

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO  
INSTITUTO DE MATEMÁTICA  
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS  
COMPUTACIONAIS

Thiago Silva de Souza

**Uma Abordagem Baseada em Especificação para Testes de Web  
Services RESTful**

RIO DE JANEIRO  
2012

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO  
INSTITUTO DE MATEMÁTICA  
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS  
COMPUTACIONAIS

Thiago Silva de Souza

**Uma Abordagem Baseada em Especificação para Testes de Web  
Services RESTful**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática (PPGI) do Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (iNCE) e do Departamento de Ciência da Computação (DCC), Instituto de Matemática (IM), da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Orientador: Eber Assis Schmitz  
Co-orientador: Alexandre Luis Correa

RIO DE JANEIRO  
2012

## CIP - Catalogação na Publicação

S729a Silva de Souza, Thiago  
Uma Abordagem Baseada em Especificação para  
Testes de Web Services RESTful / Thiago Silva de  
Souza. -- Rio de Janeiro, 2012.  
117 f.

Orientador: Eber Assis Schmitz.  
Coorientador: Alexandre Luis Correa.  
Dissertação (mestrado) - Universidade Federal  
do Rio de Janeiro, Instituto Tércio Pacitti de  
Aplicações e Pesquisas Computacionais, Programa de  
Pós-Graduação em informática, 2012.

1. Teste de software. 2. Web Services. 3.  
Arquitetura REST. 4. OCL. I. Assis Schmitz, Eber,  
orient. II. Correa, Alexandre Luis, coorient.  
III. Título.

Thiago Silva de Souza

## **Uma Abordagem Baseada em Especificação para Testes de Web Services RESTful**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática (PPGI) do Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (iNCE) e do Departamento de Ciência da Computação (DCC), Instituto de Matemática (IM), da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Aprovada em: \_\_\_\_ de fevereiro de 2012.

---

Prof. Eber Assis Schmitz, Ph.D. (Orientador)  
UFRJ/IM/DCC

---

Prof. Alexandre Luis Correa, D.Sc. (Co-orientador)  
UNIRIO

---

Prof.<sup>a</sup> Maria Luiza Machado Campos, Ph.D.  
UFRJ/IM/DCC

---

Prof. Leonardo Guerreiro Azevedo, D.Sc.  
UNIRIO

---

Prof. Marcos Kalinowski, D.Sc.  
UFRJ/COPPE

RIO DE JANEIRO  
2012

Aos meus amados pais.

## **AGRADECIMENTOS**

A Deus, primeiramente, por iluminar o meu caminho nesta jornada e por ter me dado forças para superar todos os obstáculos.

Ao meu orientador, Prof. Eber Assis Schmitz, por toda atenção dada a este trabalho e por toda dedicação, paciência, compreensão e confiança demonstradas.

Ao meu co-orientador, Prof. Alexandre Luis Correa, não somente pela orientação sempre precisa e por todo conhecimento transmitido, mas por todo o tempo empenhado na construção deste trabalho.

Aos professores Maria Luiza Machado Campos, Leonardo Guerreiro Azevedo e Marcos Kalinowski por, gentilmente, aceitarem o convite para participar da banca examinadora deste trabalho e pelas contribuições e sugestões dadas.

À Prof.<sup>a</sup> Maria Luiza Machado Campos, por todos os ensinamentos e palavras de incentivo dados desde a minha graduação até o mestrado, passando pela especialização.

Ao meu pai João Roberto e à minha mãe Selma, pelo amor incondicional e por todo sacrifício feito para que seus filhos se tornassem pessoas corretas, trabalhadoras e estudiosas.

À minha amada esposa Barbara, que sempre esteve ao meu lado nesta caminhada tão difícil, com muito amor e compreensão, suportando as minhas ausências e me ajudando em momentos de extrema dificuldade.

À minha querida irmã Thielle, que ao longo desta jornada ficou privada da companhia e dos cuidados do irmão mais velho e, mesmo assim, se tornou motivo de orgulho para a família.

Aos tios Sandra e Robson, bem como seus filhos Lucas e Luan, por todo carinho que têm comigo e por todos os momentos de alegria que vivemos.

Aos tios Erick e Lidiane e à minha afilhada Julia, pela família que nos transformamos, pelo auxílio mútuo e por todos os valores aprendidos nesses anos de convivência.

Ao meu sogro Reginaldo e à minha sogra Suely, pelo apoio dado ao longo deste período e pela torcida pela realização deste sonho.

Ao meu grande amigo, Claudio Cesar Fraga Cavalcante, pela inequívoca amizade cultivada há doze anos.

Aos professores e, sobretudo amigos, Alessandro de Almeida Castro Cerqueira e Abel Rodolfo Garcia Lozano, pelas cartas de recomendação para ingressar no programa e pelo permanente incentivo para que me dedicasse à área de ensino e pesquisa.

Ao Prof. Arnaldo Lyrio Barreto, pelo exemplo a ser seguido e pelo sopro de incentivo para que eu voltasse a cursar o Mestrado.

Aos professores Arnaldo Vieira da Rocha Filho e Wilson Chagas de Araújo, grandes responsáveis pelo meu fascínio pela ciência, por toda ajuda e apoio prestados, imprescindíveis para minha formação profissional e, conseqüentemente, para a realização desse curso.

Aos amigos que dividiram comigo as alegrias e angústias do Mestrado nos programas do IME e da UFRJ: Kele Teixeira Belloze, Carlos Henrique Pinto Rodriguez, Maria Célia Tavares e Alexandre Neves Louzada.

À Prof.<sup>a</sup> Maria Célia Tavares, Diretora da Escola de Ciência e Tecnologia da UNIGRANRIO, por aguardar tão pacientemente a conclusão desse curso.

Ao Serviço Federal de Processamento de Dados (SERPRO), pela permissão para cursar o Mestrado durante o horário de trabalho.

Ao meu chefe no Centro de Testes do SERPRO, Carlos Hernandes Oliveira, pela inestimável amizade e pela infinita capacidade de compreensão demonstrada durante a realização desta dissertação.

Ao amigo Francisco Jovando Rebelo de Albuquerque, o Chicão, pela ajuda com a análise estatística do estudo experimental.

Ao amigo Dionísio Monteiro Teodoro, o Téo, pela amizade demonstrada, especialmente por ter conseguido recuperar o arquivo desta dissertação quando o meu HD estragou a duas semanas da defesa.

A todos os participantes do estudo experimental, especialmente aos colegas do SERPRO, pelo precioso tempo dedicado e pelo empenho na realização do experimento.

Ao pessoal da secretaria do PPGI, por toda ajuda prestada durante esses anos, especialmente ao funcionário Aníbal Ribeiro Ferreira, por sua eficiência em tratar dos meus assuntos acadêmicos.

Por fim, agradeço a todos aqueles que, de alguma forma, contribuíram na elaboração deste trabalho.



## RESUMO

SOUZA, Thiago Silva de. **Uma abordagem baseada em especificação para testes de Web Services RESTful**. 2012. 117 f. Dissertação (Mestrado em Informática) – Instituto de Matemática e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2012.

Esta dissertação apresenta uma abordagem para geração de casos de teste para Web Services RESTful. Web Services RESTful possuem características que não são cobertas pelas técnicas tradicionais de teste de software. A abordagem proposta utiliza técnicas de transformação de modelos para gerar casos de teste independentes de plataforma a partir de modelos de classes UML enriquecidos com restrições *Object Constraint Language* (OCL). Tais casos de teste são, então, transformados em casos de teste específicos de plataforma que podem ser usados para verificar a implementação de Web Services RESTful do tipo CRUD. Um estudo experimental mostrou que a proporção de testadores, utilizando técnicas *ad hoc*, capaz de alcançar ao menos 95% da cobertura de restrições cobertas pela abordagem é de no máximo 20%.

Palavras-chave: Teste de software. Web Services. Arquitetura REST. OCL.

## ABSTRACT

SOUZA, Thiago Silva de. **Uma abordagem baseada em especificação para testes de Web Services RESTful**. 2012. 117 f. Dissertação (Mestrado em Informática) – Instituto de Matemática e Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2012.

This dissertation presents an approach for RESTful Web Services test case generation. RESTful Web Services have features that are not fully covered by traditional software testing techniques. The proposed approach uses model transformation techniques to generate platform independent test cases from UML class models enriched with Object Constraint Language (OCL) constraints. These test cases are then transformed into platform specific test cases that can be used to verify the implementation of CRUD RESTful Web Services. An experimental study showed that the proportion of testers, using ad hoc techniques capable of achieving at least 95% coverage restrictions approach is covered by at most 20%.

Keywords: Software testing. Web Services. REST Architecture. OCL.

## LISTA DE FIGURAS

|                                                                                |    |
|--------------------------------------------------------------------------------|----|
| FIGURA 1: EXEMPLO DE RECURSO COM UM URIs E MÚLTIPLAS REPRESENTAÇÕES.....       | 27 |
| FIGURA 2: EXEMPLO DE DESCRITOR WADL.....                                       | 28 |
| FIGURA 3: EXEMPLO DE REQUISIÇÃO HTTP.....                                      | 30 |
| FIGURA 4: EXEMPLO DE RESPOSTA HTTP.....                                        | 30 |
| FIGURA 5: MODELO DE MATURIDADE REST DE RICHARDSON (RICHARDSON, 2008).....      | 32 |
| FIGURA 6: REPRESENTAÇÃO DE UM SERVIÇO NÍVEL 0.....                             | 33 |
| FIGURA 7: REPRESENTAÇÃO DE UM SERVIÇO NÍVEL 1.....                             | 34 |
| FIGURA 8: REPRESENTAÇÃO DE UM SERVIÇO NÍVEL 2.....                             | 35 |
| FIGURA 9: REPRESENTAÇÃO DE UM SERVIÇO NÍVEL 3.....                             | 36 |
| FIGURA 10: EXEMPLO DE MODELO DE CLASSES.....                                   | 41 |
| FIGURA 11: EXEMPLO DE INVARIANTE OCL.....                                      | 41 |
| FIGURA 12: EXEMPLOS DE PRÉ E PÓS-CONDIÇÕES OCL.....                            | 41 |
| FIGURA 13: DIMENSÕES DO TESTE DE SOFTWARE (UTTING E LEGEARD, 2007).....        | 43 |
| FIGURA 14: FLUXO DE ATIVIDADES DA ABORDAGEM PROPOSTA.....                      | 52 |
| FIGURA 15: MODELO DE CLASSES DO DOMÍNIO ESCOLA.....                            | 55 |
| FIGURA 16: OPERAÇÕES AUXILIARES PARA RESTRIÇÃO DE VALORES SOBRE ATRIBUTOS..... | 57 |
| FIGURA 17: INVARIANTES DA CLASSE CURSO.....                                    | 58 |
| FIGURA 18: CONTRATO DA OPERAÇÃO CRIARCURSO().....                              | 59 |
| FIGURA 19: CONTRATO DA OPERAÇÃO CONSULTARCURSO().....                          | 60 |
| FIGURA 20: CONTRATO DA OPERAÇÃO ALTERARCURSO().....                            | 61 |
| FIGURA 21: CONTRATO DO MÉTODO REMOVERCURSO().....                              | 62 |
| FIGURA 22: RECORTE DO MODELO DE CLASSES ESTENDIDO DO DOMÍNIO ESCOLA.....       | 63 |

## LISTA DE TABELAS

|                                                                                                        |    |
|--------------------------------------------------------------------------------------------------------|----|
| TABELA 1: DADOS DE TESTE PARA OS ATRIBUTOS DA CLASSE CURSO. ....                                       | 65 |
| TABELA 2: TABELA DE DECISÃO DO PIT.....                                                                | 65 |
| TABELA 3: MODELO DE CASO DE TESTE PIT POSITIVO PARA OPERAÇÃO DE CRIAÇÃO. ....                          | 67 |
| TABELA 4: EXEMPLO DE CASO DE TESTE POSITIVO PARA A OPERAÇÃO CRIARCURSO(). ....                         | 67 |
| TABELA 5: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE CRIAÇÃO (FAIXA DE VALORES).....        | 68 |
| TABELA 6: EXEMPLO DE CASO DE TESTE PIT NEGATIVO DE FAIXA DE VALORES PARA A OPERAÇÃO CRIARCURSO(). .... | 69 |
| TABELA 7: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE CRIAÇÃO (UNICIDADE). ....              | 69 |
| TABELA 8: EXEMPLO DE CASO DE TESTE PIT NEGATIVO DE UNICIDADE PARA A OPERAÇÃO CRIARCURSO(). ....        | 69 |
| TABELA 9: MODELO DE CASO DE TESTE PIT POSITIVO PARA OPERAÇÃO DE CONSULTA. ....                         | 70 |
| TABELA 10: EXEMPLO DE CASO DE TESTE PIT POSITIVO PARA A OPERAÇÃO CONSULTARCURSO(). ....                | 70 |
| TABELA 11: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE CONSULTA. ....                        | 70 |
| TABELA 12: EXEMPLO DE CASO DE TESTE PIT NEGATIVO PARA A OPERAÇÃO CONSULTARCURSO(). ....                | 71 |
| TABELA 13: MODELO DE CASO DE TESTE PIT POSITIVO PARA OPERAÇÃO DE ALTERAÇÃO.....                        | 71 |
| TABELA 14: EXEMPLO DE CASO DE TESTE PIT POSITIVO PARA A OPERAÇÃO ALTERARCURSO().                       | 72 |
| TABELA 15: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE ALTERAÇÃO.....                        | 72 |
| TABELA 16: EXEMPLO DE CASO DE TESTE PIT NEGATIVO PARA A OPERAÇÃO ALTERARCURSO().<br>.....              | 72 |
| TABELA 17: MODELO DE CASO DE TESTE PIT POSITIVO PARA OPERAÇÃO DE REMOÇÃO.....                          | 73 |
| TABELA 18: EXEMPLO DE CASO DE TESTE PIT POSITIVO PARA A OPERAÇÃO REMOVERCURSO().<br>.....              | 73 |

|                                                                                                                    |     |
|--------------------------------------------------------------------------------------------------------------------|-----|
| TABELA 19: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE REMOÇÃO<br>(RESTRIÇÃO EXISTENCIAL).....           | 74  |
| TABELA 20: EXEMPLO DE CASO DE TESTE PIT NEGATIVO PARA A OPERAÇÃO REMOVERCURSO()<br>– RESTRIÇÃO EXISTENCIAL.....    | 74  |
| TABELA 21: MODELO DE CASO DE TESTE PIT NEGATIVO PARA OPERAÇÃO DE REMOÇÃO<br>(INTEGRIDADE REFERENCIAL). ....        | 75  |
| TABELA 22: EXEMPLO DE CASO DE TESTE PIT NEGATIVO PARA A OPERAÇÃO REMOVERCURSO()<br>– INTEGRIDADE REFERENCIAL. .... | 75  |
| TABELA 23: MAPEAMENTO ENTRE OS TIPOS DE OPERAÇÃO E OS MÉTODOS HTTP.....                                            | 76  |
| TABELA 24: CÓDIGOS DE RESPOSTA TRADICIONAIS DOS MÉTODOS HTTP.....                                                  | 76  |
| TABELA 25: MODELO GERAL DE ASSERTIVAS PARA CASOS DE TESTE PST.....                                                 | 77  |
| TABELA 26: MODELO DE CASO DE TESTE PST POSITIVO PARA OPERAÇÕES DE CRIAÇÃO. ....                                    | 78  |
| TABELA 27: EXEMPLO DE CASO DE TESTE PST POSITIVO PARA A OPERAÇÃO CRIARCURSO(). ...                                 | 78  |
| TABELA 28: MODELO DE CASO DE TESTE PST NEGATIVO PARA OPERAÇÕES DE REMOÇÃO<br>(INTEGRIDADE REFERENCIAL). ....       | 79  |
| TABELA 29: EXEMPLO DE CASO DE TESTE PST NEGATIVO PARA A OPERAÇÃO REMOVERCURSO().<br>.....                          | 80  |
| TABELA 30: ESTATÍSTICA DESCRITIVA DOS RESULTADOS.....                                                              | 92  |
| TABELA 31: VALOR P ENCONTRADO PELO TESTE DE MANN-WHITNEY.....                                                      | 94  |
| TABELA 32: FAIXA ETÁRIA DOS PARTICIPANTES.....                                                                     | 113 |
| TABELA 33: SEXO DOS PARTICIPANTES.....                                                                             | 113 |
| TABELA 34: ORIGEM DOS PARTICIPANTES. ....                                                                          | 113 |
| TABELA 35: COEFICIENTE DE RENDIMENTO DOS PARTICIPANTES ALUNOS. ....                                                | 114 |
| TABELA 36: FORMAÇÃO ACADÊMICA DOS PARTICIPANTES. ....                                                              | 114 |
| TABELA 37: EXPERIÊNCIA PROFISSIONAL DOS PARTICIPANTES EM DESENVOLVIMENTO DE<br>SOFTWARE. ....                      | 114 |
| TABELA 38: EXPERIÊNCIA DOS PARTICIPANTES EM MODELAGEM DE CLASSES COM UML. ....                                     | 115 |
| TABELA 39: EXPERIÊNCIA DOS PARTICIPANTES EM DESENVOLVIMENTO DE SISTEMAS. ....                                      | 115 |

|                                                                                                       |     |
|-------------------------------------------------------------------------------------------------------|-----|
| TABELA 40: EXPERIÊNCIA DOS PARTICIPANTES EM PROJETO DE CASOS DE TESTE A PARTIR DE ESPECIFICAÇÃO. .... | 116 |
| TABELA 41: EXPERIÊNCIA DOS PARTICIPANTES EM EXECUÇÃO DE CASOS DE TESTE CAIXA-PRETA. ....              | 116 |
| TABELA 42: EXPERIÊNCIA DOS PARTICIPANTES NA UTILIZAÇÃO DA FERRAMENTA SOAPUI. ....                     | 116 |
| TABELA 43: EXPERIÊNCIA DOS PARTICIPANTES EM DESENVOLVIMENTO DE WEB SERVICES RESTFUL.....              | 117 |
| TABELA 44: EXPERIÊNCIA DOS PARTICIPANTES EM PROJETO DE CASOS DE TESTE PARA WEB SERVICES RESTFUL.....  | 117 |
| TABELA 45: EXPERIÊNCIA DOS PARTICIPANTES EM EXECUÇÃO DE CASOS DE TESTE PARA WEB SERVICES RESTFUL..... | 117 |

## LISTA DE ABREVIATURAS E SIGLAS

1. CORBA – *Common Object Request Broker Architecture*.
2. CRUD – *Create, Retrieve, Update and Delete*.
3. DCOM – *Distributed Component Object Model*.
4. DTO – *Data Transfer Object*.
5. EP1 – Exercício Prático 1.
6. EP2 – Exercício Prático 2.
7. HATEOAS – *Hypermedia as the Engine of Application State*.
8. HTTP – *Hypertext Transfer Protocol*.
9. IDL – *Interface Definition Language*.
10. JPEG – *Joint Photographic Experts Group*.
11. JSON – *JavaScript Object Notation*.
12. MDA – *Model Driven Architecture*.
13. MIME – *Multipurpose Internet Mail Extensions*.
14. MOF – *Meta-Object Facility*.
15. OAT – *Orthogonal Array Testing*.
16. OCL – *Object Constraint Language*.
17. OMG – *Object Management Group*.
18. OWL – *Web Ontology Language*.
19. PIM – *Platform Independent Model*.
20. PIT – *Platform Independent Test*.
21. PSM – *Platform Specific Model*.
22. PST – *Platform Specific Test*.
23. PWT – *Pair-Wise Testing*.
24. QCP – Questionário de Caracterização do Participante.

25. QVT – *Query, Views and Transformations*.
26. REST – *Representational State Transfer*.
27. RMI – *Remote Method Invocation*.
28. SERPRO – Serviço Federal de Processamento de Dados.
29. SOAP – *Simple Object Access Protocol*.
30. TCLE – Termo de Consentimento Livre e Esclarecido.
31. UDDI – *Universal Description, Discovery and Integration*.
32. UML – *Unified Modeling Language*.
33. UNIGRANRIO – Universidade do Grande Rio.
34. URI – *Universal Resource Identifier*.
35. XHTML – *Extensible Hypertext Markup Language*.
36. XML – *Extensible Markup Language*.
37. WADL – *Web Application Description Language*.
38. WSDL – *Web Services Description Language*.
39. WSDL-S – *Web Service Semantics*.

# SUMÁRIO

|          |                                                 |           |
|----------|-------------------------------------------------|-----------|
| <b>1</b> | <b>Introdução .....</b>                         | <b>19</b> |
| 1.1      | Motivação.....                                  | 19        |
| 1.2      | Problema.....                                   | 22        |
| 1.3      | Objetivos e Escopo do Trabalho.....             | 23        |
| 1.4      | Organização do Trabalho.....                    | 23        |
| <b>2</b> | <b>Referencial Teórico.....</b>                 | <b>24</b> |
| 2.1      | Web Services .....                              | 24        |
| 2.2      | REST.....                                       | 26        |
| 2.2.1    | Web Services RESTful .....                      | 28        |
| 2.2.2    | Protocolo HTTP .....                            | 29        |
| 2.2.3    | Modelo de Maturidade REST .....                 | 32        |
| 2.3      | Projeto por Contrato.....                       | 37        |
| 2.4      | Object Constraint Language.....                 | 38        |
| 2.5      | Teste de Software .....                         | 41        |
| 2.5.1    | Níveis de Teste.....                            | 44        |
| 2.5.2    | Tipos de Teste .....                            | 45        |
| 2.5.3    | Técnicas e Critérios de Teste .....             | 45        |
| 2.6      | Trabalhos Relacionados .....                    | 48        |
| <b>3</b> | <b>Abordagem Proposta.....</b>                  | <b>51</b> |
| 3.1      | Visão Geral .....                               | 51        |
| 3.2      | Descrição do Domínio de Exemplo .....           | 54        |
| 3.3      | Técnica para Definição de Modelos PIM.....      | 55        |
| 3.3.1    | Modelos de Classes .....                        | 55        |
| 3.3.2    | Invariantes OCL .....                           | 57        |
| 3.3.3    | Pré e Pós-Condições OCL.....                    | 58        |
| 3.3.3.1  | Regras para Operações de Criação .....          | 58        |
| 3.3.3.2  | Regras para Operações de Consulta .....         | 59        |
| 3.3.3.3  | Regras para Operações de Alteração .....        | 60        |
| 3.3.3.4  | Regras para Operações de Remoção .....          | 61        |
| 3.4      | Técnica para Definição de Modelos PSM .....     | 62        |
| 3.5      | Técnica para Geração de Casos de Teste PIT..... | 64        |
| 3.5.1    | Tabela de Decisão .....                         | 64        |
| 3.5.2    | Modelos de Casos de Teste .....                 | 65        |
| 3.5.2.1  | Casos de Teste PIT de Criação .....             | 67        |
| 3.5.2.2  | Casos de Teste PIT de Consulta .....            | 69        |
| 3.5.2.3  | Casos de Teste PIT de Alteração .....           | 71        |
| 3.5.2.4  | Casos de Teste PIT de Remoção .....             | 73        |
| 3.6      | Técnica para Geração de Casos de Teste PST..... | 75        |
| 3.7      | Execução do Testes .....                        | 80        |
| <b>4</b> | <b>Estudo Experimental.....</b>                 | <b>81</b> |
| 4.1      | Introdução.....                                 | 81        |
| 4.1.1    | Motivação.....                                  | 81        |

|          |                                                                                         |            |
|----------|-----------------------------------------------------------------------------------------|------------|
| 4.1.2    | Objetivo Global .....                                                                   | 82         |
| 4.1.3    | Objetivo da Medição .....                                                               | 82         |
| 4.1.4    | Objetivo do Estudo .....                                                                | 82         |
| 4.1.5    | Questões .....                                                                          | 83         |
| 4.2      | <i>Planejamento</i> .....                                                               | 84         |
| 4.2.1    | Seleção do Contexto .....                                                               | 84         |
| 4.2.2    | Formulação das Hipóteses .....                                                          | 85         |
| 4.2.2.1  | Questão Q1 .....                                                                        | 85         |
| 4.2.2.2  | Questão Q2 .....                                                                        | 85         |
| 4.2.3    | Variáveis Independentes .....                                                           | 86         |
| 4.2.4    | Variáveis Dependentes .....                                                             | 86         |
| 4.2.5    | Seleção dos Participantes .....                                                         | 87         |
| 4.2.6    | Instrumentação .....                                                                    | 87         |
| 4.3      | <i>Operação</i> .....                                                                   | 89         |
| 4.4      | <i>Análise e Interpretação dos Resultados</i> .....                                     | 90         |
| 4.4.1    | Restrições Testadas pelos Participantes .....                                           | 90         |
| 4.4.2    | Testes de Hipótese .....                                                                | 92         |
| 4.4.2.1  | Questão Q1 .....                                                                        | 92         |
| 4.4.2.2  | Questão Q2 .....                                                                        | 93         |
| 4.5      | <i>Avaliação da Validade dos Resultados</i> .....                                       | 94         |
| 4.5.1    | Validade Interna .....                                                                  | 94         |
| 4.5.2    | Validade Externa .....                                                                  | 95         |
| 4.5.3    | Validade da Construção .....                                                            | 95         |
| 4.5.4    | Validade da Conclusão .....                                                             | 96         |
| 4.6      | <i>Conclusão</i> .....                                                                  | 96         |
| <b>5</b> | <b>Conclusão</b> .....                                                                  | <b>98</b>  |
| 5.1      | <i>Considerações Finais</i> .....                                                       | 98         |
| 5.2      | <i>Contribuições</i> .....                                                              | 99         |
| 5.3      | <i>Limitações e Trabalhos Futuros</i> .....                                             | 100        |
|          | <b>Referências Bibliográficas</b> .....                                                 | <b>103</b> |
|          | <b>Apêndice A – Termo de Consentimento Livre e Esclarecido para Alunos</b> .....        | <b>107</b> |
|          | <b>Apêndice B – Termo de Consentimento Livre e Esclarecido para Profissionais</b> ..... | <b>108</b> |
|          | <b>Apêndice C – Questionário de Caracterização do Participante</b> .....                | <b>109</b> |
|          | <b>Apêndice D – Exercício Prático 1</b> .....                                           | <b>110</b> |
|          | <b>Apêndice E – Exercício Prático 2</b> .....                                           | <b>111</b> |
|          | <b>Apêndice F – Perfil do Participante do Experimento</b> .....                         | <b>113</b> |

## 1 INTRODUÇÃO

Este capítulo está dividido em quatro seções. A seção 1.1 apresenta a motivação para realização deste trabalho. A seção 1.2 descreve o problema a ser tratado nesta dissertação. A seção 1.3 mostra os objetivos e o escopo deste trabalho. Por fim, a seção 1.4 descreve a estrutura segundo a qual este trabalho está organizado.

### 1.1 Motivação

Atualmente, a tecnologia de Web Services é a principal solução tecnológica no que se refere à integração e reúso de software no contexto de computação distribuída. Web Services são programas modulares, independentes e autodescritivos que, a partir da definição de sua interface, podem ser executados por qualquer outra aplicação (SILVA-DE-SOUZA e PIRES, 2005).

Recentemente uma nova categoria de Web Services, denominados RESTful, vem ganhando notoriedade tanto na indústria quanto na academia. Web Services RESTful são serviços que seguem o estilo arquitetural *REpresentational State Transfer* (REST), o qual visa promover o uso intensivo do protocolo *Hypertext Transfer Protocol* (HTTP) (FIELDING, 2000). Web Services RESTful diferenciam-se dos Web Services tradicionais, conhecidos como WS-<sup>1</sup>, principalmente por prescindirem de tantas especificações em formato *Extensible Markup Language* (XML), tais como os padrões *Simple Object Access Protocol* (SOAP) e *Web Services Description Language* (WSDL) (W3C, 2001).

Web Services RESTful manipulam recursos. Um recurso é qualquer item de informação acessível através de um *Universal Resource Identifier* (URI) (FIELDING, 2000). Todo recurso possui uma ou mais representações formadas pelos dados e metadados que o

---

<sup>1</sup> O termo WS-\* faz referência às diversas especificações para Web Services tradicionais cujos nomes são iniciados pelo prefixo WS, e.g., WSDL).

descrevem. Cada representação possui um formato em conformidade com o padrão *Multipurpose Internet Mail Extensions* (MIME) (FIELDING, 2000). A manipulação de recursos se dá pela transferência de representações entre clientes e servidores HTTP. Essa transferência de representações é apoiada pela interface uniforme do protocolo HTTP, composta, principalmente, pelos verbos POST, GET, PUT e DELETE. POST cria um novo recurso, GET recupera o estado corrente de um recurso em qualquer representação, PUT modifica o estado de um recurso já existente e, finalmente, DELETE exclui um recurso (WEBBER, PARASTATIDIS e ROBINSON, 2010). Dessa forma, Web Services RESTful vinculam um método HTTP específico a cada operação de manipulação de um recurso.

Uma das formas de controle da qualidade de Web Services, como de qualquer software, é através de testes. O teste de software tem como objetivo encontrar defeitos em programas. Ele consiste na execução de programas, para os quais são submetidos dados de entrada e as respostas geradas são avaliadas em função dos resultados esperados (DELAMARO, MALDONADO e JINO, 2007). Geralmente os desenvolvedores realizam testes estruturais (caixa branca) em nível unitário antes de publicar um serviço. Entretanto, quando os serviços são oferecidos por um fornecedor externo, o código-fonte da implementação não está disponível para o cliente testar. Portanto, para o teste de Web Services não é possível aplicar as técnicas tradicionais de teste estrutural, o que torna fundamental a realização de testes funcionais (caixa preta) (SOMMERVILLE, 2011).

As técnicas tradicionais de teste de software não são diretamente aplicáveis a Web Services em função de sua natureza dinâmica (CANFORA e PENTA, 2009). Web Services RESTful, particularmente, trazem a necessidade de técnicas específicas de teste funcional, no que diz respeito a três características presentes nessa tecnologia: i) insuficiência semântica do documento de descrição do serviço; ii) variedade de formatos para representação dos recursos e; iii) uso de requisições HTTP para executar os serviços.

Quanto à primeira característica, vale ressaltar que grande parte dos trabalhos sobre teste de Web Services utilizam descritores WSDL como insumo para geração de casos de teste. A descrição de serviços RESTful pode ser feita através do formato *Web Application Description Language* (WADL) (W3C, 2009) ou da especificação WSDL 2.0 (W3C, 2007). Alguns autores, porém, consideram desnecessária a definição deste tipo de documento de descrição, tendo em vista que a própria representação dos recursos manipulados por Web Services RESTful pode conter detalhes de como manipulá-los, usando o conceito de hipermídia (RICHARDSON e RUBY, 2007; WEBBER, PARASTATIDIS e ROBINSON, 2010). No entanto, serviços hipermídia são raros devido ao pouco suporte ferramental, tornando o WADL a opção mais natural para se descrever Web Services RESTful.

Um contrato de serviço é um compromisso entre o produtor e os consumidores de um serviço (ERL *et al.*, 2008). Um descritor WADL, portanto, é um contrato de serviço, representado em XML, que descreve um conjunto de operações permitidas sobre recursos, os modelos de URI e os formatos possíveis para representação dos recursos (WEBBER, PARASTATIDIS e ROBINSON, 2010). Porém, um documento WADL não especifica todas as restrições de negócio envolvidas em um domínio, restringindo-se a especificar aspectos tecnológicos.

Testes funcionais são dependentes da especificação do sistema. Quanto mais precisa e semanticamente rica for a especificação maior poderá ser a qualidade dos testes projetados. No caso de aplicações de controle de cadastro aderentes ao padrão *Create, Retrieve, Update and Delete* (CRUD), o WADL não representa as invariantes do domínio tratado, bem como, as pré e pós-condições de cada operação dos serviços, o que dificulta a geração de dados de teste significativos. É necessário, portanto, prover mais semântica às especificações, através de uma especificação precisa, para permitir que um conjunto mínimo de casos de teste seja gerado e, possivelmente, automatizado.

A segunda característica se refere à possibilidade dos recursos serem representados em diversos formatos, tais como XML, *JavaScript Object Notation* (JSON), *Extensible Hypertext Markup Language* (XHTML), entre outros. Em testes estruturais não há necessidade de verificar todas as operações dos serviços RESTful combinadas com todos os formatos de representação. Basta garantir alta cobertura de testes unitários sobre os objetos do domínio e testar os componentes de serialização (XML, JSON, etc.) de forma isolada. No entanto, quando se trata de testes funcionais, os componentes de serialização não podem ser testados separadamente dos objetos de domínio, tornando necessários casos de teste que combinem todas as operações com todos os formatos de representação consumidos ou produzidos.

Já a terceira característica em pauta diz respeito à necessidade de utilizar requisições HTTP para executar os serviços. Como serviços CRUD são baseados nos verbos POST, GET, PUT e DELETE, o problema neste caso está relacionado à estratégia de composição dos casos de teste em termos de requisições, considerando que há relações de precedência entre as operações CRUD e não se sabe de antemão o estado do sistema sob teste.

Este trabalho trata das questões apresentadas propondo uma abordagem de teste baseada em especificação para Web Services RESTful do tipo CRUD, na qual o domínio do serviço é representado de acordo com determinadas regras e, em seguida, são aplicadas técnicas para geração de casos de teste independentes de plataforma, que são finalmente transformados em casos de teste específicos para a tecnologia REST.

## 1.2 Problema

Um simples documento de descrição de Web Services RESTful não garante que ele atenda às necessidades do cliente, bem como não garante que ele tenha sido implementado de maneira correta. Além disso, o conjunto de informações provido a respeito do serviço torna o teste funcional (caixa-preta) uma tarefa complexa. O processo de geração de casos de teste

para Web Services RESTful, especificamente relacionados a controle de cadastro, mesmo fazendo uso das principais ferramentas existentes, demanda muito esforço no que se refere ao projeto de casos de teste, considerando as regras de negócio do domínio, os diversos formatos de representação de recursos e a composição de requisições HTTP para exercitar os serviços.

### **1.3 Objetivos e Escopo do Trabalho**

O objetivo geral deste trabalho é definir uma abordagem que possibilite a geração automática de casos de teste para Web Services RESTful do tipo CRUD, a partir de modelos de classes UML enriquecidos com restrições OCL.

### **1.4 Organização do Trabalho**

Esta dissertação está organizada em cinco capítulos. O capítulo 1 mostrou a introdução do trabalho, ressaltando a sua motivação, o problema tratado e seus objetivos. O capítulo 2 discorre sobre o referencial teórico deste trabalho, através de um resumo da revisão bibliográfica. O capítulo 3 apresenta a abordagem proposta, através de um exemplo simples. O capítulo 4 descreve um estudo experimental realizado com o objetivo de colher indícios da utilidade da abordagem proposta. O capítulo 5 conclui a dissertação, apresentando as considerações finais, contribuições e sugestões de trabalhos futuros. Em seguida, é apresentada a lista de referências bibliográficas utilizadas. Finalmente, são apresentados na forma de apêndices os instrumentos utilizados no estudo experimental.

## 2 REFERENCIAL TEÓRICO

Este capítulo trata da fundamentação teórica da dissertação. Ele está dividido em seis seções: a seção 2.1 introduz os conceitos fundamentais sobre Web Services; a seção 2.2 descreve os princípios do estilo arquitetural REST; a seção 2.3 disserta sobre projeto de software baseado em contratos; a seção 2.4 apresenta aspectos sobre a linguagem OCL, utilizada para descrever restrições sobre modelos UML; a seção 2.5 apresenta conceitos a respeito de teste de software; por fim, a seção 2.6 analisa os trabalhos relacionados com esta dissertação.

### 2.1 Web Services

Desde o seu surgimento, a Web vem sendo utilizada, principalmente, para acessar documentos e aplicações de forma interativa. Na maior parte das vezes o acesso é feito por usuários humanos, normalmente utilizando *Web browsers*. Contudo, com a evolução recente das tecnologias para Internet, principalmente com a popularização da linguagem XML e suas tecnologias relacionadas, o papel da Web está evoluindo de um ambiente baseado na troca de dados para um ambiente baseado na troca de serviços. Esta tendência está sendo motivada pela tecnologia de *Web Services*.

De modo geral, Web Services são funcionalidades disponibilizadas em um ambiente de computação distribuída na forma de serviços (ERL *et al.*, 2008). Em uma perspectiva técnica, o termo *serviço* é usado para se referir a qualquer função de software que implementa uma tarefa do negócio ou realiza uma função genérica como, por exemplo, autenticação ou registro de atividades/ocorrências (*logging*) (DAIGNEAU, 2012).

O conceito de serviço, entretanto, não é recente. Tecnologias de desenvolvimento baseado em componentes e de objetos distribuídos tais como *Common Object Request Broker*

*Architecture* (CORBA), *Distributed Component Object Model* (DCOM) e o *Remote Method Invocation* (RMI) já implementavam o conceito de serviço. No entanto, os serviços criados com tais tecnologias eram acessados por protocolos dependentes de um modelo de objetos específico, diferentemente da tecnologia de Web Services, onde os serviços são acessados por protocolos e formatos de dados ubíquos, como o HTTP e o XML, ambos padrões independentes de fabricante e largamente utilizados na Web (SILVA-DE-SOUZA e PIRES, 2005). Além disso, as interfaces de serviços CORBA, DCOM e RMI eram criadas com linguagens de definição de interfaces (*Interface Definition Language* – IDL) específicas de plataforma, o que obrigava os clientes a possuir a mesma plataforma computacional, formando “ilhas de interoperabilidade” (DAIGNEAU, 2012).

Apesar de relativamente recente, a tecnologia de Web Services já passou por algumas evoluções. A primeira geração da tecnologia foi caracterizada pelas especificações *Web Services Description Language* (WSDL) (W3C, 2001), *Simple Object Access Protocol* (SOAP) e *Universal Description, Discovery and Integration* (UDDI). Posteriormente, foram desenvolvidas diversas outras especificações com o objetivo de adicionar camadas para segurança, roteamento e endereçamento de mensagens aos padrões já existentes. Essa extensão ficou conhecida como a segunda geração da tecnologia de Web Services.

Atualmente há duas categorias de serviços: Web Services aderentes às duas primeiras gerações são conhecidos como Web Services WS-\*, enquanto que serviços alinhados com o estilo arquitetural *Representational State Transfer* (REST), descrito na seção 2.2, são denominados Web Services RESTful.

Independentemente da categoria, Web Services provêm os meios para integrar sistemas distintos e expor funções do negócio através do protocolo HTTP. No que diz respeito ao uso do protocolo HTTP, a principal diferença entre os dois estilos é que Web Services WS-\* usam HTTP apenas como um mecanismo de transporte de dados, enquanto que Web

Services RESTful o utilizam como um protocolo de aplicação completo que define a semântica do comportamento do serviço (DAIGNEAU, 2012).

## 2.2 REST

*Representational State Transfer* (REST) ou Transferência de Estado Representacional é um termo descrito por Roy Fielding (2000) em sua tese de doutorado, que representa uma série de restrições que formam um estilo arquitetural de software.

Em uma arquitetura REST, o foco são os recursos. Um recurso é qualquer item de informação acessível através de um URI (FIELDING, 2000). Segundo Richardson e Ruby (2007), um recurso é algo que pode ser armazenado em um computador e ser representado como um fluxo de bits, como um documento, uma tupla em um banco de dados ou o resultado da execução de um algoritmo.

Fielding (2000) identifica uma série de restrições que devem ser observadas no desenvolvimento de um sistema REST, dentre as quais vale destacar as seguintes:

- *Cliente-Servidor* – Os clientes estão separados do servidor, e estes se comunicam apenas pela interface uniforme do protocolo HTTP. Por esta separação, entende-se que o cliente está totalmente desacoplado do servidor, não dependendo do modo em que os dados são armazenados ou de como as mudanças de estado são implementadas.
- *Sem estado (stateless)* – Não há estado conversacional. Portanto, cada requisição deve conter todas as informações necessárias para o processamento, e todo estado deve ser mantido pelo cliente e não pelo servidor.
- *URI* – Cada recurso é identificado por um URI, utilizado para realizar as alterações no estado do recurso.

- *Interface uniforme* – Toda e qualquer operação executada em um recurso é feita a partir dos métodos HTTP, simplificando e desacoplando a arquitetura.
- *Cache* – Como na Web, clientes devem ser capazes de manter *cache* de respostas enviadas pelo servidor. Sendo assim, as respostas devem indicar se podem ser mantidas em *cache*, para prevenir que clientes façam uso de dados inconsistentes.

Outro aspecto importante do padrão REST diz respeito às múltiplas representações de recursos. Os recursos devem ter pelo menos um identificador, e cada identificador é associado a uma ou mais representações. Uma representação é uma visão do recurso em um instante do tempo (WEBBER, PARASTATIDIS e ROBINSON, 2010). Tal visão pode ser materializada em um ou mais formatos de transferência de dados, tais como XML, JSON, XHTML, texto simples ou outros formatos, podendo atender, portanto, a diversos tipos de cliente. A Figura 1 representa um recurso com um identificador URI (<http://locadora.ppgi.ufrrj.br/filme/1>) e dois formatos de representação, o primeiro em XML e o segundo com a notação JSON.

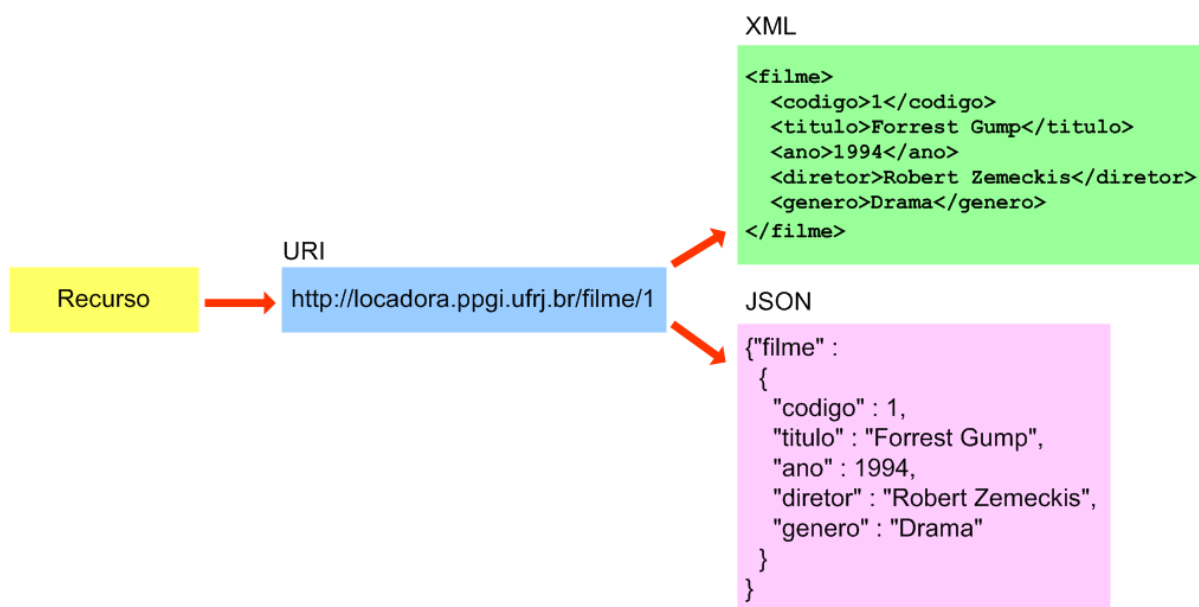


Figura 1: Exemplo de recurso com um URI e múltiplas representações.

### 2.2.1 Web Services RESTful

Web Services RESTful são serviços desenvolvidos seguindo as restrições impostas pelo padrão REST, tendo como base o protocolo HTTP. Segundo Richardson e Ruby (2007), não há um modelo oficial a ser seguido para os Web Services RESTful, como acontece com os que são baseados em SOAP, em razão de REST ser um estilo arquitetural e não um protocolo.

Um Web Service RESTful é diferente de um baseado em SOAP, pois não necessita de um documento que o descreva, já que os seus URIs e o método HTTP definem a semântica do serviço. Porém, opcionalmente, podem ser descritos utilizando a *Web Application Definition Language* (WADL) (W3C, 2009) ou a WSDL, somente na versão 2.0. Tais formatos são direcionados à leitura por máquinas e contêm elementos que informam onde o serviço está localizado e como deve ser acessado. A Figura 2 representa um descritor WADL para um serviço de reconhecimento de triângulo, que recebe três parâmetros do tipo inteiro e produz uma resposta em texto simples.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
  <resources base="http://localhost:8080/TrianguloRest/">
    <resource path="/triangulo">
      <method name="GET" id="imprimeTriangulo">
        <request>
          <param type="xs:int" style="query" name="a"/>
          <param type="xs:int" style="query" name="b"/>
          <param type="xs:int" style="query" name="c"/>
        </request>
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Figura 2: Exemplo de descritor WADL.

### 2.2.2 Protocolo HTTP

O estilo arquitetural REST é fundamentado nas capacidades do protocolo HTTP. Web Services RESTful utilizam o protocolo HTTP para transportar as representações de recursos entre clientes e servidores e para definir a semântica do comportamento dos serviços, através de um conjunto de métodos do próprio protocolo (DAIGNEAU, 2012).

O HTTP é um dos protocolos presentes na camada de aplicação da arquitetura TCP/IP, e é utilizado para troca de informações entre sistemas hipermídia distribuídos (FIELDING, 2000). Na prática, o HTTP funciona baseado em trocas de mensagens entre um cliente e um servidor, onde há sempre uma mensagem de requisição (*HTTP request*) solicitando um recurso disponível em um URI, e uma mensagem de resposta (*HTTP response*) contendo o resultado da solicitação. Para estabelecer a conexão, o cliente submete uma requisição ao servidor informando dados como o URI que deseja acessar, a versão do protocolo HTTP que está utilizando e o formato do conteúdo que porventura exista no corpo da mensagem. Por sua vez, o servidor devolve para o cliente uma resposta contendo a versão do protocolo HTTP e o um código de resposta daquela requisição, podendo ser de erro ou de sucesso, encerrando assim a conexão.

Tanto as mensagens de requisição quanto as de resposta são formadas por uma linha inicial, nenhuma, uma ou mais de uma linha de cabeçalho, uma linha vazia para indicar o fim do cabeçalho e o corpo da mensagem, que pode ser opcional em alguns momentos. O cabeçalho contém informações adicionais de interesse do cliente e do servidor, tais como o tipo de conteúdo esperado, que é definido no cabeçalho “*Content-Type*”. Já o corpo pode ser utilizado de duas formas: 1) na requisição, pode conter dados de usuário ou um arquivo a ser enviado ao servidor; 2) na resposta, pode conter o recurso solicitado pelo cliente ou uma mensagem de erro. A Figura 3 mostra um exemplo de mensagem HTTP de requisição, na qual é solicitada a criação de um recurso do tipo Filme, através do método POST.

```
POST /Locadora/filme/
User-Agent : Jakarta Commons-HttpClient/3.1
Host : 127.0.0.1:8080
Content-Length : 152
Content-Type : text/xml; charset=UTF-8

<filme>
  <codigo>1</codigo>
  <titulo>Forrest Gump</titulo>
  <ano>1994</ano>
  <diretor>Robert Zemeckis</diretor>
  <genero>Drama</genero>
</filme>
```

**Figura 3: Exemplo de requisição HTTP.**

A Figura 4 mostra um exemplo de mensagem HTTP de resposta, que informa, entre outras coisas, que o recurso referente à requisição da Figura 3 foi criado com sucesso.

```
HTTP/1.1 201 Created
Server : Apache-Coyote/1.1
Location : http://127.0.0.1:8080/Locadora/filme/1
Content-Type : text/xml
Content-Length : 0
Date : Thu, 27 Jan 2011 16:49:30 GMT
```

**Figura 4: Exemplo de resposta HTTP.**

Quando um cliente deseja enviar uma solicitação ao servidor através do protocolo HTTP, ele pode utilizar um dos métodos pertencentes ao protocolo. O protocolo HTTP, na sua versão 1.1, possui diversos métodos, dentre os quais vale ressaltar os seguintes:

- *GET*: Solicita o acesso a um determinado recurso, disponível em um URI.
- *POST*: Cria um novo recurso com as informações enviadas no corpo da requisição HTTP. Pode também efetuar o processamento de informações que não estão relacionadas diretamente ao recurso em questão.
- *PUT*: Atualiza um recurso presente no URI contido na mensagem. Caso o recurso não exista, ele pode criá-lo.

- *DELETE*: Se executado com sucesso, remove um recurso indicado por um URI. Caso não seja executado corretamente, retornará um código de resposta, indicando que o recurso não pode ser excluído ou que não existe nada a ser excluído.

Para qualquer mensagem HTTP, deve ser definido o tipo do conteúdo que seguirá no corpo da mensagem, de acordo com o padrão *Multipurpose Internet Mail Extensions* (MIME). Para especificar o tipo MIME, basta incluí-lo no cabeçalho “*Content-Type: nome\_do\_tipo\_mime*”. Os principais tipos são:

- *text/plain* – texto simples;
- *text/html* – texto no formato *HyperText Markup Language* (HTML);
- *text/xml* – texto no formato XML;
- *application/json* – texto no formato JSON;
- *image/jpeg* – imagem no formato *Joint Photographic Experts Group* (JPEG).

As mensagens de resposta do servidor devem sempre retornar um código que representa o resultado da requisição, também chamado de *status code*. No protocolo HTTP, há códigos que indicam se a requisição foi concluída com sucesso, se ocorreu um erro, se houve algum problema no servidor, se o recurso não foi encontrado, ou outras respostas. Esses códigos estão separados nos seguintes grupos:

- *1xx (Informações)* – Informa ao cliente que a requisição foi recebida com sucesso e está sendo processada.
- *2xx (Sucesso)* – Informa que a requisição foi concluída com sucesso, independente da existência do recurso.
- *3xx (Redirecionamento)* – Informa uma ação que deve ser executada para completar a requisição.

- *4xx (Erro do lado do cliente)* – Informa que houve um erro no lado do cliente que não o permitiu completar a requisição.
- *5xx (Erro do lado do servidor)* – Informa que ocorreu um erro no servidor que o impediu de completar a requisição.

### 2.2.3 Modelo de Maturidade REST

Com o objetivo de classificar aplicações que procuram seguir o estilo arquitetural REST quanto ao grau de aderência ao padrão, Richardson (2008) estabeleceu um modelo de maturidade REST. Este modelo é dividido em quatro níveis, do nível 0 ao nível 3, onde o nível 3 é o que indica maior aderência às restrições REST. A Figura 5 representa tal modelo.

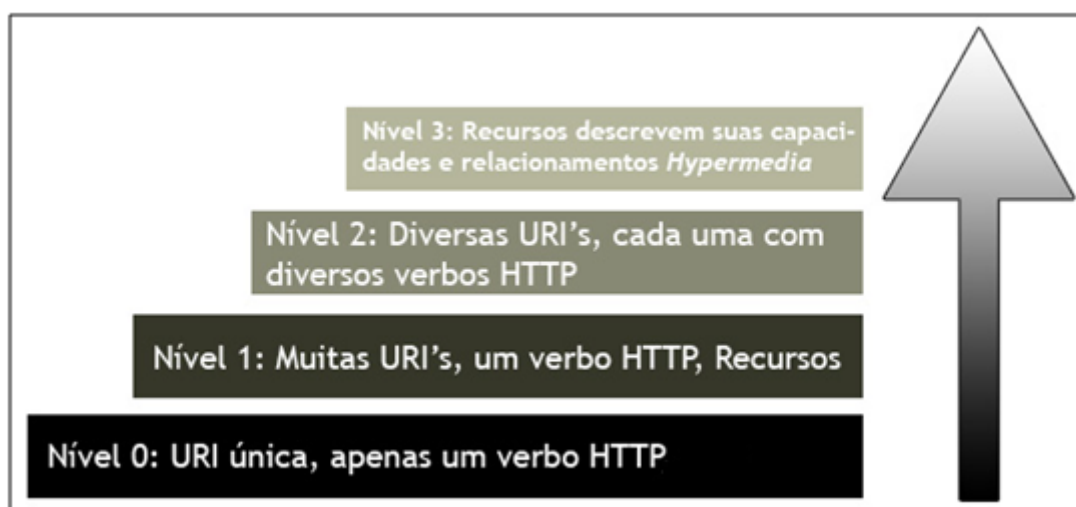
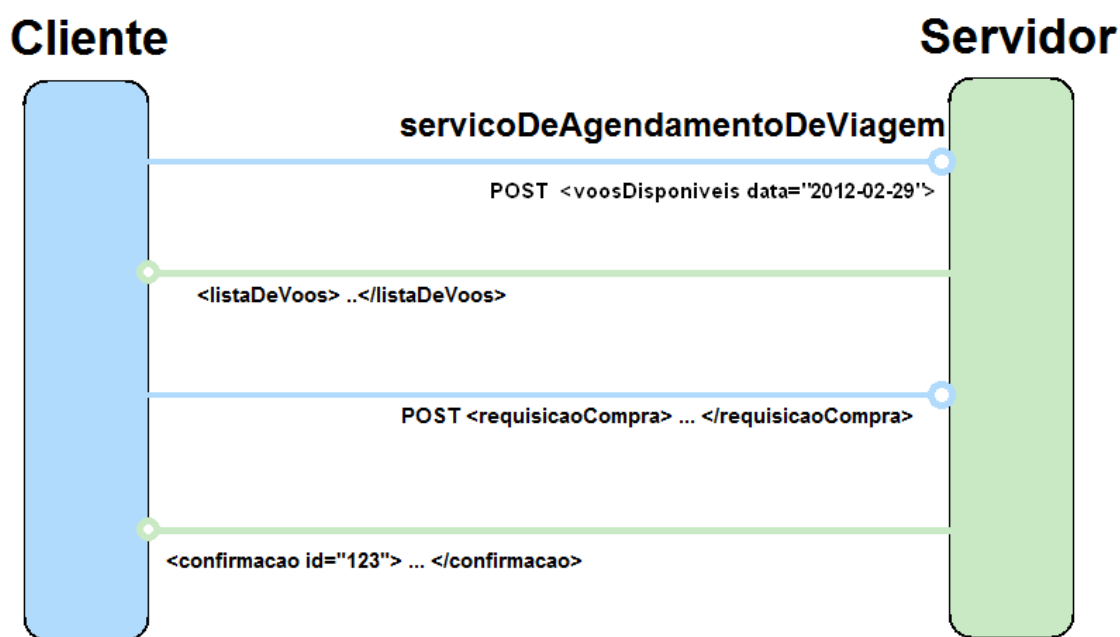


Figura 5: Modelo de maturidade REST de Richardson (RICHARDSON, 2008).

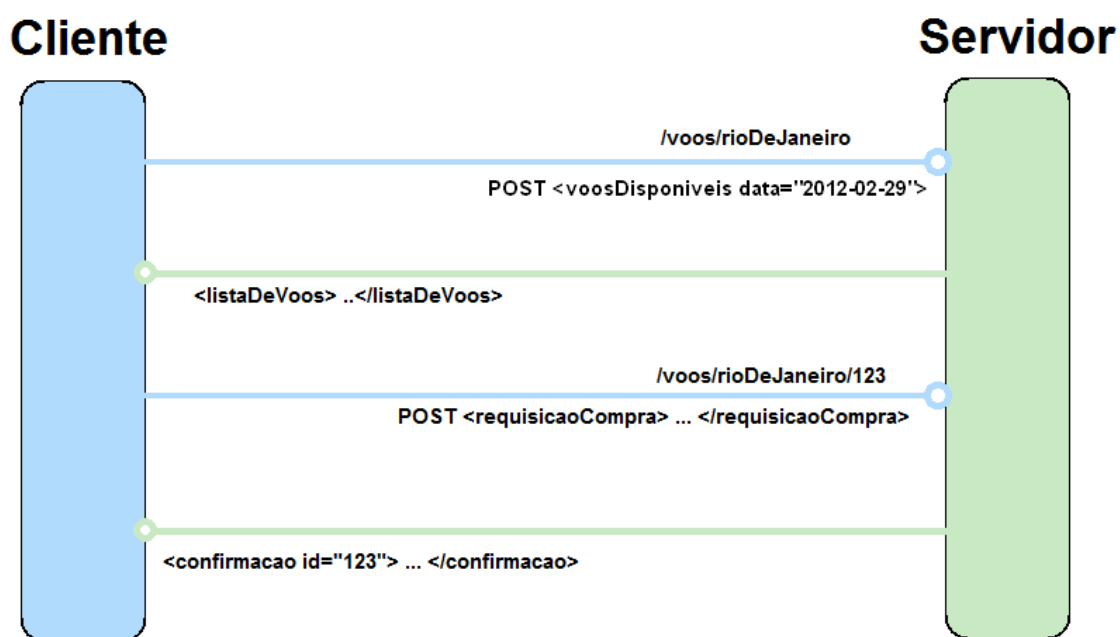
O nível 0 é o nível mais básico de maturidade de serviços, sendo caracterizado pelos serviços que possuem apenas um URI e que usam apenas um método HTTP (geralmente o método POST) (WEBBER, PARASTATIDIS e ROBINSON, 2010). Este é o tipo de abordagem seguida pelos Web Services WS-\*, nos quais os serviços são identificados por um único URI e as mensagens SOAP são transmitidas via método POST. A Figura 6 representa um serviço de nível 0 relacionado a um agendamento de viagem.



**Figura 6: Representação de um serviço nível 0.**

Inicialmente, no exemplo da Figura 6, uma requisição é feita a um serviço de agendamento de viagens requisitando os vôos disponíveis do dia 29 de fevereiro de 2012. A resposta a tal requisição retorna uma representação XML de cada um dos vôos deste dia. Em seguida, uma nova requisição é feita ao mesmo URI, através do envio de um documento XML, indicando que esta é uma requisição de compra. Logo após, o servidor retorna ao cliente uma mensagem confirmando a compra. É possível perceber que, neste exemplo, nenhuma das restrições REST é seguida.

O próximo nível de maturidade, o nível 1, continua utilizando apenas um método HTTP, mas emprega vários URIs para identificar os recursos. A Figura 7 representa um serviço de nível 1, tomando como base o exemplo da Figura 6.



**Figura 7: Representação de um serviço nível 1.**

No exemplo da Figura 7 as operações são realizadas em URIs diferentes, diferentemente do exemplo da Figura 6. O cliente, ao desejar uma lista de vôos do Rio de Janeiro, realiza uma requisição ao URI `/voos/rioDeJaneiro`, enviando, no corpo da requisição, um documento XML indicando a data desejada do vôo. O servidor envia a lista de vôos disponíveis, e o cliente realiza uma requisição ao URI `/voos/rioDeJaneiro/123`, que representa exatamente o vôo escolhido por ele, enviando uma mensagem de requisição de compra. Para concluir a operação, o servidor retorna uma mensagem indicando que a transação foi concluída com sucesso.

No nível 2 a aplicação passa a utilizar o protocolo HTTP não só como protocolo de transporte, mas também como a interface através da qual os clientes se conectam. Através desta interface os clientes podem saber quais operações podem ser realizadas em um recurso, com a utilização correta dos verbos HTTP. Neste nível, há a utilização de códigos de resposta HTTP para sinalizar sucesso, erro, recurso não encontrado, acesso não permitido, entre outros. Desta forma, no lugar de uma mensagem XML informando que uma operação foi concluída

com sucesso, o servidor retorna uma resposta HTTP com o *status* “200 Ok” ou “201 Created” (no caso da criação de um novo recurso). O nível 2 inclui tipicamente aplicações CRUD, onde o estado dos recursos pode ser manipulado pelos métodos HTTP. A Figura 8 representa um serviço de nível 2.

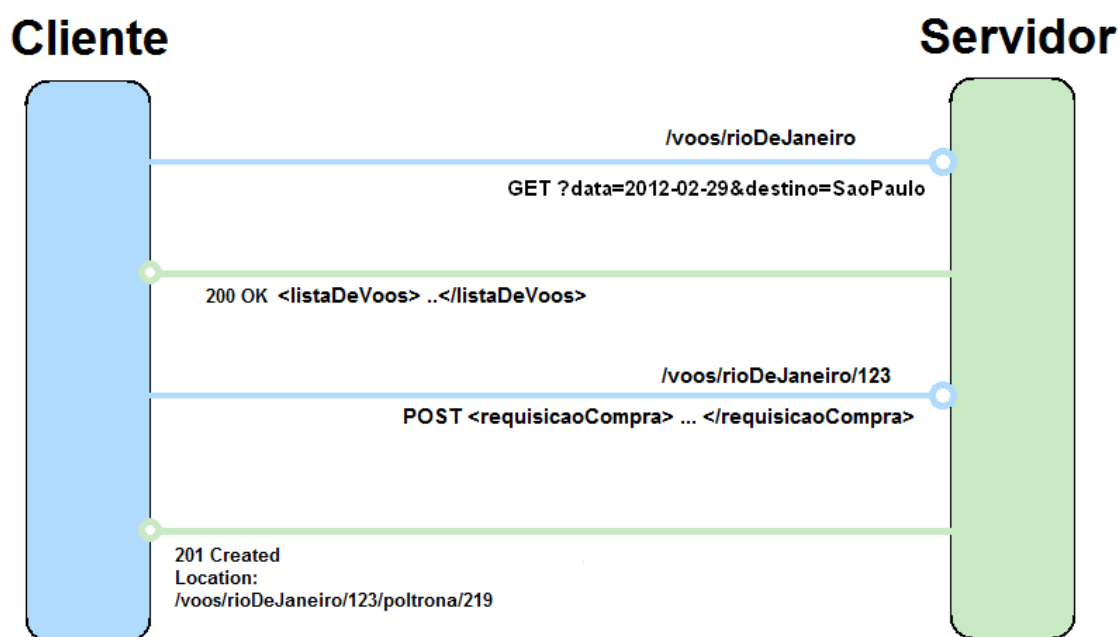


Figura 8: Representação de um serviço nível 2.

No exemplo da Figura 8 pode-se notar a diferença na requisição, na qual o cliente não utiliza *POST* para buscar os vôos. Ele realiza uma requisição por *GET* e informa os parâmetros de sua busca no próprio URI. O servidor retorna a lista de vôos conforme os dados da requisição com *status* “200 Ok” na resposta, informando que a operação foi realizada com sucesso.

Para realizar uma compra, o cliente submete ao servidor uma mensagem com seus dados, os dados da poltrona que deseja reservar e o identificador do vôo (neste caso, 123). Em seguida, o servidor retorna uma resposta com *status* “201 Created”, indicando que a operação de compra foi realizada com sucesso, e com um cabeçalho *Location*, informando a localização do novo recurso criado, que representa a poltrona recém-reservada.

O nível 3 é o menos utilizado por grande parte dos serviços, pois é difícil de ser implementado e poucas ferramentas oferecem suporte para implementá-lo. Nele, o conceito de hipermídia como motor do estado da aplicação (*Hypermedia as the Engine of Application State* - HATEOAS) é introduzido (WEBBER, PARASTATIDIS e ROBINSON, 2010). Com a utilização de HATEOAS, os recursos são descobertos e expostos dinamicamente para os clientes. As transições de estado de um recurso e os relacionamentos com outros recursos são expressos na forma de *hyperlinks* na representação de cada recurso. Isto dá ao servidor a capacidade de alterar seus URIs sem afetar os clientes, já que estes descobririam os novos URIs ao receber uma nova representação de um recurso. A Figura 9 representa um serviço de nível 3.



Figura 9: Representação de um serviço nível 3.

O exemplo da Figura 9 é semelhante ao da Figura 8, com exceção à representação dos recursos, que apresentam *links* que apontam para as mudanças de estado de cada recurso e seus relacionamentos com outros recursos.

## 2.3 Projeto por Contrato

Projeto por contrato (*design by contract*) é uma abordagem, introduzida por Meyer (MEYER, 1992), utilizada para especificar as variações de espaço-estado possíveis para um método. A idéia é aplicar mecanismos semelhantes aos que regem os relacionamentos entre consumidores e empresas fornecedoras de serviços. Um contrato é um acordo formal entre duas ou mais partes, que descreve seus direitos e obrigações de modo não-ambíguo (CHEESMAN e DANIELS, 2001). As partes envolvidas no contrato são denominadas cliente (contratante) e servidor (contratado) (BINDER, 2000). Cabe ao cliente atender aos requisitos exigidos pelo servidor, enquanto que o servidor deve cumprir suas promessas definidas no contrato.

Um contrato, segundo Meyer (1992), é formado por pré-condições, pós-condições e invariantes de classe. Uma *pré-condição* é uma condição que deve ser satisfeita antes da execução da operação. Uma *pós-condição* é a descrição dos efeitos que a operação deve produzir. Já uma *invariante de classe* é uma propriedade aplicada a todas as instâncias da classe, transcendendo rotinas particulares. Antes da execução de cada método, as invariantes e as pré-condições devem ser avaliadas em conjunto, assim como devem ser avaliadas as invariantes e as pós-condições ao término da execução de cada método. Se o cliente garantir as pré-condições, então o servidor deve garantir as pós-condições. No entanto, se o cliente não garantir as pré-condições, então o contrato será cancelado para aquela chamada, podendo a operação não ser executada e não garantir as pós-condições.

Erl *et al.* (2008), de um ponto-de-vista voltado a Web Services, afirmam que um contrato de serviço é uma coleção de metadados que descrevem vários aspectos do serviço, tais como:

- o objetivo de cada operação;
- as mensagens que precisam ser trocadas ao executar as operações;

- modelos de dados usados para definir a estrutura das mensagens;
- um conjunto de condições para executar as operações;
- informação sobre como e onde o serviço pode ser acessado.

Os formatos tradicionais para descrição de serviços (WSDL e WADL) permitem a representação dos aspectos listados por Erl *et al.* (2008). Entretanto, tais formatos, mesmo associados a documentos XML Schema, não são capazes de representar os elementos típicos de um contrato (invariantes, pré e pós-condições). Especificações enriquecidas semanticamente, por exemplo, através de contratos, podem trazer informações importantes sobre as regras de negócio dos serviços, o que possibilita a derivação de condições e dados de teste mais significativos para avaliar a qualidade da implementação dos serviços. Além disso, contratos formais de operações, quando processáveis por máquinas, tornam a geração de código de implementação e a geração de casos de teste passíveis de automação. Desta forma, para possibilitar a geração automatizada de casos de teste significativos para Web Services RESTful torna-se necessário o uso de uma linguagem voltada à especificação de contratos de operações para enriquecer semanticamente as especificações de serviços.

## 2.4 Object Constraint Language

A *Object Constraint Language* (OCL) (OMG, 2010) é uma linguagem padronizada pelo *Object Management Group* (OMG) utilizada para especificação de restrições que não podem ser descritas graficamente com modelos UML ou metamodelos aderentes ao padrão *Meta-Object Facility* (MOF). OCL pode ser usada para os mais diversos propósitos, tais como: especificar restrições que precisam ser asseguradas pelo sistema que está sendo modelado; especificar pré e pós-condições sobre operações; especificar condições de guarda e

regras de derivação; especificar regras de formação de modelos de acordo com metamodelos e servir como linguagem de consulta.

Na *Model Driven Architecture* (MDA) (OMG, 2003), um *framework* conceitual para desenvolvimento de software baseado em modelos, modelos independentes de plataforma, modelos dependentes de plataforma e transformações automáticas entre modelos têm importância fundamental no processo de desenvolvimento de software. Para aplicar os conceitos de MDA é preciso produzir modelos que possam ser processados por máquinas. Neste contexto, OCL realiza um papel importante, porque permite a elaboração de modelos precisos processáveis por computadores. OCL é também uma parte importante das linguagens de transformação de modelos, tais como o padrão *Query, Views and Transformations* (QVT) do OMG (OMG, 2005).

Em vez de usar notações matemáticas formais, OCL é uma linguagem textual com elementos sintáticos similares aos presentes em linguagens de programação orientadas a objetos. OCL foi projetada para ser uma linguagem menos intimidante em relação às linguagens formais tradicionais, tais como Z (WOODCOCK e DAVIS, 1996) e VDM-SL (JONES, 1989).

Diagramas, como o diagrama de classes da UML, tipicamente não são refinados o suficiente para representar todos os aspectos relevantes de uma especificação. Restrições, regras, contratos de operação são alguns exemplos de informação adicional sobre os elementos de um modelo que são frequentemente descritos em linguagem natural. No entanto, especificações em linguagem natural são intrinsecamente relacionadas a problemas de ambiguidade (BERRY e KAMSTIES, 2004), como aconteceu com a primeira versão da UML.

OCL é uma linguagem textual declarativa usada para especificar, de maneira precisa, as restrições sobre elementos de um modelo (OMG, 2010). Toda expressão escrita em OCL

baseia-se nos elementos (classes, atributos, operações, entre outros) definidos em um modelo específico. OCL é uma linguagem de especificação pura; portanto, uma expressão OCL garantidamente é livre de efeitos colaterais.

Expressões OCL podem resultar em um valor primitivo, um objeto, uma tupla ou em uma coleção de valores. OCL define quatro tipos de coleção (*Set*, *OrderedSet*, *Bag* e *Sequence*). Uma coleção *Set* não permite elementos duplicados, ao contrário de coleções do tipo *Bag*. Uma coleção *OrderedSet* se assemelha ao *Set*, acrescentando a ordenação dos elementos. Já uma coleção *Sequence* se assemelha ao *Bag*, também acrescentando a ordenação dos elementos. A biblioteca padrão da OCL define uma série de operações para cada tipo. Operações lógicas (*and*, *or*, *xor*, *implies*, *if-then-else-endif*), operações para coleções (*size*, *includes*, *isEmpty*) e quantificadores (*forAll*, *exists*) são exemplos de tais operações.

OCL pode ser usada para especificar restrições a respeito da estrutura e do comportamento de um sistema. Invariantes são restrições estruturais definidas no contexto de um tipo. Uma invariante é definida por uma expressão booleana que deve ser verdadeira para todas as instâncias daquele tipo durante todo o seu tempo de vida. O formato geral de uma invariante OCL é: **context** <Tipo> **inv**: <expressão booleana>.

As pré e pós-condições de uma operação também podem ser especificadas por expressões OCL. Neste caso, o contexto será identificado pelo nome da classe ou interface acompanhado do nome da operação. As palavras reservadas *pre* e *post* identificam pré e pós-condições, respectivamente. Novamente, todas as expressões que são utilizadas em pré e pós-condições devem ser expressões que retornem valor booleano.

A Figura 10 representa um modelo de classes UML que será usado para exemplificar a definição de uma invariante, de uma pré-condição e de uma pós-condição.

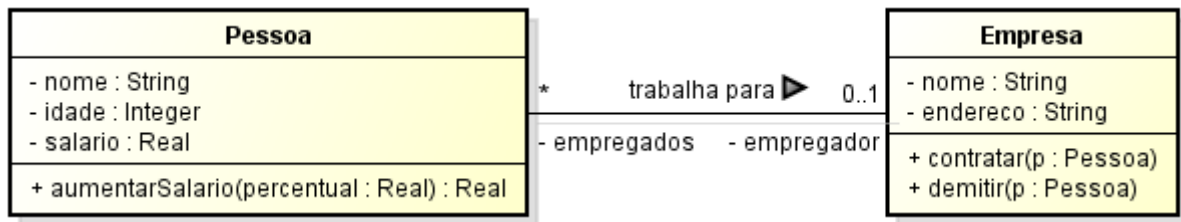


Figura 10: Exemplo de modelo de classes.

A Figura 11 mostra um exemplo de invariante OCL que especifica que todas as instâncias da classe *Pessoa* devem ter um salário não negativo. Como a invariante é definida no contexto da classe *Pessoa*, a expressão *self.salario*  $\geq 0$  deve ser verdadeira para todas as instâncias dessa classe. A palavra reservada *self* se refere à instância da classe definida como contexto (*Pessoa*).

```

context Pessoa
inv: self.salario >= 0
  
```

Figura 11: Exemplo de invariante OCL.

A Figura 12 representa uma pré-condição e uma pós-condição para a operação *contratar*(*p* : *Pessoa*), da classe *Empresa*, representada na Figura 10. A pré-condição, indicada pela cláusula *pre*, garante que a pessoa passada no parâmetro *p* já não é um empregado da empresa. A pós-condição, indicada pela palavra reservada *post*, garante que, após a execução da operação, a pessoa passada como parâmetro foi de fato adicionada ao conjunto de empregados.

```

context Empresa::contratar(p : Pessoa)
pre: empregados->excludes(p)
post: empregados->includes(p)
  
```

Figura 12: Exemplos de pré e pós-condições OCL.

## 2.5 Teste de Software

A cada dia as empresas tornam-se mais dependentes de soluções de software, qualquer que seja o ramo de atuação. À medida que seus processos de negócio são materializados em

produtos de software, a complexidade dos sistemas aumenta e, conseqüentemente, aumenta a preocupação com a qualidade desses sistemas. Quanto mais complexo o sistema, maior será a probabilidade de erros acontecerem no seu desenvolvimento. Tais erros geralmente são provocados pelas pessoas envolvidas no projeto, sejam desenvolvedores ou representantes dos usuários. Erros ocorrem porque a atividade de construção de software depende da habilidade, da capacidade de interpretação e da execução coordenada de atividades por parte das pessoas que o desenvolvem. Mesmo fazendo uso de métodos e ferramentas de desenvolvimento de software, é comum que defeitos estejam presentes em qualquer software. Para que os sistemas não sejam liberados para utilização repletos de defeitos, devem ser aplicadas atividades que garantam sua qualidade. A garantia de qualidade de software é composta por atividades de verificação e validação de software, com o objetivo de garantir que tanto o modo pelo qual o software está sendo construído quanto o produto em si estejam em conformidade com as especificações e as necessidades dos interessados (DELAMARO, MALDONADO e JINO, 2007).

A verificação consiste em avaliar a consistência de uma implementação com uma especificação. Já a validação diz respeito a avaliar se o software desenvolvido atende às reais necessidades dos usuários (PEZZÈ e YOUNG, 2008). As atividades de verificação podem ser classificadas em estáticas e dinâmicas. A verificação estática contempla técnicas de análise estática das representações do sistema, tais como a revisão dos documentos do projeto ou do código-fonte do sistema. Em contrapartida, a verificação dinâmica atua sobre o software em execução e se preocupa em observar o comportamento do produto sob diferentes condições de execução.

O termo “teste de software” refere-se à verificação dinâmica do comportamento de um programa frente a um conjunto finito de casos de teste (UTTING e LEGEARD, 2007). Trata-se de verificação dinâmica porque há a necessidade de colocar o software em operação para

avaliar se determinados dados de entrada geram determinados dados de saída (resultado esperado). Os dados de entrada também são conhecidos como *dados de teste*. O resultado esperado da execução de uma operação também é chamado de *oráculo* (BINDER, 2000). Ao par formado por dados de teste e por seus respectivos oráculos dá-se o nome de *caso de teste* (DELAMARO, MALDONADO e JINO, 2007).

O teste de software pode ser representado em três dimensões, que indicam “quando”, “o quê” e “como” testar. A primeira dimensão contempla os *níveis de teste*, que dizem respeito ao estágio da atividade de teste em relação à atividade de desenvolvimento; a segunda dimensão representa os *tipos de teste*, que se referem às características do software que serão testadas; e, finalmente, a terceira dimensão considera as *técnicas de teste*, que indicam como realizar os testes em função dos insumos recebidos para o teste, tais como especificação de requisitos ou código-fonte. Essas três dimensões estão representadas na Figura 13 e são descritas nas subseções 2.5.1, 2.5.2 e 2.5.3.

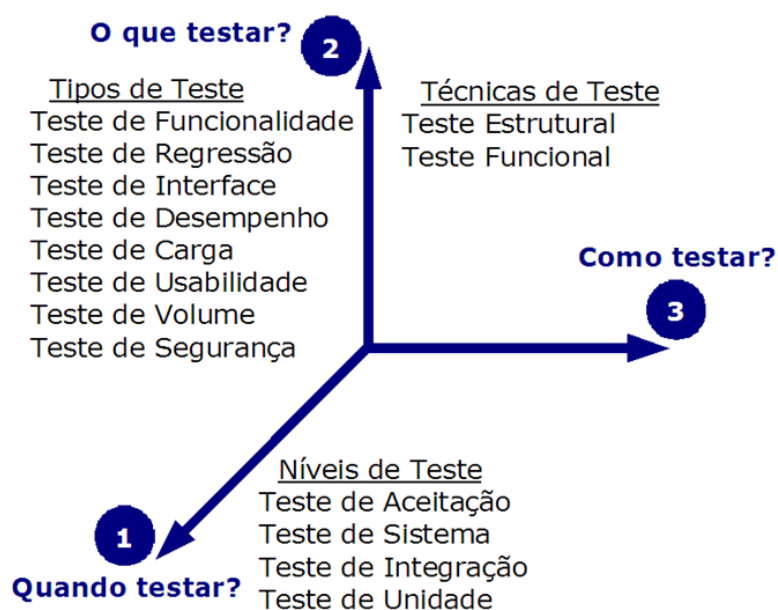


Figura 13: Dimensões do Teste de Software (UTTING e LEGEARD, 2007).

O teste de software tem como principais objetivos encontrar e prevenir defeitos, bem como, prover informações sobre o nível de qualidade do software (POL, TEUNISSEN e

VEENENDAAL, 2002). É importante salientar que a realização de testes sobre um software não permite provar que o software está correto, isto é, o teste pode revelar a presença de defeitos, mas nunca sua ausência. Outro princípio importante é o de que testar todas as combinações de entradas e resultados esperados não é viável. Em vez do teste exaustivo, riscos e prioridades devem ser levados em consideração para dar foco à atividade de teste. Além disso, podem-se utilizar métricas que indiquem a adequação dos testes aos riscos e prioridades estipulados, tais como a de cobertura de testes. A métrica de cobertura indica quão completamente um conjunto de testes exercita as capacidades de um software (MCGREGOR e SYKES, 2001). Desta forma a medida de cobertura de testes, geralmente representada em termos percentuais, pode ser utilizada para demonstrar o nível de qualidade do software e para tomada de decisões quanto à continuidade ou não dos testes.

### **2.5.1 Níveis de Teste**

As atividades de teste devem ocorrer em diferentes níveis, paralelamente às atividades de desenvolvimento de software (POL, TEUNISSEN e VEENENDAAL, 2002). Desta forma, à medida que atividades de desenvolvimento são realizadas, atividades de teste correspondentes às de desenvolvimento também são executadas. Independentemente do processo de testes utilizado os níveis de teste são os seguintes:

- *Teste de Unidade*: testa as menores unidades do software de maneira independente, tais como um método ou uma classe simples, visando garantir que estes atendam às especificações funcionais e de arquitetura; geralmente realizado por programadores.
- *Teste de Integração*: como os componentes são construídos e testados separadamente, ao serem acoplados deve-se verificar se eles interagem corretamente; geralmente realizado por programadores.

- *Teste de Sistema*: os testes de sistema têm foco no funcionamento do sistema como um todo, para validar a exatidão e a perfeição na execução das funções requeridas; geralmente realizado por testadores.
- *Teste de Aceitação*: tem a finalidade de validar se o software está apto a entrar em produção e executar as funções a que se propôs; geralmente realizado pelo cliente ou usuários.

### **2.5.2 Tipos de Teste**

Os tipos de teste dizem respeito às características do software que devem ser testadas, incluindo necessidades de negócio e restrições tecnológicas. Segundo Delamaro, Maldonado e Jino (2007), o tipo mais comum de teste é o *teste funcional* ou comportamental, cujo objetivo é identificar defeitos referentes aos requisitos funcionais do software, em qualquer nível de teste. O *teste de regressão* tem como propósito garantir que os defeitos encontrados foram corrigidos e que as correções ou inserções de novos códigos em determinados locais do software não afetaram outras partes inalteradas do produto. O *teste de desempenho* mede e avalia o tempo de resposta, o número de transações e outros requisitos sensíveis ao tempo de resposta do sistema. O *teste de usabilidade* verifica a facilidade que o software possui de ser claramente entendido e facilmente operado pelos usuários. O *teste de segurança* visa garantir que os dados ou funções de um sistema possam ser acessados apenas por atores autorizados a acessá-las.

### **2.5.3 Técnicas e Critérios de Teste**

Quanto à origem das informações utilizadas para definição dos casos de teste, o teste pode ser classificado segundo duas técnicas: 1) a técnica funcional, ou caixa preta, em que o

teste considera apenas a especificação do software, ignorando completamente sua estrutura lógica interna e; 2) a técnica estrutural, ou caixa branca, onde o teste leva em conta o código-fonte do software a ser testado e tem como objetivo avaliar o comportamento interno do componente de software.

O teste funcional ou caixa preta é utilizado para avaliar se as funções do software estão sendo realizadas de acordo com a especificação de requisitos. Nessa técnica, os detalhes de implementação não são considerados, e o software é avaliado sob a ótica do usuário. Essa técnica de teste pode revelar qualquer tipo de defeito no sistema sob teste e em qualquer nível de teste. É possível também utilizar todos os valores definidos na especificação de requisitos, exercitando todas as combinações possíveis. No entanto, como o domínio de entrada pode ser infinito ou muito grande, o teste pode se tornar impraticável em função da escassez de tempo e recursos.

Desta forma, é necessário aplicar critérios para seleção dos casos de teste visando a diminuição do domínio de entrada. Um *critério de teste* é uma regra ou diretriz utilizada de forma sistemática para avaliar um conjunto de dados e selecionar aqueles que são representativos para o teste (PEZZÈ e YOUNG, 2008). No que se refere à técnica funcional, os principais critérios são os seguintes:

- *Particionamento em classes de equivalência*: deve-se identificar o domínio de entrada na especificação e dividi-lo em duas classes: uma composta por valores válidos e outra composta por valores inválidos. Pode-se assumir que o comportamento do sistema será similar para qualquer valor da classe válida e, da mesma forma, para qualquer valor da classe inválida. Sendo assim, não é necessário utilizar todas as entradas válidas e inválidas possíveis no teste. Basta utilizar uma entrada válida e outra inválida, assumindo que são valores representativos de cada classe.

- *Análise de valor limite*: casos de teste que exploram condições-limite do software têm maior probabilidade de revelar defeitos. São essas condições que limitam as classes de equivalência. Sendo assim, a análise de valor limite considera a criação de casos de teste que exploram as fronteiras entre as classes válidas e inválidas, utilizando valores que estão imediatamente acima ou abaixo dos limites especificados.
- *Grafo de causa-efeito*: os critérios anteriores são, de certa forma, limitados por não considerarem combinações de dados de entrada. Nesse critério são utilizadas combinações de dados de entrada (causas) e suas respectivas ações esperadas (efeitos) para construir um grafo de causas e efeitos, que é convertido em uma tabela de decisão, da qual são derivados os casos de teste.

Como todos os critérios da técnica funcional baseiam-se apenas na especificação do produto testado, a qualidade de tais critérios depende fortemente da qualidade da especificação de requisitos (DELAMARO, MALDONADO e JINO, 2007). Sendo assim, é importante que sejam realizadas ações de verificação estática da documentação de requisitos que será utilizada como insumo para a atividade de teste.

A técnica de teste estrutural ou caixa branca avalia a implementação do sistema em termos de componentes elementares do programa. Tal técnica implica em testar os caminhos lógicos do sistema por casos de teste que exercitem conjuntos específicos de condições, laços e caminhos. A técnica estrutural é considerada complementar ao teste funcional por não revelar muitos defeitos, tendo em vista que os casos de teste produzidos pelo programador são pouco representativos.

## 2.6 Trabalhos Relacionados

Diversas abordagens para teste de Web Services já foram propostas, sendo, inclusive, descritas e comparadas em alguns *surveys* (CANFORA e PENTA, 2009; BOZKURT, HARMAN e HASSOUN, 2010; ENDO e SIMÃO, 2010). Grande parte das abordagens para teste de Web Services existentes refere-se a Teste Baseado em Especificação (*Specification-Based Testing*), cujo propósito é derivar os casos de teste a partir dos contratos dos serviços.

Os trabalhos relacionados a Teste de Web Services Baseado em Especificação podem ser divididos em duas categorias: a) teste baseado na especificação padrão de Web Services e; b) teste baseado em especificações semânticas de serviços. A primeira categoria inclui trabalhos que se caracterizam por realizar a geração de casos de teste a partir da análise sintática das especificações WSDL e XML Schema (BAI *et al.*, 2005; TSAI, CHEN e PAUL, 2005; HANNA e MUNRO, 2007; BARTOLINI *et al.*, 2009). A segunda categoria de trabalhos inclui abordagens para geração de casos de teste a partir de linguagens da Web Semântica para descrição dos serviços, nos quais as restrições de negócio são representadas como ontologias no formato *Web Ontology Language* (OWL) (SINHA e PARADKAR, 2006; PARADKAR *et al.*, 2007; DAI *et al.*, 2007; WANG *et al.*, 2007; BAI *et al.*, 2008; LEE, BAI e CHEN, 2008; JOKHIO, 2009; JOKHIO, DOBBIE e SUN, 2009; RAMOLLARI *et al.*, 2009).

Mais recentemente, os trabalhos de Noikajana e Suwannasart (NOIKAJANA e SUWANNASART, 2009) e Askaruinisa e Abirami (ASKARUINISA e ABIRAMI, 2010) propuseram o uso de restrições em OCL para descrição de pré e pós-condições de Web Services, considerando que OCL é mais expressiva que OWL. Tais trabalhos propõem abordagens para geração de casos de teste para Web Services baseada em contratos, usando o formato *Web Service Semantics* (WSDL-S) (W3C, 2005) e OCL. Ambas as abordagens utilizam métodos para seleção de casos de teste baseados em análise combinatória. Tais

métodos têm como objetivo reduzir a quantidade de combinações a serem testadas, proporcionando níveis de cobertura satisfatórios. A abordagem de Noikajana e Suwannasart utiliza o método denominado *Pair-Wise Testing* (PWT). Já a abordagem de Askaruinisa e Abirami utiliza o método de *array* ortogonal, ou *Orthogonal Array Testing* (OAT). Nesse último trabalho, os autores comparam o método proposto com o método PWT de Noikajana e Suwannasart. Tais trabalhos, no entanto, não se aplicam diretamente ao teste de Web Services RESTful do tipo CRUD, tendo em vista que cobrem apenas o problema de geração de dados de teste em Web Services que: i) não modificam o estado do sistema; ii) manipulam tipos de dados simples e; iii) se baseiam no formato WSDL-S, específico para descrição de serviços WS-\*.

A área de teste de serviços RESTful ainda é pouco explorada e há poucos trabalhos publicados, dentre os quais destacam-se os de Chakrabarti e Rodriquez (CHAKRABARTI e RODRIQUEZ, 2010) e Reza e Van Gilst (REZA e VAN GILST, 2010). O primeiro apresenta uma notação formal baseada no formato WADL e um algoritmo para testar automaticamente a conectividade (*connectedness*) de Web Services RESTful através de seus endereços URI. O segundo trabalho apresenta uma proposta de *framework* para teste de serviços RESTful, realizando perturbação de dados sobre uma especificação dos parâmetros de entrada. Entretanto, o trabalho utiliza formatos XML próprios para representação de serviços e de parâmetros de entrada, o que pode dificultar sua aceitação. Ambos os trabalhos, no entanto, ainda estão em fase de desenvolvimento e não apresentam resultados práticos, limitando-se a demonstrações dos métodos sem o apoio de ferramentas.

Outra categoria de trabalhos que merece destaque inclui as abordagens para especificação semântica de Web Services RESTful. Ontologias OWL podem ser referenciadas por descrições OWL-S e WSDL-S. No entanto, tais formatos são aderentes à especificação WSDL 1.1, que não prevê a descrição de Web Services RESTful. Apenas o formato

SAWSDL (W3C, 2007), lançado mais recentemente, é compatível com a especificação WSDL 2.0 (W3C, 2007), que proporciona a descrição de tal categoria de serviços. Desta forma, trabalhos como os de Sheth, Gomadam e Lathem (2007), Battle e Benson (2008), Alowisheq, Millard e Tiropanis (2009) e Maleshkova, Pedrinaci e Domingue (2009) propuseram formatos para descrição semântica de Web Services RESTful. Tais abordagens, poderiam, a princípio, auxiliar a geração de casos de teste para Web Services RESTful, adicionando semântica aos descritores de serviço. No entanto, elas têm em comum o fato de estender os descritores de serviço com formatos próprios para representar as restrições semânticas, o que tornaria necessário avaliá-las e compará-las com as linguagens de especificação tradicionais.

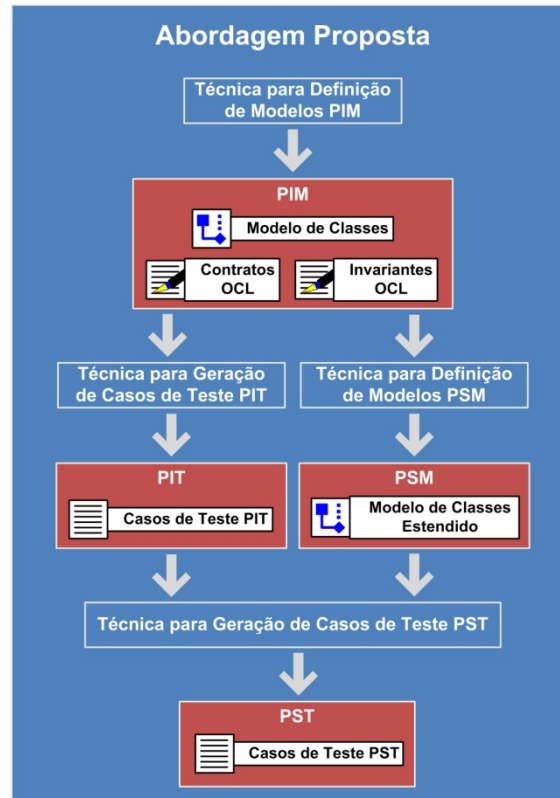
### **3 ABORDAGEM PROPOSTA**

Este capítulo descreve a abordagem proposta para teste de Web Services RESTful. A seção 3.1 apresenta uma visão geral da abordagem proposta. A seção 3.2 descreve um domínio fictício que será usado para exemplificar a aplicação da abordagem ao longo do capítulo. A seção 3.3 descreve a primeira técnica da abordagem, utilizada para definir um modelo independente de plataforma. A seção 3.4 apresenta a técnica para definição de modelos específicos para Web Services RESTful. A seção 3.5 descreve a técnica para geração de casos de teste independentes de plataforma. A seção 3.6 mostra a técnica para geração de casos de teste específicos para Web Services RESTful. A seção 3.7 comenta aspectos relacionados à execução dos casos de teste projetados.

#### **3.1 Visão Geral**

A abordagem proposta neste trabalho é baseada na especificação MDA (OMG, 2003), onde o processo de desenvolvimento de software é dirigido pela atividade de modelagem. Na MDA, os modelos produzidos são refinados sucessivamente até a geração do código-fonte do software. A abordagem aqui proposta segue uma linha similar à de MDA, visando produzir casos de teste para Web Services RESTful do tipo CRUD, que manipulam recursos baseados em tipos de dados complexos (nível 2 do modelo de maturidade de Richardson, descrito na seção 2.2.3).

A abordagem é composta por quatro técnicas inter-relacionadas. Tais técnicas têm como objetivo geral sistematizar o processo de especificação e geração de casos de teste. A Figura 14 representa o fluxo de execução das técnicas da abordagem proposta.



**Figura 14: Fluxo de atividades da abordagem proposta.**

A primeira técnica da abordagem tem como objetivo definir um modelo estrutural de domínio, conhecido na abordagem MDA como *Platform Independent Model* (PIM) (OMG, 2003). O PIM, neste caso, é composto por três artefatos: i) um modelo de classes UML, representando as entidades do domínio (com seus atributos e operações) e seus respectivos relacionamentos; ii) invariantes OCL, usadas para definir restrições sobre os valores dos atributos e; iii) pré e pós-condições OCL que devem ser satisfeitas para execução de cada operação do serviço.

O uso de modelos UML na primeira técnica se justifica por ser um padrão de mercado amplamente utilizado e por possuir elementos de notação diretamente mapeáveis para elementos das linguagens de programação orientadas a objetos. O uso da OCL para especificação dos contratos é justificado por três motivos: i) OCL é considerada mais representativa que ontologias para definição de restrições semânticas sobre um modelo de domínio (NOIKAJANA e SUWANNASART, 2009); ii) OCL é menos intimidante do que as

linguagens formais tradicionais (WARMER e KLEPPE, 2003); iii) OCL é aderente aos padrões do OMG e, conseqüentemente, mais adequada para uma abordagem baseada na MDA (WARMER e KLEPPE, 2003).

A segunda técnica objetiva a definição do modelo específico de plataforma (*Platform Specific Model* – PSM) (OMG, 2003) a partir do PIM produzido pela técnica anterior. Nesta técnica, o modelo de classes UML é estendido através da adição de estereótipos e *tagged values* que permitem a representação de restrições inerentes à tecnologia de Web Services RESTful. Desta forma, informações como o formato de representação consumido ou produzido por uma operação, bem como o modelo de URI utilizado são representados no próprio modelo, tornando desnecessária a geração de um descritor WADL.

A terceira técnica pode ser realizada paralelamente à definição do PSM e tem como propósito produzir casos de teste independentes de plataforma (*Platform Independent Test* – PIT). Essa técnica aplica algoritmos que derivam casos de teste a partir das restrições do modelo de classes e das restrições OCL, combinando critérios tradicionais de projeto de casos de teste baseado em especificação tais como: particionamento em classes de equivalência, análise de valor-limite e tabela de decisão.

Finalmente, a quarta técnica é aplicada para geração dos casos de teste específicos de plataforma (*Platform Specific Test* – PST). Essa técnica recebe como entrada o PSM e o PIT produzidos nas etapas anteriores e aplica um algoritmo que combina os testes do PIT com testes sobre as restrições tecnológicas do PSM, produzindo os casos de teste específicos para tecnologia REST. Nessa técnica o caso de teste será composto por um determinado conjunto de requisições HTTP e assertivas de acordo com o objetivo do teste.

### 3.2 Descrição do Domínio de Exemplo

Para exemplificar a aplicação da abordagem, será utilizado um domínio simples, denominado “Escola”, incluindo serviços que manipulam instâncias das classes *Curso* e *Aluno*. Os serviços devem permitir a criação, consulta, alteração e remoção de instâncias dessas classes.

Um curso possui as propriedades código e nome. Tais atributos devem obedecer às seguintes restrições:

- o código do curso deve estar entre 1 e 99;
- não podem existir dois ou mais cursos com o mesmo código;
- o tamanho do nome do curso deve estar entre 1 e 20 caracteres.

Um aluno, por sua vez, é descrito por matrícula, nome, sexo, um atributo que indica se a matrícula está ativa ou não. Um aluno pode ter diversos números de telefone. Os valores desses atributos devem respeitar as seguintes restrições:

- a matrícula do aluno deve estar entre 1 e 999;
- não podem existir dois ou mais alunos com a mesma matrícula;
- o tamanho do nome do aluno deve estar entre 1 e 60 caracteres;
- o sexo do aluno deve ser “masculino” ou “feminino”;
- um aluno pode ter zero, um ou vários números de telefone.
- cada número de telefone deve possuir exatamente 10 caracteres.

Além disso, todo aluno deve estar matriculado em um curso, e um curso pode ter vários alunos matriculados.

### 3.3 Técnica para Definição de Modelos PIM

#### 3.3.1 Modelos de Classes

O primeiro artefato que compõe o PIM é o modelo de classes. Tal modelo contempla classes de serviço e de domínio, bem como, classes de representação baseadas nas classes de domínio. As classes de serviço contêm as operações CRUD que manipulam instâncias de cada classe de domínio. Cada classe de domínio representa uma entidade do negócio cujas instâncias são tratadas pelas operações das classes de serviço. As classes de representação servem como estruturas de transmissão de dados entre camadas, de modo similar ao padrão *Data Transfer Object* (DTO) (DAIGNEAU, 2012). Neste modelo de classes cada tipo de classe utiliza um estereótipo específico: i) classes de serviço recebem o estereótipo `<<crud>>`; ii) classes de domínio utilizam o estereótipo `<<entity>>` e; iii) classes de representação são estereotipadas com `<<representation>>`. Essa técnica de modelagem permite a criação de outros PSMs para sistemas CRUD aproveitando o mesmo PIM. A Figura 15 representa um exemplo de modelo de classes produzido através da técnica descrita para o domínio Escola.

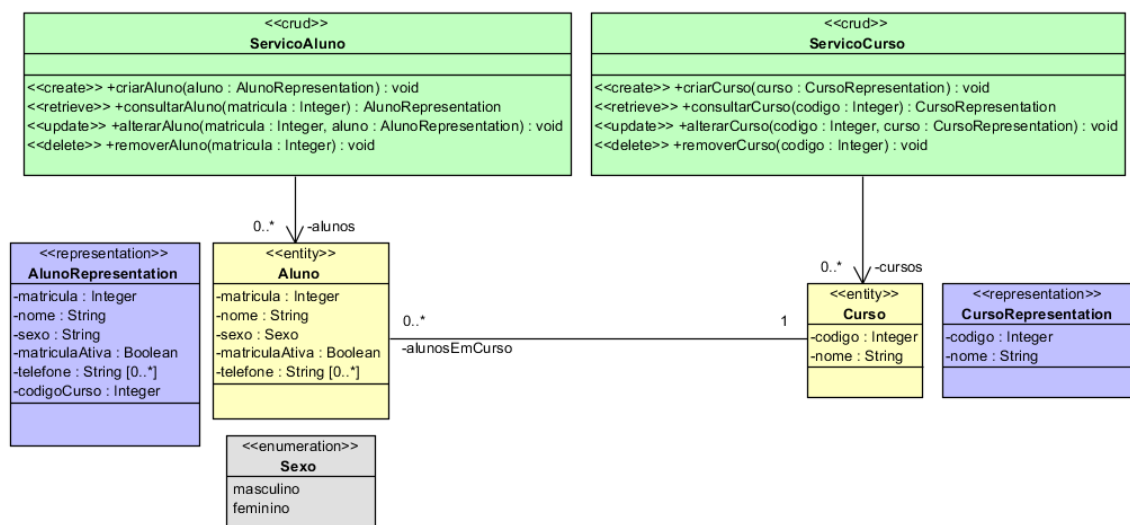


Figura 15: Modelo de classes do domínio Escola.

O uso de classes de representação, indicadas pelo estereótipo <<representation>>, se faz necessário na especificação do serviço porque determinadas operações podem ter uma lista de parâmetros muito extensa e, de forma recorrente, outras operações do mesmo serviço podem possuir a mesma lista de parâmetros. Desta forma, a classe de representação serve para agregar os dados que são passados como parâmetros nas operações, simplificando a definição das listas de parâmetros. Além disso, em termos de implementação, geralmente Web Services RESTful são criados utilizando o padrão DTO, para não expor as classes de domínio para o cliente e porque, geralmente, os objetos de domínio estão conectados em uma rede complexa que é difícil de serializar (FOWLER, 2006).

As classes de representação são derivadas das classes de domínio. Cada classe de domínio deve ter uma classe de representação. A definição de uma classe de representação deve obedecer às seguintes regras:

- para cada classe de domínio deve-se criar uma classe de representação, cujo nome será o mesmo nome da classe de domínio sucedido pelo sufixo “*Representation*”;
- para cada atributo da classe de domínio cujo tipo de dado for numérico inteiro ou de ponto flutuante, assim como booleano ou String, criar um atributo idêntico na classe de representação;
- para cada atributo da classe de domínio que represente um relacionamento com outra classe de domínio, criar um atributo na classe de representação com tipo de dado correspondente ao atributo identificador da classe referenciada (no exemplo da Figura 15, em vez de criar um atributo de tipo *Curso* em *AlunoRepresentation* foi criado um atributo de tipo *Integer* para se referenciar ao código do curso);
- para cada atributo da classe de domínio cujo tipo de dado for uma enumeração, criar um atributo na classe de representação com tipo de dado *String* (no exemplo da Figura

15 o tipo de dado do atributo “sexo” da classe *AlunoRepresentation* foi definido como String, apesar de seu atributo correspondente ter tipo de dado definido como uma enumeração.

### 3.3.2 Invariantes OCL

Um modelo de classes pode não ser suficiente para representar todas as restrições do domínio. Desta forma, esta técnica utiliza a OCL para expressar invariantes de classe e pré e pós-condições de operações. Invariantes OCL são utilizadas nesta técnica para definir restrições típicas de sistemas CRUD, como restrições sobre os valores dos atributos, restrições de unicidade de instâncias e restrições de existência.

As faixas de valores dos atributos das classes de domínio devem ser delimitadas com invariantes. Já os valores dos atributos das classes de representação devem ser restringidos através de pré-condições, tendo em vista que objetos das classes de representação são passados como parâmetros aos métodos. Portanto, para evitar redundância, as restrições sobre valores de atributos são concentradas na respectiva classe de serviço e definidas através de operações auxiliares. Essas operações são referenciadas tanto pelas invariantes quanto pelas pré-condições. A Figura 16 representa um exemplo com duas operações auxiliares da classe *ServicoCurso*, que aplicam regras sobre os atributos da classe *Curso*. A primeira operação restringe os valores do atributo código entre 1 e 99, enquanto que a segunda operação determina que o tamanho do atributo nome deve estar entre 1 e 20 caracteres.

```
context ServicoCurso::codigoValido(codigo : Integer) : Boolean
body: codigo >= 1 and codigo <= 99

context ServicoCurso::nomeValido(nome : String) : Boolean
body: (nome.size() >= 1) and (nome.size() <= 20)
```

**Figura 16: Operações auxiliares para restrição de valores sobre atributos.**

Em seguida são definidas as invariantes para os atributos das classes de domínio. Invariantes sobre valores de atributos devem referenciar as operações auxiliares, representadas na Figura 16. A Figura 17 apresenta exemplos de invariantes para a classe *Curso*. As invariantes das linhas 2 e 3 referenciam operações auxiliares que delimitam os valores dos atributos “código” e “nome”, respectivamente, e a invariante da linha 4 representa uma restrição de unicidade de instâncias, para garantir que não existam dois ou mais cursos com o mesmo código.

```

1 context Curso
2   inv invCodigo: servicoCurso.codigoValido(codigo)
3   inv invNome: servicoCurso.nomeValido(nome)
4   inv invCursoUnico: Curso.allInstances()->isUnique(codigo)

```

**Figura 17: Invariantes da classe Curso.**

### 3.3.3 Pré e Pós-Condições OCL

O terceiro passo da técnica consiste na definição de pré e pós-condições OCL. Considerando que operações CRUD possuem comportamentos similares independentemente do domínio, a técnica orienta a escrita de contratos com base em regras específicas para cada tipo de operação. As subseções 3.3.3.1, 3.3.3.2, 3.3.3.3 e 3.3.3.4 apresentam as regras para definição de pré e pós-condições para operações de criação, consulta, alteração e remoção, respectivamente.

#### 3.3.3.1 Regras para Operações de Criação

Os contratos para operações de criação devem obedecer ao seguinte conjunto de regras:

- O contexto do contrato deve indicar a assinatura do método de criação, passando um parâmetro do tipo da classe de representação.

- Para cada invariante sobre valores de atributos definida anteriormente criar uma pré-condição, referenciando a operação auxiliar que define a invariante.
- Definir uma pré-condição indicando que o objeto passado como parâmetro não deve existir no sistema.
- Definir uma pós-condição que garanta que uma nova instância da classe de domínio foi criada, com os mesmos valores passados pelo parâmetro e que tal instância não existia no sistema antes da execução da operação.

A Figura 18 mostra um exemplo de contrato para o método *criarCurso()*, no qual as pré-condições verificam se os parâmetros passados são válidos e se a instância a ser criada já existe, enquanto que a pós-condição indica que após a execução da operação uma nova instância da classe *Curso* deve existir.

```

1 context ServicoCurso::criarCurso(cr : CursoRepresentation)
2   pre codigoValido: self.codigoValido(cr.codigo)
3   pre nomeValido: self.nomeValido(cr.nome)
4   pre cursoNaoExistente: not cursos->exists(c : Curso | c.codigo = cr.codigo)
5
6   post cursoNovoCriado:
7     let novoCurso : Curso =
8       cursos->select(c : Curso | c.oclIsNew() and
9         c.codigo = cr.codigo and
10        c.nome = cr.nome)->asSequence()->first() in
11     cursos = cursos@pre->including(novoCurso)

```

Figura 18: Contrato da operação *criarCurso()*.

### 3.3.3.2 Regras para Operações de Consulta

As operações de consulta geralmente recebem como parâmetro um valor chave que identifica a instância a ser retornada. O contrato para operações de consulta proposto não indica uma pré-condição sobre os valores recebidos como parâmetros, considerando que o tipo de dado do parâmetro já determina o seu domínio de valores possíveis, assim como se a instância não existir, a operação simplesmente não produzirá qualquer retorno. Os contratos para operações de consulta devem, portanto, obedecer às seguintes regras:

- O contexto do contrato deve indicar a assinatura do método de consulta, recebendo um parâmetro correspondente ao atributo identificador da instância da classe a ser recuperada.
- Definir uma pós-condição que garanta o retorno de uma instância que possua o mesmo valor do atributo identificador recebido como parâmetro.

A Figura 19 mostra um exemplo de contrato para o método *consultarCurso()*, no qual um parâmetro equivalente ao atributo “código” é recebido como identificador para a busca, e a pós-condição assegura que os dados do curso correspondente serão retornados.

```

1 context ServicoCurso::consultarCurso(codigo : Integer) : CursoRepresentation
2
3   post cursoRetornado:
4     let curso : Curso = cursos->select(c : Curso | c.codigo = codigo)->
5       asSequence()->first() in
6       result.codigo = curso.codigo and result.nome = curso.nome

```

**Figura 19: Contrato da operação consultarCurso().**

### 3.3.3.3 Regras para Operações de Alteração

Os contratos para operações de alteração devem obedecer as regras a seguir:

- O contexto do contrato deve indicar a assinatura do método de alteração, recebendo um parâmetro correspondente ao atributo identificador da instância da classe a ser alterada e um parâmetro do tipo da classe de representação.
- Para cada invariante sobre valores de atributos da classe cuja instância será alterada, criar uma pré-condição, referenciando a operação auxiliar que define a invariante.
- Definir uma pré-condição indicando que a instância identificada pelo primeiro parâmetro já deve existir no sistema.
- Definir uma pós-condição que garanta que a instância da classe de domínio foi alterada, com os mesmos valores passados pelo segundo parâmetro.

- Definir uma pós-condição que garanta que apenas as alterações sobre a instância indicada tenham sido realizadas sobre as instâncias da classe de domínio, ou seja, as demais instâncias dessa classe devem permanecer inalteradas.

A Figura 20 mostra um exemplo de contrato para o método *alterarCurso()*, no qual as pré-condições verificam se o novo nome do curso é válido e se a instância a ser alterada já existe. As pós-condições também seguem as regras apresentadas.

```

1 context ServicoCurso::alterarCurso(codigo : Integer, cr : CursoRepresentation)
2 pre novoNomeValido: not self.nomeValido(cr.nome)
3 pre cursoExistente: cursos->exists(c : Curso | c.codigo = cr.codigo)
4
5 post cursoAlterado:
6   let cursoAlterado : Curso =
7     cursos@pre->select(c : Curso | c.codigo@pre = codigo)->
8       asSequence()->first() in
9       cursoAlterado.codigo = cursoAlterado.codigo@pre and
10      cursoAlterado.nome = cr.nome
11 post nenhumaModificacaoNoConjuntoDeCursos:
12   cursos = cursos@pre

```

**Figura 20: Contrato da operação *alterarCurso()*.**

### 3.3.3.4 Regras para Operações de Remoção

Os contratos para operações de remoção se baseiam nas seguintes regras:

- O contexto do contrato deve indicar a assinatura da operação de remoção, recebendo um parâmetro correspondente ao atributo identificador da instância da classe a ser removida.
- Definir uma pré-condição indicando que a instância identificada pelo parâmetro deve existir no sistema.
- Definir uma pós-condição que assegure que a instância identificada pelo parâmetro tenha deixado de existir no sistema.

A Figura 21 representa o contrato do método *removerCurso()*. A pré-condição impõe que deve existir uma instância da classe *Curso* cujo valor do atributo código seja igual ao

valor passado como parâmetro. Já a pós-condição obriga que, após a execução da operação, essa instância da classe *Curso* tenha sido removida da memória do serviço.

```

1 context ServicoCurso::removerCurso(codigo : Integer)
2   pre cursoExistente: cursos->exists(c : Curso | c.codigo = codigo)
3
4   post cursoRemovido:
5     let cursoExcluido : Curso =
6       cursos@pre->select(c : Curso | c.codigo@pre = codigo)->
7         asSequence()->first() in
8         cursos = cursos@pre->excluding(cursoExcluido)

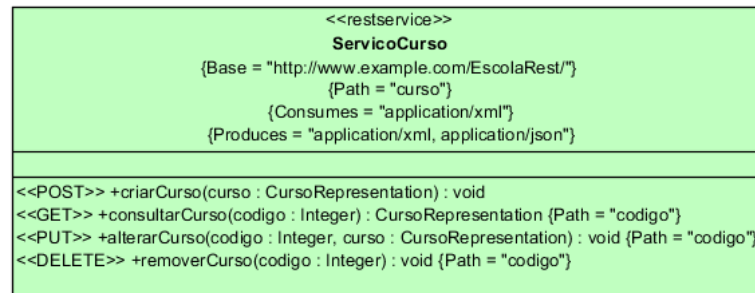
```

Figura 21: Contrato do método `removerCurso()`.

### 3.4 Técnica para Definição de Modelos PSM

A geração do PSM consiste basicamente em estender o modelo de classes do PIM através da redefinição de estereótipos e da inclusão de *tagged values* nas classes de serviço. O estereótipo `<<crud>>` definido no compartimento do nome da classe é substituído pelo estereótipo `<<restservice>>`, para indicar que se trata de um serviço RESTful. Os estereótipos definidos para as operações são substituídos pelos estereótipos `<<POST>>`, `<<GET>>`, `<<PUT>>` e `<<DELETE>>`, em referência aos verbos HTTP, de maneira adequada ao tipo de operação.

As *tagged values* são utilizadas de duas formas: (i) no escopo da classe e (ii) no escopo de uma operação. No escopo da classe são definidas as *tagged values* *Base*, *Path*, *Consumes* e *Produces*. *Base* indica o URI base da aplicação, *Path* adiciona um caminho relativo ao URI base, *Consumes* e *Produces* informam, respectivamente, os tipos de representação consumidos e produzidos pelo serviço, caso sejam comuns a todas as operações. No contexto de uma operação, as três últimas *tagged values* também podem ser utilizadas para adicionar informações específicas. A Figura 22 exemplifica o uso dos estereótipos e *tagged values* utilizados nesta técnica, no contexto da classe *ServicoCurso*.



**Figura 22: Recorte do modelo de classes estendido do domínio Escola.**

Neste caso, o recorte do PSM apresentado indica que:

- o URI base dos serviços da aplicação é `http://www.example.com/EscolaRest` (valor do *tag Base*);
- o URI do serviço *ServicoCurso* é `http://www.example.com/EscolaRest/curso`, dado pela concatenação do URI base com o valor do *tag Path* (“curso”);
- as operações consomem dados no formato *application/xml* (valor do *tag Consumes*) e produzem respostas nos formatos *application/xml* e *application/json* (valor do *tag Produces*);
- o *tag Path* com o valor “código”, utilizado pelas operações de consulta, alteração e remoção, é passado concatenando-se o seu valor ao URI do serviço (por exemplo, a consulta do curso com código = 99 seria transformado no URI `http://www.example.com/EscolaRest/curso/99`).

Além do uso de estereótipos e *tagged values* nas classes de serviço é possível redefinir os tipos de dados utilizados nas classes de domínio e nas classes de serviço, para adequar o modelo a alguma tecnologia de implementação específica. Por exemplo, tipos de dados definidos no PIM como *Integer*, podem ser substituídos no PSM por *int*, *long* ou outro tipo correspondente da plataforma-alvo. Da mesma forma, uma enumeração pode ser

implementada através de valores codificados com outros tipos de dados, como *char* ou *int*, se a tecnologia de implementação não possuir tipos enumerados.

### **3.5 Técnica para Geração de Casos de Teste PIT**

A geração de casos de teste PIT envolve tanto as restrições expressas no modelo de classes quanto aquelas representadas em OCL. A técnica é dividida em duas fases, descritas nas subseções 3.5.1 e .

#### **3.5.1 Tabela de Decisão**

A primeira fase consiste na criação de uma tabela de decisão a partir das restrições expressas no modelo e das invariantes sobre valores de atributos, de acordo com os seguintes passos:

- Gerar dados de teste a partir das faixas de valores definidas nas invariantes, nas enumerações e nos indicadores de multiplicidade dos relacionamentos associados a esta entidade, através da análise de valor-limite, separando-os em valores válidos e inválidos. Para atributos obrigatórios deve-se incluir valor nulo (*null*) nos seus valores inválidos.
- Gerar combinações de entrada com valores válidos e combinações de entrada com apenas um valor inválido e os demais valores válidos. Desta forma, evita-se a criação de combinações com mais de um valor inválido, as quais seriam desnecessárias nesse contexto porque teriam comportamento equivalente.

- Preencher uma tabela de decisão na qual o retorno esperado para as combinações de valores válidos seja positivo (sucesso) e o retorno esperado para as combinações inválidas seja negativo (erro).

Ao aplicar o primeiro passo desta técnica sobre as restrições dos atributos “código” e “nome” da classe *Curso*, definidas na Figura 16, são obtidos os dados de teste representados na Tabela 1:

**Tabela 1: Dados de teste para os atributos da classe Curso.**

| Atributo | Valores Válidos             | Valores Inválidos    |
|----------|-----------------------------|----------------------|
| código   | 1 e 99                      | null, 0 e 100        |
| nome     | 1 caractere e 20 caracteres | null e 21 caracteres |

Ao aplicar o segundo e o terceiro passos sobre os dados de teste da Tabela 1 é produzida uma tabela de decisão, representada pela Tabela 2.

**Tabela 2: Tabela de decisão do PIT.**

| #  | Entrada |               | Resultado Esperado |      |
|----|---------|---------------|--------------------|------|
|    | codigo  | nome          | Sucesso            | Erro |
| 1  | null    | 1 caractere   |                    | X    |
| 2  | null    | 20 caracteres |                    | X    |
| 3  | 0       | 1 caractere   |                    | X    |
| 4  | 0       | 20 caracteres |                    | X    |
| 5  | 1       | null          |                    | X    |
| 6  | 1       | 1 caractere   | X                  |      |
| 7  | 1       | 20 caracteres | X                  |      |
| 8  | 1       | 21 caracteres |                    | X    |
| 9  | 99      | null          |                    | X    |
| 10 | 99      | 1 caractere   | X                  |      |
| 11 | 99      | 20 caracteres | X                  |      |
| 12 | 99      | 21 caracteres |                    | X    |
| 13 | 100     | 1 caractere   |                    | X    |
| 14 | 100     | 20 caracteres |                    | X    |

### 3.5.2 Modelos de Casos de Teste

A segunda fase da técnica consiste na definição dos passos que irão compor os casos de teste. A estratégia de composição é especialmente importante em testes caixa-preta porque

os casos de teste dependem da definição do estado do sistema, realizada através das operações do serviço. O problema é que as operações CRUD possuem relações de precedência, o que pode tornar os testes repletos de passos referentes a operações que ainda não foram testadas.

Uma estratégia utilizada para evitar o problema de precedência de operações e, conseqüentemente, minimizar a quantidade de passos por caso de teste é o padrão Testes Encadeados (*Chained Tests*), descrito por Meszaros (MESZAROS, 2007). Nesse padrão os testes são projetados sequencialmente, de forma que o estado final de um teste seja utilizado como estado inicial do teste subsequente. Entretanto, o uso desse padrão não é recomendado porque fere o princípio de independência entre os testes: a falha de um teste compromete a execução dos testes executados em seguida.

A técnica aqui proposta considera que cada teste deve assegurar a criação e destruição de um estado necessário ao próprio teste. Essa técnica, no entanto, se limita a utilizar apenas as operações fornecidas na interface pública dos serviços, sem o uso de métodos auxiliares de teste para controle do estado do sistema, como sugere o padrão *Back Door Manipulation* (MESZAROS, 2007). Desta forma, deve-se considerar que o sistema sob teste possui estado vazio e que, dependendo do caso de teste, algumas operações dos serviços podem ser utilizadas como auxiliares no seu início (*setup*) e no seu fim (*tearDown*).

A definição dos passos de um caso de teste PIT deve levar em consideração as pré-condições de existência e as pós-condições definidas no modelo PIM. São considerados três tipos de casos de testes: i) os casos de teste positivos, que utilizam dados de teste válidos e esperam por resultados de sucesso; ii) os casos de teste negativos para faixa de valores, que utilizam dados de teste inválidos e esperam por resultados de erro; iii) os casos de teste negativos de unicidade ou integridade referencial, que utilizam dados de teste válidos, porém esperam por resultados de erro. Os dados de teste utilizados nos passos são obtidos na tabela de decisão criada.

Nesta técnica os casos de teste devem ser criados com base em modelos de casos de teste. Nesses modelos, o problema de precedência de operações é contornado assumindo-se que as operações usadas nos casos de teste para verificar pré e pós-condições, bem como, para criar um conjunto de instâncias necessárias ao caso de teste (*setup*) ou remover os objetos criados no caso de teste (*teardown*), estão corretas.

### 3.5.2.1 Casos de Teste PIT de Criação

A Tabela 3 apresenta o modelo de caso de teste positivo para operação de criação. O primeiro passo utiliza a operação de consulta para verificar que a instância a ser criada não existe no sistema. O segundo passo executa a operação de criação, utilizando dados de teste válidos. O terceiro passo utiliza novamente a operação de consulta, para verificar se a instância criada realmente existe no sistema. Finalmente, o quarto passo utiliza a operação de remoção para excluir a instância do sistema e, assim, não afetar os demais testes.

**Tabela 3: Modelo de caso de teste PIT positivo para operação de criação.**

| Sequência de Passos                        | Resultado Esperado | Função do Passo                     |
|--------------------------------------------|--------------------|-------------------------------------|
| 1. consultar<Classe>(<identificador>)      | Erro               | Verifica pré-condição de existência |
| 2. criar<Classe>(<dados de teste válidos>) | Sucesso            | Executa operação principal          |
| 3. consultar<Classe>(<identificador>)      | Sucesso            | Verifica pós-condição               |
| 4. remover<Classe>(<identificador>)        | Sucesso            | <i>Teardown</i>                     |

A Tabela 4 representa um caso de teste PIT para a operação *criarCurso()*, feito com base no modelo da Tabela 3 e com dados de teste provenientes da linha 6 da tabela de decisão (Tabela 2).

**Tabela 4: Exemplo de caso de teste positivo para a operação criarCurso().**

| Sequência de Passos            | Resultado Esperado |
|--------------------------------|--------------------|
| 1. consultarCurso(1)           | Erro               |
| 2. criarCurso({1,1 caractere}) | Sucesso            |
| 3. consultarCurso(1)           | Sucesso            |
| 4. removerCurso(1)             | Sucesso            |

O passo da linha 1 verifica, através da operação *consultarCurso()*, se existe no sistema um curso com código igual a “1”; o resultado esperado é de erro, tendo em vista que não se deseja criar um curso cujo código já existe, para respeitar a invariante de unicidade de instâncias definida na Figura 17. A linha 2 representa a operação sob teste, a qual passa como parâmetros os valores “1”, para código, e “a”, para o nome do curso. Em seguida, na linha 3, novamente a operação de consulta é realizada, mas desta vez se espera que o resultado seja de sucesso, uma vez que a operação de criação de curso foi executada. Finalmente, é executada a operação *removerCurso()* para excluir do sistema os objetos criados no contexto deste caso de teste.

A Tabela 5 representa o modelo de casos de teste negativos para operações de criação, relacionados a restrições sobre valores de atributos. Este modelo considera que o objeto criado com a operação não depende de outros objetos para existir. Se houver necessidade de definir um estado para testar a operação de criação devem-se executar as operações correspondentes antes da execução da operação sob teste. Desta forma, ao se passar um conjunto de dados de teste inválidos é possível determinar que o erro é oriundo de valores inválidos e não de restrições referenciais não contempladas no caso de teste.

**Tabela 5: Modelo de caso de teste PIT negativo para operação de criação (faixa de valores).**

| <b>Sequência de Passos</b>                   | <b>Resultado Esperado</b> | <b>Função do Passo</b>     |
|----------------------------------------------|---------------------------|----------------------------|
| 1. criar<Classe>(<dados de teste inválidos>) | Erro                      | Executa operação principal |

A Tabela 6 representa um exemplo de caso de teste negativo de faixa de valores para a operação *criarCurso()*. Os dados de teste são provenientes da linha 1 da Tabela 2. Como a classe Curso não possui dependência referencial com outra classe do domínio Escola basta exercitar a operação *criarCurso()*, sem usar operações auxiliares.

Tabela 6: Exemplo de caso de teste PIT negativo de faixa de valores para a operação `criarCurso()`.

| Sequência de Passos                            | Resultado Esperado |
|------------------------------------------------|--------------------|
| 1. <code>criarCurso({null,1 caractere})</code> | Erro               |

O teste negativo de unicidade para operações de criação consiste em executar a operação duas vezes, usando os mesmos dados de teste em ambas as execuções. Vale ressaltar que tais dados de teste devem ser dados válidos, para garantir que, se um erro ocorrer, será unicamente pelo motivo de duplicidade de identificadores. Neste caso a primeira execução da operação tem como objetivo definir um estado na memória do sistema que permita avaliar, a partir da segunda execução, se o serviço atende à restrição de unicidade. A Tabela 7 representa o modelo para casos de teste negativos, em relação à unicidade, para operações de criação.

Tabela 7: Modelo de caso de teste PIT negativo para operação de criação (unicidade).

| Sequência de Passos                                                 | Resultado Esperado | Função do Passo            |
|---------------------------------------------------------------------|--------------------|----------------------------|
| 1. <code>criar&lt;Classe&gt;(&lt;dados de teste válidos&gt;)</code> | Sucesso            | <i>Setup</i>               |
| 2. <code>criar&lt;Classe&gt;(&lt;dados de teste válidos&gt;)</code> | Erro               | Executa operação principal |

A Tabela 8 apresenta um exemplo de caso de teste negativo de unicidade para a operação `criarCurso()`, criado utilizando o modelo da Tabela 7 com dados de teste da linha 6 da Tabela 2.

Tabela 8: Exemplo de caso de teste PIT negativo de unicidade para a operação `criarCurso()`.

| Sequência de Passos                         | Resultado Esperado |
|---------------------------------------------|--------------------|
| 1. <code>criarCurso({1,1 caractere})</code> | Sucesso            |
| 2. <code>criarCurso({1,1 caractere})</code> | Erro               |

### 3.5.2.2 Casos de Teste PIT de Consulta

Os casos de teste positivos para operações de consulta devem ser criados com base no modelo da Tabela 9. Para retornar uma instância através de uma operação de consulta tal instância deve ter sido previamente criada. Desta forma, o passo 1 deste modelo corresponde

à operação de criação, utilizando dados de teste válidos. Em seguida é executada a operação de consulta, esperando que a instância correspondente ao identificador passado como parâmetro seja retornada. Ao final, é executada a operação de remoção para limpar a memória do sistema.

**Tabela 9: Modelo de caso de teste PIT positivo para operação de consulta.**

| <b>Sequência de Passos</b>                 | <b>Resultado Esperado</b> | <b>Função do Passo</b>     |
|--------------------------------------------|---------------------------|----------------------------|
| 1. criar<Classe>(<dados de teste válidos>) | Sucesso                   | <i>Setup</i>               |
| 2. consultar<Classe>(<identificador>)      | Sucesso                   | Executa operação principal |
| 3. remover<Classe>(<identificador>)        | Sucesso                   | <i>Teardown</i>            |

A Tabela 10 representa um exemplo de caso de teste positivo para a operação *consultarCurso()*. Novamente são usados os dados de teste da linha 6 da Tabela 2.

**Tabela 10: Exemplo de caso de teste PIT positivo para a operação consultarCurso().**

| <b>Sequência de Passos</b>     | <b>Resultado Esperado</b> |
|--------------------------------|---------------------------|
| 1. criarCurso({1,1 caractere}) | Sucesso                   |
| 2. consultarCurso(1)           | Sucesso                   |
| 3. removerCurso(1)             | Sucesso                   |

O modelo de casos de teste negativos para operações de consulta considera apenas a execução da operação de consulta, sem a criação de instâncias para serem consultadas. Cabe ressaltar que esses testes devem ser aplicados em um sistema com estado inicial vazio. Sendo assim, o identificador passado como parâmetro na operação pode tanto ser um valor válido quanto um valor inválido, porque de qualquer forma nenhuma instância deverá ser retornada. A Tabela 11 representa este modelo.

**Tabela 11: Modelo de caso de teste PIT negativo para operação de consulta.**

| <b>Sequência de Passos</b>            | <b>Resultado Esperado</b> | <b>Função do Passo</b>     |
|---------------------------------------|---------------------------|----------------------------|
| 1. consultar<Classe>(<identificador>) | Erro                      | Executa operação principal |

A Tabela 12 representa um exemplo de caso de teste negativo para a operação *consultarCurso()*. O parâmetro passado é um dado válido, proveniente da Tabela 2,

entretanto, o resultado esperado da execução da operação é um erro, tendo em vista que o caso de teste não garantiu a criação de uma instância de curso com código igual a “1”.

**Tabela 12: Exemplo de caso de teste PIT negativo para a operação consultarCurso().**

| Sequência de Passos  | Resultado Esperado |
|----------------------|--------------------|
| 1. consultarCurso(1) | Erro               |

### 3.5.2.3 Casos de Teste PIT de Alteração

Assim como a consulta, a operação de alteração é dependente da operação de criação, de forma que para uma instância ser alterada ela deve ter sido previamente criada. Desta forma, o modelo de casos de teste positivos para operação de alteração, representado na Tabela 13, considera como passo 1 a execução da operação de criação correspondente à classe de domínio cujo objeto deva ser alterado. Em seguida, no passo 2, a operação de alteração é executada, passando como parâmetros o identificar da instância e dados válidos para os demais atributos. O passo 3 serve para verificar a pós-condição definida para operações de alteração, como mostra o exemplo da Figura 20. O último passo limpa a memória do sistema através da operação *removerCurso()*.

**Tabela 13: Modelo de caso de teste PIT positivo para operação de alteração.**

| Sequência de Passos                                           | Resultado Esperado | Função do Passo            |
|---------------------------------------------------------------|--------------------|----------------------------|
| 1. criar<Classe>(<dados de teste válidos>)                    | Sucesso            | <i>Setup</i>               |
| 2. alterar<Classe>(<identificador>, <dados de teste válidos>) | Sucesso            | Executa operação principal |
| 3. consultar<Classe>(<identificador>)                         | Sucesso            | Verifica pós-condição      |
| 4. remover<Classe>(<identificador>)                           | Sucesso            | <i>Teardown</i>            |

Um exemplo de caso de teste positivo para a operação *alterarCurso()* é representado na Tabela 14. Neste caso, uma instância de curso é criada com código igual a “1” e o nome recebe um caractere qualquer. A alteração consiste em mudar o nome para uma sequência aleatória de 20 caracteres.

Tabela 14: Exemplo de caso de teste PIT positivo para a operação `alterarCurso()`.

| Sequência de Passos                                 | Resultado Esperado |
|-----------------------------------------------------|--------------------|
| 1. <code>criarCurso({1, 1 caractere})</code>        | Sucesso            |
| 2. <code>alterarCurso(1, {1, 20 caracteres})</code> | Sucesso            |
| 3. <code>consultarCurso(1)</code>                   | Sucesso            |
| 4. <code>removerCurso(1)</code>                     | Sucesso            |

O modelo de casos de teste negativos para operação de alteração, representado na Tabela 15, é parecido com o apresentado na Tabela 14. A diferença é que após a operação de alteração não é realizada uma operação de consulta, tendo em vista que não se espera que a operação de alteração seja efetuada com sucesso, já que os dados de teste passados são inválidos. Mesmo assim, são necessárias as operações auxiliares de criação e remoção que atuam como *setup* e *teardown*, respectivamente.

Tabela 15: Modelo de caso de teste PIT negativo para operação de alteração.

| Sequência de Passos                                                                            | Resultado Esperado | Função do Passo            |
|------------------------------------------------------------------------------------------------|--------------------|----------------------------|
| 1. <code>criar&lt;Classe&gt;(&lt;dados de teste válidos&gt;)</code>                            | Sucesso            | <i>Setup</i>               |
| 2. <code>alterar&lt;Classe&gt;(&lt;identificador&gt;, &lt;dados de teste inválidos&gt;)</code> | Erro               | Executa operação principal |
| 3. <code>remover&lt;Classe&gt;(&lt;identificador&gt;)</code>                                   | Sucesso            | <i>Teardown</i>            |

A Tabela 16 mostra um exemplo de caso de teste negativo para a operação `alterarCurso()`. Neste caso de teste inicialmente é criada uma instância de *Curso* com dados válidos, obtidos na linha 6 da Tabela 2, através da operação `criarCurso()`. No passo 2 é executada a operação de alteração, passando os dados da linha 8 da Tabela 2, dos quais o segundo valor (“21 caracteres”) é inválido, o que deve resultar em um erro na execução da operação de alteração. Em seguida, a instância criada é removida pela operação `removerCurso()`.

Tabela 16: Exemplo de caso de teste PIT negativo para a operação `alterarCurso()`.

| Sequência de Passos                                 | Resultado Esperado |
|-----------------------------------------------------|--------------------|
| 1. <code>criarCurso({1, 1 caractere})</code>        | Sucesso            |
| 2. <code>alterarCurso(1, {1, 21 caracteres})</code> | Erro               |
| 3. <code>removerCurso(1)</code>                     | Sucesso            |

### 3.5.2.4 Casos de Teste PIT de Remoção

A Tabela 17 representa o modelo de casos de teste positivos para operação de remoção. A operação de remoção também é dependente da operação de criação. Desta forma, no passo 1 deste modelo é executada a operação de criação, passando dados de teste válidos. O passo 2 executa a operação de remoção, passando o identificador da instância criada no passo 1. Ao final, é executada a operação de consulta para verificar se a instância foi realmente removida, atendendo a pós-condição para operações de remoção.

**Tabela 17: Modelo de caso de teste PIT positivo para operação de remoção.**

| Sequência de Passos                        | Resultado Esperado | Função do Passo            |
|--------------------------------------------|--------------------|----------------------------|
| 1. criar<Classe>(<dados de teste válidos>) | Sucesso            | Setup                      |
| 2. remover<Classe>(<identificador>)        | Sucesso            | Executa operação principal |
| 3. consultar<Classe>(<identificador>)      | Erro               | Verifica pós-condição      |

A Tabela 18 apresenta um exemplo de caso de teste positivo para a operação *removerCurso()*. O primeiro passo cria, a partir dos dados de teste da linha 6 da Tabela 2, a instância que será removida no segundo passo. Ao final, a operação de consulta é executada esperando que o curso não exista mais na memória do sistema.

**Tabela 18: Exemplo de caso de teste PIT positivo para a operação *removerCurso()*.**

| Sequência de Passos             | Resultado Esperado |
|---------------------------------|--------------------|
| 1. criarCurso({1, 1 caractere}) | Sucesso            |
| 2. removerCurso(1)              | Sucesso            |
| 3. consultarCurso(1)            | Erro               |

Esta técnica considera dois tipos de casos de teste negativos para operações de remoção: i) casos de teste que verificam a existência da instância a ser removida e ii) casos de teste que verificam se o sistema permite a remoção de instâncias referenciadas por outros objetos, provocando um erro de integridade referencial. Desta forma há dois modelos de casos de teste negativos para operação de remoção.

A Tabela 19 representa o modelo de casos de teste negativos para operações de remoção, visando testar a restrição existencial inerente a operações de remoção. Neste modelo não são executadas operações de criação, tendo em vista que o objetivo deste caso de teste é executar a operação de remoção passando um identificador inexistente e receber um erro como resposta.

**Tabela 19: Modelo de caso de teste PIT negativo para operação de remoção (restrição existencial).**

| Sequência de Passos                 | Resultado Esperado | Função do Passo            |
|-------------------------------------|--------------------|----------------------------|
| 1. remover<Classe>(<identificador>) | Erro               | Executa operação principal |

O exemplo da Tabela 20 representa um caso de teste negativo para a operação *removerCurso()*, no qual somente esta operação é executada, sobre um sistema com estado vazio, o que deve produzir um erro como resposta.

**Tabela 20: Exemplo de caso de teste PIT negativo para a operação *removerCurso()* – restrição existencial.**

| Sequência de Passos | Resultado Esperado |
|---------------------|--------------------|
| 1. removerCurso(1)  | Erro               |

No que diz respeito à integridade referencial, a Tabela 21 representa o modelo de casos de teste negativos. O passo 1 consiste na criação da instância que será usada posteriormente para avaliar a operação de remoção. No passo 2 é criada a instância dependente estruturalmente daquela criada no passo 1. Em seguida, no passo 3 a operação de remoção é executada, passando como parâmetro o identificador da instância criada no passo 1. Tendo em vista que esta instância é referenciada por outra instância, o resultado esperado é uma mensagem de erro. As operações executadas nos passos 4 e 5 servem para limpar a memória do sistema.

**Tabela 21: Modelo de caso de teste PIT negativo para operação de remoção (integridade referencial).**

| <b>Sequência de Passos</b>                           | <b>Resultado Esperado</b> | <b>Função do Passo</b>     |
|------------------------------------------------------|---------------------------|----------------------------|
| 1. criar<Classe>(<dados de teste válidos>)           | Sucesso                   | <i>Setup</i>               |
| 2. criar<ClasseDependente>(<dados de teste válidos>) | Sucesso                   | <i>Setup</i>               |
| 3. remover<Classe>(<identificador>)                  | Erro                      | Executa operação principal |
| 4. remover <ClasseDependente> (<identificador>)      | Sucesso                   | <i>Teardown</i>            |
| 5. remover<Classe>(<identificador>)                  | Sucesso                   | <i>Teardown</i>            |

A Tabela 22 mostra um exemplo de caso de teste negativo para a operação `removerCurso()`, visando avaliar a restrição de integridade referencial existente entre as classes *Aluno* e *Curso*, tendo em vista que, de acordo com a Figura 15, toda instância de *Aluno* deve referenciar uma instância de *Curso*.

**Tabela 22: Exemplo de caso de teste PIT negativo para a operação `removerCurso()` – integridade referencial.**

| <b>Sequência de Passos</b>                                      | <b>Resultado Esperado</b> |
|-----------------------------------------------------------------|---------------------------|
| 1. criarCurso({1, 1 caractere})                                 | Sucesso                   |
| 2. criarAluno({1, 1 caractere, masculino, true, 1111111111, 1}) | Sucesso                   |
| 3. removerCurso(1)                                              | Erro                      |
| 4. removerAluno(1)                                              | Sucesso                   |
| 5. removerCurso(1)                                              | Sucesso                   |

### 3.6 Técnica para Geração de Casos de Teste PST

A geração dos casos de teste PST consiste em traduzir os passos dos casos de teste PIT para requisições HTTP, de acordo com o método apropriado. Desta forma, as operações indicadas nos casos de teste PIT são transformadas em requisições HTTP, de acordo com a Tabela 23.

**Tabela 23: Mapeamento entre os tipos de operação e os métodos HTTP.**

| <b>Tipo de Operação</b> | <b>Método HTTP</b> |
|-------------------------|--------------------|
| Criação                 | POST               |
| Consulta                | GET                |
| Alteração               | PUT                |
| Remoção                 | DELETE             |

A verificação dos resultados esperados é realizada pela avaliação de assertivas sobre os códigos de resposta e os cabeçalhos das requisições e das respostas. Para cada requisição são feitas uma ou mais assertivas, de modo que o resultado esperado é alcançado se todas as assertivas forem verdadeiras.

A técnica considera nas assertivas apenas os códigos de resposta de sucesso e de erro no cliente (2xx e 4xx), citados na seção 2.2.2. Toda resposta a uma requisição HTTP trás um código de resposta (*Status Code*), frequentemente usado para controle de erros. A Tabela 24 apresenta os códigos de resposta para situações de sucesso e de erro em função de cada método HTTP.

**Tabela 24: Códigos de resposta tradicionais dos métodos HTTP.**

| <b>Método HTTP</b> | <b>Código de Sucesso</b> | <b>Código de Erro</b> |
|--------------------|--------------------------|-----------------------|
| POST               | 201                      | 400                   |
| GET                | 200                      | 404                   |
| PUT                | 201                      | 400                   |
| DELETE             | 200                      | 400 ou 404            |

O código de resposta 201, usado pelos métodos POST e PUT em situações de sucesso, indica que o recurso foi criado ou alterado e pode ser referenciado por um URI. O código 200, compartilhado pelos métodos GET e DELETE em respostas a requisições bem-sucedidas, indica que a resposta contém a representação do recurso ou que o recurso foi encontrado e devidamente removido. Em relação aos códigos de erro, o código 400 usado por POST, PUT e DELETE indica que a operação não foi realizada porque a requisição do cliente contém um conjunto de dados que violam alguma regra da operação. Por fim, o código 404, usado por GET e DELETE, indica que o recurso não foi encontrado no URI especificado.

As assertivas sobre situações de erro baseiam-se apenas no *status code*. Já as assertivas sobre situações de sucesso para os métodos POST, GET e PUT utilizam alguns cabeçalhos do próprio protocolo. Tais cabeçalhos são usados para verificar o cumprimento da implementação com as restrições tecnológicas definidas no modelo PSM.

Requisições com método POST utilizam o cabeçalho *Location* que indica o URI do recurso recém-criado. Desta forma, a verificação da criação de recursos não fica restrita ao código de resposta. O método POST também utiliza, juntamente com o método PUT, o cabeçalho *Content-Type* da resposta para compará-lo com o cabeçalho *Content-Type* da requisição. Este cabeçalho indica os formatos de representação suportados por cada operação (tipos MIME). Assim, é possível verificar se o serviço implementa corretamente o que está definido nas *tagged values Produces* e *Consumes* do PSM. Este tipo de verificação também é realizada com requisições do tipo GET: a diferença é que o conteúdo do cabeçalho *Content-Type* da resposta deve ser comparado com o conteúdo do cabeçalho *Accept* da requisição. Tais regras estão representadas na Tabela 25, que serve como um modelo geral de assertivas para cada tipo de método, considerando respostas HTTP de sucesso e de erro.

**Tabela 25: Modelo geral de assertivas para casos de teste PST.**

| <b>Método HTTP</b> | <b>Resposta HTTP de Sucesso</b>                           | <b>Resposta HTTP de Erro</b>    |
|--------------------|-----------------------------------------------------------|---------------------------------|
| POST               | <i>Status Code</i> = 201                                  | <i>Status Code</i> = 400        |
|                    | <i>Location</i> $\neq$ null                               |                                 |
|                    | <i>Content-Type</i> = <i>Content-Type</i> da requisição   |                                 |
| GET                | <i>Status Code</i> = 200                                  | <i>Status Code</i> = 404        |
|                    | <i>Content-Type</i> $\subset$ <i>Accept</i> da requisição |                                 |
| PUT                | <i>Status Code</i> = 201                                  | <i>Status Code</i> = 400        |
|                    | <i>Content-Type</i> = <i>Content-Type</i> da requisição   |                                 |
| DELETE             | <i>Status Code</i> = 200                                  | <i>Status Code</i> = 400 ou 404 |

A partir deste modelo geral de assertivas é possível derivar modelos de casos de teste específicos por tipo de operação. Para isso, é necessário substituir as operações definidas nos casos de teste PIT pelos métodos HTTP correspondentes, conforme mostra a Tabela 23, e associar as assertivas correspondentes, conforme a Tabela 25.

A Tabela 26 representa o modelo de casos de teste PST positivos para operações de criação. A linha 1 realiza uma requisição do tipo GET, equivalente a uma operação de consulta do PIT, com objetivo de garantir que o recurso a ser criado já não existe na memória do sistema. A linha 2 executa a requisição principal do caso de teste, na qual um recurso deve ser criado utilizando dados válidos. A linha 3 realiza uma nova consulta via GET para verificar se o recurso agora existe no sistema. A requisição com DELETE ao final do caso de teste remove o recurso criado pela requisição da linha 2.

**Tabela 26: Modelo de caso de teste PST positivo para operações de criação.**

| Sequência de Requisições | Resultado Esperado                                            |
|--------------------------|---------------------------------------------------------------|
| 1. GET                   | <i>Status Code = 404</i>                                      |
| 2. POST                  | <i>Status Code = 201</i>                                      |
|                          | <i>Location &lt;&gt; null</i>                                 |
|                          | <i>Content-Type = Content-Type da requisição</i>              |
| 3. GET                   | <i>Status Code = 200</i>                                      |
|                          | <i>Content-Type <math>\subset</math> Accept da requisição</i> |
| 4. DELETE                | <i>Status Code = 200</i>                                      |

Com base no PIT definido na Tabela 4 e no modelo apresentado na Tabela 26, a Tabela 27 representa um caso de teste positivo para a operação *criarCurso()*. O URI é obtido pela concatenação dos valores das *tagged values Base* da classe *ServicoCurso* e *Path* da operação *criarCurso()*, representadas na Figura 22.

**Tabela 27: Exemplo de caso de teste PST positivo para a operação criarCurso().**

| Sequência de Requisições                                                                                                                                                                                                                            | Resultado Esperado na Resposta                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| GET <a href="http://www.example.com/EscolaRest/curso/1">http://www.example.com/EscolaRest/curso/1</a><br>Accept: application/xml, application/json                                                                                                  | <i>Status Code = 404</i>                                                                                      |
| POST <a href="http://www.example.com/EscolaRest/curso">http://www.example.com/EscolaRest/curso</a><br>Accept: application/xml, application/json<br>Content-Type: application/xml<br><br><curso><br><codigo>1</codigo><br><nome>a</nome><br></curso> | <i>Status Code = 201</i><br><i>Location &lt;&gt; null</i><br><i>Content-Type = Content-Type da requisição</i> |
| GET <a href="http://www.example.com/EscolaRest/curso/1">http://www.example.com/EscolaRest/curso/1</a><br>Accept: application/xml, application/json                                                                                                  | <i>Status Code = 200</i><br><i>Content-Type <math>\subset</math> Accept da requisição</i>                     |
| DELETE<br><a href="http://www.example.com/EscolaRest/curso/1">http://www.example.com/EscolaRest/curso/1</a><br>Accept: application/xml, application/json                                                                                            | <i>Status Code = 200</i>                                                                                      |

A Tabela 28 representa um modelo de caso de teste PST negativo para operações de remoção, cujo objetivo é verificar uma restrição de integridade referencial. A linha 1 indica a execução de uma operação de criação de um recurso que será referenciado por outro recurso após a execução do método definido na linha 2. Ao se tentar remover o primeiro recurso criado o resultado esperado deve ser de erro, para não violar a restrição de integridade referencial. As linhas 4 e 5 funcionam como *teardown*.

**Tabela 28: Modelo de caso de teste PST negativo para operações de remoção (integridade referencial).**

| <b>Sequência de Requisições</b> | <b>Resultado Esperado</b>                               |
|---------------------------------|---------------------------------------------------------|
| 1. POST (recurso tipo 1)        | <i>Status Code</i> = 201                                |
|                                 | <i>Location</i> $\neq$ null                             |
|                                 | <i>Content-Type</i> = <i>Content-Type</i> da requisição |
| 2. POST (recurso tipo 2)        | <i>Status Code</i> = 201                                |
|                                 | <i>Location</i> $\neq$ null                             |
|                                 | <i>Content-Type</i> = <i>Content-Type</i> da requisição |
| 3. DELETE (recurso tipo 1)      | <i>Status Code</i> = 400                                |
| 4. DELETE (recurso tipo 2)      | <i>Status Code</i> = 200                                |
| 5. DELETE (recurso tipo 1)      | <i>Status Code</i> = 200                                |

A Tabela 29, apresentada em seguida, representa um caso de teste PST baseado na Tabela 28, no qual se tenta remover um curso que possui um aluno matriculado. Tal operação tenta violar a restrição de integridade referencial, tendo em vista que, conforme o PIM definido na Figura 15, todas as instâncias da classe *Aluno* fazem referência a alguma instância da classe *Curso*.

Tabela 29: Exemplo de caso de teste PST negativo para a operação `removerCurso()`.

| Sequência de Requisições                                                                                                                                                                                                                                                         | Resultado Esperado na Resposta                                                                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| POST http://www.example.com/EscolaRest/curso<br>Content-Type: application/XML<br><br><curso><br><codigo>1</codigo><br><nome>a</nome><br></curso>                                                                                                                                 | <i>Status Code</i> = 201<br><i>Location</i> <> null<br><i>Content-Type</i> = <i>Content-Type</i> da requisição |
| POST http://www.example.com/EscolaRest/aluno<br>Content-Type: application/xml<br><br><aluno><br><matricula>1</matricula><br><nome>a</nome><br><sexo>masculino</sexo><br><matriculaAtiva>>true</matriculaAtiva><br><telefone>111111111</telefone><br><curso>1</curso><br></aluno> | <i>Status Code</i> = 201<br><i>Location</i> <> null<br><i>Content-Type</i> = <i>Content-Type</i> da requisição |
| DELETE<br>http://www.example.com/EscolaRest/curso/1<br>Accept: application/xml, application/json                                                                                                                                                                                 | <i>Status Code</i> = 400                                                                                       |

### 3.7 Execução do Testes

A abordagem proposta nesta dissertação não determina uma ferramenta para execução dos casos de teste produzidos. Qualquer ferramenta ou biblioteca HTTP pode ser utilizada para executá-los. Pode-se, entretanto, recomendar a soapUI (EVIWARE, 2010), uma ferramenta para automação de testes funcionais e não-funcionais para Web Services (WS-\* e REST). Tal ferramenta cobre as principais especificações relacionadas à tecnologia de Web Services e é distribuída em duas versões: uma gratuita (*community*) e outra paga (*professional*).

## **4 ESTUDO EXPERIMENTAL**

Este capítulo apresenta o estudo experimental realizado com o objetivo de avaliar a abordagem proposta. A seção 5.1 introduz o estudo destacando sua motivação, objetivos e questões a serem respondidas. A seção 5.2 descreve o planejamento realizado para condução do experimento. A seção 5.3 detalha os aspectos operacionais do estudo. A seção 5.4 é destinada à análise e interpretação dos resultados obtidos. A seção 5.5 diz respeito à avaliação da validade dos resultados. Por fim, a seção 5.6 tece comentários finais sobre o estudo em questão.

### **4.1 Introdução**

#### **4.1.1 Motivação**

A abordagem descrita neste estudo propõe técnicas para especificação e geração de casos de teste para Web Services RESTful. A própria abordagem indica os tipos de restrição que devem ser testados e propõe a realização de um teste positivo e um teste negativo sobre cada restrição, alcançando sempre cobertura total sobre as restrições identificadas.

Diante disso, um estudo experimental foi realizado com o objetivo de identificar indícios de que a abordagem proposta contribui para a qualidade dos testes. O experimento leva em consideração o fato de que, sem conhecimento da abordagem e seguindo seus próprios critérios de teste, a maioria dos participantes não alcança uma cobertura de restrições arbitrariamente definida como satisfatória. O estudo experimental foi realizado com dois grupos de participantes: o primeiro, com 28 testadores, formado por alunos concluintes do curso de Bacharelado em Sistemas de Informação da Universidade do Grande Rio

(UNIGRANRIO) e o segundo com 7 profissionais de Teste de Software do Serviço Federal de Processamento de Dados (SERPRO) num total de 35 participantes.

#### **4.1.2 Objetivo Global**

O objetivo global deste estudo é verificar qual a proporção de testadores que consegue alcançar a cobertura mínima satisfatória em relação às restrições contidas em um modelo PSM criado conforme preconizado pela própria abordagem. A cobertura mínima satisfatória foi estabelecida em 95% do total de restrições, percentual equivalente a dois desvios-padrão acima da média (considerando uma distribuição normal).

#### **4.1.3 Objetivo da Medição**

O objetivo da medição é caracterizar o percentual de cobertura de restrições positivas e negativas exercitadas pelos casos de teste projetados pelos participantes, seguindo seus próprios critérios de teste, tendo como base um exercício prático específico, com restrições de negócio e restrições tecnológicas previamente definidas.

Além disso, é necessário saber se há diferenças significativas nas observações das duas amostras, de forma que, se houver, será necessário aplicar tratamentos específicos a cada uma.

#### **4.1.4 Objetivo do Estudo**

A estrutura a seguir, proposta por Wohlin *et al.* (2000), sintetiza o objetivo do estudo:

- **Analisar** os casos de teste projetados pelos participantes do estudo a respeito de um exercício específico;

- **Com o propósito** de caracterizar;
- **Referente** à proporção de participantes que alcança ao menos 95% da cobertura de testes sugerida pela abordagem proposta;
- **Do ponto-de-vista** do pesquisador;
- **No contexto de** alunos concluintes do curso de Bacharelado em Sistemas de Informação da UNIGRANRIO e de profissionais de Teste de Software do SERPRO.

#### **4.1.5 Questões**

**Q1:** Qual a proporção de participantes que alcança ao menos 95% da cobertura de restrições alcançada pela abordagem proposta?

**Métrica:** O método de intervalo de confiança para proporções, que determina uma faixa de valores, baseada em uma margem de erro, para se estimar o verdadeiro valor de um parâmetro populacional.

**Q2:** Existem diferenças significativas entre a cobertura média de restrições entre os dois grupos constituintes da amostra?

**Métrica:** O teste U de Mann-Whitney, utilizado para testar se duas amostras independentes provêm de populações com medianas iguais.

## 4.2 Planejamento

### 4.2.1 Seleção do Contexto

Segundo Wohlin *et al.* (2000), a seleção do contexto de um experimento é caracterizada por quatro dimensões: i) o processo (*on-line* ou *off-line*), ii) os participantes, iii) a realidade (real ou modelado) e iv) a generalidade (específico ou geral).

Este estudo utiliza um processo *on-line*, tendo em vista que os participantes produzem casos de teste durante o período do experimento, sob a supervisão do pesquisador.

Os participantes do estudo são alunos e profissionais, conforme informa a seção 5.1.4. Além do fator conveniência, a escolha de tais grupos foi determinada pelo perfil de ambos os grupos, supostamente compatível com a realização das tarefas sugeridas pelo experimento. Os alunos, por estarem no último período do curso em questão, já passaram por disciplinas que envolvem modelagem UML e desenvolvimento de Web Services RESTful, apesar de a maioria não conhecer Teste de Software. Os profissionais, por sua vez, possuem alguma experiência com projeto de casos de teste baseado em especificação, entretanto, a maioria desconhece a tecnologia de Web Services RESTful.

O problema é classificado como modelado, porque os objetos a serem interpretados pelos participantes foram criados especificamente para o experimento, de acordo com critérios arbitrados pelo pesquisador.

O nível de cobertura dos casos de teste de cada participante é determinado em relação a um PSM definido através da abordagem proposta, portanto, o contexto possui caráter específico.

### 4.2.2 Formulação das Hipóteses

Para cada questão relacionada na seção 4.1.5 foram formuladas uma hipótese nula e uma hipótese alternativa.

#### 4.2.2.1 Questão Q1

A hipótese nula declara que a abordagem proposta pouco acrescenta ao resultado prático obtido utilizando o conhecimento acadêmico ou prático dos participantes. Em outras palavras, a cobertura obtida pela aplicação do método é trivial. Para isso, precisamos definir dois parâmetros:

- 1) a porcentagem de cobertura caracterizada como cobertura eficaz ( $ce$ ). Consideramos que o nível crítico para esta fração ( $ce_{crit}$ ) como sendo 95% do total de restrições indicado pela abordagem.
- 2) a fração dos testadores ( $p$ ) que consegue a cobertura eficaz. Consideramos o nível crítico para esta fração ( $ft_{crit}$ ) como sendo 68% (um desvio padrão acima da média numa distribuição normal).

$$H_0: p \geq ft_{crit} \text{ para } ce \geq ce_{crit}$$

A hipótese alternativa considera que a proporção de participantes que alcançam a cobertura mínima estabelecida (95%) é menor que 68%.

$$H_1: p < ft_{crit} \text{ para } ce \geq ce_{crit}$$

#### 4.2.2.2 Questão Q2

A hipótese nula declara que os dois grupos de participantes apresentam um desempenho equivalente no que diz respeito à cobertura de testes. Considerando  $m_a$  como a

mediana de restrições cobertas pelo grupo de alunos e  $m_b$  a mediana de restrições cobertas pelos profissionais, simbolicamente, temos:

$$H_0: m_a = m_b$$

A hipótese alternativa é a afirmativa de que os dois grupos de participantes não são equivalentes em termos de média de restrições cobertas pelos testes.

$$H_a: m_a \neq m_b$$

#### **4.2.3 Variáveis Independentes**

A principal variável independente do estudo é o número de restrições a serem testadas incluídas no modelo PSM submetido aos participantes. Esta variável foi contabilizada, de acordo com a abordagem proposta, com cento e trinta e duas restrições (sessenta e seis positivas e sessenta e seis negativas).

O tipo de participante (aluno ou profissional), bem como, seu nível de conhecimento sobre modelagem com UML, teste de software e Web Services RESTful também são informações independentes que poderiam afetar de forma indesejável os resultados do estudo.

#### **4.2.4 Variáveis Dependentes**

Neste estudo a variável dependente é o número de restrições contidas em um dado modelo PSM cobertas pelos casos de teste projetados por cada participante.

#### **4.2.5 Seleção dos Participantes**

Os participantes do experimento foram escolhidos por conveniência, por se tratarem de pessoas inseridas em empresas do relacionamento do pesquisador. A partir de uma consulta prévia a um universo de cerca de oitenta pessoas, incluindo alunos concluintes do curso de Bacharelado em Sistemas de Informação da UNIGRANRIO e profissionais de teste de software do SERPRO, trinta e cinco pessoas se candidataram, voluntariamente, a participar do estudo experimental. Dos trinta e cinco voluntários, vinte e oito são alunos da UNIGRANRIO e sete são profissionais do SERPRO.

#### **4.2.6 Instrumentação**

Neste estudo foram utilizados os seguintes instrumentos:

##### **a) Termo de Consentimento Livre e Esclarecido (TCLE)**

O TCLE é um documento que contém informações a respeito do procedimento a ser realizado no experimento, ressaltando a liberdade de desistência do participante e o acordo de confidencialidade dos resultados.

##### **b) Questionário de Caracterização do Participante (QCP)**

O questionário QCP foi utilizado para o participante caracterizar algumas informações pessoais e descrever alguns itens interessantes para o estudo como, por exemplo, tempo de experiência em desenvolvimento de software, experiência com teste de software e conhecimento de Web Services RESTful.

**c) Tutorial sobre Teste de Software (T1)**

Os participantes alunos receberam um treinamento assistido sobre Teste de Software, no qual foi apresentado e distribuído um tutorial sobre Teste de Software, focado nos critérios de teste para testes funcionais.

**d) Tutorial sobre Web Services RESTful (T2)**

Os participantes profissionais receberam um treinamento assistido sobre Web Services RESTful. Nesse treinamento, foi apresentado e distribuído um tutorial sobre Web Services RESTful focado em aspectos conceituais do estilo arquitetural REST e tópicos fundamentais a respeito do protocolo HTTP.

**e) Tutorial sobre soapUI (T3)**

Todos os participantes receberam um treinamento assistido acerca da utilização da ferramenta soapUI, no contexto de Web Services RESTful, apoiado por um tutorial.

**f) Exercício Prático 1 (EP1)**

O Exercício Prático 1 consiste de um modelo PSM, criado conforme a abordagem proposta, representando um serviço de controle de filmes. Tal exercício apoiou o treinamento sobre a ferramenta soapUI e serviu para familiarizar os participantes com a forma de especificação de Web Services proposta pela abordagem.

**g) Exercício Prático 2 (EP2)**

O Exercício Prático 2 também consiste de um modelo PSM, criado conforme a abordagem proposta, sobre o domínio Escola, apresentado no capítulo 2. A partir de tal exercício os participantes produziram os casos de teste que são objetos de estudo deste experimento.

Todos os instrumentos utilizados estão disponíveis no endereço <http://sites.google.com/site/thiagoein/instrumentos>. Os instrumentos TCLE, QCP, EP1 e EP2 também são apresentados nos apêndices A, B, C, D e F.

### **4.3 Operação**

A execução do experimento foi realizada em duas semanas distintas. Na primeira semana, o experimento foi realizado com o grupo de alunos, e na segunda semana foi realizado com o grupo de profissionais. Cada grupo realizou 8 horas de treinamento e 8 horas para resolução do exercício EP2, divididas em quatro seções de 4 horas.

O treinamento realizado teve como objetivo assegurar que os participantes do experimento tivessem um conhecimento mínimo para produção dos casos de teste. Cada grupo de participantes recebeu um treinamento específico, de acordo com seus conhecimentos prévios indicados no questionário QCP.

Os grupos foram acomodados em laboratórios de computadores, preparados previamente pelo pesquisador, onde todo o experimento foi realizado, para evitar interferências indesejáveis tanto do ambiente computacional como do ambiente externo. O grupo de alunos utilizou um laboratório da UNIGRANRIO, enquanto que o grupo de profissionais utilizou um laboratório do próprio SERPRO.

A primeira seção de treinamento foi diferenciada para cada grupo. O grupo dos alunos recebeu treinamento teórico sobre Testes de Software, enquanto que o grupo dos profissionais recebeu treinamento teórico sobre Web Services RESTful.

A segunda seção de treinamento de ambos os grupos cobriu a utilização da ferramenta soapUI para definição e execução de testes para Web Services RESTful. A escolha da ferramenta se justifica pelos recursos oferecidos que facilitam a definição de casos de teste.

Nesta seção, ambos os grupos receberam o exercício EP1 para se familiarizarem com o formato de especificação e para servir como referência para definição de testes durante o aprendizado do soapUI.

Na terceira seção ambos os grupos receberam o exercício EP2 e cada participante iniciou a produção de casos de teste seguindo seus próprios critérios de teste. Os casos de teste de cada participante foram criados na ferramenta soapUI e gravados em arquivos identificados e recolhidos pelo pesquisador. A interrupção desta atividade se mostrou necessária para evitar a fadiga dos participantes e, conseqüentemente, interferir negativamente nos resultados.

Finalmente, na quarta seção, os participantes deram continuidade ao exercício EP2, interrompido na terceira seção. Cada participante, ao final da seção, entregou ao pesquisador o arquivo contendo o projeto de casos de teste do soapUI para posterior análise.

#### **4.4 Análise e Interpretação dos Resultados**

Todo o processo de análise e interpretação dos resultados foi apoiado pela ferramenta MS-Excel 2007 e pelo pacote estatístico *Statistica*, versão 10, da StatSoft (STATSOFT, 2011).

A estatística descritiva a respeito do perfil dos participantes encontra-se no Apêndice F.

##### **4.4.1 Restrições Testadas pelos Participantes**

Após a realização do exercício EP2 com todos os participantes, foram analisados todos os casos de teste produzidos, perfazendo um total aproximado de 1.700 casos de teste. A pedido do pesquisador, cada caso de teste possuía um nome que permitisse identificar

facilmente seu objetivo, visando agilizar o processo de contagem. Mesmo assim, cada caso de teste foi analisado para verificar sua intenção e contabilizar a restrição que o caso de teste se propunha a cobrir.

Durante a análise, foram identificados quatro problemas recorrentes em alguns casos de teste, descritos como *smells* por Meszaros (2007): i) casos de teste duplicados, ii) casos de teste compartilhados, iii) casos de teste encadeados e iv) casos de teste compartilhados e encadeados. Diante disso, para efeito de contabilização, os casos de teste duplicados foram desconsiderados, enquanto que os demais foram contados, mas com a indicação do *smell* correspondente. Vale ressaltar que, desta forma, um único caso de teste compartilhado pode contabilizar mais de uma restrição.

Considerando a cobertura mínima fixada (95%) e o total de restrições do PSM (66 positivas e 66 negativas), os participantes deveriam testar 63 restrições de cada tipo ou 126 restrições no total para serem considerados casos de sucesso. Os participantes que não alcançaram tal cobertura foram considerados casos de falha.

A Tabela 30 apresenta a estatística descritiva dos resultados, onde se observa que a média de restrições positivas testadas (62,6) é próxima do total de restrições positivas incluídas no PSM do exercício EP2 (66). Entretanto, a média de restrições negativas testadas é de apenas 39,1 para um total de 66 restrições negativas a serem testadas. Quando ambos os tipos de restrição são agregados observa-se uma média de 101,7 restrições testadas de um total de 132 restrições. Diante desses resultados, percebe-se que há uma tendência maior de se testar as situações válidas das operações, em detrimento das situações com dados inválidos.

**Tabela 30: Estatística descritiva dos resultados.**

|                    | Restrições Positivas Testadas | Restrições Negativas Testadas | Total de Restrições Testadas |
|--------------------|-------------------------------|-------------------------------|------------------------------|
| Observações        | 35                            | 35                            | 35                           |
| Média              | 62,6                          | 39,1                          | 101,7                        |
| Mediana            | 66                            | 40                            | 104                          |
| Moda               | 66                            | 32                            | múltipla                     |
| Frequência da moda | 24                            | 5                             | 3                            |
| Valor mínimo       | 46                            | 18                            | 80                           |
| Valor máximo       | 66                            | 62                            | 128                          |
| Desvio padrão      | 5,9                           | 9,8                           | 12,5                         |

#### **4.4.2 Testes de Hipótese**

##### **4.4.2.1 Questão Q1**

Como será visto na questão 2, a seguir, as duas amostras estudadas são equivalentes em termos de mediana da quantidade de restrições testadas, tratamos ambos os grupos como parte da mesma amostra.

A questão Q1 trata da proporção de participantes que atingem a cobertura mínima satisfatória das restrições definidas no PSM do exercício EP2. Para determinar tal proporção foi utilizado o “Método do Intervalo de Confiança para Proporção”. Um intervalo de confiança é uma faixa de valores usada para se estimar o verdadeiro valor de um parâmetro populacional. Segundo Triola (2008), a margem de erro é a diferença máxima provável (com probabilidade  $(1 - \alpha)$  entre a proporção amostral observada e o verdadeiro valor da proporção populacional. A margem de erro para proporções pode ser calculada pelo método de Agresti e Coull (AGRESTI e COULL, 1998). O intervalo Agresti-Coull é um tipo de aproximação para o intervalo de confiança para a distribuição binomial. Dado que observamos  $X$  sucessos em  $n$  tentativas, podemos definir:

$$\tilde{n} = n + z_{1-\alpha/2}^2$$

$$\tilde{p} = \frac{X + z_{1-\alpha/2}^2 / 2}{\tilde{n}}$$

De onde um intervalo de confiança para  $p$  pode ser dado por:

$$\tilde{p} \pm z_{1-\alpha/2} \sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}$$

Onde:  $z_{1-\alpha/2}$  é o percentil  $(1-\alpha/2)$  da distribuição normal padrão. Usando as fórmulas acima podemos calcular:

$$\tilde{p} = 0,101 \text{ e } z_{1-\frac{\alpha}{2}} \sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}} = 0,095.$$

e

$(p_{min}..p_{max}) = (0,01 .. 0,20)$ , um intervalo de confiança para  $p$ .

Recapitulando as hipóteses previamente definidas:

$$\begin{aligned} H_0: p_{max} &\geq ft_{crit} \\ H_a: ft_{crit} &> p_{max} \end{aligned}$$

Como  $ft_{crit} = 0,68$  podemos rejeitar a hipótese nula e aceitar a hipótese alternativa.

#### 4.4.2.2 Questão Q2

Para verificar se os grupos de participantes são equivalentes foi aplicado o teste de Mann-Whitney. O teste de Mann-Whitney é um teste não-paramétrico que estima a probabilidade que dois grupos independentes (alunos e profissionais) sejam provenientes da mesma população (GUIMARÃES, 2008).

O teste de Mann-Whitney considera as seguintes hipóteses:

$$\begin{aligned} H_0: m_a &= m_b \\ H_a: m_a &\neq m_b \end{aligned}$$

Onde  $m_a$  e  $m_b$  representam as medianas das populações em teste. A Tabela 31 mostra os valores- $p$  ( $p$ -values) obtidos usando o software *Statistica*.

**Tabela 31: Valor  $p$  encontrado pelo teste de Mann-Whitney.**

| Parâmetro                     | $p$ -value |
|-------------------------------|------------|
| Restrições Positivas Testadas | 0,300      |
| Restrições Negativas Testadas | 0,531      |
| Total de Restrições Testadas  | 0,983      |

Diante dos valores- $p$  obtidos, a um nível de significância de 0,05, não podemos rejeitar a hipótese nula que os grupos provêm da mesma população.

## 4.5 Avaliação da Validade dos Resultados

### 4.5.1 Validade Interna

Os alunos envolvidos no experimento correspondiam a aproximadamente metade dos alunos do último período do curso de Sistemas de Informação da UNIGRANRIO, enquanto que os profissionais que participaram do estudo correspondiam a 1/3 dos profissionais do Centro de Testes do SERPRO no Rio de Janeiro. Tais proporções de participantes mostram-se significativas para esses dois contextos, entretanto, não se pode afirmar que são significativas em contextos mais amplos de alunos e profissionais.

Nenhum dos participantes que responderam ao questionário QCP deixou de participar do experimento até o seu final. Os participantes do experimento possuíam, na sua maioria, conhecimento sobre Web Services RESTful ou conhecimento sobre Teste de Software. Tal diferença de conhecimentos não exerceu influência significativa nos resultados, como mostrado pelos resultados do teste de Mann-Whitney.

#### **4.5.2 Validade Externa**

A amostragem deste experimento foi realizada por conveniência. A amostragem por conveniência usa os resultados que são mais fáceis de coletar. Essa técnica também pode ser escolhida se houver impossibilidade de coletar resultados da população de interesse. No caso deste experimento não foi identificada uma população de “testadores de Web Services RESTful”. O que mais se aproximou dessa população foram os grupos de alunos e de profissionais, tendo em vista que o primeiro grupo tinha conhecimento de programação de Web Services RESTful e o segundo tinha experiência prática com Teste de Software.

Desta forma, como a amostragem foi localizada, não é possível afirmar que os resultados deste experimento se repetiriam com outros grupos de alunos ou profissionais, bem como com Web Services RESTful reais e mais complexos. Este experimento representa um passo inicial a ser dado antes de qualquer aprofundamento em projetos reais.

#### **4.5.3 Validade da Construção**

Um dos principais desafios deste estudo foi coletar dados significativos em um espaço curto de tempo. O modelo PSM representado no exercício EP2 é relativamente simples e foi criado levando-se em consideração os principais tipos de restrição encontrados em serviços CRUD. No entanto, foi necessário realizar o exercício EP2 em duas seções para evitar a fadiga dos participantes e, conseqüentemente, prejudicar os resultados.

Outra questão importante está relacionada ao controle do ambiente. O experimento foi realizado em laboratórios de informática previamente preparados pelo pesquisador. Todos os alunos realizaram o experimento no mesmo local e horário, assim como os profissionais (cada grupo em seu local de atuação). Os participantes foram orientados a não trocar informações e eram constantemente supervisionados pelo pesquisador.

#### **4.5.4 Validade da Conclusão**

A validade das conclusões obtidas neste estudo é corroborada pelas técnicas estatísticas utilizadas, que visam minimizar as ameaças à validade das conclusões. Embora o número de participantes seja relativamente pequeno (35), o estudo produziu uma quantidade de itens de avaliação razoável, totalizando aproximadamente 1.700 (mil e setecentos) casos de teste.

### **4.6 Conclusão**

Este capítulo apresentou detalhes sobre o planejamento e a execução de um estudo experimental realizado sobre uma população composta por alunos de Sistemas de Informação e profissionais de Teste de Software. O experimento consistiu, basicamente, de um treinamento específico e de uma atividade de produção de casos de teste a partir de um modelo PSM para Web Services RESTful.

O estudo mostrou que apenas 2 (5%) participantes alcançaram a meta de cobertura eficaz de restrições e, levando em consideração o erro de amostragem, o valor máximo para esta proporção seria de 20% ou seja 7 participantes que é muito inferior ao valor especificado na nossa hipótese nula (no mínimo 24 testadores). Portanto, dentro dos limites experimentais, podemos afirmar que a utilização do método proposto permitiria uma melhoria significativa na cobertura de testes, fazendo com que 80% dos testadores tenham um desempenho melhor do que usando as técnicas *ad-hoc*.

Finalmente, é possível afirmar que o estudo apresenta indícios de que a abordagem proposta para teste de Web Services RESTful baseada em especificação pode contribuir com a qualidade dos testes nos ambientes experimentais. Entretanto, a repetição deste experimento

é uma condição necessária para que as conclusões do estudo possam ser generalizados para os ambientes de teste encontrados no mundo real.

## **5 CONCLUSÃO**

Este capítulo apresenta a conclusão da dissertação. A seção 5.1 tece as considerações finais do trabalho. A seção 5.2 apresenta as contribuições produzidas com esta dissertação. Por fim, a seção 5.3 descreve as limitações deste trabalho, bem como aponta sugestões de futuros trabalhos.

### **5.1 Considerações Finais**

A tecnologia de Web Services tradicionais (WS-\*) trouxe consigo uma série de características que fomentaram a pesquisa relacionada à qualidade dos serviços. Diversos trabalhos abordaram o assunto propondo a geração de casos de teste a partir dos descritores de serviço ou de especificações semânticas feitas com ontologias.

A tecnologia de Web Services RESTful, no entanto, possui características que não são cobertas pelos trabalhos existentes para teste de Web Services WS-\*. Além disso, há poucos trabalhos que abordam o teste de Web Services RESTful, cada qual tratando de um assunto específico, distantes do problema de especificação semântica dos serviços.

Este trabalho, portanto, procurou contribuir para a qualidade de Web Services RESTful, propondo uma abordagem sistematizada para geração de casos de teste para serviços CRUD. A abordagem é composta por duas técnicas voltadas à especificação e outras duas técnicas voltadas à geração de casos de teste. As técnicas de especificação são empregadas para garantir a precisão dos modelos e, consequentemente, propiciar a criação de casos de teste mais significativos em relação à geração de casos de teste a partir de descritores de serviços.

A técnica para definição de modelos PIM tem como objetivo garantir que os principais tipos de restrição presentes em serviços CRUD sejam representados, fazendo uso do diagrama

de classes ou de restrições OCL. Cabe ressaltar que o PIM é genérico o bastante para que outros modelos PSM sejam criados, possibilitando, assim, a aplicação da abordagem em outras tecnologias de desenvolvimento de software.

A técnica para definição de modelos PSM consiste basicamente na representação de estereótipos e *tagged values* no modelo de classes que indicam determinadas características inerentes à tecnologia REST, tais como o modelo de URI e os tipos MIME consumidos ou produzidos por cada operação do serviço.

A técnica para geração de casos de teste PIT inicialmente identifica as restrições contidas no modelo PIM para, em seguida, gerar os casos de teste aplicando uma combinação das técnicas de particionamento em classes de equivalência, análise de valor-limite e tabela de decisão. Além disso, nessa técnica são definidos os passos que compõem os casos de teste, considerando que o estado do sistema é desconhecido.

Em seguida, é aplicada a técnica para geração de casos de teste PST, na qual os casos de teste PIT são traduzidos para a tecnologia REST, seguindo determinados modelos como base. Tais modelos determinam assertivas que devem ser testadas considerando os cabeçalhos das respostas HTTP.

## 5.2 Contribuições

As principais contribuições deste trabalho dizem respeito à sistematização da produção de casos de teste a partir de restrições semânticas de domínio associadas a restrições tecnológicas, específicas da tecnologia REST. Cada técnica que compõe a abordagem possui uma contribuição específica. Tais contribuições puderam ser observadas no experimento realizado, que mostrou indícios de que a abordagem contribui para a realização de testes no contexto de Web Services RESTful.

A primeira contribuição específica da abordagem são as técnicas para definição de modelos PIM e PSM. Os modelos PIM fazem uso de contratos OCL, criados a partir de regras específicas. Tais regras podem ser utilizadas para especificar qualquer sistema CRUD.

Já a técnica para definição de modelos PSM sugere regras para modelagem para Web Services RESTful. Considerando que Web Services RESTful são cada vez mais comuns na indústria de software torna-se necessário definir técnicas que permitam a representação das restrições tecnológicas dessa categoria de serviços. Sendo assim, a técnica para definição de modelos PSM configura a segunda contribuição específica desta dissertação, tendo em vista que sugere uma forma de representação dessa categoria de serviços.

A terceira e quarta contribuições específicas consistem dos modelos de casos de teste PIT e PST, cujos passos são ordenados seguindo uma lógica de precedência, considerando que o estado do sistema sob teste é vazio. Uma característica comum observada nos casos de teste projetados pelos participantes do experimento foi o aproveitamento de estado de outros casos de teste. Os casos de teste definidos pela abordagem, no entanto, são independentes, garantindo, com isso, que outros casos de teste não serão afetados.

### **5.3 Limitações e Trabalhos Futuros**

O uso da abordagem no exemplo apresentado serviu para mostrar a utilização prática da proposta. Apesar de não ser suficiente para a validação efetiva do trabalho, o estudo experimental mostrou indícios de que a abordagem proposta contribui para a melhoria da qualidade de Web Services RESTful.

Este trabalho, no entanto, possui algumas limitações, colocadas a seguir como possíveis trabalhos futuros. No que se refere à abordagem foram identificadas as seguintes limitações:

- *Hipermídia/HATEOAS* - A abordagem proposta cobre apenas Web Services RESTful do tipo CRUD, alcançando o nível 2 no modelo de maturidade de Richardson (RICHARDSON, 2008). Novos trabalhos podem tratar de serviços hipermídia/HATEOAS que correspondem ao terceiro e último nível do citado modelo.
- *Tipos de restrição* - A abordagem proposta identifica um determinado conjunto de restrições a serem testadas em função do tipo de operação. Outros trabalhos podem sugerir novos tipos de restrição, tanto de negócio quanto tecnológicas, para tornar a abordagem mais abrangente.
- *Automação* - A abordagem é passível de automação. Trabalhos futuros podem automatizar cada técnica que compõe a abordagem tornando sua utilização mais produtiva.

O estudo experimental também possui algumas limitações, indicadas a seguir:

- *Eficiência dos testes* - Uma importante contribuição da abordagem diz respeito à composição dos casos de teste em termos de passos. Uma boa parte dos casos de teste produzidos pelos participantes do experimento atingiu o objetivo proposto, no entanto, utilizou uma quantidade diferente de passos. Sendo assim, é possível analisar os casos de teste em função do esforço de construção, considerando a quantidade de passos ou requisições HTTP utilizados para construí-los.
- *Qualidade dos testes* - Uma importante contribuição da abordagem é o fato dos casos de teste propostos serem livres de determinados *smells* catalogados por Meszaros (2007). Grande parte dos casos de teste produzidos pelos participantes do experimento possuíam problemas de construção, apesar de atingirem o objetivo proposto. Desta forma, os casos de teste dos participantes podem ser analisados a fim de caracterizar os tipos de *smell* mais recorrentes.

- *Comparação entre grupos* - O estudo experimental deu o mesmo tratamento aos dois grupos de participantes. É possível, no entanto, aplicar tratamentos diferenciados, como por exemplo, treinar um grupo na abordagem e comparar os casos de teste produzidos com os de um grupo sem conhecimento da abordagem.
- *Aplicações reais* - O experimento foi baseado em um exercício prático fictício, criado com o objetivo de simular situações reais. Novos trabalhos podem ser realizados sobre serviços REST reais existentes na indústria.
- *Outras populações* - A amostragem do experimento foi realizada por conveniência, tendo em vista que os participantes fazem parte do convívio do pesquisador. Outros trabalhos poderiam envolver populações diferentes e mais representativas.

## REFERÊNCIAS BIBLIOGRÁFICAS

1. AGRESTI, A.; COULL, B. A. Approximate is better than "exact" for interval estimation of binomial proportions. **The American Statistician**, 52, 1998. 119-126.
2. ALOWISHEQ, A.; MILLARD, D. E.; TIROPANIS, T. **EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies**. Proceedings of the 8th International Semantic Web Conference (ISWC '09). Chantilly, VA: Springer-Verlag. 2009. p. 941-948.
3. ASKARUINISA, A.; ABIRAMI, A. M. Test Case Reduction Technique for Semantic Based Web Services. **International Journal on Computer Science and Engineering (IJCSE)**, 02, n. 03, 2010. 566-576.
4. BAI, X. et al. **WSDL-Based Automatic Test Case Generation for Web Services Testing**. Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE '05). Washington, DC: IEEE Computer Society. 2005. p. 215-220.
5. BAI, X. et al. **Ontology-Based Test Modeling and Partition Testing of Web Services**. Proceedings of the 2008 IEEE International Conference on Web Services (ICWS '08). Washington, DC: IEEE Computer Society. 2008. p. 465-472.
6. BARTOLINI, C. et al. **WS-TAXI: A WSDL-based Testing Tool for Web Services**. Proceedings of the 2009 International Conference on Software Testing Verification and Validation (ICST '09). Washington, DC: IEEE Computer Society. 2009. p. 326-335.
7. BATTLE, R.; BENSON, E. Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). **Journal of Web Semantics**, Amsterdam, The Netherlands, 1, n. 6, 2008. 61-69.
8. BERRY, D. M.; KAMSTIES, E. Ambiguity in Requirements Specification. In: LEITE, J. C. S. P.; DOORN, J. H. **Perspectives on Software Requirements**. [S.l.]: Kluwer Academic Publishers, 2004.
9. BINDER, R. V. **Testing object-oriented systems: models, patterns, and tools**. Upper Saddle River: Pearson, 2000.
10. BOZKURT, M.; HARMAN, M.; HASSOUN, Y. **Testing Web Services: A Survey**. King's College London. Londres. 2010. (TR-10-01).
11. CANFORA, G.; PENTA, M. Service-Oriented Architectures Testing: A Survey. **Software Engineering: International Summer Schools, ISSSE 2006-2008**, Berlin, Heidelberg, 2009. 78-105.
12. CHAKRABARTI, S. K.; RODRIQUEZ, R. **Connectedness testing of RESTful web-services**. Proceedings of the 3rd India Software Engineering Conference (ISEC '10). New York, NY: ACM. 2010. p. 143-152.
13. CHEESMAN, J.; DANIELS, J. **UML Components: a simple process for specifying component-based software**. Upper Saddle River: Addison-Wesley, 2001.
14. DAI, G. et al. **Contract-Based Testing for Web Services**. Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC '07). Washington, DC: IEEE Computer Society. 2007. p. 517-526.

15. DAIGNEAU, R. **Service Design Patterns: fundamental design solutions for SOAP/WSDL and RESTful Web Services**. Upper Saddle River: Pearson, 2012.
16. DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. Rio de Janeiro: Elsevier, 2007.
17. ENDO, A. T.; SIMÃO, A. D. S. **Formal Testing Approaches for Service-Oriented Architectures and Web Services: A Systematic Review**. Universidade de São Paulo. São Carlos. 2010. (348).
18. ERL, T. et al. **Web service contract: design e versioning for SOA**. Crawfordsville: Prentice Hall, 2008.
19. EVIWARE. **soapUI**, 2010. Disponível em: <<http://www.soapui.org>>. Acesso em: 18 outubro 2010.
20. FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. University of California. Irvine. 2000.
21. FOWLER, M. **Refactoring: Improving the Design of Existing Code**. Boston: Addison-Wesley, 1999.
22. FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. Porto Alegre: Bookman, 2006.
23. GUIMARÃES, P. R. B. **Métodos quantitativos estatísticos**. Curitiba: IESDE Brasil, 2008.
24. HANNA, S.; MUNRO, M. **An Approach for Specification-based Test Case Generation for Web Services**. Proceedings of ACS/IEEE International Conference on Computer Systems and Applications. Los Alamitos, CA: IEEE Computer Society. 2007. p. 16-23.
25. JOKHIO, M. S. **Goal-based Testing of Semantic Web Services**. 2009 IEEE/ACM International Conference on Automated Software Engineering. [S.l.]: [s.n.]. 2009. p. 707-711.
26. JOKHIO, M. S.; DOBBIE, G.; SUN, J. **Towards Specification Based Testing for Semantic Web Services**. Proceedings of the 2009 Australian Software Engineering Conference (ASWEC '09). Washington, DC: IEEE Computer Society. 2009. p. 54-63.
27. JONES, C. B. **Systematic software development using VDM**. Upper Saddle River: Prentice Hall, 1989.
28. LEE, S.; BAI, X.; CHEN, Y. **Automatic Mutation Testing and Simulation on OWL-S Specified Web Services**. Proceedings of the 41st Annual Simulation Symposium (ANSS-41 '08). Washington, DC: IEEE Computer Society. 2008. p. 149-156.
29. MALESHKOVA, M.; PEDRINACI, C.; DOMINGUE, J. **Semantically Annotating RESTful Services with SWEET**. Proceedings of the 8th International Semantic Web Conference (ISWC 2009). Washington, DC: Springer. 2009.
30. MCGREGOR, J. D.; SYKES, D. A. **A Practical Guide to Testing Object-Oriented Software**. Upper Saddle River: Addison-Wesley, 2001.
31. MESZAROS, G. **xUnit Test Patterns: refactoring test code**. Boston: Pearson, 2007.
32. MEYER, B. Applying "Design by Contract". **Computer**, Los Alamitos, outubro 1992. 40-51.

33. NOIKAJANA, S.; SUWANNASART, T. **An Improved Test Case Generation Method for Web Service Testing from WSDL-S and OCL with Pair-Wise Testing Technique**. Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09). Washington, DC: IEEE Computer Society. 2009. p. 115-123.
34. OMG, OBJECT MANAGEMENT GROUP. **MDA guide version 1.0.1**. [S.l.]. 2003.
35. OMG, OBJECT MANAGEMENT GROUP. **MOF QVT Final Adopted Specification**. Object Management Group. [S.l.]. 2005.
36. OMG, OBJECT MANAGEMENT GROUP. Object Constraint Language, fevereiro 2010. Disponível em: <<http://www.omg.org/spec/OCL/2.2/PDF>>. Acesso em: 18 outubro 2010.
37. PARADKAR, A. M. et al. **Automated Functional Conformance Test Generation for Semantic Web Services**. Proceedings of the 2007 IEEE International Conference on Web Services (ICWS 2007). [S.l.]: [s.n.]. 2007. p. 110-117.
38. PEZZÈ, M.; YOUNG, M. **Teste e Análise de Software**. Porto Alegre: Bookman, 2008.
39. POL, M.; TEUNISSEN, R.; VEENENDAAL, E. V. **Software testing: a guide to the TMap approach**. Harlow: Pearson, 2002.
40. RAMOLLARI, E. et al. **Leveraging Semantic Web Service Descriptions for Validation by Automated Functional Testing**. ESWC 2009 Heraklion: Proceedings of the 6th European Semantic Web Conference on The Semantic Web. Berlin, Heidelberg: Springer-Verlag. 2009. p. 593-607.
41. REZA, H.; VAN GILST, D. **A Framework for Testing RESTful Web Services**. Proceedings of the Seventh International Conference on Information Technology. [S.l.]: IEEE Computer Society. 2010. p. 216-221.
42. RICHARDSON, L. The Maturity Heuristic, 2008. Disponível em: <<http://www.crummy.com/writing/speaking/2008-QCon/act3.html>>. Acesso em: 11 novembro 2011.
43. RICHARDSON, L.; RUBY, S. **Restful web services**. Sebastopol: O'Reilly, 2007.
44. SHETH, A. P.; GOMADAM, K.; LATHEM, J. Semantically Interoperable and Easier-to-Use Services and Mashups. **IEEE Internet Computing**, Piscataway, NJ, USA, n. 11, 2007. 91-94.
45. SILVA-DE-SOUZA, T.; PIRES, P. F. Tecnologia de Serviços Web: Conceitos e Aplicações. In: \_\_\_\_\_ **Minicursos do 20º SBBD e 19º SBES**. Uberlândia: Sociedade Brasileira de Computação, 2005. Cap. 11, p. 303-335.
46. SINHA, A.; PARADKAR, A. **Model-based functional conformance testing of web services operating on persistent data**. Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications (TAV-WEB '06). New York, NY: ACM. 2006. p. 17-22.
47. SOMMERVILLE, I. **Engenharia de software**. 9. ed. ed. São Paulo: Pearson, 2011.
48. STATSOFT. **StatSoft South America**, 2011. Disponível em: <<http://www.statsoft.com.br>>. Acesso em: 15 janeiro 2012.
49. TRIOLA, M. F. **Introdução à estatística**. Rio de Janeiro: LTC, 2008.

50. TSAI, W.-T.; CHEN, Y.; PAUL, R. **Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems**. Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '05). Washington, DC: IEEE Computer Society. 2005. p. 139-147.
51. UTTING, M.; LEGEARD, B. **Practical Model-Based Testing: A Tools Approach**. San Francisco: Morgan Kaufmann Publishers Inc., 2007.
52. W3C, WORLD WIDE WEB CONSORTIUM. Web Services Description Language (WSDL) 1.1, 2001. Disponivel em: <<http://www.w3.org/TR/wsdl>>. Acesso em: 15 novembro 2010.
53. W3C, WORLD WIDE WEB CONSORTIUM. OWL Web Ontology Language, 2004. Disponivel em: <<http://www.w3.org/TR/owl-features/>>. Acesso em: 15 novembro 2010.
54. W3C, WORLD WIDE WEB CONSORTIUM. OWL-S: Semantic Markup for Web Services, 2004. Disponivel em: <<http://www.w3.org/Submission/OWL-S>>. Acesso em: 18 outubro 2010.
55. W3C, WORLD WIDE WEB CONSORTIUM. Web Service Modeling Ontology (WSMO), 2005. Disponivel em: <<http://www.w3.org/Submission/WSMO/>>. Acesso em: 15 novembro 2010.
56. W3C, WORLD WIDE WEB CONSORTIUM. Web Service Semantics - WSDL-S, 2005. Disponivel em: <<http://www.w3.org/Submission/WSDL-S/>>. Acesso em: 17 outubro 2010.
57. W3C, WORLD WIDE WEB CONSORTIUM. Semantic Annotations for WSDL and XML Schema, 2007. Disponivel em: <<http://www.w3.org/TR/sawSDL/>>. Acesso em: 17 outubro 2010.
58. W3C, WORLD WIDE WEB CONSORTIUM. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2007. Disponivel em: <<http://www.w3.org/TR/wsdl20/>>. Acesso em: 15 novembro 2010.
59. W3C, WORLD WIDE WEB CONSORTIUM. Web Application Description Language, 2009. Disponivel em: <<http://www.w3.org/Submission/wadl/>>. Acesso em: 15 novembro 2010.
60. WANG, Y. et al. **Ontology-Based Test Case Generation for Testing Web Services**. Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems (ISADS '07). Washington, DC: IEEE Computer Society. 2007. p. 43-50.
61. WARMER, J.; KLEPPE, A. **The Object Constraint Language: getting your models ready for MDA**. 2. ed. Boston: Addison-Wesley, 2003.
62. WEBBER, J.; PARASTATIDIS, S.; ROBINSON, I. **REST in Practice**. Sebastopol: O'Reilly, 2010.
63. WOHLIN, C. et al. **Experimentation in software engineering: an introduction**. Norwell: Kluwer Academic Publishers, 2000.
64. WOODCOCK, J.; DAVIS, J. **Using Z: specification, refinement and proof**. Upper Saddle River: Prentice Hall, 1996.

## APÊNDICE A – Termo de Consentimento Livre e Esclarecido para Alunos

|      |                                                           |
|------|-----------------------------------------------------------|
| TCLE | <b>Termo de Consentimento Livre e Esclarecido (Aluno)</b> |
|------|-----------------------------------------------------------|

Eu declaro ter mais de 18 anos de idade e concordo em participar de um estudo conduzido por Thiago Silva de Souza como parte das atividades de Mestrado do Programa de Pós-Graduação em Informática (PPGI) do Departamento de Ciência da Computação (DCC/IM) e do Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (NCE) da UFRJ. Este estudo visa analisar a viabilidade de uma abordagem para geração de casos de teste para Web Services RESTful.

### Procedimento

Este estudo acontecerá em duas etapas: uma de treinamento dos participantes e outra de realização de testes sobre Web Services RESTful. Na primeira etapa, o participante receberá um treinamento teórico sobre Testes de Software e um treinamento prático sobre a ferramenta soapUI, usada para testes de Web Services. Na segunda etapa o participante irá projetar e executar testes sobre Web Services RESTful utilizando a ferramenta soapUI. Os testes produzidos por cada participante serão analisados pelo pesquisador, em comparação com testes produzidos pela abordagem proposta.

### Benefícios e Liberdade de Desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado adquirido com o treinamento ministrado e com o material distribuído, independentemente de minha participação ou não no estudo. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas para a Engenharia de Software.

### Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será identificado nas publicações decorrentes deste estudo. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

### Equipe

Pesquisador Responsável: Thiago Silva de Souza

Professores Responsáveis: Eber Assis Schmitz, Ph.D. e Alexandre Luis Correa, D.Sc.

### Participante da Pesquisa

Nome (em letra de forma): \_\_\_\_\_  
 \_\_\_\_\_, \_\_\_\_\_ de \_\_\_\_\_ de 201\_\_\_\_.

\_\_\_\_\_  
 Assinatura

## APÊNDICE B – Termo de Consentimento Livre e Esclarecido para Profissionais

|      |                                                                  |
|------|------------------------------------------------------------------|
| TCLE | <b>Termo de Consentimento Livre e Esclarecido (Profissional)</b> |
|------|------------------------------------------------------------------|

Eu declaro ter mais de 18 anos de idade e concordo em participar de um estudo conduzido por Thiago Silva de Souza como parte das atividades de Mestrado do Programa de Pós-Graduação em Informática (PPGI) do Departamento de Ciência da Computação (DCC/IM) e do Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais (NCE) da UFRJ. Este estudo visa analisar a viabilidade de uma abordagem para geração de casos de teste para Web Services RESTful.

### Procedimento

Este estudo acontecerá em duas etapas: uma de treinamento dos participantes e outra de realização de testes sobre Web Services RESTful. Na primeira etapa, o participante receberá um treinamento teórico sobre Web Services RESTful e um treinamento prático sobre a ferramenta soapUI, usada para testes de Web Services. Na segunda etapa o participante irá projetar e executar testes sobre Web Services RESTful utilizando a ferramenta soapUI. Os testes produzidos por cada participante serão analisados pelo pesquisador, em comparação com testes produzidos pela abordagem proposta.

### Benefícios e Liberdade de Desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado adquirido com o treinamento ministrado e com o material distribuído, independentemente de minha participação ou não no estudo. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas para a Engenharia de Software.

### Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será identificado nas publicações decorrentes deste estudo. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

### Equipe

Pesquisador Responsável: Thiago Silva de Souza

Professores Responsáveis: Eber Assis Schmitz, Ph.D. e Alexandre Luis Correa, D.Sc.

### Participante da Pesquisa

Nome (em letra de forma): \_\_\_\_\_  
 \_\_\_\_\_, \_\_\_\_\_ de \_\_\_\_\_ de 201 \_\_\_\_.

\_\_\_\_\_  
 Assinatura

## APÊNDICE C – Questionário de Caracterização do Participante

|            |                                                       |
|------------|-------------------------------------------------------|
| <b>QCP</b> | <b>Questionário de Caracterização do Participante</b> |
|------------|-------------------------------------------------------|

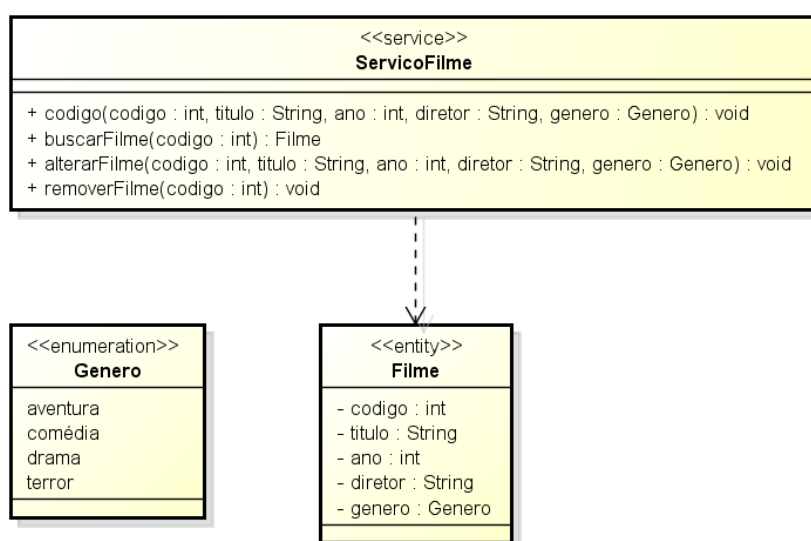
  

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|---------------------------------|-------------------|--------------------------------|-------------------|--------------------------------------------------------|-------------------|-------------------------------------------|-------------------|------------------------------------|-------------------|--------------------------------------------|-------------------|--------------------------------------------------------|-------------------|---------------------------------------------------------|-------------------|
| <b>Nome Completo</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <b>Dados Pessoais</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| Idade: <input type="checkbox"/> Até 18 anos<br><input type="checkbox"/> De 19 a 24 anos<br><input type="checkbox"/> De 25 a 29 anos<br><input type="checkbox"/> De 30 a 34 anos<br><input type="checkbox"/> A partir de 35 anos                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Sexo: <input type="checkbox"/> Masculino<br><input type="checkbox"/> Feminino |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <b>Origem</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <input type="checkbox"/> Acadêmico da Unigranrio                      CR (se acadêmico) _____<br><input type="checkbox"/> Profissional do Serpro                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <b>Formação Universitária</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| Nível: <input type="checkbox"/> Doutor ou Mestre<br><input type="checkbox"/> Especialista<br><input type="checkbox"/> Graduado<br><input type="checkbox"/> Graduando                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <b>Experiência Profissional em Desenvolvimento de Software</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| Tempo de Experiência: <input type="checkbox"/> Nenhuma experiência<br><input type="checkbox"/> Menos de 1 ano<br><input type="checkbox"/> Entre 1 e 3 anos<br><input type="checkbox"/> Entre 3 e 5 anos<br><input type="checkbox"/> Entre 5 e 10 anos<br><input type="checkbox"/> Acima de 10 anos                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                               |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| <p>Por favor, indique sua experiência nas questões a seguir, assinalando uma das alternativas da escala de 1 a 5, de acordo com a seguinte orientação:</p> <p>1 – nenhuma<br/>         2 – estudei em cursos ou livros<br/>         3 – pratiquei em projetos de curso<br/>         4 – utilizei em alguns (1 a 3) projetos na indústria<br/>         5 – utilizei em mais de 3 projetos na indústria</p> <p>No contexto de qualquer tecnologia exceto a de Web Services RESTful:</p> <table style="width: 100%;"> <tr> <td>a) Modelagem de classes com UML</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>b) Desenvolvimento de sistemas</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>c) Projeto de casos de teste a partir de especificação</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>d) Execução de casos de teste caixa-preta</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>e) Utilização da ferramenta soapUI</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> </table> <p>No contexto de Web Services RESTful:</p> <table style="width: 100%;"> <tr> <td>f) Desenvolvimento de Web Services RESTful</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>g) Projeto de casos de teste para Web Services RESTful</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> <tr> <td>h) Execução de casos de teste para Web Services RESTful</td> <td style="text-align: right;">1   2   3   4   5</td> </tr> </table> |                                                                               | a) Modelagem de classes com UML | 1   2   3   4   5 | b) Desenvolvimento de sistemas | 1   2   3   4   5 | c) Projeto de casos de teste a partir de especificação | 1   2   3   4   5 | d) Execução de casos de teste caixa-preta | 1   2   3   4   5 | e) Utilização da ferramenta soapUI | 1   2   3   4   5 | f) Desenvolvimento de Web Services RESTful | 1   2   3   4   5 | g) Projeto de casos de teste para Web Services RESTful | 1   2   3   4   5 | h) Execução de casos de teste para Web Services RESTful | 1   2   3   4   5 |
| a) Modelagem de classes com UML                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| b) Desenvolvimento de sistemas                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| c) Projeto de casos de teste a partir de especificação                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| d) Execução de casos de teste caixa-preta                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| e) Utilização da ferramenta soapUI                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| f) Desenvolvimento de Web Services RESTful                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| g) Projeto de casos de teste para Web Services RESTful                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |
| h) Execução de casos de teste para Web Services RESTful                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | 1   2   3   4   5                                                             |                                 |                   |                                |                   |                                                        |                   |                                           |                   |                                    |                   |                                            |                   |                                                        |                   |                                                         |                   |

## APÊNDICE D – Exercício Prático 1

|     |                     |
|-----|---------------------|
| EP1 | Exercício Prático 1 |
|-----|---------------------|

Considere o modelo de classes a seguir. Ele contém uma classe de serviço cujos métodos manipulam objetos de uma classe de domínio. Observe que determinadas regras de negócio estão explícitas no modelo.



powered by astah

A seguir estão relacionadas regras de negócio que não estão explícitas no modelo de classes, mas que também devem ser consideradas no projeto de testes:

- O atributo `código` identifica univocamente cada instância.
- O atributo `código` deve ser  $\geq 1$  e  $\leq 999$ .
- O atributo `título` deve possuir pelo menos um caractere e no máximo 40 caracteres.
- O atributo `ano` deve estar entre 1900 e 2020.
- O atributo `diretor` deve possuir pelo menos um caractere e no máximo 30 caracteres

Além das regras de negócio há restrições tecnológicas em função do uso da tecnologia REST. Sendo assim, o quadro a seguir representa as restrições presentes no exemplo apresentado.

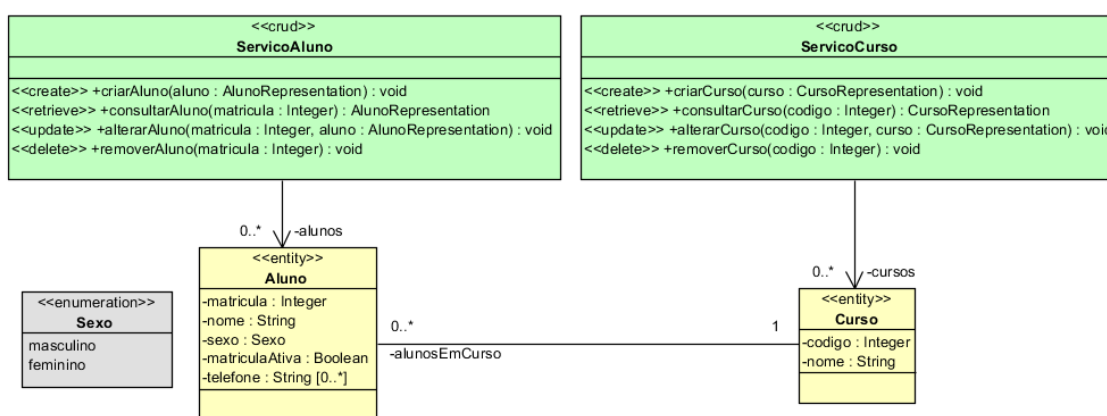
| Recurso         | GET | POST | PUT | DELETE | Conteúdo                                          |
|-----------------|-----|------|-----|--------|---------------------------------------------------|
| /filme          |     | X    | X   |        | POST e PUT consomem "application/xml"             |
| /filme/{codigo} | X   |      |     | X      | GET produz "application/xml" e "application/json" |

Com base em todas as restrições apresentadas crie e execute casos de teste no soapUI para verificar se cada restrição foi implementada corretamente na aplicação sob teste.

## APÊNDICE E – Exercício Prático 2

|     |                     |
|-----|---------------------|
| EP2 | Exercício Prático 2 |
|-----|---------------------|

Considere o modelo de classes a seguir. Ele contém classes de serviço cujos métodos manipulam objetos das respectivas classes de domínio. Observe que determinadas regras de negócio estão explícitas no modelo. Além das regras de negócio há restrições tecnológicas em função do uso da tecnologia REST na forma de anotações.



A seguir estão relacionadas regras de negócio que não estão explícitas no modelo de classes, mas que também devem ser consideradas no projeto de testes:

### 1 Curso

O atributo **código** identifica univocamente cada instância.

O valor do atributo **código** deve ser  $\geq 1$  e  $\leq 99$ .

O valor do atributo **nome** deve possuir pelo menos um caractere e no máximo 20 caracteres.

### 2 Aluno

O atributo **matricula** identifica univocamente cada instância.

O valor do atributo **matricula** deve ser  $\geq 1$  e  $\leq 999$ .

O valor do atributo **nome** deve possuir pelo menos um caractere e no máximo 60 caracteres.

O valor de cada instância de **telefone** deve possuir exatamente 10 caracteres.

Além das regras de negócio há restrições tecnológicas em função do uso da tecnologia REST. Sendo assim, o quadro a seguir representa as restrições presentes no exemplo apresentado.

| Recurso            | GET | POST | PUT | DELETE | Conteúdo                                                                           |
|--------------------|-----|------|-----|--------|------------------------------------------------------------------------------------|
| /curso             |     | X    |     |        | POST consome "application/xml"                                                     |
| /curso/{codigo}    | X   |      | X   | X      | PUT consome "application/xml"<br>GET produz "application/xml" e "application/json" |
| /aluno             |     | X    |     |        | POST consome "application/xml"                                                     |
| /aluno/{matricula} | X   |      | X   | X      | PUT consome "application/xml"<br>GET produz "application/xml"                      |

Para as operações de entrada de dados devem-se utilizar as seguintes estruturas de documentos XML:

### 3 Curso

```
<curso>
  <codigo></codigo>
  <nome></nome>
</curso>
```

### 4 Aluno

```
<aluno>
  <matricula></matricula>
  <nome></nome>
  <sexo></sexo>
  <matriculaAtiva></matriculaAtiva>
  <telefone></telefone>
  ...
  <telefone></telefone> (um elemento para cada telefone)
  <curso></curso>
</aluno>
```

Com base em todas as restrições apresentadas projete os casos de teste no soapUI.

## APÊNDICE F – Perfil do Participante do Experimento

Com base nas respostas dos participantes ao questionário QCP foram criadas as tabelas apresentadas nesta seção. A Tabela 32 apresenta a distribuição de faixa etária dos participantes, onde é possível verificar que a maior parte dos participantes possui entre 19 e 24 anos. Esse número é explicado pela maior quantidade de participantes do grupo de alunos no experimento.

**Tabela 32: Faixa etária dos participantes.**

| Faixa etária        | Frequência |
|---------------------|------------|
| Até 18 anos         | 0          |
| De 19 a 24 anos     | 19         |
| De 25 a 29 anos     | 6          |
| De 30 a 34 anos     | 5          |
| A partir de 35 anos | 5          |

A Tabela 33 apresenta distribuição de gênero dos participantes, na qual se observa que a grande maioria de participantes é do sexo masculino.

**Tabela 33: Sexo dos participantes.**

| Sexo      | Frequência |
|-----------|------------|
| Masculino | 29         |
| Feminino  | 6          |

Quanto à origem os participantes foram divididos em alunos e profissionais. A Tabela 34 mostra a distribuição de participantes de cada grupo.

**Tabela 34: Origem dos participantes.**

| Origem       | Frequência |
|--------------|------------|
| Aluno        | 28         |
| Profissional | 7          |

Os participantes do grupo de alunos tiveram seu coeficiente de rendimento (CR) registrado, com o intuito de avaliar se o CR do aluno exerce influência sobre os testes que projeta no experimento. A Tabela 35 apresenta a distribuição entre faixas de CR dos alunos envolvidos no experimento.

**Tabela 35: Coeficiente de rendimento dos participantes alunos.**

| Faixa de CR        | Frequência |
|--------------------|------------|
| $6,5 < x \leq 7,0$ | 3          |
| $7,0 < x \leq 7,5$ | 5          |
| $7,5 < x \leq 8,0$ | 11         |
| $8,0 < x \leq 8,5$ | 8          |
| $8,5 < x \leq 9,0$ | 1          |

A distribuição de frequência da formação acadêmica dos participantes ficou concentrada na categoria “Graduando”, caracterizada por todos os alunos envolvidos no estudo. As demais categorias foram representadas pelos participantes profissionais. A Tabela 36 apresenta essa distribuição de frequência.

**Tabela 36: Formação acadêmica dos participantes.**

| Formação         | Frequência |
|------------------|------------|
| Doutor ou Mestre | 0          |
| Especialista     | 4          |
| Graduado         | 2          |
| Graduando        | 28         |
| Técnico          | 1          |

No que diz respeito à experiência profissional dos participantes na área de desenvolvimento de software, as categorias “Nenhuma experiência” e “Entre 1 e 3 anos” foram as mais frequentemente informadas, como mostra a Tabela 37. Mais uma vez, esses números são explicados pela maior quantidade de alunos do que de profissionais.

**Tabela 37: Experiência profissional dos participantes em desenvolvimento de software.**

| Experiência Profissional em Desenvolvimento de Software | Frequência |
|---------------------------------------------------------|------------|
| Nenhuma experiência                                     | 14         |
| Menos de 1 ano                                          | 5          |
| Entre 1 e 3 anos                                        | 10         |
| Entre 3 e 5 anos                                        | 1          |
| Entre 5 e 10 anos                                       | 2          |
| Acima de 10 anos                                        | 3          |

Quanto à experiência dos participantes em modelagem de classes com UML a maioria informou que apenas praticou em projetos de curso, como mostra a Tabela 38. Esse número é

influenciado pela maior quantidade de alunos que, na sua maioria, aprenderam e praticaram modelagem com UML nas disciplinas do curso de Bacharelado em Sistemas de Informação.

**Tabela 38: Experiência dos participantes em modelagem de classes com UML.**

| Modelagem de classes com UML                     | Frequência |
|--------------------------------------------------|------------|
| Nenhuma                                          | 1          |
| Estudou em cursos ou livros                      | 3          |
| Praticou em projetos de curso                    | 23         |
| Utilizou em alguns (1 a 3) projetos na indústria | 7          |
| Utilizou em mais de 3 projetos na indústria      | 1          |

Em relação à experiência dos participantes em desenvolvimento de sistemas, a Tabela 39 mostra que a maioria apenas praticou em projetos de curso, fato este que tem como explicação o fato dos alunos aprenderem e praticaram este assunto no âmbito do próprio curso.

**Tabela 39: Experiência dos participantes em desenvolvimento de sistemas.**

| Desenvolvimento de sistemas                      | Frequência |
|--------------------------------------------------|------------|
| Nenhuma                                          | 1          |
| Estudou em cursos ou livros                      | 1          |
| Praticou em projetos de curso                    | 18         |
| Utilizou em alguns (1 a 3) projetos na indústria | 7          |
| Utilizou em mais de 3 projetos na indústria      | 8          |

A maioria dos participantes não possui experiência em projeto de casos de teste a partir de especificação, como mostra a Tabela 40. Isso se deve, provavelmente, ao fato do curso que realizam não possuir disciplinas relacionadas a Teste de Software. Cabe ressaltar que dos sete participantes profissionais apenas um declara que não utilizou em mais de três projetos na indústria, o que mostra que a maioria dos profissionais envolvidos possui experiência prática com o assunto.

**Tabela 40: Experiência dos participantes em projeto de casos de teste a partir de especificação.**

| Projeto de casos de teste a partir de especificação | Frequência |
|-----------------------------------------------------|------------|
| Nenhuma                                             | 24         |
| Estudou em cursos ou livros                         | 2          |
| Praticou em projetos de curso                       | 3          |
| Utilizou em alguns (1 a 3) projetos na indústria    | 0          |
| Utilizou em mais de 3 projetos na indústria         | 6          |

A Tabela 41 representa a experiência dos participantes em execução de casos de teste caixa-preta. Ela mostra uma distribuição parecida com a da Tabela 40, provavelmente, pelos mesmos motivos.

**Tabela 41: Experiência dos participantes em execução de casos de teste caixa-preta.**

| Execução de casos de teste caixa-preta           | Frequência |
|--------------------------------------------------|------------|
| Nenhuma                                          | 20         |
| Estudou em cursos ou livros                      | 3          |
| Praticou em projetos de curso                    | 4          |
| Utilizou em alguns (1 a 3) projetos na indústria | 2          |
| Utilizou em mais de 3 projetos na indústria      | 6          |

Considerando que os participantes iriam projetar seus casos de teste na ferramenta soapUI, foi registrado o nível de conhecimento prévio da ferramenta que cada participante possuía, não necessariamente relacionado a Web Services RESTful. A Tabela 42 mostra que a grande maioria não possuía qualquer conhecimento sobre a utilização da ferramenta. Apenas dois participantes declararam já ter praticado em projetos de curso. Tais participantes realizavam Trabalho de Conclusão de Curso sobre a ferramenta.

**Tabela 42: Experiência dos participantes na utilização da ferramenta soapUI.**

| Utilização da ferramenta soapUI                  | Frequência |
|--------------------------------------------------|------------|
| Nenhuma                                          | 33         |
| Estudou em cursos ou livros                      | 0          |
| Praticou em projetos de curso                    | 2          |
| Utilizou em alguns (1 a 3) projetos na indústria | 0          |
| Utilizou em mais de 3 projetos na indústria      | 0          |

A Tabela 43 mostra a distribuição em relação à experiência dos participantes em desenvolvimento de Web Services RESTful. O resultado já era esperado: a maioria dos alunos

praticou no próprio curso, enquanto que a maioria dos profissionais informou ter nenhuma experiência.

**Tabela 43: Experiência dos participantes em desenvolvimento de Web Services RESTful.**

| Desenvolvimento de Web Services RESTful          | Frequência |
|--------------------------------------------------|------------|
| Nenhuma                                          | 6          |
| Estudou em cursos ou livros                      | 4          |
| Praticou em projetos de curso                    | 25         |
| Utilizou em alguns (1 a 3) projetos na indústria | 0          |
| Utilizou em mais de 3 projetos na indústria      | 0          |

Apesar da maioria dos alunos informar ter conhecimento em desenvolvimento de Web Services RESTful, todos participantes, inclusive os profissionais, relataram ter nenhuma experiência em projeto de casos de teste para Web Services RESTful, como mostra a Tabela 44.

**Tabela 44: Experiência dos participantes em projeto de casos de teste para Web Services RESTful.**

| Projeto de casos de teste para Web Services RESTful | Frequência |
|-----------------------------------------------------|------------|
| Nenhuma                                             | 35         |
| Estudou em cursos ou livros                         | 0          |
| Praticou em projetos de curso                       | 0          |
| Utilizou em alguns (1 a 3) projetos na indústria    | 0          |
| Utilizou em mais de 3 projetos na indústria         | 0          |

Apesar de este trabalho ser relacionado ao projeto de casos de teste, foi medida a experiência dos participantes em execução de casos de teste para Web Services RESTful, tendo em vista que uma experiência prévia neste aspecto poderia influenciar os resultados de um participante. Entretanto, novamente, todos os participantes relataram ter nenhuma experiência, como mostra a Tabela 45.

**Tabela 45: Experiência dos participantes em execução de casos de teste para Web Services RESTful.**

| Execução de casos de teste para Web Services RESTful | Frequência |
|------------------------------------------------------|------------|
| Nenhuma                                              | 35         |
| Estudou em cursos ou livros                          | 0          |
| Praticou em projetos de curso                        | 0          |
| Utilizou em alguns (1 a 3) projetos na indústria     | 0          |
| Utilizou em mais de 3 projetos na indústria          | 0          |