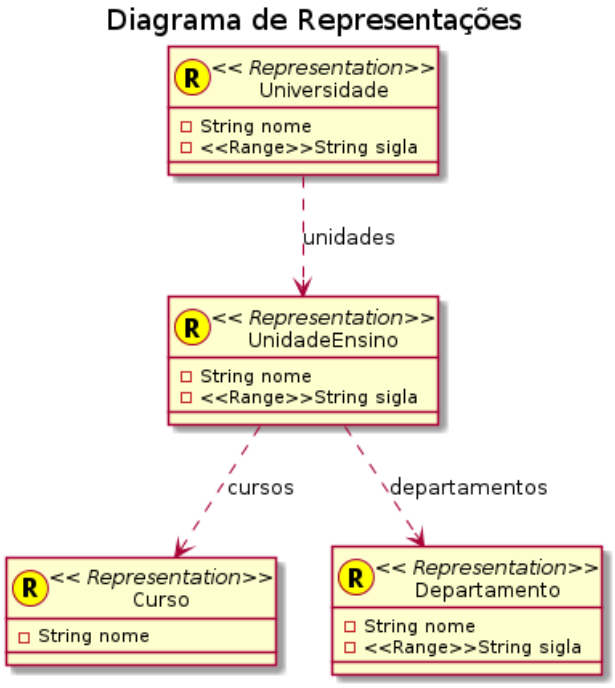


**Figura 6.7:** Diagrama de classes para representações

Para construir esse diagrama o usuário deve acessar o menu "Arquivo", submenu "Novo", submenu "Modelo" e clicar na opção "Diagrama de classes para representações". O comportamento do sistema nesse caso é igual a construção do diagrama de classes para recursos.

### 6.3.14 Construir diagramas de classes para as exceções e serviços

Para cada recurso modelado deve ser criada uma classe de serviço para realizar a regra de negócio exigida pelo caso de uso. Esses serviços devem acessar uma fonte de dados para

buscar e gravar informações. O acesso aos dados deve ser realizado através de uma classe única responsável pela comunicação com a fonte de dados.

Essas classes de serviços e a classe de acesso a dados são modeladas em um diagrama de classes conforme pode ser visto Figura 6.8.

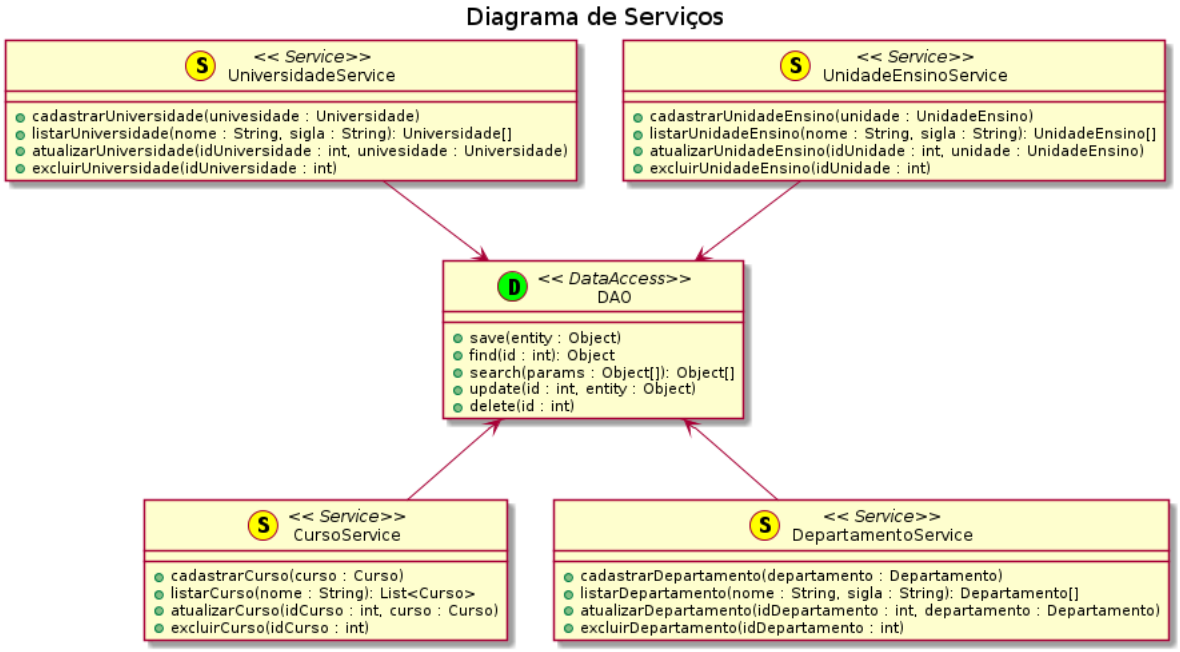


**Figura 6.8:** Diagrama de classes para serviços

Além disso, os serviços e recursos podem retornar condições de erro que foram definidas no cadastro de exceções. Essas exceções também são modeladas em um diagrama de classes conforme Figura 6.9.

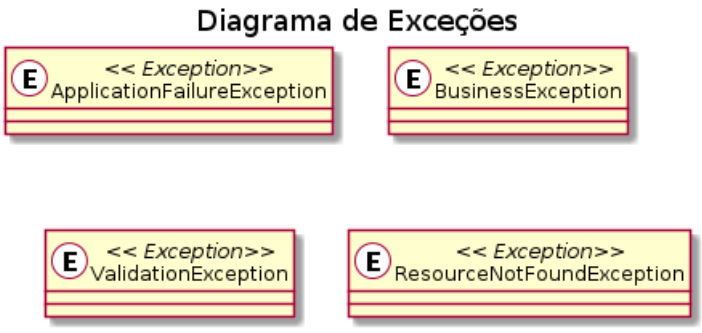


**Figura 6.9:** Diagrama de classes para exceções

Para construir esses diagramas o usuário deve acessar o menu “Arquivo”, submenu “Novo”, submenu “Modelo” e clicar na opção “Diagrama de classes para serviços e exceções”. O comportamento do sistema nesse caso é idêntico a construção dos demais diagrama de classes.

### 6.3.15 Construir diagramas de sequência

Os diagramas de sequência são modelos da UML usados para representar interações entre objetos de um cenário, realizadas através de métodos, dando ênfase na ordenação temporal em que as mensagens são trocadas entre os objetos. Estes diagramas também tem sua estrutura representada no formato XMI.

Usando a linguagem RestML, o usuário deverá criar um diagrama de sequência para cada diagrama de atividades já existente no projeto. Para modelar estes diagramas de sequência o usuário deve acessar o menu "Arquivo", submenu "Novo", submenu "Modelo" e clicar na opção "Diagrama de sequência". Na janela que será exibida o usuário deverá informar qual o diagrama de atividades que servirá de referência para a construção do diagrama de sequência e para cada atividade deve escolher quais componentes irão trocar mensagem e ainda qual o método será invocado nessa mensagem.

Essa modelagem possui algumas restrições:

- É permitido apenas um componente de cada tipo;
- O ator deve iniciar o fluxo de mensagens dentro do diagrama e deve se comunicar apenas com o recurso;
- O recurso só pode enviar mensagens para o seu serviço, o ator ou para ele mesmo (auto-mensagem);
- O serviço se comunica apenas com o recurso e com o objeto de acesso a dados;
- O objeto de acesso a dados apenas se comunica com o serviço.

Nesse momento, todos os modelos independentes de plataforma (PIM) previstos pela linguagem RestML já foram modelados. Agora estes modelos sofrerão transformações para se tornarem modelos específicos de plataforma (PSM).

### 6.3.16 Transformação de modelos PIM para modelos PSM

A transformação dos modelos independentes de plataforma (PIM) para modelos específicos de plataforma (PSM) será realizada utilizando o padrão QVT (*Queries/Views/Transformations*) (OMG, 2011) definido pela OMG.

Para realizar essa transformação, o usuário deve acessar o menu "Transformação", e clicar na opção "Construir modelos específicos da plataforma". A transformação segue as regras descritas na seção 5.2.6.

Como resultado dessa transformação, são gerados novos diagramas de classes e sequência a partir dos já existentes só que agora contendo os estereótipos específicos da plataforma desejada. Estes diagramas são apresentados na árvore de projetos dentro da seção Plataforma.

### 6.3.17 Geração de código-fonte

Depois de contruídos todos os modelos especificos da plataforma, o ambiente RestMDD é capaz de gerar o código-fonte da aplicação transformando os modelos do formato XMI para código Java. Esta transformação é realizada com o apoio da *engine Velocity*[3].

Em suma, o *Velocity* é uma *engine* Java baseada em *templates* que pode ser usada para gerar código-fonte, documentos e outros artefatos a partir de modelos que são transformados de acordo com *templates* pré-definidos. Assim, é possivel criar componentes nas mais diversas linguagens de programação e plataformas apenas criando *templates* que suportam a transformação dos modelos XMI em código-fonte.

Para a linguagem RestML foram construídos *templates* para que fosse possível transformar os diagramas criados anteriormente em código-fonte na linguagem Java para a plataforma JavaEE utilizando os frameworks Spring e JAX-RS.

Para realizar essa etapa do processo de desenvolvimento, o usuário deve acessar o menu "Transformação", e clicar na opção "Construir código-fonte". O usuário então deverá informar um nome de pacote e a versão do código-fonte. Como resultado, será construído um projeto Maven[4] com quatro módulos: *common*, *model*, *business* e *web*. O código-fonte presente em cada um desses módulos é apresentado na árvore de projetos dentro da seção "Codificação". Um exemplo de código-fonte gerado por esse processo encontra-se no Apêndice A.

### 6.3.18 Empacotamento

Para melhor distribuição da aplicação e também para que seja possível a implantação da mesma em um servidor Web é fundamental que seja gerado um pacote contendo todo o código-fonte compilado da aplicação.

Assim, o ambiente RestMDD permite o empacotamento da aplicação em um arquivo WAR (*Web Archive*), previsto pela especificação JavaEE.

Para realizar o empacotamento o usuário deve selecionar o menu "Construir", entrar no submenu "Empacotamento" e clicar na opção "Construir pacote". Uma janela será exibida com todos os projetos que tem código-fonte gerado e sua respectiva versão. É permitido

[3]http://velocity.apache.org
[4]http://maven.apache.org/

criar pacotes de um mesmo projeto para diferentes versões. O usuário deve selecionar o projeto desejado e clicar no botão "Gerar pacote". Caso já tenha sido realizado o empacotamento desejado para o projeto escolhido, o usuário será questionado se deseja atualizar o pacote já existente.

O usuário é capaz também de consultar todos os pacotes gerados através do menu "Construir", acessar o submenu "Empacotamento" e clicar na opção "Consultar pacotes". Uma janela contendo todos os pacotes gerados no espaço de trabalho será exibida e o usuário pode realizar o download desse pacote ou ainda excluí-lo.

## 6.4 Considerações finais

Neste capítulo foi apresentado o ambiente colaborativo RestMDD e como desenvolver serviços Web RESTful usando a linguagem específica de domínio RestML seguindo a arquitetura orientada a recursos proposta por (Richardson e Ruby, 2007). Foram descritos todos os passos para a criação dos serviços Web RESTful desde a criação do espaço de trabalho compartilhado até o empacotamento do código-fonte gerado.

Capítulo

7

# Experimento

A experimentação é uma disciplina muito importante para diversas áreas de pesquisa, incluindo engenharia de software. Estudos empíricos são necessários para avaliar a eficácia ou quão promissor processos, metodologias e técnicas são em relação a outras abordagens. Assim, neste capítulo é apresentado um estudo experimental sobre o ambiente colaborativo RestMDD, a fim de melhor compreender o impacto dessa abordagem nas atividades de desenvolvimento de software.

## 7.1 Engenharia de software experimental

A realização de experimentos na Engenharia de Software tem como objetivos a caracterização, avaliação, previsão, controle e melhoria em relação ao produtos, processos, recursos, modelos e outros resultados e atividades do processo de desenvolvimento de software (Travassos et al., 2002). Além de avaliar técnicas e atividades da Engenharia de Software, a experimentação contribui com a evolução, previsão, compreensão, controle e aperfeiçoamento do processo de desenvolvimento de software e produtos (Basili et al., 1986).

A experimentação pode ser vista como um processo sistemático composto de fases, subprocessos e produtos gerados ao final de cada fase. As fases propostas por (Wohlin et al., 2000) são: definição do experimento, planejamento, operação, análise e interpretação, e por fim apresentação e empacotamento.

A presente dissertação de mestrado tenta investigar as vantagens no uso do desenvolvimento orientado a modelos de maneira colaborativa para a produção de Serviços Web RESTful em termos de produtividade. Dessa forma, foi conduzido um estudo experimental conforme as fases propostas por (Wohlin et al., 2000). Cada uma dessas fases serão explicadas em detalhes nas próximas seções.

## 7.2 Definição do experimento

A definição do experimento é a primeira fase de um estudo experimental. A fase de definição descreve os objetivos, o objeto do estudo, o foco da qualidade, o ponto de vista e o contexto. Como resultado, a fase de definição fornece a direção geral do experimento, o seu escopo, a base para a formulação das hipóteses e as notações preliminares para a avaliação da validade (Travassos et al., 2002).

Portanto, para a definição deste experimento foi utilizado o paradigma GQM (*Goal-Question Metric*) (Basili et al., 1994), que prevê a definição dos objetivos da avaliação, que são rastreados em um conjunto de dados que representam os objetivos de forma operacional, e a interpretação destes dados com respeito aos objetivos definidos. O rastreamento entre os objetivos e os dados é feito através de questões que caracterizam os objetivos de forma mais específica (Lucrédio, 2009).

### 7.2.1 Objetivos do estudo

Seguindo o formato sugerido no GQM, foi definido o seguinte objetivo para esta avaliação:

> **Analisar** a abordagem orientada a modelos colaborativa para construção de Serviços Web RESTful **com o objetivo de** determinar se ela promove aumento da produtividade no processo de desenvolvimento de software, quando comparada com o desenvolvimento não orientado a modelos, **com respeito aos** artefatos de software produzidos **do ponto de vista** do desenvolvedor de software **no contexto de** projetos de software que visam a construção de Serviços Web RESTful.

Esse objetivo pode ser descrito pelas seguintes questões:

**Q1:** Analisando-se um mesmo projeto desenvolvido com e sem a abordagem, é possível observar um aumento da produtividade no processo de desenvolvimento de software no projeto que utilizou a abordagem ?

**Q2:** O uso de um ambiente colaborativo influenciou no aumento da produtividade ?

**Q3:** Os desenvolvedores que usaram a abordagem orientada a modelos tiveram dificuldades ou limitações que prejudicaram as atividades de desenvolvimento ?

O objetivo do experimento é demonstrar que o uso de uma abordagem orientada a modelos de maneira colaborativa nesse cenário pode propiciar um aumento de produtividade durante o processo de desenvolvimento de software. Assim, a questão Q1 busca observar este aumento da produtividade observando duas abordagens: uma delas com o uso do ambiente colaborativo RestMDD e outra com o uso de um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*) e ferramenta de modelagem UML convencional.

A colaboração entre os desenvolvedores é fundamental para a realização do experimento e construção dos serviços propostos. Assim, a questão Q2 tem por finalidade avaliar os benefícios no uso do ambiente colaborativo para o desenvolvimento do software em termos de produtividade comparado com outras formas de colaboração.

Já a questão Q3 pretende verificar se a utilização do MDD pode ocasionar problemas durante o desenvolvimento do software que podem vir a prejudicar a produtividade dos desenvolvedores.

### 7.2.2 Métricas

Para responder as questões propostas em 7.2.1, foram definidas formas qualitativas e quantitativas de obtenção dos dados. Assim, se torna possível avaliar a percepção subjetiva dos indivíduos participantes do experimento em conjunto com medidas extraídas durante a execução do estudo empírico.

A questão Q1 prevê a avaliação da produtividade no processo de desenvolvimento de software tomando como base dois projetos de software usando abordagens distintas. A medição da produtividade no software é semelhante a outras formas de medição da produtividade, isto é, ela é medida como a proporção de unidades de saída divididas por unidades de entrada. As entradas consistem no esforço dispendido para produzir um determinado resultado. Para software, os resultados tangíveis são o código-fonte e documentação (IEEE, 1993).

As métricas utilizadas para a avaliação da produtividade foram obtidas no documento *IEEE Standard for Software Productivity Metrics* (IEEE, 1993) que padroniza uma terminologia para as métricas de produtividade de software assegurando a compreensão dos dados medidos, tanto na construção de código-fonte quanto para documentação.

Estas métricas foram escolhidas pois utilizam a relação produtos x esforço, o que as tornam mais estáveis se comparadas às relações de valor como por exemplo produto x unidades monetárias, que podem variar conforme as necessidades de mercado tornando-as métricas mais instáveis.

Assim, a fórmula para cálculo de produtividade do produto $a$ é definida como:

$$P_a = \frac{S_a}{E_a}$$

Onde:
$S_a$ = Primitiva de saída
$E_a$ = Esforço

A primitiva de saída utilizada para medir as funcionalidades entregues do software é o ponto por função (Albrecht e Gaffney, 1983). Já o esforço a ser considerado nesse cenário é o tempo gasto para implementação das funcionalidades. Assim, a medida final de produtividade nos permitirá descobrir quantos pontos por função os desenvolvedores podem implementar por unidade de tempo.

Para avaliar a questão Q2 referente aos possíveis benefícios que o uso de um ambiente colaborativo pode proporcionar na produtividade dos desenvolvedores, foi aplicado um questionário com os indivíduos participantes do experimento contendo perguntas que avaliaram se os benefícios desejados foram claramente percebidos pelos sujeitos. Segue as perguntas realizadas no questionário, com respostas em aberto:

- O uso de um espaço de trabalho compartilhado contribui para a redução do tempo gasto no desenvolvimento do projeto como um todo?
- A capacidade de trabalhar de forma isolada garante a integridade dos artefatos produzidos pelo desenvolvedor?
- O desenvolvimento do projeto de forma simultânea reduz o tempo de construção dos artefatos?
- Quais foram as vantagens de se utilizar um ambiente colaborativo no desenvolvimento do software?
- Quais foram as desvantagens de se utilizar um ambiente colaborativo no desenvolvimento do software?
- Na sua opinião, houve aumento de produtividade no desenvolvimento do software devido ao uso de um ambiente colaborativo?

A questão Q3 consiste em avaliar especificamente se a prática do MDD no contexto da construção Serviços Web RESTful pode trazer dificuldades que sejam prejudiciais na produtividade durante o processo de desenvolvimento de software. Para avaliar essas possíveis dificuldades de utilização da abordagem MDD, foram incluídas as seguintes perguntas, no questionário realizado na questão Q2, referente às dificuldades percebidas pelos participantes do experimento:

- Quais foram as dificuldades para o aprendizado da abordagem?
- Quais foram as dificuldades para a definição e cadastro dos recursos e atores do sistema?
- Quais foram as dificuldades para a construção dos modelos de casos de uso?
- Quais foram as dificuldades para a construção dos diagramas de casos de uso?
- Quais foram as dificuldades para a construção dos diagramas de atividades?
- Quais foram as dificuldades para o cadastro das representações?
- Quais foram as dificuldades para o cadastro das exceções?
- Quais foram as dificuldades para a construção dos diagramas de classes?
- Quais foram as dificuldades para a construção dos diagramas de sequência?
- Quais foram as dificuldades para a construção dos modelos específicos da plataforma?
- Quais foram as dificuldades para a geração do código-fonte?
- Quais foram as dificuldades para o empacotamento do código-fonte?
- Cite outras dificuldades percebidas durante a utilização da abordagem de MDD.
- Na sua percepção, as dificuldades encontradas fez com que o tempo total no desenvolvimento das funcionalidades fosse superior ao tempo gasto com uma abordagem não orientada a modelos?
- O uso do ambiente colaborativo RestMDD deve ser desconsiderado devido as dificuldades em seu uso?

## 7.3 Planejamento do experimento

A fase de planejamento vem logo a seguir da fase de definição do experimento. Nessa fase ocorre o projeto do experimento com a seleção do contexto e dos indivíduos, formulação das hipóteses, seleção das variáveis, a preparação da instrumentação e a consideração da validade do experimento. O resultado dessa fase apresenta o experimento totalmente elaborado e pronto para execução (Travassos et al., 2002).

### 7.3.1 Hipóteses

A hipótese deste estudo é que a construção de Serviços Web RESTful usando uma abordagem orientada a modelos de maneira colaborativa pode aumentar a produtividade dos desenvolvedores em relação a abordagens não orientada a modelos. O aumento de produtividade, nesse contexto, consiste que os programadores levam um tempo menor para modelar e codificar um Serviço Web RESTful.

O uso de abordagens orientada a modelos para a construção de Serviços Web RESTful ainda é uma área pouco explorada, e portanto as hipóteses propostas podem ser utilizadas como referência para novos estudos na área.

Assim, foram definidas as seguintes hipóteses para esta avaliação que podem ser descritas com base nas questões e métricas definidas anteriormente:

**Hipótese 1**

**Hipótese Nula($H_{0a}$):** Não existe diferença de produtividade entre a abordagem orientada a modelos colaborativa e a abordagem não orientada a modelos (codificação).

$H_{0a}$: Produtividade(MDD) = Produtividade(Codificação)

**Hipótese Alternativa($H_{1a}$):** Existe aumento de produtividade com o uso da abordagem orientada a modelos colaborativa em relação abordagem não orientada a modelos (codificação).

$H_{1a}$: Produtividade(MDD) > Produtividade(Codificação)

**Hipótese 2**

**Hipótese Nula($H_{0b}$):** O uso de um ambiente colaborativo não influenciou na produtividade dos desenvolvedores.

**Hipótese Alternativa($H_{1b}$):** O uso de um ambiente colaborativo proporcionou um aumento na produtividade dos desenvolvedores.

**Hipótese 3**

**Hipótese Nula($H_{0c}$):** Os desenvolvedores que usaram a abordagem orientada a modelos tiveram dificuldades que prejudicaram sua produtividade.

**Hipótese Alternativa($H_{1c}$):** Os desenvolvedores que usaram a abordagem orientada a modelos não tiveram dificuldades que prejudicaram sua produtividade.

### 7.3.2 Seleção do contexto

Uma atividade importante no âmbito universitário consiste na apresentação de Trabalhos de Conclusão de Curso (TCC), em que os alunos em geral apresentam seu trabalho final para uma banca de professores. A necessidade de comunicação entre alunos e professores a fim de coletar as informações referentes aos trabalhos de conclusão de curso desenvolvidos e as datas convenientes e disponíveis para realização das apresentações foi a principal motivação para o desenvolvimento de um sistema de agenda de grupo.

Os sistemas de agenda de grupo são calendários que tipicamente podem ser compartilhados em uma rede e visam suprir a necessidade de integração de informações para uso pessoal ou para uso de um determinado grupo. Neste contexto universitário foi proposta a evolução de um sistema de agenda de grupo já existente na universidade, com a inclusão de novas funcionalidades e permitindo que os dados pudessem ser acessados por diferentes aplicações presentes na universidade.

Para atender estes requisitos, vislumbrou-se o desenvolvimento da aplicação AgendaWS contendo Serviços Web RESTful capazes de proporcionar uma interface simples e uniforme permitindo a integração com outras aplicações. Assim, a modelagem e implementação desses serviços foi o alvo do estudo experimental. Como base para a criação dos serviços foi construído um documento de requisitos de acordo com a observação do sistema de agenda de grupo já existente. A aplicação deveria portanto ser construída a partir do zero, desde a modelagem dos casos de uso até a implantação em um Servidor Web.

### 7.3.3 Seleção dos indivíduos

Para participar do estudo foram selecionados desenvolvedores de software com formação superior e com experiência de 4 a 6 anos na indústria. Todos os indivíduos obrigatoriamente deveriam conhecer linguagem de programação Java para Web, modelagem de casos de uso, noções básicas de UML bem como uso de ferramentas de modelagem e desenvolvimento de Serviços Web RESTful usando *frameworks* Java.

No processo seletivo foram aplicados questionários ao desenvolvedores para avaliar se a experiência de cada um corresponde aos requisitos desejados, bem como quais ferramentas de desenvolvimento e modelagem UML estes estavam mais familiarizados. No processo seletivo foram escolhidos 10 indivíduos, os quais atenderam aos requisitos estabelecidos. Estes então foram divididos em 2 grupos para execução do experimento.

### 7.3.4 Variáveis

**Variáveis independentes**

- Linguagem de programação: Java
- Linguagem de modelagem: UML
- Requisitos da aplicação
- Experiência dos indivíduos
- Funcionalidades a serem desenvolvidas

**Variáveis dependentes**

- Ambiente de desenvolvimento
- Tempo gasto para modelagem dos casos de uso e diagramas UML usando ferramenta de modelagem UML - $TM_{UML}$
- Tempo gasto para codificação dos Serviços Web RESTful usando IDE - $TC_{IDE}$
- Tempo gasto para modelagem dos casos de uso e diagramas UML usando ambiente colaborativo RestMDD - $TM_{MDD}$
- Tempo gasto para codificação dos Serviços Web RESTful usando ambiente colaborativo RestMDD - $TC_{MDD}$

### 7.3.5 Instrumentação do experimento

Para que fosse possível realizar o estudo experimental conforme o planejado foi necessário a produção de um documento de requisitos contendo a especificação formal das funcionalidades que a aplicação AgendaWS deveria conter. A partir deste documento que os desenvolvedores realizariam a modelagem dos casos de uso, diagramas UML e também a codificação destes casos de uso.

Após a elaboração deste documento foram escolhidos quatro requisitos funcionais como tarefas a serem executadas pelos desenvolvedores. Os requisitos foram selecionados de tal forma que todas as tarefas previstas tenham uma complexidade semelhante, para que estas não influenciem no estudo experimental.

Como os desenvolvedores selecionados não possuíam experiência em desenvolvimento orientado a modelos, foi necessário a criação de uma ajuda dentro do ambiente colaborativo RestMDD explicando o processo de desenvolvimento usando o paradigma MDD.

### 7.3.6 Validade

Para redução da influência dos fatores que não são interesse do nosso estudo e, portanto, para aumento da validade interna do estudo, foram selecionados indivíduos com níveis de experiência semelhantes e que possuíam as compentências necessárias para o desenvolvimento do experimento. Outro fator que pode afetar a validade interna é a complexidade das funcionalidades a serem implementadas. Para reduzir esse fator foram propostas funcionalidades simples e de fácil compreensão pelos participantes.

A validade do experimento corresponde à habilidade de chegar a uma conclusão correta sobre os relacionamentos entre o tratamento e os resultados obtidos no experimento (Travassos et al., 2002). Uma ameaça à validade de conclusão é a confiabilidade das métricas. Por essa razão, nesse estudo experimental foram utilizadas as métricas para a avaliação da produtividade descritas no documento *IEEE Standard for Software Productivity Metrics* (IEEE, 1993).

## 7.4 Execução do experimento e coleta de dados

A execução do experimento deve seguir o planejamento definido na fase anterior. Nesse momento os dados experimentais são coletados para serem analisados e avaliados na fase de análise de resultados.

Um aspecto importante que deve ser considerado na fase de execução é o envolvimento da parte humana no experimento, ou seja, os participantes devem ser preparados para a experimentação do ponto de vista moral e metodológica de modo a evitar resultados errôneos devido ao mal-entendido ou falta de interesse (Travassos et al., 2002). A coleta dos dados não deve causar nenhuma interferência no estudo que está sendo realizado.

### 7.4.1 Execução do experimento

Conforme mencionado na Seção 7.3.3 foram selecionados 10 indivíduos para execução do experimento. Cada um deles deveria construir quatro funcionalidades previstas no documento de requisitos, sendo duas delas feitas com o uso da ferramenta de modelagem UML *Enterprise Architect*[1], que mostrou ser a mais familiar dentre todos os participantes, e uma IDE de sua preferência. As outras duas funcionalidades deveriam ser construídas com o uso do ambiente colaborativo RestMDD.

Este experimento foi executado individualmente com cada participante em momentos distintos. A primeira etapa da execução do experimento é a configuração do ambiente de desenvolvimento. Em primeiro lugar, o desenvolvedor deveria instalar a ferramenta

[1] http://www.sparxsystems.com.au/

*Enterprise Architect*. Por ser uma ferramenta comercial, e portanto precisa de licença para o seu uso, nesse estudo empírico foi utilizado a versão *Trial* que é gratuita. Uma vez instalado, o participante precisaria também instalar uma IDE de sua preferência.

Com as ferramentas devidamente instaladas na máquina do desenvolvedor foi possível o início da construção das funcionalidades. Essa parte do experimento foi acompanhada integralmente pelo pesquisador. Nesse instante foi disponibilizado ao participante um documento contendo quatro funcionalidades que este deveria implementar. O desenvolvedor pode analisar as funcionalidades o tempo que fosse necessário para que obtivesse um completo entendimento do funcionamento das funcionalidades a serem implementadas. O pesquisador também estava à disposição para explicar e sanar todas as dúvidas e possíveis ambiguidades existentes no documento.

Uma vez compreendido o funcionamento das funcionalidades propostas o desenvolvedor passaria para a fase de desenvolvimento. Conforme mencionado na Seção 7.3.3 os indivíduos foram divididos em 2 grupos, que serão referenciados neste documento como Grupo 1 e Grupo 2. Os indivíduos do Grupo 1 começariam o desenvolvimento das duas primeiras funcionalidades com o uso das ferramentas instaladas em sua máquina e posteriormente as duas últimas funcionalidades com o uso do ambiente colaborativo RestMDD. Já com os indivíduos do Grupo 2 a ordem de execução foi invertida, as duas primeiras funcionalidades com o uso do ambiente colaborativo RestMDD e por último as outras duas funcionalidades com o uso das ferramentas instaladas em sua máquina.

Ambos os grupos deveriam construir os seguintes artefatos:

- Modelos de caso de uso
- Diagrama de casos de uso
- Diagramas de atividades
- Diagramas de classes (recursos, serviços, representações e exceções)
- Diagramas de sequência
- Código-fonte

Para todos os indivíduos foi disponibilizado um arquivo contendo o projeto inicial criado na ferramenta *Enterprise Architect* contendo cada um dos artefatos listados anteriormente para servir de exemplo para o desenvolvedor. O participante deveria então importar esse projeto na ferramenta *Enterprise Architect* e a partir dele construir os novos modelos. A mesma abordagem foi proposta para o código-fonte, que também foi disponibilizado para que o desenvolvedor importasse em sua IDE. Tanto o projeto quanto o código-fonte eram iguais para todos os participantes.

**Coleta de dados**

Os tempos de construção das funcionalidades foram divididos em tempo gasto na modelagem (TM) e tempo gasto na codificação (TC). O indivíduo obrigatoriamente deveria começar o desenvolvimento pela construção dos modelos e em seguida passaria para a codificação. Ao terminar a modelagem, o desenvolvedor informa ao pesquisador que valida se os modelos estão corretos e completos e coleta o tempo gasto na execução da atividade. O mesmo procedimento acontece com a codificação.

Ao final do desenvolvimento das funcionalidades o questionário proposto na Seção 7.2.2 foi encaminhado aos participantes para que a percepção subjetiva destes no uso das duas abordagens pudesse ser analisada.

## 7.5 Análise de resultados

A análise dos resultados obtidos com a execução do experimento apresentam as conclusões sobre a verificação das hipóteses, a possibilidade da rejeição da hipótese nula, e a redução do conjunto de dados. Nesse momento, os aspectos mais importantes são eliminar dados fora da distribuição normal (*outliers*), escolher o teste estatístico apropriado, explicar os resultados considerando os aspectos da validade, realizar a análise custo-benefício, e interpretar corretamente os resultados negativos (Travassos et al., 2002).

### 7.5.1 Apresentação dos resultados

Após realizada a coleta dos dados do experimento estes foram sumarizados e são apresentados nas tabelas 7.1 e 7.2, que mostram os dados coletados com o Grupo 1 e Grupo 2, respectivamente.

**Tabela 7.1:** Tabela de dados referentes ao Grupo 1

| ID | Abordagem | TM | TC | Esforço | P.F. | Produtividade(P.F./min) |
|---|---|---|---|---|---|---|
| devel01 | Tradicional | 44 | 17 | 61 | 20 | 0,327868852 |
| | MDD | 49 | 0 | 49 | 20 | 0,392156863 |
| devel02 | Tradicional | 31 | 25 | 56 | 20 | 0,357142857 |
| | MDD | 38 | 0 | 38 | 20 | 0,526315789 |
| devel03 | Tradicional | 37 | 21 | 58 | 20 | 0,344827586 |
| | MDD | 43 | 0 | 43 | 20 | 0,465116279 |
| devel04 | Tradicional | 41 | 19 | 60 | 20 | 0,333333333 |
| | MDD | 52 | 0 | 52 | 20 | 0,384615385 |
| devel05 | Tradicional | 46 | 19 | 65 | 20 | 0,307692308 |
| | MDD | 47 | 0 | 47 | 20 | 0,425531915 |

**Tabela 7.2:** Tabela de dados referentes ao Grupo 2

| ID | Abordagem | TM | TC | Esforço | P.F. | Produtividade(P.F./min) |
|---|---|---|---|---|---|---|
| devel06 | Tradicional | 38 | 20 | 58 | 20 | 0,344827586 |
| | MDD | 44 | 0 | 44 | 20 | 0,454545455 |
| devel07 | Tradicional | 43 | 22 | 65 | 20 | 0,307692308 |
| | MDD | 45 | 0 | 45 | 20 | 0,444444444 |
| devel08 | Tradicional | 49 | 17 | 66 | 20 | 0,303030303 |
| | MDD | 57 | 0 | 57 | 20 | 0,350877193 |
| devel09 | Tradicional | 40 | 23 | 63 | 20 | 0,317460317 |
| | MDD | 47 | 0 | 47 | 20 | 0,425531915 |
| devel10 | Tradicional | 39 | 21 | 60 | 20 | 0,333333333 |
| | MDD | 52 | 0 | 52 | 20 | 0,384615385 |

A coluna ID identifica os desenvolvedores participantes com valores de 1 a 10. A coluna Abordagem indica qual o paradigma de desenvolvimento de software utilizado. As colunas TM e TC indicam o tempo gasto em minutos para realizar a modelagem e a codificação, respectivamente, usando uma abordagem específica, enquanto que a coluna Esforço é a soma destes valores indicando o tempo total em minutos que o participante levou para concluir as atividades propostas. A coluna P.F. determina a complexidade das tarefas executadas, e por se tratarem de funcionalidades semelhantes este valor permanece inalterado em todos os registros da tabela. A coluna Produtividade aplica a métrica definida na Seção 7.2.2, determinando a produtividade do desenvolvedor em razão de pontos por função por minuto.

Além destes dados objetivos coletados do experimento, foram também analisadas as respostas recebidas no questionário aplicado ao final do experimento. Por se tratarem de dados subjetivos, estes serão discutidos na próxima seção.

### 7.5.2 Análise das hipóteses e conclusões

Os resultados deste estudo empírico apresentam indícios para a rejeição das hipóteses nulas. Esta rejeição é baseada apenas em análise descritiva, e não está embasada em estudos estatísticos rigorosos.

Observando as tabelas 7.1 e 7.2 foi possível verificar que o tempo total gasto no desenvolvimento das funcionalidades é sempre menor com o uso da abordagem MDD colaborativa. Isso se deve principalmente ao fato de que no experimento não há esforço na construção do código-fonte, que é gerado de forma automatizada.

Além disso, não foi encontrada nenhuma relação entre produtividade e a ordem de execução do experimento, ou seja, não foi percebido o aumento ou a redução da produtividade se o participante realizasse o experimento primeiro com a abordagem MDD e depois com a abordagem tradicional, ou vice-versa.

As tabelas 7.3 e 7.4 apresentam os cálculos de estatística básica referente ao uso da abordagem MDD e da abordagem tradicional, respectivamente. É possível observar que todas as amostras analisadas obtiveram uma variância pequena, aumentando dessa forma a confiança do experimento.

**Tabela 7.3:** Estatística Básica referente a abordagem MDD

| **Informação** | **Valor Produtividade** |
|---|---|
| Mínimo | 0,350877193 |
| Média | 0,425375062 |
| Mediana | 0,425531915 |
| Máximo | 0,526315789 |
| Desvio Padrão | 0,050507514 |
| Variância | 0,002551009 |

**Tabela 7.4:** Estatística Básica referente a abordagem Tradicional

| **Informação** | **Valor Produtividade** |
|---|---|
| Mínimo | 0,303030303 |
| Média | 0,327720878 |
| Mediana | 0,330601092 |
| Máximo | 0,357142857 |
| Desvio Padrão | 0,018351941 |
| Variância | 0,000336794 |

Assim, com relação à hipótese nula $H_{0a}$, conclui-se que a mesma é passível de rejeição, pois na comparação dos dados do estudo empírico foi verificado que todos os participantes obtiveram um aumento de produtividade utilizando a abordagem MDD colaborativa, contrariando esta hipótese.

Em relação ao questionário aplicado aos participantes foi possível observar alguns aspectos importantes sobre as dificuldades encontradas no uso do ambiente colaborativo, bem como suas vantagens em relação a outras ferramentas. As principais vantagens destacadas pelos participantes foram:

- Não é necessário "gastar tempo" na preparação do ambiente de trabalho;
- Alterações realizadas nos artefatos são disponibilizadas aos demais colaboradores de maneira transparente;
- Os colaboradores podem trabalhar em paralelo, porém sem o receio de prejudicar ou ser prejudicado pelos demais desenvolvedores que compartilham o espaço de trabalho;
- Os diagramas de classes são construídos de forma simples e rápida;

- Tempo reduzido para a construção de um cadastro simples.

Analisando as respostas do questionário foi possível constatar que os participantes consideram que houve aumento da produtividade com o uso do ambiente colaborativo RestMDD. Assim, há indícios de rejeição da hipótese nula $H_{0b}$.

Entre as desvantagens e/ou limitações apresentadas pela abordagem MDD colaborativa é possível destacar:

- Curva de aprendizado para utilização da abordagem MDD;
- Dificuldades para definir fluxos alternativos nos modelos de casos de uso;
- É obrigatório criar diferentes diagramas de classes, um para cada tipo de componente;
- Dificuldade na criação dos diagramas de sequência;
- Rigidez no código-fonte;
- Ausência de ferramentas de comunicação;
- Ausência de versionamento de artefatos.

Mesmo diante das dificuldades encontradas pelos participantes, nenhum destes considerou estas limitações como impeditivas para o uso da abordagem MDD, uma vez que os participantes conseguiram concluir as atividades em um tempo menor que com o uso da abordagem tradicional. Assim, diante desses dados é possível considerar a rejeição da hipótese nula $H_{0c}$.

## 7.6 Considerações finais

Neste capítulo foi apresentado o estudo experimental realizado para avaliação do ambiente colaborativo RestMDD. Esta avaliação foi conduzida de acordo com as fases propostas por (Wohlin et al., 2000). Primeiramente, foi apresentada a fase de definição do experimento, onde foi exposto o objetivo do experimento e as questões relacionadas a este objetivo, bem como as métricas a serem consideradas na avaliação. A seguir, foi apresentada a fase de planejamento, com todas as suas etapas para a elaboração do projeto do experimento. Logo após foi discutido sobre a execução do experimento e a coleta de dados, e por último foi realizada a análise dos resultados, o que permitiu rejeitar as hipóteses propostas na fase de planejamento.

Capítulo

8

# Conclusão

## 8.1 Caracterização da Pesquisa Realizada

Este trabalho definiu uma linguagem específica de domínio para a modelagem de Serviços Web RESTful utilizando MDD e construiu o ambiente de desenvolvimento colaborativo RestMDD para utilizar esta linguagem na produção dos serviços, além de apresentar os resultados de um experimento realizado para examinar a relação em termos de produtividade alcançada pelos desenvolvedores usando um processo de desenvolvimento de software convencional, orientado a codificação, comparado a uma abordagem de desenvolvimento de software orientado a modelos colaborativa, para a produção de Serviços Web RESTful.

Conforme a análise dos resultados do experimento realizada no capítulo anterior foi possível observar que o uso de uma abordagem orientada a modelos colaborativa pode produzir um aumento na produtividade dos desenvolvedores se comparado a uma abordagem de desenvolvimento de software orientada a codificação. Em contrapartida, foi possível observar que a prática do MDD não é familiar para grande parte dos desenvolvedores e portanto esse fato pode causar dificuldades quanto a sua adoção de forma ampla.

## 8.2 Contribuições

Podem-se destacar como principais contribuições deste trabalho:

1. Definição de uma linguagem específica de domínio para modelagem de Serviços Web RESTful;
2. Definição dos requisitos necessários para a construção de um ambiente de desenvolvimento de software colaborativo usando o paradigma de desenvolvimento orientado a modelos (MDD);
3. Construção de um ambiente colaborativo capaz de produzir Serviços Web RESTful usando o paradigma MDD;
4. Resultados sobre a comparação de produtividade entre o desenvolvimento de software tradicional (codificação) e o desenvolvimento orientado a modelos colaborativo.

## 8.3 Produção científica

Este projeto de mestrado resultou na elaboração e publicação de alguns artigos em eventos científicos que são listados a seguir:

1. SANCHEZ, R. V. V. ; OLIVEIRA, R. R. ; NETO, D. F. ; FORTES, R. P. M. ; PIMENTEL, M. G. C. Services Mobile: Incorporando serviços da Web 2.0 aos dispositivos móveis mediante o uso de informações de contexto. In: Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), 2011, Florianopolis SC. Anais do Webmedia 2011, 2011. v. v. 1. p. 1-4.
2. OLIVEIRA, R. R. ; SANCHEZ, R. V. V. ; NETO, D. F. ; FROTA, P. C. ; FORTES, R. P. M. Notificação de Wiki por meio de Mensageiros Instantâneos via bot reforçando a colaboração na Web. In: Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), 2011, Florianopolis SC. Anais do Webmedia 2011, 2011. v. v. 1. p. 1-7.
3. OLIVEIRA, R. R. ; SANCHEZ, R. V. V. ; FORTES, R. P. M. Desenvolvimento de web services RESTful e SOAP-WSDL utilizando as Implementações de Referência JAX-RS e JAX-WS. Material de Treinamento do Curso de web services RESTful e SOAP-WSDL em Java. 2011. Universidade de São Paulo. São Carlos - SP.
4. DIAS, A. L. ; FORTES, R. P. M. ; MASIERO, P. C. ; RAMOS, M. E. ; BADIALE, L. ; SANCHEZ, R. V. V. ; GRILLO, F. Adjustment and evolution of the AgendAloca

system focused on user requirements and accessibility: a case study. Submetido ao 9th International Cross-Disciplinary Conference on Web Accessibility - W4A 2012 - Lyon França.

5. SANCHEZ, R. V. V. ; OLIVEIRA, R. R. ; FORTES, R. P. M. RestML: Modeling RESTful Web Services. Submetido ao Call for Chapters - REST: Advanced Research Topics and Practical Applications 2012.

6. OLIVEIRA, R. R. ; SANCHEZ, R. V. V. ; ESTRELLA, J. C.; BRUSAMOLIN, V.; FORTES, R. P. M. Evaluation of Maintainability among RESTful and SOAP-WSDL web services approaches. Submetido ao 14th International Conference on Information Integration and Web-based Applications & Services - iiWAS 2012. Bali Indonésia.

## 8.4 Dificuldades e Limitações

O presente trabalho apresentou algumas dificuldades e limitações durante a preparação e execução:

A primeira dificuldade encontrada durante o desenvolvimento do ambiente colaborativo foi a criação dos modelos XMI e a geração automática dos modelos UML, pois as opções de API's e *frameworks* disponíveis para essa finalidade são bastante reduzidas.

Outra grande dificuldade apresentada ainda no desenvolvimento do ambiente colaborativo foi a definição dos *templates* para geração automática do código-fonte, pois devido ao grande número de artefatos a serem criados foi necessário a definição de diversos *templates* distintos.

Já em relação ao experimento realizado, outra dificuldade foi encontrar as métricas mais adequadas para avaliação da produtividade, de modo que fosse possível avaliar tanto de forma quantitativa quanto qualitativa o uso da abordagem MDD colaborativa.

Foi identificada como uma limitação do experimento a quantidade de indivíduos participantes. Com a replicação do experimento com uma quantidade maior de participantes os resultados obtidos neste estudo podem ser comparados e o impacto do fator humano pode ser melhor avaliado.

## 8.5 Trabalhos Futuros

Com a realização do estudo experimental do ambiente colaborativo RestMDD e graças ao *feedback* recebido dos participantes, foi possível encontrar possibilidades de melhoria no ambiente e também novas questões a serem abordadas na linguagem RestML.

Assim, como trabalhos futuros decorrentes dessa dissertação é possível destacar:

- **Inclusão de ferramentas de comunicação ao ambiente RestMDD**: foi sugerido pela maior parte dos participantes no experimento a inclusão de ferramentas que melhorassem a comunicação entre os desenvolvedores. Entre as sugestões estão: *wikis*, mensageiros instantâneos e *chats*;

- **Versionamento dos artefatos**: outro aspecto importante no ponto de vista dos usuários do ambiente colaborativo é o versionamento dos artefatos. Portanto, seria necessário submeter os modelos criados no ambiente a um sistema de controle de versões;

- **Encapsulamento de aplicações legadas**: uma das características dos Serviços Web é a integração de aplicações, e portanto é interessante que os Serviços Web RESTful modelados pela linguagem RestML sejam capazes de se comunicar com componentes de software já existente, dessa forma fornecendo apenas uma interface uniforme para integração com outras aplicações;

- **Geração de código-fonte em outras linguagens e plataformas**: a linguagem RestML descrita nesse trabalho propõe apenas geração de código-fonte para linguagem de programação Java. Assim, seria importante a possibilidade de geração de código em outras linguagens e plataformas, melhorando assim a portabilidade das aplicações.

Por fim, seria importante a replicação desse estudo aplicado a outros domínios de softwares e com um número maior de participantes.

# 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.

Albrecht, A.; Gaffney, J. E. Software function, source lines of code, and development effort prediction: A software science validation. *Software Engineering, IEEE Transactions on*, v. SE-9, n. 6, p. 639–648, 1983.

Aldawud, O.; Elrad, T.; Bader, A. Uml profile for aspect-oriented software development. In: *The Third International Workshop on Aspect Oriented Modeling*, 2003.

Ambler, S. W. *The object primer: Agile model-driven development with uml 2.0.* Cambridge University Press, 2004.

Ambler, S. W. *The elements of uml 2.0 style.* Cambridge University Press, 2005.

Ambler, S. W. *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise.* IBM Press, 2012.

Basili, V. R.; Caldiera, G.; Rombach, H. D. The goal question metric approach. In: *Encyclopedia of Software Engineering*, Wiley, 1994.

Basili, V. R.; Selby, R. W.; Hutchens, D. H. Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, v. 12, n. 7, p. 733–743, 1986.

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.

Cockburn, A. *Writing effective use cases*. Addison-Wesley Professional, 2000.

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/`

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.

Fuentes-Fernández, L.; Vallecillo-Moreno, A. An introduction to UML profiles. *UPGRADE, The European Journal for the Informatics Professional*, v. 5, n. 2, p. 5–13, 2004.

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.

IEEE IEEE Standard for Software Productivity Metrics. *IEEE Std 1045-1992*, p. 1–38, 1993.

JCP Jsr 339: Jax-rs 2.0: The java api for restful web services. 2012.
Disponível em `http://jcp.org/en/jsr/detail?id=339`

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.

Merriman, T. Mda in action: An anatomy of a platform-independent model. 2003.

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/`

Oracle Your first cup: An introduction to the java ee platform. 2012.
Disponível em `http://docs.oracle.com/javaee/6/firstcup/doc/`

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

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*, ).

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.

Rosenberg, D.; Stephens, M. *Use case driven object modeling with uml*. Apress, 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.

SpringSource Introduction to spring framework. 2012.
Disponível em `http://www.springsource.org/`

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.

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*, ).

Travassos, G. H.; Gurov, D.; do Amaral, E. A. G. *Introdução a engenharia de software experimental*. Relatório Técnico, COPPE / UFRJ, 2002.

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

WebRatio Webratio web site. 2013.
Disponível em `www.webratio.com`

Wohlin, C.; Runeson, P.; Host, M. *Experimentation in software engineering: An introduction*. Kluwer Academic Publishers, 2000.

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.

Apêndice

# A

# Exemplo de Código-Fonte gerado pelo ambiente RestMDD

O código-fonte apresentado a seguir foi gerado pelo ambiente colaborativo RestMDD e representa uma parte de todo o código-fonte produzido durante a realização dos experimentos apresentados no Capítulo 7.

```
package br.usp.icmc.agenda.business.exception;

/**
 * Classe que representa uma falha do tipo "Erro desconhecido".
 *
 * @author developer
 *
 */
public class ApplicationFailureException extends Exception {

    private static final long serialVersionUID = 1L;

    public ApplicationFailureException(Exception cause) {
        super(cause);
    }

}

package br.usp.icmc.agenda.business.exception;
```

```
/**
 * Classe que representa uma falha do tipo "Falha de negocio".
 *
 * @author developer
 *
 */
public class BusinessException extends Exception {

    private static final long serialVersionUID = 1L;

    public BusinessException(String message, Exception cause) {
        super(message, cause);
    }

}

package br.usp.icmc.agenda.web.exception;

/**
 * Classe que representa uma falha do tipo "Requisicao mal-formada".
 *
 * @author developer
 *
 */
public class ValidationException extends Exception {

    private static final long serialVersionUID = 1L;

    public ValidationException(String message) {
        super(message);
    }

}

package br.usp.icmc.agenda.web.exception;

/**
 * Classe que representa uma falha do tipo "Recurso nao encontrado".
 *
 * @author developer
 *
 */
public class ResourceNotFoundException extends Exception {

    private static final long serialVersionUID = 1L;
```

```
}
```

```
package br.usp.icmc.agenda.business.dao;

import java.util.List;
import java.util.Map;

/**
 * Interface para acesso aos dados externos da aplicacao.
 *
 * @author developer
 *
 */
public interface DAO {

    void save(Object entity);

    Object find(Class persistentClass, Integer id);

    List search(Class persistentClass, boolean fullName,
            Map<String, Object> params);

    Object update(Integer id, Object entity);

    void delete(Class persistentClass, Integer id);

}
```

```
package br.usp.icmc.agenda.business.dao;

import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import org.springframework.stereotype.Repository;

/**
 * Classe para acesso aos dados externos da aplicacao.
 *
 * @author developer
 *
```

```
 */
@Repository("jpaDao")
public class SpringJPADAO implements DAO {

    @PersistenceContext
    private EntityManager entityManager;

    public void save(Object entity) {
        entityManager.persist(entity);
    }

    @SuppressWarnings({ "unchecked", "rawtypes" })
    public Object find(Class persistentClass, Integer id) {
        return entityManager.find(persistentClass, id);
    }

    @SuppressWarnings("rawtypes")
    public List search(Class persistentClass, boolean fullName,
            Map<String, Object> params) {
        StringBuilder command = new StringBuilder("SELECT e FROM ");
        command.append(persistentClass.getSimpleName());
        command.append(" e");

        if ((params != null) && (params.size() > 0)) {
            StringBuilder where = new StringBuilder();

            for (Entry<String, Object> entry : params.entrySet()) {
                if (entry.getValue() == null) {
                    continue;
                } else {
                    if (where.length() == 0) {
                        where.append(" WHERE ");
                    } else {
                        where.append(" AND ");
                    }

                    if (entry.getValue() instanceof String) {
                        where.append("UPPER(e.");
                        where.append(entry.getKey());
                        if (fullName) {
                            where.append(") LIKE UPPER(:");
                            where.append(entry.getKey());
                            where.append(")");
                        } else {
                            where.append(") LIKE UPPER('%' || :");
```

```
                                where.append(entry.getKey());
                                where.append(" || '%')");
                            }
                        } else {
                            where.append("e.");
                            where.append(entry.getKey());
                            where.append(" = :");
                            where.append(entry.getKey());
                        }
                    }
                }

                command.append(where.toString());
            }

            Query query = entityManager.createQuery(command.toString());

            if ((params != null) && (params.size() > 0)) {
                for (Entry<String, Object> entry : params.entrySet()) {
                    if (entry.getValue() != null) {
                        query.setParameter(entry.getKey(), entry.getValue());
                    }
                }
            }

            return query.getResultList();
        }

        public Object update(Integer id, Object entity) {
            return entityManager.merge(entity);
        }

        @SuppressWarnings("rawtypes")
        public void delete(Class persistentClass, Integer id) {
            Object bean = find(persistentClass, id);
            if (bean != null) {
                entityManager.remove(bean);
            }
        }

}


package br.usp.icmc.agenda.business.services;

import java.util.List;
```

```
import br.usp.icmc.agenda.business.exception.ApplicationFailureException;
import br.usp.icmc.agenda.business.exception.BusinessException;
import br.usp.icmc.agenda.model.Universidade;

/**
 * Interface de negocio do servico UniversidadeService
 *
 * @author developer
 *
 */
public interface UniversidadeService {

    void cadastrarUniversidade(Universidade universidade)
            throws BusinessException, ApplicationFailureException;

    List<Universidade> listarUniversidade(String nome, String sigla)
            throws ApplicationFailureException;

    Universidade recuperarUniversidade(Integer idUniversidade)
            throws ApplicationFailureException;

    void atualizarUniversidade(Integer idUniversidade, Universidade universidade)
            throws BusinessException, ApplicationFailureException;

    void excluirUniversidade(Integer idUniversidade)
            throws ApplicationFailureException;
}


package br.usp.icmc.agenda.business.services;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import br.usp.icmc.agenda.business.dao.DAO;
import br.usp.icmc.agenda.business.exception.ApplicationFailureException;
import br.usp.icmc.agenda.business.exception.BusinessException;
import br.usp.icmc.agenda.model.Universidade;
```

```
/**
 * Classe de negocio do servico UniversidadeService
 *
 * @author developer
 *
 */
@Service
@Transactional(rollbackFor = {ApplicationFailureException.class,
        BusinessException.class})
@Scope("request")
public class UniversidadeServiceImpl implements UniversidadeService {

    private static Logger logger = Logger.getLogger(UniversidadeServiceImpl.class);

    @Autowired
    private DAO dao;

    @Override
    public void cadastrarUniversidade(Universidade universidade)
            throws BusinessException, ApplicationFailureException {
        try {
            Map<String, Object> params = new HashMap<String, Object>();
            params.put("sigla", universidade.getSigla());

            List<Universidade> universidades = dao.search(Universidade.class,
                    true, params);
            if ((universidades == null)
                    || (universidades.isEmpty())
                    || (!universidades.get(0).getSigla()
                            .equals(universidade.getSigla()))) {
                dao.save(universidade);
            } else {
                throw new BusinessException("universidade.sigla.exists");
            }
        } catch (BusinessException e) {
            logger.error(e);
            throw e;
        } catch (Exception e) {
            logger.error(e);
            throw new ApplicationFailureException(e);
        }
    }

    @Override
```

```
public List<Universidade> listarUniversidade(String nome, String sigla)
        throws ApplicationFailureException {
    try {
        Map<String, Object> params = new HashMap<String, Object>();
        params.put("nome", nome);
        params.put("sigla", sigla);

        return dao.search(Universidade.class, false, params);
    } catch (Exception e) {
        logger.error(e);
        throw new ApplicationFailureException(e);
    }
}

@Override
public Universidade recuperarUniversidade(Integer idUniversidade)
        throws ApplicationFailureException {
    try {
        Object entity = dao.find(Universidade.class, idUniversidade);
        if (entity != null) {
            return ((Universidade) entity);
        } else {
            return null;
        }
    } catch (Exception e) {
        logger.error(e);
        throw new ApplicationFailureException(e);
    }
}

@Override
public void atualizarUniversidade(Integer idUniversidade,
        Universidade universidade) throws BusinessException,
        ApplicationFailureException {
    try {
        Map<String, Object> params = new HashMap<String, Object>();
        params.put("sigla", universidade.getSigla());

        List<Universidade> universidades = dao.search(Universidade.class,
                true, params);
        if ((universidades == null)
                || (universidades.isEmpty())
                || (!universidades.get(0).getSigla()
                        .equals(universidade.getSigla()))
                || (universidades.get(0).getId().equals(idUniversidade))) {
```

```
                dao.update(idUniversidade, universidade);
            } else {
                throw new BusinessException("universidade.sigla.exists");
            }
        } catch (BusinessException e) {
            logger.error(e);
            throw e;
        } catch (Exception e) {
            logger.error(e);
            throw new ApplicationFailureException(e);
        }
    }

    @Override
    public void excluirUniversidade(Integer idUniversidade)
            throws ApplicationFailureException {
        try {
            dao.delete(Universidade.class, idUniversidade);
        } catch (Exception e) {
            logger.error(e);
            throw new ApplicationFailureException(e);
        }
    }

}


package br.usp.icmc.agenda.model;

import java.io.Serializable;
import java.util.Arrays;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlAttribute;
```

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;

/**
 * Representacao do recurso Universidade
 *
 * @author developer
 *
 */
@Entity
@SequenceGenerator(name = "universidadeSeq", sequenceName = "SEQ_UNIVERSIDADE",
                   initialValue = 1, allocationSize = 1)
@XmlRootElement(namespace = "http://garapa.intermidia.icmc.usp.br/agenda/ws/model")
@XmlType(name = "universidade", propOrder = { "descricao", "sigla", "links" })
public class Universidade implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "universidadeSeq")
    @NotNull
    @Min(value = 1, message = "validate.id.invalid")
    private Integer id;

    @Column(nullable = false)
    private String nome;

    @Size(min = 3, max = 20, message = "validate.universidade.sigla.size")
    private String sigla;

    @OneToMany(cascade = CascadeType.PERSIST, mappedBy = "universidade")
    private List<UnidadeEnsino> unidadesEnsino;

    @XmlAttribute
    public Integer getId() {
        return this.id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
```

```
public String getNome() {
    return this.nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getSigla() {
    return sigla;
}

public void setSigla(String sigla) {
    this.sigla = sigla;
}

@XmlTransient
public List<UnidadeEnsino> getUnidadesEnsino() {
    return unidadesEnsino;
}

public void setUnidadesEnsino(List<UnidadeEnsino> unidadesEnsino) {
    this.unidadesEnsino = unidadesEnsino;
}

@XmlElementWrapper(name = "links")
@XmlElement(name = "link")
public List<Link> getLinks() {
    return Arrays.asList(new Link("self", "/universidades/" + id),
            new Link("unidades", "/universidades/" + id + "/unidades"));
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((nome == null) ? 0 : nome.hashCode());
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
```

```
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Universidade other = (Universidade) obj;
        if (nome == null) {
            if (other.nome != null) {
                return false;
            }
        } else if (!nome.equals(other.nome)) {
            return false;
        }
        if (id == null) {
            if (other.id != null) {
                return false;
            }
        } else if (!id.equals(other.id)) {
            return false;
        }
        return true;
    }

}


package br.usp.icmc.agenda.web.resources;

import java.net.URI;
import java.util.List;
import java.util.Set;

import javax.validation.ConstraintViolation;
import javax.validation.Validator;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Response;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

import br.usp.icmc.agenda.business.exception.ApplicationFailureException;
import br.usp.icmc.agenda.business.exception.BusinessException;
import br.usp.icmc.agenda.business.services.UniversidadeService;
import br.usp.icmc.agenda.model.Universidade;
import br.usp.icmc.agenda.web.exception.ResourceNotFoundException;
import br.usp.icmc.agenda.web.exception.ValidationException;

@Path("/universidades")
@Component
@Scope("request")
public class UniversidadeResource {

    @Autowired
    private UniversidadeService service;

    @Autowired
    private Validator validator;

    private static final String UNIVERSIDADE_INVALID = "universidade.invalid";

    @POST
    @Produces({ "application/json", "application/xml" })
    public Response cadastrarUniversidade(Universidade universidade)
            throws ValidationException, BusinessException,
            ApplicationFailureException {
        validate(universidade);
        service.cadastrarUniversidade(universidade);
        URI uri = URI.create("/" + universidade.getId());
        Response response = Response.created(uri).entity(universidade).build();
        return response;
    }

    @GET
    @Produces({ "application/json", "application/xml" })
    public List<Universidade> listarUniversidade(
            @QueryParam("nome") String nome, @QueryParam("sigla") String sigla)
            throws ValidationException, ResourceNotFoundException,
            ApplicationFailureException {
        List<Universidade> lstUniversidade = service.listarUniversidade(nome,
                sigla);
```

```
        if ((lstUniversidade == null) || (lstUniversidade.isEmpty())) {
            throw new ResourceNotFoundException();
        }

        return lstUniversidade;
    }

    @GET
    @Produces({ "application/json", "application/xml" })
    @Path("{id}")
    public Universidade recuperarUniversidade(
            @PathParam("id") Integer idUniversidade)
            throws ValidationException, ResourceNotFoundException,
            ApplicationFailureException {
        Universidade universidade = service
                .recuperarUniversidade(idUniversidade);
        if (universidade == null) {
            throw new ResourceNotFoundException();
        }
        return universidade;
    }

    @PUT
    @Produces({ "application/json", "application/xml" })
    @Path("{id}")
    public Response atualizarUniversidade(Universidade universidade,
            @PathParam("id") Integer id) throws ValidationException,
            BusinessException, ApplicationFailureException {
        validate(universidade);
        service.atualizarUniversidade(id, universidade);
        URI uri = URI.create("/" + id);
        Response response = Response.ok(uri).entity(universidade).build();
        return response;
    }

    @DELETE
    @Produces({ "application/json", "application/xml" })
    @Path("{id}")
    public Response excluirUniversidade(@PathParam("id") Integer id)
            throws ValidationException, ApplicationFailureException {
        service.excluirUniversidade(id);
        URI uri = URI.create("/" + id);
        Response response = Response.ok(uri).build();
        return response;
    }
```

```
    private void validate(Universidade universidade) throws ValidationException {
        if (universidade != null) {
            Set<ConstraintViolation<Universidade>> errors = validator
                    .validate(universidade);
            if (!errors.isEmpty()) {
                ConstraintViolation<Universidade> constraint = errors.iterator()
                        .next();
                throw new ValidationException(constraint.getMessage());
            }
        } else {
            throw new ValidationException(UNIVERSIDADE_INVALID);
        }
    }
}
```