REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA

TESTES EM APLICAÇÕES WEB
Abordagem de testes sobre Web Services

Ricardo Antunes Barbosa – *Universidade Estadual de Campinas - Unicamp*

RESUMO: Atualmente os Web Services estão ganhando cada vez mais popularidade principalmente por facilitar a integração de sistemas de diferentes plataformas e por trabalhar com tecnologias solidificadas no cenário da Internet como, por exemplo, XML e HTTP. Diferentemente dos softwares tradicionais, os Web Services possuem características específicas que proporcionam desafios para realização de testes. Os Web Services trabalham sobre a uma arquitetura orientada a serviços (SOA) onde as técnicas de testes tradicionais dificilmente podem ser utilizadas sem adaptações devido ao dinamismo da arquitetura e outros fatores. Este trabalho tem como objetivo esclarecer as principais dificuldades na realização de testes além de explanar algumas técnicas de testes propostas recentemente como, por exemplo, teste de mutação sobre documentos WSDL e teste de perturbação de dados em mensagens SOAP.

PALAVRAS-CHAVE:
Teste de Web Services, Testes de mutação em Web Services, Perturbação de dados em Web Services, Teste de conformidade

ABSTRACT: Web Services have been gaining more popularity mainly because they enable the integration of systems which belong to different platforms and because they work with highly accepted technologies for the Internet environment, for example, XML and HTTP. Unlike traditional software, Web Services have specific characteristics that provide challenges for testing. Web Services work on a Service-Oriented Architecture (SOA) in which tradicional testing techniques can hardly be used without adjustments due to the dynamism of its architecture, among other factors. This paper aims to clarify the main difficulties in testing as well as explain some recently proposed testing techniques, for instance, mutation tests on WSDL documents and data perturbation tests in SOAP messages.

KEYWORDS:
Testing Web Services, Mutation Testing in Web Services, Data Perturbation in Web Services, Compliance Testing

Artigo Original
Recebido em: 14/10/2011
Avaliado em: 18/10/2014
Publicado em: 28/04/2014

Publicação
Anhanguera Educacional Ltda.

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Correspondência
Sistema Anhanguera de Revistas Eletrônicas - SARE
rc.ipade@anhanguera.com

1. INTRODUÇÃO

Nos últimos anos as aplicações Web tem se tornado cada vez mais complexas. Originalmente, as aplicações Web tinham como principal função apresentar informações estáticas, ou seja, documentos em formato texto com pouca interação com o usuário. Com o passar dos anos e com o surgimento de novas tecnologias, a simples apresentação de documentos estáticos passou a não atender as necessidades dos usuários da Internet.

Desta forma, novos recursos foram desenvolvidos e adicionados às aplicações Web, sendo alguns deles o dinamismo de páginas (com uso de tecnologias como CGI, ASP, JSP, PHP e outros), scripts para interação com o usuário (linguagens de scripts como VBScript e Javascript), documentos XML (como proposta para troca entre aplicações ou componentes Web) e Web Services (que agregam a aplicação Web uma coleção de funções ou serviços disponíveis em uma rede).

Aplicações Web do tipo comércio eletrônico, e-learning, serviços de busca entre outras estão fazendo uso atualmente de Web Services pois estes permitem uma integração entre diversos ambientes além de estimular o desenvolvimento distribuído de aplicações Web. Assim como qualquer sistema, os Web Services precisam ser testados a fim de garantir sua qualidade e principalmente sua confiabilidade. Em alguns casos, a falha de um Web Service pode gerar grandes prejuízos tanto para os consumidores do serviço quanto para os seus mantenedores. Desta forma, pesquisas sobre técnicas para realização de testes em Web Services têm sido propostas a fim de melhorar a qualidade e eficiência destes novos componentes de software. Devido a algumas características únicas dos Web Services e também por causa da arquitetura de funcionamento onde estão envolvidos, novas técnicas de testes estão sendo recomendadas além de novas ferramentas de testes.

A seção 2 tem como objetivo esclarecer conceitos chaves como testes de software, validação e verificação. Uma descrição sobre Web Services e algumas tecnologias associadas serão relatadas na seção 3. A seção 4 descreverá algumas propostas de pesquisadores para realização de testes em Web Services. A seção 5 descreverá algumas ferramentas disponíveis no mercado que realizam testes automatizados em Web Services. As considerações finais serão relatadas na seção 6.

2. VALIDAÇÃO, VERIFICAÇÃO E TESTE DE SOFTWARE

A construção de um software seja ele desktop, aplicação Web, aplicação móvel, etc., não é uma tarefa simples. O processo de construção de software atua sob um ambiente complexo e propício ao surgimento de defeitos, podendo fazer com que o software não funcione adequadamente. Segundo Delamaro (2007), muitos fatores podem causar o surgimento de problemas em um software, mas há um consenso sobre a origem da maioria destes problemas: o erro humano.

A concepção de um software ocorre através da participação de pessoas em processos de análise, interpretação, construção, etc. onde o surgimento de equívocos normalmente pode ocorrer. Hoje, mesmo com a adoção cada fez maior de ferramentas que visam orientar, padronizar e acompanhar o processo de desenvolvimento de software não extingue a possibilidade de surgimento de erros.

Para que os defeitos existentes em um software possam ser localizados antes que o produto final seja disponibilizado, tem-se adotado um conjunto de atividades conhecidas como Validação, Verificação e Teste que são atividades relacionadas à garantia de qualidade do software. De acordo com Pressman (2002), o processo de validação consiste em assegurar que o produto final corresponda exatamente aos requisitos de software. Desta forma, a validação procurar garantir que se esteja construindo o produto certo. Já as atividades de verificação procuram garantir que o produto esteja sendo construído da maneira correta, ou seja, buscam assegurar a conformidade e consistência do software sendo aplicadas em cada um dos estágios do processo de desenvolvimento.

É um equívoco dizer que o teste de software é um processo para demonstrar que erros não estão presentes em um programa. "Testes, no entanto criativos e aparentemente completos, não podem garantir a ausência de todos os erros." (MYERS, 2004, p. 43).

De acordo com Myers (2004), o principal objetivo do teste de software é revelar a presença de erros. Em outras palavras, o teste de software é um processo de investigação onde o software é executado com a intenção de localizar erros. Assim, considera-se um teste bem sucedido aquele que gera um caso de teste capaz de fazer com que o programa falhe.

Juntos, a validação, verificação e testes procuram proporcionar um software que faça exatamente o que está definido em seu contexto de projeto e nada além disso.

3. PRIMEIROS PASSOS

Nos últimos anos novas tecnologias têm sido desenvolvidas e sendo amplamente utilizadas em aplicações Web. Estas tecnologias em parceria com frameworks vêm favorecendo a construção de aplicações Web cada vez mais complexas que utilizam componentes e serviços distribuídos. Estes serviços conhecidos como Web Services possuem uma interface simples e funcionalidades bem definidas proporcionando um cenário de integração entre as aplicações.

> Um sistema de software projetado para suportar a interoperabilidade entre máquinas sobre rede. Tem uma interface descrita em um formato processável por máquina (especificamente WSDL). Outros sistemas interagem com o serviço Web de uma maneira prescrita por sua descrição usando mensagens SOAP, tipicamente transmitida usando HTTP com serialização XML em conjunto com outras normas relacionadas à Web (W3C. WEB SERVICE GLOSSARY, 2004)

Web Service é uma solução que permite integrar aplicações web independente de plataforma ou linguagem que foram escritas proporcionando agilidade e eficiência na troca de informações. Através de uma "linguagem universal", no caso a XML, um Web Service poderá receber dados e processá-los em sua linguagem de origem e em seguida disponibilizar a informação ou serviço para o qual foi projetado para a aplicação que o invocou. Um Web Service é descrito e definido usando XML (*eXtensible Markup Language*) e são identificados por uma URI (*Unique Resource Identifier*).

Uma das grandes motivações para a utilização de Web Services é a possibilidade de diferentes aplicações comunicarem entre si utilizando recursos diferentes. Em outras palavras, o Web Service permite que qualquer aplicação possa utilizá-lo extraindo e operando informações disponibilizadas por estes. Os Web Services fazem uso de algumas tecnologias como XML, WSDL, SOAP e UDDI. Este artigo não propõe abordar em detalhes as tecnologias citadas anteriormente, mas para um maior aproveitamento da leitura, um breve relato destas tecnologias será descrita a seguir:

XML: *Extensible Markup Language* ou Linguagem de Marcação Extensível é uma linguagem atualmente padrão na Internet desenvolvida para estrutura, transporte e armazenamento de dados. Semelhantemente a linguagem HTML a linguagem XML é uma linguagem de marcação diferenciando-se por não estar limitada a algumas tags pré-definidas. Sendo uma linguagem auto-descritiva e amplamente extensível tornou-se um dos pivôs para o sucesso e disseminação do uso de Web Services. Por possuir uma descrição bem clara de sua utilidade tornou-se um dos padrões mais utilizados não só na Internet mas também para outros aplicativos sendo ainda base para a criação de outras linguagens como WSDL, XHTML, WAP e WML. "XML prove uma linguagem que pode ser utilizada por diferentes plataformas e linguagens de programação podendo ainda expressar mensagens e funções complexas." (W3SCHOOLS. WEB SERVICE).

SOAP: Originado do acrônimo em inglês *Simple Object Access Protocol* é considerado um padrão de comunicação de Web Services. É o protocolo de comunicação que permite a troca de mensagens em um ambiente descentralizado e independente de plataforma e linguagem de programação. O SOAP tem se tornado amplamente utilizado por empregar tecnologias padrões da Internet como a XML e HTTP. A principal função do SOAP é garantir a interoperabilidade e intercomunicação entre sistemas utilizando para isso a linguagem XML sobre um mecanismo padrão de transporte, no caso, HTTP. Partindo do princípio que uma mensagem SOAP é um documento XML podemos descrever os seguintes elementos para este documento:

Envelope: é o elemento que identifica o documento XML como sendo uma mensagem SOAP. O envelope pode conter declarações de *namespaces* como declarações de outros atributos, como por exemplo, o estilo de codificação.

Header: é um cabeçalho opcional contendo informações adicionais como dados específicos de uma aplicação (por exemplo, autenticação). Se o *header* for utilizado em uma mensagem SOAP deve ser o primeiro elemento filho do elemento *envelope*.

Body: diferente do elemento *header* este elemento é obrigatório. Este elemento contém a informação (ou *payload*) a ser transportada para seu destino final.

Fault: é um elemento opcional usado para indicar mensagens de erro. Se o elemento *fault* é presente deve aparecer como um elemento filho do elemento *body*.

WSDL: É uma das linguagens mais importantes no cenário de Web Services e possui um papel fundamental. Originado do acrônimo em inglês *Web Service Descriptor Language,* um arquivo WSDL é um documento escrito em linguagem XML para descrever um Web Service. Um documento WSDL funciona semelhantemente a um contrato de serviço. O WSDL além de descrever minuciosamente o serviço também descreve a sua forma de acesso e a suas operações disponíveis. Segundo o W3C (W3C. WSDL, 2001), o WSDL descreve os serviços de rede como um conjunto de *endpoints* (endereços) sobre as mensagens contendo informações orientadas ao documento (*document-oriented*) ou orientadas a procedimento (*procedure-oriented*). Um documento WSDL descreve um Web Service utilizando os seguintes elementos:

<types>, local onde são definidos os tipos de dados utilizados pelo Web Service.

<message>, define os dados a serem transmitidos pelo Web Service podendo conter um ou mais elementos **<part>**, que formam as partes reais da mensagem.

<operation>, descrição abstrata de uma ação suportada por um serviço. É neste elemento onde é definido o relacionamento entre os parâmetros de entrada e saída.

<portType>, um conjunto abstrato de operações suportadas por um ou mais *endpoints*. Pode ser comparada a uma biblioteca de funções em uma linguagem de programação tradicional.

<binding>, um protocolo concreto para um **<portType>** específico, isto é, define o protocolo utilizado para acessar as operações de um objeto (HTTP, SOAP ou MIME).

<port>, *endpoint* único definido pela combinação de um binding e um endereço de rede.

<service>, uma coleção de *endpoint* relacionados.

UDDI: Acrônimo em inglês *Universal Description, Discovery and Integration* é um diretório para armazenamento de informações sobre Web Services, ou seja, é um diretório de interfaces de Web Services descritas em WSDL. Os provedores de serviços que desejam publicar seus Web Services podem utilizar um registro de serviço UDDI. Os usuários de serviços podem consultar um UDDI para localizar um serviço de interesse e obter as informações necessárias para sua utilização.

4. PROPOSTAS DE TESTES PARA WEB SERVICES

Com o surgimento de novas tecnologias, novas arquiteturas e novos paradigmas de programação é inevitável que haja evolução dos métodos e técnicas de testes de software. Quando há o aparecimento de novos conceitos, é analisado se as técnicas de testes existentes conseguem atender satisfatoriamente os novos desafios propostos. As técnicas de testes tradicionais são bases sólidas para a criação de novas técnicas de testes devendo seus conceitos serem fortemente considerados.

Em alguns casos, as técnicas de testes tradicionais passam por simples adaptações para se ajustarem as novas metas de testes. Em outros casos, as técnicas de testes tradicionais precisam sofrer ajustes tão grandes que acabam descaracterizando-se havendo perda de sua identidade inicial. Desta forma, surgem novas técnicas de testes para fazer frente aos novos desafios e dificuldades propostas por novas tecnologias e arquiteturas.

> Web Services têm o potencial de reduzir drasticamente a complexidade e os custos envolvidos em um projeto de integração de software. [...] O método de comunicação introduz complexidades para os problemas da verificação e validação de serviços da Web que não existem no software tradicional. [...] (Offutt; Xu, 2004, p. 1)

Devido ao dinamismo proposto pela arquitetura SOA (Service Oriented Architeture) torna-se necessária uma proposta de teste e validação apropriada para os Web Services. Para Canfora e Di Penta (2006) este cenário dinâmico (publicação, vinculação, invocação e integração em tempo de execução) trás, sem dúvida, novos desafios para as técnicas de teste atuais. Os Web Services possuem características únicas justificando a necessidade de implantação de novos métodos e ferramentas de testes já que os métodos e ferramentas de testes atuais não trabalham com serviços. (CANFORA; DI PENTA, 2006).

Como a adoção de interfaces gráficas não faz parte do conceito de Web Service, existe uma dificuldade para a realização de testes manuais tornando-os penosos e complexos. Desta forma, os Web Services são fortes candidatos a utilização de testes automatizados. Métodos de testes baseados no código fonte como testes de fluxo de dados, cobertura de código, teste de mutação e outros, ficam comprometidos visto a falta de acesso ao código fonte. A falta de acesso ao código fonte se dá uma vez que os Web Services são vistos como componentes de softwares encapsulados sendo acessíveis por uma interface de comunicação. Testar um Web Service assemelhasse a ao teste do tipo caixa preta onde as especificações são avaliadas, mas a estrutura do código fonte, que pode estar em C#, Java, etc., não é verificada.

Como os Web Services utilizam fortemente um conjunto de tecnologias que fazem uso de padrão baseado em XML, técnicas de testes precisam ser ajustadas para trabalhar com este padrão, ou seja, casos de testes precisam ser gerados considerando o formato padrão baseado em XML.

A partir destas particularidades do cenário SOA, pesquisadores vem sugerindo algumas propostas que podem ser adotadas para testes de Web Services. Este artigo propõe a apresentação das propostas de testes citadas a seguir:

a. Testes baseados em documento WSDL: Um documento WSDL descreve uma interface para a utilização de um Web Service contendo informações que auxiliarão os testadores a especificarem um plano de testes apropriado. As funcionalidades e características de um Web Service serão verificadas a partir de sua interface, ou seja, a partir do documento WSDL.
b. Testes baseados em perturbação de dados: Esta abordagem tem como objetivo alterar mensagens SOAP fazendo uso de operadores de perturbação. As mensagens alteradas por estes operadores são utilizadas como base de testes para a interação entre pares de Web Services.
c. Teste de mutação sobre documentos WSDL: Com o uso da técnica de testes conhecida como Análise de Mutantes, operadores de mutação são desenvolvidos e aplicados em documentos WSDL com o intuito de gerar interfaces modificadas para Web Services que serão utilizadas para testar os serviços.
d. Testes de conformidade para Web Services: A aplicação de testes de conformidade procuram identificar problemas na implementação dos Web Services. O teste de conformidade visa responder a seguinte questão: A implementação do Web Service apresenta o comportamento especificado? Esta abordagem proporcionará aos utilizadores do Web Service garantias que sua implementação corresponde corretamente a descrição do serviço.

4.1. Testes baseados em documentos WSDL

Recentemente alguns pesquisadores vêm demonstrando interesse em explorar as informações descritas em um documento WSDL. As informações contidas em um documento WSDL podem auxiliar na elaboração de casos de testes eficazes proporcionando um nível de confiabilidade cada vez maior a respeito de um Web Service. A partir de um documento WSDL, casos de testes podem ser gerados automaticamente e com eles executar um plano de testes sobre um Web Service monitorando-o e analisando-o em tempo de execução. Desta forma, não só é possível verificar a eficácia do plano de testes proposto mas também analisar criteriosamente o funcionamento do Web Service em estudo.

Em seu artigo Bai *et al* (2005) propõe um framework de testes distribuído baseado em um documento WSDL. Segundo os autores, o framework pode ser aplicado em 4 níveis de testes sendo eles: 1) Teste de operações individuais em um único serviço, 2) Testes de combinações de operações em um único serviço, 3) Testes de operações individuais em serviços compostos e 4) Teste de combinações de operações em serviços compostos. O framework proposto inclui os seguintes componentes:

1. **Gerador de casos de testes**: Este módulo permite a submissão de um documento WSDL e gera um conjunto de casos de testes para serem utilizados em um plano de testes. Este módulo poderá ser estendido para utilizar outras linguagens de descrição de Web Services, como por exemplo, a OWL-S (Marcação Semântica para Web Services). Os casos de testes gerados por este módulo são armazenados em um banco de dados.
2. **Controlador de execução de teste**: Tem como finalidade controlar e monitorar a execução dos testes em um ambiente distribuído e coletar os resultados dos testes. O controlador de execução seleciona os casos de testes armazenados em um banco de dados e os fornece aos agentes de teste.
3. **Agentes de teste**: São componentes (no caso *proxys*) espalhados na rede que realizam testes remotamente em um Web Service utilizando os dados de testes fornecidos pelo controlador de execução de teste.
4. **Analisador de teste**: Tem como objetivo analisar os resultados dos testes produzidos pelos agentes de teste, avaliar a qualidade do serviço e fornecer um relatório de testes.

A figura 3 representa o framework proposto na pesquisa de Bal *et al* (2005)

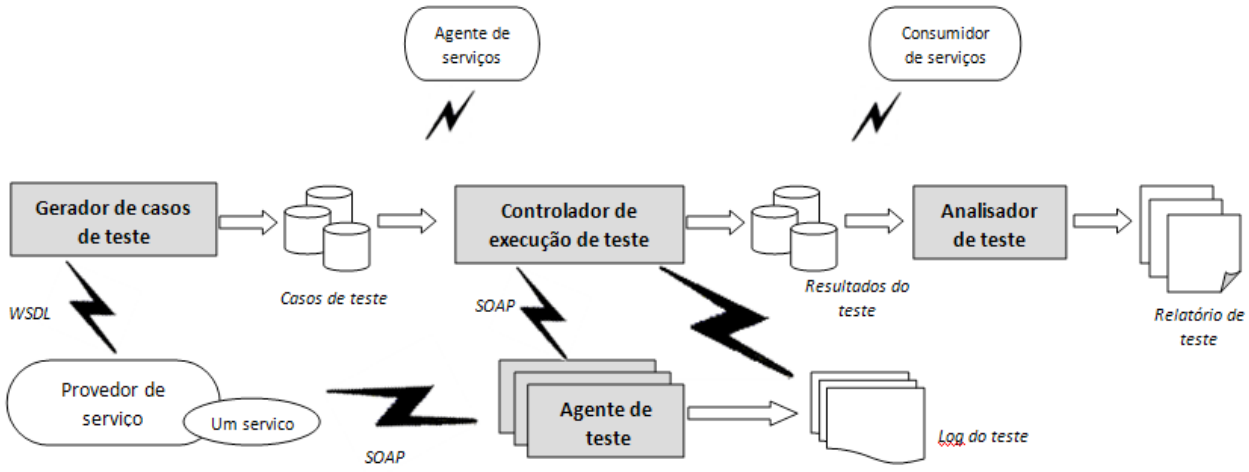

Figura 3 – Figura adaptada de Bal *et al* (2005)

Mesmo que um documento WSDL possua informações de alta relevância para descrição de um Web Service, estas informações ainda não são suficientes para proporcionar testes mais eficazes para avaliar o funcionamento de um serviço. Com um documento WSDL em mãos podemos saber apenas o número de entradas e saídas (*inputs* e *outputs*), os tipos de variáveis para cada entrada e saída, a ordem da entrada e saída dos dados e como um serviço deve ser invocado. De acordo com TSAI *et al* (2002), um documento WSDL padrão não contém informações suficientes para proporcionar, por exemplo, testes de caixa preta e testes de regressão.

Como os documentos WSDL não possuem informações como sequências de invocações, dependência de entradas e saídas (*input/output dependence*), os documentos WSDL atuais são ineficazes para testes de caminho e testes de fluxo de dados. TSAI *et al* (2002) propõem uma extensão da WSDL de forma que o documento de descrição do serviço contemple as seguintes informações:

a. Dependência de entrada e saída: com a utilização de um tipo complexo *WSInputOutputDependenceType* contendo pelo menos um elemento do tipo *WSIOPairType* (também um tipo complexo que representa um par de entrada-saída) será possível descrever as associações entre dados de entrada e dados de saída favorecendo estratégias de testes como teste de regressão e testes de fluxo de dados.

b. Sequências de invocação: para representar o relacionamento ou dependência entre os serviços o documento WSDL deveria conter o nome do serviço e o link do Web Service a qual está relacionado. Uma estratégia proposta seria a descrição destas informações usando o elemento *WSInvocationDependencyType*. Este elemento conterá os sub-elementos *WSICallers* e *WSICallees* para representar os serviços chamadores e os serviços chamados. Testes de fluxo de dados e testes de caminhos poderão ser realizados com a descrição destas informações.

c. Descrição funcional hierárquica: descrições funcionais de um Web Service poderão ser incorporadas de forma hierárquica em um documento WSDL. Estendendo o WSDL incluindo os sub-elementos *WSFParents* e *WSFChildren* do tipo *WSFParentType* e *WSFChildrenType* respectivamente, possuiriam um link para um arquivo WSDL contendo a estrutura hierárquica. Com as descrições funcionais, algumas estratégias de testes poderão ser melhoradas como, por exemplo, os testes funcionais e testes de regressão.

d. Especificação de sequências: um documento WSDL que contenha informações de especificação de sequência pode esclarecer aspectos comportamentais de um Web Service. Mesmo que as especificações de sequência sejam muito comuns em testes de software orientado a objetos elas acrescentam informações úteis que ajudariam na elaboração de um ambiente para testes em Web Services. Segundo TSAI *et al* (2002), uma descrição de uma especificação de sequência em um arquivo WSDL para um serviço que trata transações bancárias poderia adotar o modelo a seguir:

AberturaDeConta – Depósito – (Depósito | Saque) - FechamentoDeConta

Um exemplo detalhado de um documento WSDL contendo uma especificação de sequência é visto em detalhes no trabalho de TSAI *et al* (2002).

4.2. Testes baseados em Perturbação de Dados

Outra abordagem de testes para Web Services é a perturbação de dados. Partindo do princípio que os Web Services interagem entre si através de trocas de mensagens em formato XML, pesquisadores estão estudando maneiras de testar estes serviços através de alterações feitas nestas mensagens.

Offutt e Xu (2004) apresentam uma abordagem de testes de Web Services baseado em perturbação de dados onde mensagens XML são modificadas seguindo regras previamente definidas. Em seguida, essas mensagens modificadas são reenviadas para o serviço onde posteriormente é feita uma análise sobre o comportamento do Web Service. A proposta da perturbação de dados em testes em Web Services ocorre em dois cenários principais:

- **perturbação de dados**: modifica os valores em mensagens SOAP.
- **perturbação de interação**: este cenário é subdividido em perturbação de comunicação RPC que modifica chamadas RPC (chamadas de procedimentos remotos) e perturbação na comunicação de dados que modifica mensagens de comunicação de dados, isto é, mensagens cujo propósito é transmitir dados.

A perturbação de dados (*Data Value Perturbation*) tem como objetivo a alteração dos valores em mensagens SOAP de acordo com regras definidas pelos tipos dos dados. Esse tipo de perturbação é baseada na teoria de Testes de Valores Limites que exercitam os limites do domínio de entrada dos dados. Offutt e Xu (2004) propõem testes de valores limites para os dados do tipo texto (*string*), número (*decimal, float* e *double*) e *boolean*.

Para os dados do tipo texto os valores limites avaliados são: comprimento máximo, comprimento mínimo, texto todo em maiúscula (*upper case*) e texto todo em minúsculas (*lower case*). Para os dados do tipo numérico os valores limites avaliados são: zero, valor máximo e valor mínimo de acordo com um *XML schema*. Já, para os dados do tipo *boolean* os valores testados são *true* e *false*.

A perturbação na comunicação de chamadas de procedimentos (*RPC Communication Perturbation*) tem como principal objetivo efetuar alterações nos parâmetros informados em uma chamada RPC. Baseando-se no princípio da técnica de testes Análise de Mutantes, Offutt e Xu (2004) propuseram alguns operadores de mutação com a finalidade de modificar os valores utilizados nas chamadas RPC, valores que são classificados em: *dados de uso normal* (dados usados dentro do programa) e *dados de uso SQL* (dados que são usados como entrada em um banco de dados).

Para esta tarefa foi utilizado uma representação formal de um schema XML denominada RTG (*Regular Tree Grammar*). Esta gramática de árvore regular é uma 6-tupla $<E, D, N, A, P, n_s>$ onde: E é um conjunto finito de tipos de elementos; D é um conjunto finito de tipos de dados; *N* é um conjunto finito de não-terminais; A um conjunto finito de tipos de atributos; P um conjunto de regras de produção e n_s é o não-terminal inicial, $n_x \in N$.

Dado um conjunto de todos os elementos instâncias de *N*, um operador de mutação é dado por $r = f(n_1,..., n_i)$, onde f é uma função, $i \geq 1$, cada $n_1,..., n_i \in N$ e tem o mesmo tipo de dado, e *r* tem o mesmo tipo de dado que $n_1,... , n_i$.

Os operadores de mutação propostos por Offutt e Xu (2004) para perturbação na comunicação RPC foram:

- *Divide (n):* Modifica o valor de n para $1 \div n$, onde *n* é do tipo *double*.
- *Multiply (n):* Modifica o valor de *n* para $n \times n$.
- *Negative (n):* Modifica o valor de *n* para *-n*.
- *Absolute (n).* Modifica o valor de *n* para $|n|$.
- *Exchange (n1, n2):* Substitui o valor de *n1* com *n2* e vice-versa, quando *n1* e *n2* são do mesmo tipo.
- *Unauthorized (str):* Muda o valor da string *str* para *str' OR '1' =' 1* (semelhante às técnicas de *SQL Injection*)

O objetivo principal da perturbação na comunicação dos dados é testar os relacionamentos e restrições nesses dados. Com um XML Schema é possível realizar relacionamentos entre elementos de dados dentro do XML. Em um XML Schema, através dos indicadores de ocorrência (*minOccurs* e *maxOccurs*) é possível indicar com que freqüência um elemento pode ocorrer. Através destes indicadores é possível definir restrições de cardinalidade tornando-se parte essencial para uma provável estratégia de teste no que se diz respeito a relacionamento. Assim como os relacionamentos, outras restrições também podem ser definidas em um XML Schema. Restrições em tipos de dados numéricos (ex. *enumeration*, *totalDigits*, *maxExclusive*, *minExclusive*, e outros) podem ser facilmente utilizadas para geração de casos de testes.

Ainda no cenário de perturbação de dados, uma alternativa de testes para Web Services é a perturbação em XML Schemas. "XML Schemas é uma linguagem para expressar restrições em documentos XML" (W3C. XML SCHEMA), ajudando a definir a estrutura, conteúdo e semântica destes documentos. Nesta abordagem, a principal diferença esta na perturbação não ser realizada no documento XML utilizado pelo Web Service mas sim sobre o arquivo que o define e o valida, no caso o XML Schema. Nesta abordagem, a proposta de avaliação de um Web Service é verificar o quão bem a aplicação valida uma mensagem XML. Xu, Offutt e Luo (2005) propuseram um método para realizar perturbação em XML Schemas a fim de criarem mensagens inválidas. A proposta além de avaliar o quão bem um Web Service valida uma mensagem XML, avalia também quão bem o Web Service processa os dados sem validação. Esta técnica permitirá analisar a capacidade do Web Service de adaptar a mensagem XML segundo as regras definidas no XML Schema.

Xu *et al* (2005) propuseram alguns operadores de perturbação de XML Schemas com a função de inserir, remover e alterar nós (ou sub-árvores) em XML Schemas originais.

Mensagens XML são geradas a partir dos XML Schemas alterados formando um conjunto de casos de testes. Em seguida, o Web Service é submetido a testes com os casos de testes gerados. Os seguintes operadores de perturbação foram propostos em (XU *et al*, 2005):

- *InsertN*: Insere um novo nó entre dois nós que estão conectados entre si.
- *DeleteN*: Remove um nó de uma árvore. Os nós-filhos são movidos para o nó-pai.
- *InsertND*: Insere um novo nó abaixo de outro nó.
- *DeleteND*: Remove um nó com seus tipos de dados.
- *InsertT*: Insere uma nova árvore abaixo de um determinado nó.
- *DeleteT*: Remove uma sub-árvore.
- *ChangeE*: Modifica as restrições de um XML Schema.

Com exceção dos operadores *InsertT* e *DeleteT*, os demais foram classificados em três critérios de cobertura de testes. Os critérios definidos foram: a) **Cobertura de remoção e Cobertura de Inserção** que são critérios que modificam a estrutura de um XML Schema utilizando para isso os operadores de inserção (*insertND()* e *insertN()*) e remoção (*deleteND()* e *deleteN()*) e b) **Cobertura de restrição** que cobrem todas as restrições existentes em um XML Schema utilizando o operador *changeE().* Dois estudos empíricos utilizando estes operadores podem ser observados em Xu *et al* (2005).

4.3. Testes de Mutação sobre documentos WSDL

A Análise de Mutantes é uma técnica de testes baseada em defeito cujo objetivo é medir a adequação de um conjunto de dados de testes criados externamente (DEMILLO; LIPTON, 1978). A proposta da técnica de Análise de Mutantes é inserir alterações (defeitos produzidos por operadores de mutação) em um programa ou arquivo gerando uma versão modificada denominada "mutante". Ao executar a versão mutante, o testador comparará seu resultado de saída com o resultado de saída produzido pelo programa original. Se os resultados produzidos forem diferentes é dito que o programa mutante é *morto,* caso contrário, é dito que o mutante é *vivo*. Dentre os mutantes vivos, é necessária a análise dos mutantes considerados *equivalentes*, ou seja, aqueles onde casos de testes não conseguem diferenciar a versão mutante da versão original. O escore de mutação é dado pela seguinte expressão:

$$Escore\ de\ Mutação = \frac{\text{Total de Mutantes Mortos}}{\text{Total de Mutantes} - \text{Total de Mutantes Equivalentes}}$$

Um conjunto de dados de testes que produz o escore de mutação mais próximo de 1,0 tem melhores condições de indicar que o objeto testado não possui os defeitos produzidos pelos operadores de mutação.

A figura 4 descreve resumidamente o funcionamento da técnica de testes Análise de Mutantes.

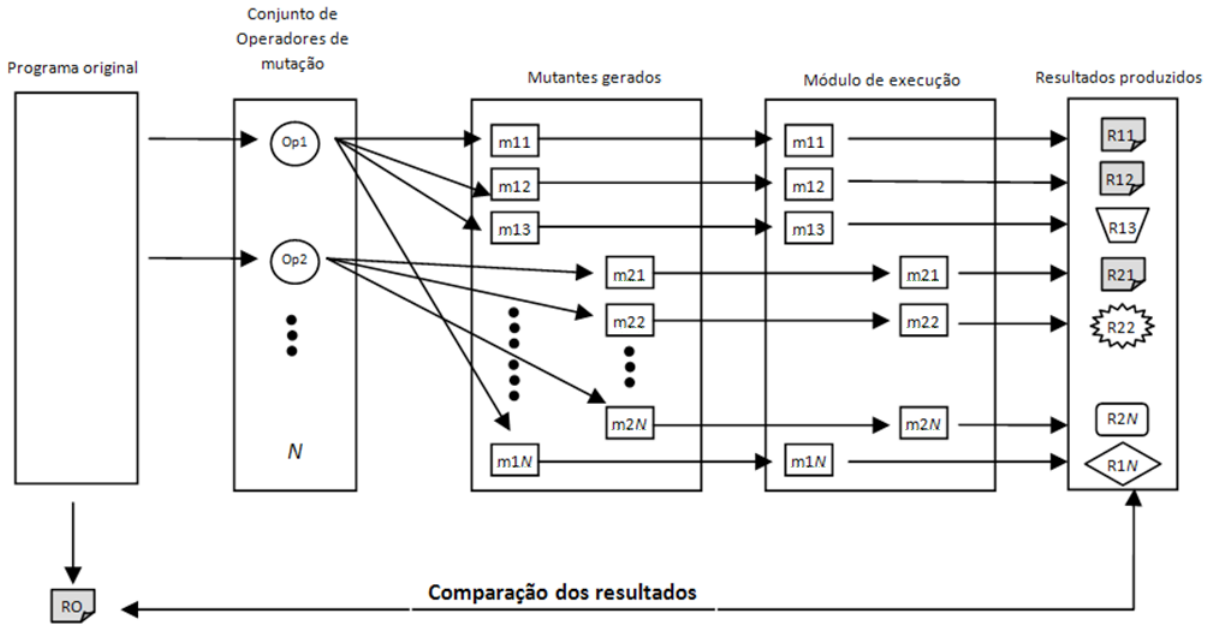

Figura 4 – Resumo da técnica de testes Análise de Mutantes

Em observação a figura 4 percebe-se que um resultado é produzido para cada execução dos mutantes gerados. Alguns destes resultados são iguais ao resultado original (RO) produzido pelo programa original. Como os resultados R11, R12 e R21 são iguais a RO, podemos dizer que os mutantes m11, m12 e m21 são *vivos* ou são *mutantes equivalentes*. Os resultados R13, R22, R1*N* e R2*N* são diferentes a RO. Desta forma, os mutantes m13, m22, m1*N* e m2*N* são mutantes *mortos*.

A dificuldade de acesso ao código fonte do Web Service faz com que os testadores geralmente direcionem os testes sobre artefatos quem possam dar uma idéia de como o Web Service funciona, como por exemplo, os documentos WSDL. Siblini e Mansour (2005) propuseram uma técnica de testes baseada em análise de mutantes que utiliza documentos WSDL. Os pesquisadores sugeriram nove operadores de mutação específicos para documentos WSDL com o objetivo de gerar documentos WSDL modificados. Estes documentos modificados (mutantes) permitirão a geração de casos de testes que poderão, segundo os pesquisadores, revelar erros na interface e até possíveis erros lógicos em um Web Service.

Os nove operadores propostos por Siblini e Mansour (2005) são divididos em 3 grupos sendo: a) **grupo de troca**: onde os operadores deste grupo realizam a troca da sequência de um elemento em um WSDL, b) **grupo especial:** onde os operadores deste grupo modificam o valor de um elemento e c) **grupo de ocorrência:** onde os operadores deste grupo adicionam ou removem a ocorrência de um elemento. A seguir uma breve descrição destes operadores de mutação:

Grupo de troca

- **SwitchTypesComplexTypeElement**: troca os elementos que sejam do mesmo tipo que estão dentro do elemento *complexType*
- **SwitchTypesComplexTypeAttribute**: troca os atributos que sejam do mesmo tipo que estão dentro do elemento *complexType*
- **SwitchTypesSimpleTypeElement**: troca os elementos que sejam do mesmo tipo que estão dentro do elemento *simpleType*
- **SwitchTypesSimpleTypeAttribute**: troca os atributos que sejam do mesmo tipo que estão dentro do elemento *simpleType*
- **SwitchMessagePart**: troca partes de um elemento do mesmo tipo na mensagem
- **SwitchPortTypeMessage**: troca mensagens do mesmo tipo no elemento *operation* que é definido no *PortType*

Grupo especial

- **SpecialTypesElementNil**: ajusta o atributo *nil* para verdadeiro em um elemento do tipo *complexType*

Grupo de ocorrência

- **OccurrenceTypesComplexTypeElement**: adiciona ou apaga uma ocorrência de um elemento em um tipo *complexType*
- **OccurrenceTypesComplexTypeAttribute**: adiciona ou apaga um atributo opcional em um tipo *complexType*

Para explanar um exemplo de aplicação de um dos operadores de mutação sugeridos em (SIBLINI; MANSOUR, 2005) segue uma demonstração do uso do operador *SwitchTypesComplexTypeElement* para um Web Service fictício que valida o login de acesso de um usuário.

O Web Service fictício possui uma função chamada *ChecarLoginUsuario* que receberá dois valores de entrada: o login do usuário e sua senha. A seguir uma proposta de um trecho de documento WSDL para este Web Service.

<s:element minOccurs="0" maxOccurs="1" name="NomeUsuario" type="s:string">
<s:element minOccurs="0" maxOccurs="1" name="Senha" type="s:string">

O trecho do documento WSDL demonstra que a sequência de entrada é primeiro o nome do usuário e depois a senha. Ao fazer uso do operador de mutação *SwitchTypesComplexTypeElement* é gerado o seguinte trecho WSDL mutante

<s:element minOccurs="0" maxOccurs="1" name="Senha" type="s:string">
<s:element minOccurs="0" maxOccurs="1" name="NomeUsuario" type="s:string">

Supondo que um usuário válido (acesso permitido) possua os dados NomeUsuario = "joao" e Senha = "abracadabra", ao utilizar o WSDL mutante os dados de entrada passariam a ser NomeUsuario = "abracadabra" e Senha = "joao". Se com a utilização destes últimos dados de entrada for obtida permissão de acesso, um defeito seria detectado no Web Service.

Ainda no contexto de testes baseados em Análise de Mutantes, Solino e Vergilio (2009) propuseram outros operadores de mutação para atuação em documentos WSDL. Assim como em (SIBLINI; MANSOUR, 2005), os operadores sugeridos por Solino e Vergilio (2009) foram baseados em defeitos comuns em documentos WSDL e possuem condições para revelar defeitos na própria implementação do Web Service. Os operadores sugeridos em (SOLINO; VERGILIO, 2009) são:

- **changeServiceSigning:** muda a ordem dos parâmetros para um serviço disponível em um documento WSDL.
- **changeTypeonServiceSigning:** muda a ordem dos tipos dos parâmetros para um serviço disponível em um documento WSDL. A ordem dos parâmetros não é alterada.
- **changeTypeofMessageElement:** troca o tipo de um elemento definido em uma mensagem por outro tipo de um elemento definido em outra mensagem também presente no documento WSDL.
- **changeInputOutput:** troca o tipo de uma mensagem de entrada por um tipo de uma mensagem de saída (da mesma transação) ambas presentes em um documento WSDL.

O operador *changeServiceSigning* é baseado no operador *SwitchTypesComplexTypeElement* proposto por Siblini e Mansour (2005). Conforme (SOLINO; VERGILIO, 2009), a principal diferença entre eles é que o operador *SwitchTypesComplexTypeElement* troca a ordem dos parâmetros de um serviço de forma aleatória, visto que o operador *changeServiceSigning* troca a ordem apenas dos elementos responsáveis pela definição da assinatura do serviço.

4.4. Teste de Conformidade para Web Service

Outra abordagem sobre testes em Web Services é o teste de conformidade. Este tipo de teste procura verificar se a implementação de um Web Service está de acordo com sua descrição. É fundamental que a implementação de um Web Service esteja em plena conformidade com sua descrição para reduzir as chances de surgimento de problemas e manter a sua eficiência e interoperabilidade. Este tipo de teste aumenta as chances de um serviço estar devidamente implementado garantindo sua Qualidade de Serviço (QoS).

O teste de conformidade foi utilizado por Heckel e Mariani (2005) em uma abordagem voltada para a descoberta de serviços. A proposta dos autores é um framework de *Serviço de Descoberta de Alta Qualidade* onde seriam realizados testes de conformidade para validar um Web Service antes de seu registro em um sistema de registro UDDI. A idéia é que os serviços de descoberta de Web Services gerem automaticamente casos de testes de conformidade a partir da descrição disponibilizada pelo fornecedor do serviço. Só após os Web Services passarem pelos testes seus registros serão efetivados. Isso proporcionará maior confiança

por parte do cliente em estar utilizando um serviço com implementação de qualidade e com comportamento previsto e desejável.

Como visto anteriormente, o documento WSDL é o elemento utilizado para descrição sintática do Web Service. No entanto, o documento WSDL padrão não possui expressividade semântica e não fornece uma descrição comportamental do funcionamento do Web Service. Isso pode, por exemplo, levar um cliente a encontrar e utilizar equivocadamente um serviço. Uma possível solução é as especificações comportamentais do Web Service estarem disponibilizadas da mesma forma como a descrição sintática do serviço.

Para realizar a descrição comportamental do Web Service alguns pesquisadores usam recursos como Regras de Transformação (HECKEL; MARIANI, 2005) e WSDL-S (SINHA; PARADKAR, 2006).

Em seu trabalho Heckel e Mariani (2005) propõem a utilização de grafos com Regras de Transformação (GT Rules). O processo de registro de um Web Service ocorreria em 5 passos:

1. O provedor do serviço realiza o *upload* do documento WSDL e das regras de transformação (GT rules).
2. O serviço de descoberta gera automaticamente casos de testes a partir das especificações contidas nas regras de transformações.
3. Casos de testes "concretos" são executados remotamente utilizando uma interface de teste fornecida pelo Web Service.
4. Os resultados de testes são verificados com base nos resultados produzidos e no estado conceitual do serviço após a execução dos testes.
5. Se o Web Service passar por todos os casos de testes, o serviço de descoberta efetua o registro junto com o documento WSDL e com as regras de transformação.

Segundo Heckel e Mariani (2005), regras de transformação especificam como os parâmetros e dados internos são utilizados e modificados. Estas regras são baseadas no modelo de dados da interface do serviço (um documento WSDL, por exemplo) e também no modelo conceitual do estado interno do serviço. Cada serviço estaria associado a um conjunto de regras de transformação que representam diferentes computações que podem ser feitas quando um serviço é executado com diferentes valores de entrada e diferentes estados. Cada regra de transformação possui uma *pré-condição* para poder ser aplicada e um efeito que consiste em objetos e links que são adicionados, removidos ou possuem os valores de atributos alterados.

As condições verificadas no teste de conformidade proposto por Heckel e Mariani (2005) são a **completude** que é validada por casos de testes dentro do domínio das regras e a **estabilidade** (*soundness*) validada por casos de testes fora do domínio das regras de transformação. Conceitos como partições de domínio, análise de valor limite e análise de fluxo de dados são adequadas ao contexto de grafos de regras de transformação.

Sinha e Paradkar (2006) apresentam em seu trabalho uma abordagem de geração de teste de conformidade para Web Services que operam com dados persistentes, ou seja, dados que continuam a existir após o término da execução do Web Service que os criou. Por causa da limitação dos documentos WSDL padrão, os autores utilizam documentos WSDL-S que possuem anotações semânticas que definem entradas, saídas, pré-condições e efeitos das operações descritas na interface do serviço. As pré-condições são requisitos necessários para que uma operação esteja disponível, já o efeito representa o estado do serviço após a execução da operação.

"Um documento WSDL-S (Web Service Semântico) define mecanismos para associar anotações semânticas com os Web Services que são descritos por meio de um documento WSDL." (W3C. WEB SERVICE SEMANTIC, 2005). Em outras palavras um documento Web Service semântico é uma extensão da especificação da linguagem WSDL.

Em (SINHA; PARADKAR, 2006), os autores propõe um algoritmo para criar uma representação baseada na Máquina de Estados Finita Estendida (EFSM) a partir do documento WSDL-S de forma que seja possível representar o estado do Web Service. As pré-condições e efeitos são descritos no formato de ontologias usando a linguagem OWL via extensões SWRL. Para cada operação descrita na ontologia OWL o algoritmo extrai informações das pré-condições e efeitos definindo também um possível conjunto de entradas e saídas para cada operação. O algoritmo devolve uma Máquina de Estados Finita Estendida N onde $N = \{I, O, D, T\}$ sendo I um conjunto de entradas, O um conjunto de saídas, D um espaço n dimensional $D_1 x \ldots x D_n$ de dados associados a N e T uma relação de transação T: Q x D x $I \rightarrow Q$ x D x O onde Q é um conjunto de controle de estados.

O algoritmo de Sinha e Paradkar (2006) gera uma Máquina de Estados Finita Estendida para apenas um controle de estado, ou seja, $Q = \{q_0\}$ onde q_0 recebe n auto-looping sendo n o número de operações do Web Service. Os autores adotaram esta estratégia para assemelhar-se ao comportamento proposto pelo documento WSDL-S que não indica explicitamente uma sequência de operações.

5. FERRAMENTAS AUTOMATIZADAS PARA TESTES EM WEB SERVICES

Sem dúvida os estudos realizados por pesquisadores da área de testes de software colaboram com idéias e sugestões que auxiliam grandes empresas a produzirem ferramentas para testes automatizados. O grande benefício destas ferramentas é a geração e execução de inúmeros casos de testes proporcionando redução de custos, economia de tempo e maior controle e fluidez na execução dos testes. As ferramentas para testes automatizados podem prover uma melhoria na qualidade dos Web Services assim como para qualquer sistema de software.

A seguir uma breve descrição de algumas ferramentas para testes em Web Services disponíveis atualmente no mercado. Algumas das ferramentas citadas não utilizam as abordagens vistas neste artigo estando principalmente focadas na avaliação de performance, integração e segurança. Não é intuito desta seção fazer qualquer tipo de comparação entre as ferramentas nem descrever suas arquiteturas de funcionamento.

soapUI: O soapUI (EVIWARE, 2010) é uma das ferramentas mais conhecidas para realização de testes automatizados em Web Services. Esta ferramenta além de realizar testes funcionais realiza outros tipos de testes como teste de carga, teste de qualidade e teste de conformidade. O soapUI é uma ferramenta open source produzida pela empresa Eviware e suporta diferentes tipos de protocolos entre eles SOAP, REST e HTTP. Possui como características a inspeção e invocação de serviço, simulação de serviço (Service Mocking), monitoramento de desempenho, etc. O soapUI é uma ferramenta feita em Java sendo independente de plataforma.

SOAtest: O SOAtest (PARASOFT, 2010) da Parasoft é uma plataforma comercial de testes de ambientes SOA permitindo ao desenvolvedor criar cenários de testes para Web Services. Possui como principais características o teste funcional *End-to-End* (que valida aspectos críticos de transações complexas), virtualização do comportamento do serviço, teste de carga e performance, teste de segurança, testes de regressão, detecção de erros em tempo de execução e outras.

TestMaker: O TestMaker (PUSHTOTEST, 2010) da PushToTest é um framework open source baseado na arquitetura SOA. Este ambiente de teste permite basicamente a realização de testes de funcionalidade, teste de regressão, teste de desempenho em aplicações RIA (Rich Internet Applications) e aplicações SOA. Uma das principais características é o Cloud Testing (Teste nas nuvens) que é uma forma de testar aplicações web que usam computação nas nuvens simulando situações reais de tráfego. A ferramenta possui recurso de gravação e reprodução de testes.

Algumas das pesquisas realizadas nas abordagens de testes vistas neste artigo também contribuem com ferramentas experimentais para testes em Web Services.

6. CONSIDERAÇÕES FINAIS

O processo de teste de software é uma das atividades fundamentais para fornecer qualidade do produto. Este artigo procurou demonstrar algumas das várias abordagens existentes para testes em Web Services. Cada uma das abordagens apontadas neste artigo procura atender alguma necessidade específica e contornar alguns desafios e dificuldades propostos pela arquitetura SOA.

A abordagem de testes utilizando documentos WSDL utiliza informações básicas a respeito do Web Service. Os pesquisadores utilizam estes documentos como fonte principal

para geração de casos de testes que visam testar principalmente o acesso às operações dos serviços. Algumas ferramentas disponíveis no mercado como, por exemplo, a soapUI utilizam as informações contidas no documento WSDL como base para a geração de casos de teste.

O uso de perturbação de dados nas mensagens XML trocadas pelos Web Services permitem verificar o comportamento dos serviços ao utilizarem dados "inesperados". Esta abordagem procura verificar principalmente erros muito comuns como erros nos valores limites dos tipos de dados, falta de tratamento de dados de entrada (que costumam gerar os triviais erros do tipo exceções) entre outros.

Documentos WSDL mutantes permitem indicar se um Web Service apresenta um determinado tipo de defeito. Esta abordagem que usa análise de mutantes possui como principal objetivo verificar defeitos na interface dos serviços. Já a abordagem para testar a conformidade do Web Service possuem como principal finalidade verificar a implementação do serviço a partir de sua descrição comportamental.

A cada ano, os Web Services estão se tornando uma tecnologia promissora para softwares na Internet e para a integração entre diversas aplicações. Por isso, novas pesquisas estão sendo desenvolvidas para gerar soluções a fim de atender a necessidade de novas técnicas e ferramentas de teste para o cenário de Web Services.

Com técnicas de testes mais precisas será possível garantir que Web Services possuam um nível de qualidade cada vez mais maior**.**

REFERÊNCIAS

BAI, X.; DONG, W.; TSAI, W.; CHEN Y. **WSDL-Based Automatic Test Case Generation form Web Service Testing**. In: IEEE International Workshop on Service-Oriented System Engineering, 2005, Beijing, IEEE, 2005, p. 215–220.

CANFORA, G.; DI PENTA, M. **Testing Services and Service-Centric Systems: Challenges and Opportunities**, IT Professional, v. 8, n. 2, p. 10-17, abr. 2006.

DELAMARO, M. E.; MALDONADO, J. C.; JINO M. **Introdução ao Teste de Software –** ed. Rio de Janeiro: Campos, 2007

DEMILLO, R.A.; LIPTON, R.J. SAYWARD, F.G. **Hints on Test Data Selection: Help for the Practicing Programmer**, IEEE Computer, v. C-11, p. 34-41, abr. 1978.

EVIWARE. **soapUI** :, Disponível em: <http://www.soapui.org/>. Acesso em: 13 jul. 2010

HECKEL, R; MARIANI, L. **Automatic conformance testing of Web Service**, In: Fundamental Approaches to Software Enginneering, Springer, p. 34-48, 2005

MYERS, G. J. **The Art of Software Testing**, 2 ed., New Jersey: John Wiley & Sons, 2004

OFFUTT, J.; XU, W. **Generating Test Cases for Web Services Using Data Perturbation**, ACM SIGSOFT Software Engineering Notes, v. 29 n. 5, p. 1-10, set. 2004.

PARASOFT. **SOAtest**:, Disponível em: <http://www.parasoft.com/jsp/home.jsp> . Acesso em: 13 jul. 2010

PRESSMAN, R. S. **Engenharia de Software**, 5 ed., Rio de Janeiro: MC Graw Hill, 2002

PUSHTOTEST. **TestMaker**:, Disponível em: <http://www.pushtotest.com> . Acesso em: 6 out. 2010

SIBLINI, R; MANSOUR, N. **Testing Web Services**. In AICCSA'05: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, 2005, Washington, DC, USA, IEEE Computer Society, 2005, p. 135-vii

SINHA, A; PARADKAR, A. **Model-Based Functional Conformance Testing of Web Services Operating on Persistent Data**, In: Proceedings of TAV-WEB, ACM Press, 2006, p. 17–22

SOLINO, A. L. S.; VERGILIO, S. R.; **Mutation based testing of Web Services**, In: Test Workshop 2009. LATW '09. 10th Latin American , 2009, Búzios, IEEE, 2009, p. 1-6.

TSAI, W. T; PAUL, R.; WANG, Y.; FAN, C.; WANG, D. **Extending WSDL to Facilitate Web Services Testing**, In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, 2002, Tóquio, IEEE, 2002, p. 171-172

W3C. **OWL Ontology Web Language**, 2004, Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em: 1 out. 2010

W3C. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**, 2004, Disponível em: <http://www.w3.org/Submission/SWRL/>. Acesso em: 01 out. 2010

W3C. **WEB SERVICE GLOSSARY**, 2004, Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>. Acesso em: 04 jun. 2010

W3C. **WEB SERVICE SEMANTICS**, 2005, Disponível em: <http://www.w3.org/Submission/WSDL-S/>. Acesso em: 01 out. 2010

W3C. **WSDL**, 2001, Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em: 06 de julho de 2010

W3C. **XML SCHEMA**. Disponível em: <http://www.w3.org/standards/xml/schema>. Acesso em: 28 ago. 2010

W3SCHOOLS. **WEB SERVICE.** Disponível em: <http://www.w3schools.com/webservices/ws_intro.asp>. Acesso em: 10 jul. 2010

XU, W.; OFFUTT, J.; LUO, J. **Testing Web Services by XML Perturbation**. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, 2005, Chicago, IEEE, 2005, p. 257-266