Testes Funcionais de Web Services RESTful a partir de Modelos UML

Thiago Silva-de-Souza[1,2], Alexandre Luis Correa[3], Eber Assis Schmitz[1], Antonio Juarez Alencar[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, eber@nce.ufrj.br, juarezalencar@dcc.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. 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%.*

Resumo. *Este trabalho apresenta uma abordagem para geração de casos de teste de Web Services RESTful. 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%.*

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 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

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

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 cinco seções. A seção 2 discute os trabalhos relacionados. A seção 3 apresenta a abordagem proposta através de um exemplo. A seção 4 apresenta um estudo experimental realizado para avaliar a abordagem. Por fim, a seção 5 apresenta as conclusões.

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) [Bozkurt, Harman e Hassoun 2010].

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 visa produzir casos de teste para serviços web RESTful do tipo CRUD que manipulam recursos baseados em tipos de dados complexos. A abordagem segue uma linha similar à empregada em abordagens de desenvolvimento dirigido por modelos e, em particular, à estrutura PIM (*Platform Independent Model*) – PSM (*Platform Dependent Model*) presente na especificação *Model-Driven Architecture* (MDA) [OMG 2003]. A definição dos casos de teste para o serviço é feita a partir de dois modelos: um modelo que especifica o serviço de forma independente da tecnologia de implementação (PIM), e um modelo que especifica os aspectos ligados à sua implementação na forma de serviço web RESTful (PSM).

3.1. Visão Geral

A abordagem é composta por quatro atividades inter-relacionadas. Tais atividades têm como objetivo geral sistematizar a produção tanto da especificação do serviço como dos casos de teste.

A primeira atividade tem como objetivo definir, de forma independente de implementação, as informações manipuladas pelo serviço, as restrições de integridade e as operações de manipulação dessas informações, ou seja, as operações de criação, recuperação, atualização e exclusão. Essas definições são feitas por meio de um modelo conhecido na abordagem MDA como *Platform Independent Model* (PIM) [OMG 2003]. No caso de serviços CRUD, esse modelo é composto por três elementos: i) um modelo de classes UML que define as entidades do domínio, seus atributos e associações; ii) invariantes expressas em OCL que definem restrições sobre os valores dos atributos e

associações; iii) contratos para as operações do serviço, na forma de pré e pós-condições OCL que devem ser satisfeitas na execução de cada operação do serviço.

A segunda atividade objetiva a elaboração do modelo específico de plataforma (*Platform Specific Model* – PSM) [OMG 2003] a partir do modelo PIM produzido pela atividade anterior. Nesta atividade, o modelo de classes UML é estendido através da adição de estereótipos e etiquetas (*tagged values*) que permitem a representação de características inerentes à tecnologia de serviços web *RESTful*. Desta forma, informações como o formato de representação consumido ou produzido por uma operação, bem como o padrão de URI utilizado como referência a um recurso são representadas no próprio modelo, tornando desnecessária a geração de um descritor WADL.

O modelo independente de plataforma gerado pela primeira atividade é a entrada para a realização da terceira atividade, cujo propósito é produzir casos de teste independentes de plataforma (*Platform Independent Test* – PIT). Essa atividade consiste na derivação de casos de teste a partir das entidades, atributos, operações e restrições definidas no modelo PIM, 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 atividade consiste na geração dos casos de teste específicos de plataforma (*Platform Specific Test* – PST) que serão utilizados para verificar a implementação real do serviço. Essa técnica recebe como entradas o modelo específico de plataforma e os casos de teste independentes de plataforma produzidos nas atividades anteriores, e combina os testes independentes de plataforma com testes sobre as restrições tecnológicas especificadas no modelo PSM, produzindo os casos de teste específicos para serviços web RESTful. Cada caso de teste produzido por esta atividade será composto por um conjunto de requisições HTTP e um conjunto de assertivas sobre o resultado retornado.

Para exemplificar a aplicação da abordagem, será utilizado um simples domínio de escola com serviços que manipulam instâncias das entidades *Curso* e *Aluno*. As regras do domínio são típicas de sistemas CRUD e estão representadas no PIM descrito na subseção 3.2.

3.2. Especificação Independente de Plataforma

O primeiro elemento que compõe a especificação independente de plataforma (PIM) do serviço 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 definem as operações CRUD que manipulam instâncias de cada classe de domínio. Cada classe de serviço recebe o estereótipo <<crud>> e define, no mínimo, as quatro operações básicas sobre a entidade manipulada (criarXXX, consultarXXX, alterarXXX, removerXXX, onde XXX corresponde ao nome da entidade), cada qual recebendo o respectivo estereótipo (<<create>>, <<retrieve>>, <<update>>, <<delete>>). Cada classe de domínio representa uma entidade do negócio cujas instâncias são tratadas pelas operações das classes de serviço, recebendo o estereótipo <<entity>>.

As classes de representação servem para definir estruturas de dados de entrada e saída utilizadas pelas operações do serviço, similar ao padrão *Data Transfer Object* (DTO) [Daigneau 2012]. Por exemplo, a operação *criarAluno* recebe um elemento do tipo *AlunoRepresentation* que define as informações de entrada que originarão, como resultado da operação, uma instância de *Aluno* adicionada à memória do serviço. Em geral, a mesma representação pode ser utilizada nas operações do serviço, mas é possível definir representações específicas para cada operação, por exemplo, em casos onde os dados recebidos na criação de uma instância correspondam a um subconjunto dos atributos da entidade. Classes de representação recebem o estereótipo `<<representation>>`. A Figura 1 ilustra um exemplo de modelo de classes produzido para dois serviços CRUD do domínio Escola.

Um modelo de classes, no entanto, pode não ser suficiente para representar todas as restrições do domínio. Desta forma, consideramos que as restrições adicionais devem ser especificadas em OCL por meio de invariantes e de pré e pós-condições associadas ao modelo de classes.

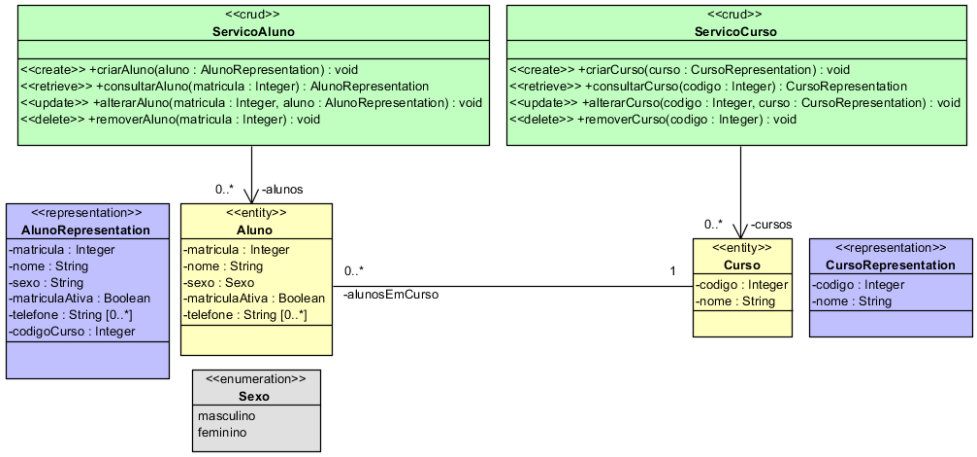

Figura 1: Modelo de classes do domínio Escola.

As regras que restringem os valores dos atributos das classes de domínio devem ser definidas com o uso de invariantes, que podem influenciar a definição dos contratos das operações do serviço. A Figura 2 ilustra um exemplo dessa situação. Nas linhas 1-2, uma invariante restringe os valores válidos para o código de um curso para o intervalo 1 a 99. A operação *criarCurso* de *ServicoCurso*, por sua vez, recebe uma estrutura de dados denominada *CursoRepresentation*. Como existe uma restrição sobre os valores válidos para o código de um curso, definida pela invariante, é preciso especificar uma pré-condição sobre o valor recebido na invocação da operação (linhas 4-5).

```
1 context Curso
2   inv: codigo >= 1 and codigo <= 99
3
4 context ServicoCurso::criarCurso(curso : CursoRepresentation)
5   pre: curso.codigo >= 1 and curso.codigo <= 99
```

Figura 2: Restrições sobre o código de um curso.

Para evitar redundância nas especificações de restrições e pré-condições das operações do serviço, sugerimos que as restrições sobre valores de atributos sejam 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 a reestruturação das restrições presentes na Figura 2 com a definição de uma operação auxiliar na classe *ServicoCurso*. Desta forma, tanto a invariante como as pré-condições passam a utilizar uma definição centralizada da restrição sobre o código do curso.

```
1 context ServicoCurso::codigoValido(codigo : Integer) : Boolean
2   body: codigo >= 1 and codigo <= 99
3
4 context Curso
5   inv: servicoCurso.codigoValido(código)
6
7 context ServicoCurso::criarCurso(curso : CursoRepresentation)
8   pre: self.codigoValido(curso.codigo)
```

Figura 3: Operações auxiliares para restrição de valores sobre atributos.

Além das invariantes, o modelo deve especificar o contrato de cada operação dos serviços na forma de pré e pós-condições OCL. Considerando que operações CRUD possuem comportamentos similares, utilizamos modelos (*templates*), descritos em [Silva-de-Souza 2012] para guiar a escrita de contratos para cada tipo de operação. A Figura 4 mostra um exemplo de contrato para a operação *criarCurso()*, no qual as pré-condições estabelecem que um curso só pode ser criado se possuir código e nomes válidos (cujas regras são definidas como operações da classe ServicoCurso) e se não existir nenhum curso com o mesmo código (restrição de unicidade sobre o código de um curso). A pós-condição estabelece que, após a execução da operação, uma nova instância da classe *Curso* deve ter sido adicionada às instâncias existentes no momento da chamada da operação.

```
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 4: Contrato da operação criarCurso().

3.3. Especificação para Plataforma Específica

A elaboração da especificação para plataforma específica (PSM) consiste em estender o modelo de classes do independente de plataforma (PIM) através da redefinição de estereótipos e da inclusão de etiquetas (*tagged values*) nas classes de serviço. O estereótipo `<<crud>>` definido no compartimento do nome da classe é substituído pelo estereótipo `<<restservice>>`, para indicar um serviço web RESTful. Os estereótipos definidos para as operações são substituídos pelos estereótipos `<<POST>>`, `<<GET>>`, `<<PUT>>` e `<<DELETE>>`, em referência aos verbos HTTP, conforme o tipo de operação.

As etiquetas podem ser utilizadas de duas formas: (i) no escopo da classe e (ii) no escopo de uma operação. No escopo da classe, são definidas as etiquetas *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 etiquetas também podem ser utilizadas para adicionar informações específicas a cada operação. A Figura 5 exemplifica o uso dos estereótipos e etiquetas utilizados no contexto da classe *ServicoCurso*.

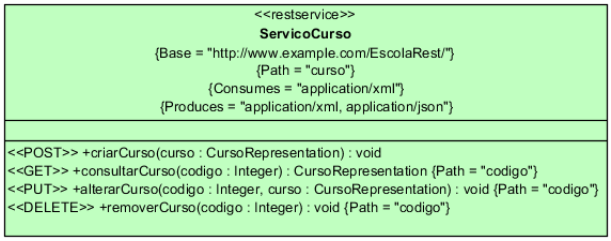

Figura 5: Recorte do modelo de classes estendido do domínio Escola.

3.4. Testes Independentes de Plataforma (PIT)

A definição de casos de teste PIT envolve tanto as restrições representadas no modelo de classes (multiplicidade de atributos e associações) como aquelas especificadas 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	tamanho 1		X
2	0	tamanho 1		X
3	1	Null		X
4	1	tamanho 0		X
5	1	tamanho 1	X	
6	1	tamanho 20	X	
7	1	tamanho 21		X
8	99	tamanho 1	X	
9	99	tamanho 20	X	
10	100	tamanho 1		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 por ferir 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 as operações fornecidas na interface dos serviços, sem o uso de operações 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 serviço sob teste possui estado vazio e que, dependendo do caso de teste, algumas operações do serviço podem ser utilizadas como auxiliares no seu início (*setup*) e no seu término (*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 5 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*

3.5. Testes Específicos para Plataforma (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. Operações de criação são substituídas por requisições do tipo POST; operações de consulta são transformadas em requisições do tipo GET; operações de alteração são mapeadas em requisições PUT; operações de remoção são representadas por requisições do tipo 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 (categorias 2xx e 4xx, respectivamente). Toda resposta HTTP possui um código de resposta (*status code*), frequentemente usado para controle de erros. A Tabela 3 apresenta os códigos de resposta para situações de sucesso e de erro.

Tabela 3: Códigos de resposta tradicionais dos métodos HTTP.

Método HTTP	Código de Sucesso	Código de Erro
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 eventuais violações das restrições tecnológicas definidas no PSM.

Requisições com o 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, assim como o método PUT, também utiliza 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 etiquetas *Produces* e *Consumes* do PSM. Este tipo de verificação também é realizado 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 4, que serve como um modelo geral de assertivas para cada tipo de método.

Tabela 4: Modelo geral de assertivas para casos de teste PST.

Método HTTP	Resposta HTTP de Sucesso	Resposta HTTP de Erro
POST	*Status Code* = 201	*Status Code* = 400
	Location <> null	
	Content-Type = *Content-Type* da requisição	
GET	*Status Code* = 200	*Status Code* = 404
	Content-Type ⊂ *Accept* da requisição	
PUT	*Status Code* = 201	*Status Code* = 400
	Content-Type = *Content-Type* da requisição	
DELETE	*Status Code* = 200	*Status Code* = 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, e associar as assertivas correspondentes, conforme a Tabela 4.

A Tabela 5 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 não existe na

memória do serviço. 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 passou a existir. A requisição com DELETE ao final do caso de teste remove o recurso criado pela requisição da linha 2.

Tabela 5: Modelo de caso de teste PST positivo para operações de criação.

Sequência de Requisições	Resultado Esperado
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 ⊂ *Accept* da requisição
4. DELETE	*Status Code* = 200

Com base no PIT definido na Tabela 2 e no modelo apresentado na Tabela 5, a Tabela 6 representa um caso de teste positivo para a operação criarCurso(). O URI é obtido pela concatenação dos valores das etiquetas *Base* da classe ServicoCurso e *Path* da operação criarCurso(), representadas na Figura 5.

Tabela 6: Caso de teste PST 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

4. Estudo Experimental

4.1. Introdução

A abordagem aqui descrita propõe técnicas para especificação e geração de casos de teste para serviços web *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 para serviços web RESTful. 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. A amostragem foi realizada por

conveniência, por se tratarem de pessoas inseridas em empresas do relacionamento do primeiro autor.

4.1.1. Objetivos

O objetivo principal deste estudo é verificar 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 em uma distribuição normal.

Além disso, é necessário verificar se há diferenças significativas nas observações das duas amostras (alunos e profissionais), de forma a identificar a necessidade de aplicação de tratamentos específicos a cada uma.

4.2. Hipóteses

As hipóteses foram definidas em função de duas questões a serem respondidas com o estudo, denominadas Q1 e Q2:

- Q1: Qual a proporção de participantes que alcança ao menos 95% da cobertura de restrições alcançada pela abordagem proposta?
- Q2: Existem diferenças significativas entre a cobertura média de restrições entre os dois grupos constituintes da amostra?

Quanto à Q1, a hipótese nula declara que a abordagem proposta pouco acrescenta ao resultado obtido utilizando o conhecimento prévio dos participantes. Em outras palavras, a cobertura obtida pela aplicação da abordagem é trivial. Desta forma, a hipótese nula considera que a proporção de testadores que alcança ao menos 95% da cobertura de restrições usando seus próprios critérios de teste é igual ou superior a 68%. A hipótese alternativa considera que tal proporção é menor que 68%.

Para a questão Q2, a hipótese nula declara que os dois grupos de participantes apresentam um desempenho equivalente no que diz respeito à cobertura de testes. A hipótese alternativa declara que os dois grupos de participantes não são equivalentes em termos de média de restrições cobertas pelos testes.

4.3. Operação

A execução do experimento foi realizada em duas etapas. Cada grupo realizou 8 horas de treinamento e 8 horas para resolução de um exercício prático cujo objetivo era projetar casos de teste para o domínio Escola, 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 em um questionário de caracterização do participante.

Os participantes projetaram os casos de teste na ferramenta soapUI [SmartBear 2011], tendo em vista que tal ferramenta apresenta recursos que facilitam o projeto de casos de teste para Web Services. Tais casos de teste foram gravados em arquivos identificados e recolhidos pelo pesquisador para posterior análise.

4.4. Análise e Interpretação dos Resultados

4.4.1. Restrições Testadas pelos Participantes

Após a realização do exercício prático 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 permitia identificar facilmente seu objetivo, visando acelerar 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.

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 média de restrições positivas testadas foi de 62,6, próxima do total de restrições positivas incluídas no PSM do exercício prático (66). Entretanto, a média de restrições negativas testadas foi 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 a testar as situações válidas das operações, em detrimento das situações com dados inválidos.

4.4.2. Testes de Hipótese

A questão Q1 trata da proporção de participantes que atingem a cobertura mínima satisfatória das restrições definidas no PSM do exercício prático. 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. 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 p [Triola 2008]. A margem de erro para proporções pode ser calculada pelo método de Agresti e Coull [Agresti e Coull 1998]. Tal método é um tipo de aproximação para o intervalo de confiança para a distribuição binomial.

Apenas dois participantes (5%) alcançaram a meta de cobertura eficaz de restrições. Desta forma, o intervalo de confiança encontrado para p foi $(p_{min}..p_{max})$= (0,01 .. 0,20). Levando em consideração o erro de amostragem, o valor máximo para esta proporção seria de 20%, ou seja, 7 participantes, o que é muito inferior ao valor especificado na hipótese nula (no mínimo 68%, 24 testadores). Sendo assim, podemos rejeitar a hipótese nula e aceitar a hipótese alternativa para Q1. 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 técnicas *ad-hoc*.

Para verificar se os grupos de participantes são equivalentes (questão Q2) 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. O teste de Mann-Whitney considera as seguintes hipóteses:

$$H_0: m_a = m_b \qquad H_1: m_a \neq m_b$$

Onde m_a e m_b representam as medianas das populações em teste. A Tabela 7 mostra os valores-p (*p-values*) obtidos.

Tabela 7: 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.

5. Conclusão

Este trabalho apresentou uma abordagem sistematizada para geração de casos de teste de 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 definição 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 combinadas com restrições tecnológicas específicas da tecnologia REST. Este trabalho, no entanto, não determina uma ferramenta para execução dos casos de teste produzidos, ou seja, qualquer ferramenta ou biblioteca HTTP pode ser utilizada para executá-los.

Referências

Agresti, A., & Coull, B. A. (1998). Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician , 52*, p. 119-126.

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

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

Chakrabarti, S. K., & Rodriquez, R. (2010). Connectedness testing of RESTful web-services. *Proceedings of the 3rd India Software Engineering Conference (ISEC '10)* (p. 143-152). New York, NY: ACM.

Daigneau, R. (2012). *Service Design Patterns: fundamental design solutions for SOAP/WSDL and RESTful Web Services.* Upper Saddle River: Pearson.

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

Endo, A. T., & Simão, A. d. (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., & 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 33rd Annual IEEE International Computer Software and Applications Conference* (pp. 115-123). Washington, DC: 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., & 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.

Richardson, L., & Ruby, S. (2007). *Restful web services.* Sebastopol: O'Reilly.

Silva-de-Souza, T. (2012). *Uma Abordagem Baseada em Especificação para Testes de Web Services RESTful.* Dissertação de Mestrado, PPGI, Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro.

SmartBear. (2011). Acesso em 20 de dezembro de 2011, disponível em soapUI: http://www.soapui.org

Sommerville, I. (2011). *Engenharia de software* (9. ed. ed.). São Paulo: Pearson.

Triola, M. F. (2008). *Introdução à estatística.* Rio de Janeiro: LTC.

W3C, World Wide Web Consortium. (2009). *Web Application Description Language*. Acesso em 15 de novembro 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/

W3C, World Wide Web Consortium. (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1*. Acesso em 15 de nov de 2010, disponível em http://www.w3.org/TR/wsdl20/

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

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

Wohlin, C., Runeson, P., Höst, M., Olhsson, M. C., Regnell, B., & Wesllén, A. (2000). *Experimentation in software engineering: an introduction.* Norwell: Kluwer Publishers.