Uma Abordagem Baseada em Especificação para Testes de Web Services RESTful

Thiago Silva-de-Souza[1,2], Alexandre Luis Correa[3], Antonio Juarez Alencar[1], Eber Assis Schmitz[1]

[1]Programa de Pós-Graduação em Informática (PPGI), Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro-RJ, Brasil

[2]Escola de Ciência e Tecnologia, Universidade do Grande Rio (UNIGRANRIO)

[3]Departamento de Informática Aplicada
Universidade Federal do Estado do Rio de Janeiro (UNIRIO) – Rio de Janeiro-RJ

thiagoein@gmail.com, alexandre.correa@uniriotec.br, juarezalencar@dcc.ufrj.br, eber@nce.ufrj.br

Abstract. *This paper 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.*

Resumo. *Este trabalho 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.*

1. Introdução

Recentemente uma nova categoria de serviços web, denominados serviços web *RESTful*, vem ganhando popularidade tanto na indústria quanto na academia. Serviços web *RESTful* são serviços que seguem o estilo arquitetural REST (*REpresentational State Transfer*) e que usam intensivamente os recursos disponíveis no protocolo HTTP (*Hypertext Transfer Protocol*) [Fielding 2000]. Serviços web *RESTful* diferenciam-se dos serviços web tradicionais, conhecidos como WS-*[1], principalmente por

[1] O termo WS-* faz referência às diversas especificações para Web Services tradicionais cujos nomes iniciam com o prefixo WS, tais como WSDL.

prescindirem de tantas especificações em formato XML (*Extensible Markup Language*), tais como os padrões *Simple Object Access Protocol* (SOAP) e *Web Services Description Language* (WSDL).

Serviços web *RESTful* manipulam recursos. Um recurso é qualquer item de informação acessível através de um *Universal Resource Identifier* (URI). Todo recurso possui uma ou mais representações formadas pelos dados e metadados que o descrevem. O formato de cada representação deve seguir o padrão MIME (*Multipurpose Internet Mail Extensions*) [Fielding 2000]. Os recursos são manipulados por meio de transferências de representações entre clientes e servidores utilizando a interface uniforme do protocolo HTTP. Tal interface é 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; *DELETE* exclui um recurso [Webber, Parastatidis e Robinson 2010]. Desta forma, serviços web *RESTful* vinculam um método HTTP específico a cada operação de manipulação de um recurso.

Em ambientes onde as soluções de software seguem uma arquitetura orientada a serviços, é importante que cada serviço desempenhe corretamente as suas funções. Uma das técnicas de controle da qualidade de serviços web, como de qualquer software, é a realização de testes. O teste de software tem como objetivo encontrar falhas 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]. Testes funcionais são definidos a partir da especificação do elemento a ser testado e, portanto, sua qualidade está diretamente relacionada com a precisão e riqueza semântica presente nessa especificação.

Testes de serviços web *RESTful* devem levar em consideração três aspectos particularmente importantes 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 execução dos serviços [Canfora e Penta 2009].

Em geral, a descrição de serviços web *RESTful* é realizada na forma de um contrato de serviço no formato *Web Application Description Language* (WADL) [W3C 2009]. Um contrato é um compromisso entre o produtor e os consumidores de um serviço [Erl *et al*. 2008]. Um descritor WADL é, portanto, um contrato de serviço, representado em XML, que descreve o conjunto das operações permitidas sobre recursos, os padrões de URI e os formatos possíveis para representação dos recursos [Webber, Parastatidis e Robinson 2010]. Um documento WADL, porém, não especifica todas as restrições de negócio que devem ser respeitadas pelas operações do serviço, restringindo-se a especificar aspectos tecnológicos. No caso de serviços de dados aderentes ao padrão *Create, Retrieve, Update, Delete* (CRUD), por exemplo, um documento WADL não representaria as invariantes do domínio, tampouco as pré e pós-condições de cada operação. Portanto, é necessário prover mais semântica às especificações dos serviços, de modo a permitir a definição de um conjunto adequado de casos de teste.

Os recursos manipulados por serviços web *RESTful* podem ser representados em diversos formatos. Desta forma, testes de serviços web *RESTful* devem levar em

consideração não apenas as informações presentes no recurso, como também a forma de representação dessas informações na invocação de cada operação. Uma operação de recuperação de um recurso *Empresa*, por exemplo, pode retornar as informações em diversos formatos possíveis, e.g., *JavaScript Object Notation* (JSON), *Extensible Hypertext Markup Language* (XHTML). Portanto, os formatos de entrada e saída dos dados manipulados por uma operação são variáveis adicionais que devem ser consideradas na definição dos casos de teste de operações de serviços web *RESTful*.

O terceiro aspecto diz respeito à necessidade de utilizar requisições HTTP na invocação das operações do serviço. Considerando, por exemplo, o contexto no qual não temos acesso à estrutura interna de um serviço de dados do tipo CRUD implementado a partir dos 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.g. para recuperar um recurso, é necessário que ele tenha sido inserido em um momento anterior), e não se sabe de antemão o estado do sistema sob teste.

Este trabalho propõe uma abordagem de teste baseada em especificação para serviços web *RESTful* do tipo CRUD, considerando que o serviço seja especificado utilizando os padrões UML e OCL. A OCL [OMG 2010] é uma linguagem formal para especificar restrições em modelos UML. Tais restrições podem representar regras de negócio, descrições contratuais da semântica de operações do modelo e expressões associadas a atributos derivados [Warmer e Kleppe 2003].

O trabalho está estruturado em mais quatro seções. A seção 2 discute os trabalhos relacionados. A seção 3 apresenta uma visão geral da abordagem proposta. A seção 4 descreve a abordagem proposta através de um exemplo. Por fim, a seção 5 apresenta as conclusões do trabalho.

2. Trabalhos Relacionados

Diversas abordagens para teste de serviços web já foram propostas, sendo, inclusive, descritas e comparadas em alguns trabalhos [Canfora e Penta 2009, Bozkurt, Harman e Hassoun 2010, Endo e Simão 2010]. Grande parte das abordagens existentes para teste de serviços web compartilha o objetivo de derivar os casos de teste a partir dos contratos dos serviços, podendo ser dividida em duas categorias: a) testes baseados na especificação padrão de serviços web e; b) testes baseados 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. 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) [W3C 2004].

Mais recentemente, alguns trabalhos propuseram abordagens para a geração de casos de teste para serviços web baseadas 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 [Noikajana e

Suwannasart 2009] utiliza o método denominado *Pair-Wise Testing* (PWT). Já a abordagem de Askaruinisa e Abirami [Askaruinisa e Abirami 2010] 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 serviços web *RESTful* do tipo CRUD, tendo em vista que cobrem apenas o problema de geração de dados de teste em serviços web 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 web *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 uma notação formal baseada no formato WADL e um algoritmo para testar automaticamente a conectividade (*connectedness*) de serviços web *RESTful* através de seus endereços URI. O segundo trabalho apresenta uma proposta de *framework* para teste de serviços web *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.

3. Abordagem Proposta

A abordagem proposta neste trabalho é baseada na especificação *Model-Driven Architecture* (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 serviços web *RESTful* do tipo CRUD, que manipulam recursos baseados em tipos de dados complexos.

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 1 representa o fluxo de execução das técnicas da abordagem proposta.

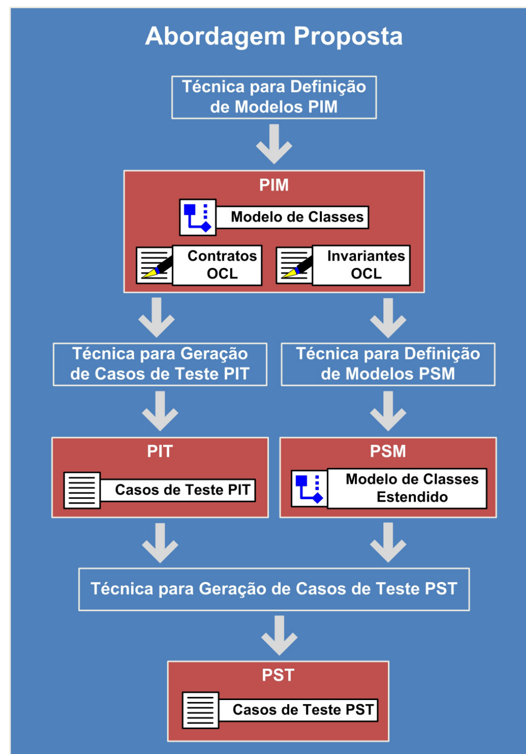

Figura 1: Fluxo de atividades da abordagem proposta.

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) contratos OCL, indicando as pré e pós-condições que devem ser satisfeitas para execução de cada operação do serviço.

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 serviços web *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.

Paralelamente à definição do PSM pode ser realizada a terceira técnica, cujo propósito é 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.

4. Exemplo de Aplicação

Para exemplificar a aplicação da abordagem será utilizado um simples domínio de escola, incluindo serviços que manipulem instâncias de *Curso* e *Aluno*. As regras de negócio do domínio são típicas de sistemas CRUD e estão representadas no PIM descrito na subseção 4.1.

4.1. PIM

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, 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 estereotipas com `<<representation>>`. Essa técnica de modelagem permite a criação de outros PSMs para sistemas CRUD aproveitando o mesmo PIM. A Figura 2 representa um exemplo de modelo de classes produzido através da técnica descrita para o domínio Escola.

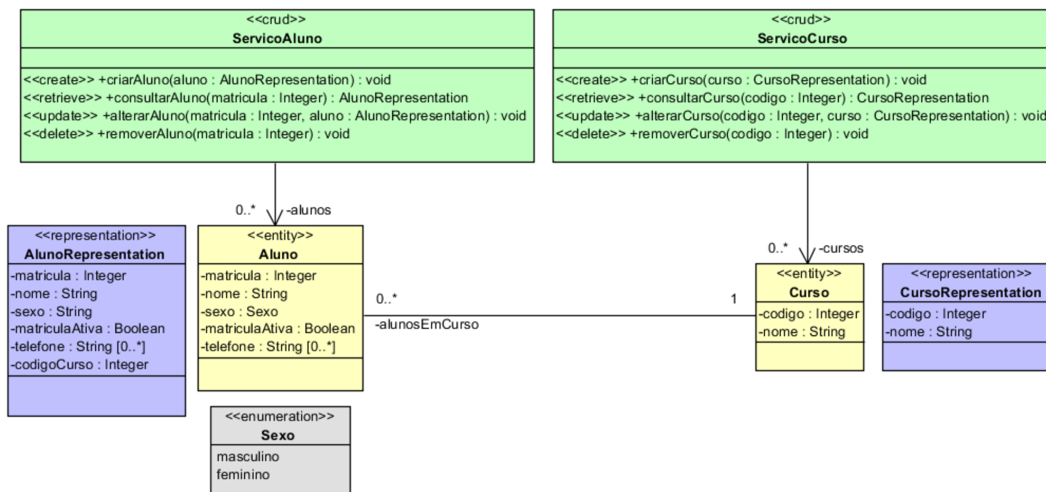

Figura 2: Modelo de classes do domínio Escola.

Um modelo de classes, no entanto, pode não ser suficiente para representar todas as restrições do domínio. Desta forma, esta técnica utiliza a OCL para expressar

invariantes e contratos na forma de pré e pós-condições. Invariantes OCL são utilizadas nesta técnica para definir restrições sobre os valores dos atributos, bem como, para definir restrições de unicidade para as instâncias das classes de domínio.

Os valores dos atributos das classes de domínio devem ser delimitados 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 3 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 3: Operações auxiliares para restrição de valores sobre atributos.

Em seguida são definidas as invariantes para os atributos das classes de domínio. Essas invariantes podem referenciar as operações auxiliares representadas na Figura 3. A Figura 4 apresenta exemplos de invariantes para a classe *Curso*, das quais as duas primeiras tratam de restrição de valores de atributos e a última representa uma restrição de unicidade de instâncias.

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

Figura 4: Invariantes da classe Curso.

O terceiro passo da técnica consiste na definição de contratos na forma de pré e pós-condições OCL. Considerando que operações CRUD possuem comportamentos similares, a técnica propõe o uso de modelos (*templates*) para guiar a escrita de contratos para cada tipo de operação. A Figura 5 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 5: Contrato da operação criarCurso().

4.2. 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 a URI base da aplicação, *Path* adiciona um caminho relativo à 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 cada operação. A Figura 6 exemplifica o uso dos estereótipos e *tagged values* utilizados nesta técnica, no contexto da classe *ServicoCurso*.

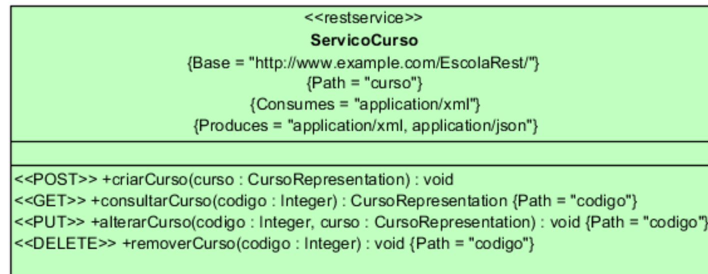

Figura 6: Recorte do modelo de classes estendido do domínio Escola.

4.3. PIT

A geração de casos de teste PIT envolve tanto as restrições expressas no modelo de classes quanto àquelas representadas em OCL. A técnica é dividida em duas fases. 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 nas multiplicidades dos relacionamentos associados a esta entidade, através da análise de valor-limite, separando-os em valores válidos e inválidos.
- Gerar combinações de entrada com valores válidos e combinações de entrada com pelo menos 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 e o retorno esperado para as combinações inválidas seja negativo.

A Tabela 1 apresenta a tabela de decisão produzida para testar as restrições sobre valores de atributos da classe *Curso*.

Tabela 1: 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

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 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 Tabela 2 representa um caso de teste PIT para a operação *criarCurso()*, cujos passos são as próprias operações do serviço e os dados passados como parâmetros são provenientes da linha 6 da tabela de decisão (Tabela 1).

Tabela 2: Exemplo de caso de teste PIT para o método criarCurso().

Sequência de Passos	Resultado Esperado	Função do Passo
1. consultarCurso(1)	Erro	Verifica pré-condição de existência
2. criarCurso({1,a})	Sucesso	Executa operação principal
3. consultarCurso(1)	Sucesso	Verifica pós-condição
4. removerCurso(1)	Sucesso	*Teardown*

4.4. 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 verbo apropriado. A verificação dos resultados esperados é realizada com base nos códigos de resposta e nos cabeçalhos das requisições e das respostas. A técnica considera apenas os códigos de resposta de sucesso e de erro no cliente. A Tabela 3 apresenta padrões de casos de teste PST para operações de criação e remoção, indicando a sequência de requisições HTTP e os resultados esperados para cada requisição. Cabe ressaltar que para garantir o estado necessário ao teste, algumas requisições assumem papel de *setup* ou *tearDown*, criando ou removendo recursos.

Tabela 3: Padrões de casos de teste PST.

Sequência de Requisições	Resultado Esperado
Criação Positiva	
1. GET	*Status Code* = 404
2. POST	*Status Code* = 201
	Location <> null
	Content-Type = *Content-Type* da requisição
3. GET	*Status Code* = 200
	Content-Type $\subset$ *Accept* da requisição
4. DELETE	*Status Code* = 200
Criação Negativa	
1. POST	*Status Code* = 400
Remoção Positiva	
1. POST	*Status Code* = 201
	Location <> null
	Content-Type = *Content-Type* da requisição
2. DELETE	*Status Code* = 200
3. GET	*Status Code* = 200
	Content-Type $\subset$ *Accept* da requisição
Remoção Negativa	
1. DELETE	*Status Code* = 404 (recurso inexistente) ou *Status Code* = 400 (erro de integridade referencial)

Considerando o padrão "Criação Positiva", a Tabela 4 representa um caso de teste positivo para a operação *criarCurso()*, utilizando dados válidos. A Tabela 5, apresentada em seguida, representa um caso de teste com o padrão "Remoção Negativa", 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, todas as instâncias da classe *Aluno* fazem referência a alguma instância da classe *Curso*.

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

Sequência de Requisições	Resultado Esperado na Resposta
GET http://www.example.com/EscolaRest/curso/1 Accept: application/xml, application/json	*Status Code* = 404
POST http://www.example.com/EscolaRest/curso Accept: application/xml, application/json Content-Type: application/xml <curso><codigo>1</codigo><nome>a</nome></curso>	*Status Code* = 201 *Location* <> null *Content-Type* = *Content-Type* da requisição
GET http://www.example.com/EscolaRest/curso/1 Accept: application/xml, application/json	*Status Code* = 200 *Content-Type* ⊂ *Accept* da requisição
DELETE http://www.example.com/EscolaRest/curso/1 Accept: application/xml, application/json	*Status Code* = 200

Tabela 5: Exemplo de caso de teste negativo para a operação removerCurso().

Sequência de Requisições	Resultado Esperado na Resposta
POST http://www.example.com/EscolaRest/curso Content-Type: application/xml <curso><codigo>1</codigo><nome>a</nome></curso>	*Status Code* = 201 *Location* <> null *Content-Type* = *Content-Type* da requisição
POST http://www.example.com/EscolaRest/aluno Content-Type: application/xml <aluno> <matricula>1</matricula><nome>a</nome> <sexo>masculino</sexo> <matriculaAtiva>true</matriculaAtiva> <telefone>11111111</telefone> <curso>1</curso> </aluno>	*Status Code* = 201 *Location* <> null *Content-Type* = *Content-Type* da requisição
DELETE http://www.example.com/EscolaRest/curso/1 Accept: application/xml, application/json	*Status Code* = 400
DELETE http://www.example.com/EscolaRest/aluno/1 Accept: application/xml	*Status Code* = 200
DELETE http://www.example.com/EscolaRest/curso/1 Accept: application/xml, application/json	*Status Code* = 200

5. Conclusão

Este trabalho apresentou uma abordagem sistematizada para geração de casos de teste para serviços web *RESTful* do tipo 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 sua precisão e, consequentemente, propiciar a criação de casos de teste significativos.

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. Este trabalho, no entanto, não determina uma ferramenta para execução dos casos de teste produzidos. Qualquer ferramenta ou biblioteca HTTP pode ser utilizada para executá-los.

Resultados obtidos em um estudo experimental mostraram que os casos de teste gerados através da abordagem proposta possuem qualidade significativamente melhor do que os casos de teste produzidos por estudantes e profissionais da área de desenvolvimento de software utilizando critérios de teste *ad hoc*.

Referências

Askaruinisa, A. and Abirami, A. M. (2010). Test Case Reduction Technique for Semantic Based Web Services. *International Journal on Computer Science and Engineering (IJCSE), 02* (03), p. 566-576.

Bozkurt, M., Harman, M. and Hassoun, Y. (2010). *Testing Web Services: A Survey.* King's College London, Department of Computer Science, Londres.

Canfora, G. and Penta, M. (2009). Service-Oriented Architectures Testing: A Survey. *Software Engineering: International Summer Schools, ISSSE 2006-2008*, p. 78-105.

Chakrabarti, S. K. and Rodriquez, R. (2010). Connectedness testing of RESTful web-services. *Proceedings of the ISEC '10* (p. 143-152). New York, NY: ACM.

Daigneau, R. (2012). *Service Design Patterns*. Upper Saddle River: Pearson.

Delamaro, M. E., Maldonado, J. C. and Jino, M. (2007). *Introdução ao Teste de Software.* Rio de Janeiro: Elsevier.

Endo, A. T. and Simão, A. S. (2010). *Formal Testing Approaches for Service-Oriented Architectures and Web Services: A Systematic Review.* USP, São Carlos.

Erl, T., Karmarkar, A., Walmsley, P., Haas, H., Yalcinalp, U., Liu, C. K., et al. (2008). *Web service contract: design e versioning for SOA.* Crawfordsville: Prentice Hall.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures.* Tese de Doutorado, University of California, Irvine.

Meszaros, G. (2007). *xUnit Test Patterns: refactoring test code.* Boston: Pearson.

Noikajana, S. and Suwannasart, T. (2009). An Improved Test Case Generation Method for Web Service Testing from WSDL-S and OCL with Pair-Wise Testing Technique. *Proceedings of COMPSAC'09* (p. 115-123). Washington: IEEE Computer Society.

OMG, Object Management Group. (2003). *MDA guide version 1.0.1.* OMG.

OMG, Object Management Group. (2010). *Object Constraint Language*, 2.2. Acesso em 18 de outubro de 2010, disponível em http://www.omg.org/spec/OCL/2.2/PDF

Reza, H. and Van Gilst, D. (2010). A Framework for Testing RESTful Web Services. *Proceedings of the Seventh International Conference on Information Technology* (p. 216-221). IEEE Computer Society.

W3C, World Wide Web Consortium. (2004). *OWL Web Ontology Language*. Acesso em 15 de novembro de 2011, disponível em http://www.w3.org/TR/owl-features/

W3C, World Wide Web Consortium. (2009). *Web Application Description Language*. Acesso em 15 de nov. de 2011, disponível em http://www.w3.org/Submission/wadl/

W3C, World Wide Web Consortium. (2005). *Web Service Semantics - WSDL-S*. Acesso em 17 de outubro de 2011, disponível em http://www.w3.org/Submission/WSDL-S/

Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language: getting your models ready for MDA* (2. ed.). Boston: Addison-Wesley.

Webber, J., Parastatidis, S. and Robinson, I. (2010). *REST in Practice.* Sebastopol: O'Reilly.