Uma Linguagem para Testes de Aceitação de Sistemas Web

Eduardo Aranha[1]*, **Paulo Borba**[1]†

[1]Centro de Informática – Universidade Federal de Pernambuco
Caixa Postal 7851 – 50732-970 Recife, PE

{ehsa, phmb}@cin.ufpe.br

Abstract. *Although having known advantages, the activity of test is frequently ignored or only partially done. One of the main reasons of it is the inadequate support of available languages and tools for tests construction and execution. Aiming to offer a more adequate support to test programmers, this article presents the WSat language, used for description of acceptance test cases of Web systems. We also argue the execution environment constructed for it.*

Resumo. *Apesar de ter vantagens conhecidas, a atividade de teste é, com frequência, ignorada ou apenas parcialmente realizada. Um dos principais motivos para isso é o suporte inadequado das linguagens e ferramentas disponíveis para construção e execução de testes. Visando oferecer um suporte mais adequado ao programador de testes, este artigo apresenta a linguagem WSat, utilizada para descrição de casos de teste de aceitação de sistemas Web. Discutimos também o ambiente de execução construído para a mesma.*

1. Introdução

Visando desenvolver sistemas corretos e robustos, metodologias de desenvolvimento tradicionais da engenharia de software vêm enfatizando a construção e automação de testes para sistemas como os desenvolvidos para Web [Nguyen, 2000]. Processos e metodologias práticas mais recentes como RUP [Jacobson, 1999] e *Extreme Programming* (XP) [Beck, 1999], por exemplo, tem destacado a atividade de teste (em particular, os testes de aceitação e de unidade) como uma das práticas chave para o sucesso de sua utilização. No caso de XP, a construção destes testes é realizada antes mesmo da implementação do código a ser testado (*Test First Design*) [Feathers, 2000].

Apesar de ter suas vantagens reconhecidas, a atividade de teste é, com frequência, ignorada ou apenas parcialmente realizada, principalmente quando deixada para o final do desenvolvimento. Um dos principais motivos para isso é o suporte inadequado das linguagens e ferramentas disponíveis para construção e execução de testes. É comum, por exemplo, a utilização de linguagens de programação convencionais para a construção destes testes. Estas linguagens são bastante poderosas, mas seu uso causa uma dificuldade desnecessária na programação dos testes [Liu and Richardson, 1999a]. De fato, elas foram desenvolvidas para implementar funcionalidades complexas, não provendo mecanismos naturais para a definição de testes.

*Suportado parcialmente pelo IPAD.
†Suportado parcialmente pela CNPq, processo 521994/96-9.

Já as ferramentas de testes disponíveis fornecem, em geral, linguagens de teste de difícil aprendizado e programação devido ao seu baixo nível de abstração. No caso de testes de aceitação, onde verifica-se se os requisitos funcionais e de GUI (*Graphical User Interface*) são satisfeitos, não podemos indicar explicitamente a estrutura da GUI do sistema testado. Isto pode dificultar, por exemplo, a identificação das partes da GUI sendo testada. Algumas ferramentas fornecem a possibilidade de gravar os casos de teste a partir da execução do sistema testado. Este recurso facilita bastante a criação dos casos de teste. Entretanto, os testes são gravados em linguagens com pouca flexibilidade para eventuais manutenções nos casos de teste e no sistema testado.

Com o objetivo de oferecer um melhor suporte para a realização de testes de sistemas Web, em particular os testes de aceitação, este artigo apresenta a linguagem de teste WSat (*Web System Acceptance Test*). Esta linguagem é utilizada para descrição de casos de teste de aceitação e possui um alto nível de abstração e reuso. Para isto, WSat permite definir tipos que representam componentes Web a serem testados. Para possibilitar a utilização desta linguagem na prática, apresentamos um ambiente de execução desenvolvido para a mesma.

Primeiro apresentamos as construções de WSat, progressivamente, através de um exemplo de programa de teste simples, porém completo e suficiente para o entendimento da linguagem. Depois discutimos a arquitetura do ambiente de execução desenvolvido para WSat e mostramos como certas construções de WSat são compiladas para Java. Por fim, apresentamos as conclusões obtidas sobre o uso de WSat, assim como mostramos trabalhos relacionados e sugestões para trabalhos futuros.

2. A linguagem WSat

A GUI é a interface utilizada para a interação entre o sistema e seus usuários, sendo a mesma responsável por apresentar aos usuários as informações geradas pelo sistema. Desta forma, as principais características que guiaram a definição de WSat foram a abstração, o reuso e a capacidade de expressar explicitamente a estrutura da GUI dos sistemas testados. Para isto, WSat permite a definição de tipos que representam os tipos de entidades a serem testadas. Estas entidades são componentes Web como páginas Web, formulários HTML, imagens e *links*. Além disto, definimos casos de teste para descrever o comportamento do sistema. Estes casos de teste utilizam as propriedades e serviços definidos nos tipos WSat para simular e testar o uso do sistema.

Nesta seção, apresentamos as principais estruturas e comandos utilizados na construção de programas WSat. Nos exemplos mostrados, fazemos uso de um sistema de busca na Web simples, cujas telas principal e de resposta são vistas na Figura 1.

Como podemos ver, a tela principal do sistema de busca possui um formulário HTML de busca que, por sua vez, possui um parâmetro chamado `palavras` e um botão para sua submissão. A seguir mostramos como exemplo parte do código HTML da tela principal do sistema.

```
<html>
    <head>
        <title>Sistema de Busca</title>
```

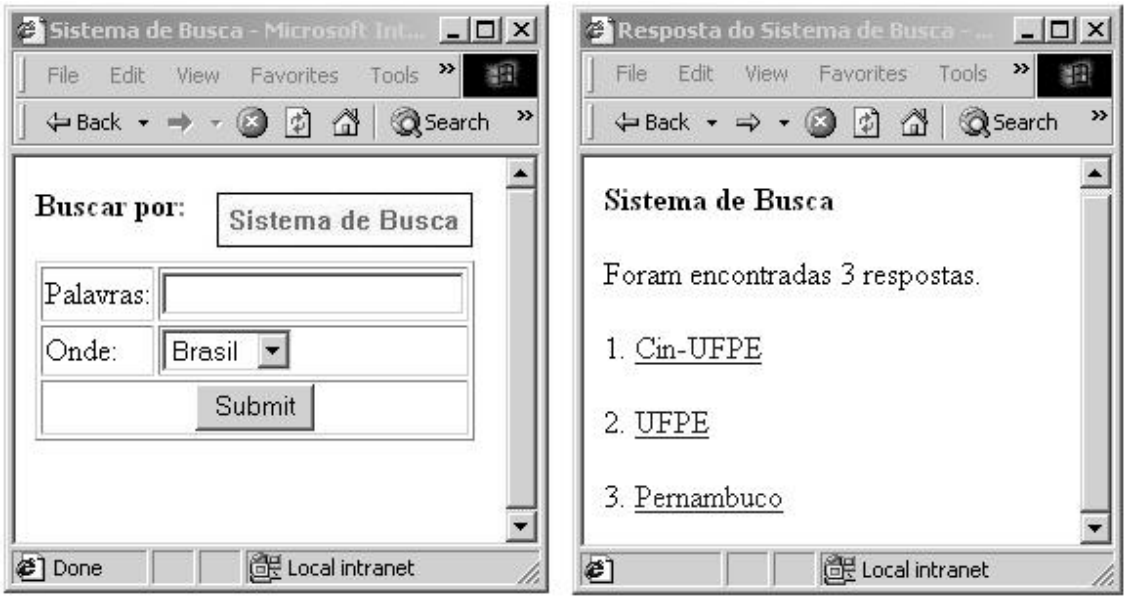

Figura 1: Telas principal e de resposta do sistema de busca.

```
    </head>
    <body>
        Buscar por:
        <form name="formBusca" method="POST"
              action="pagResposta.jsp">
            <input type="text" name="palavras"
                               value="">
            ...
        </form>
    </body>
</html>
```

Como veremos mais adiante, para programar os testes faz-se necessário o acesso ao código HTML fonte das páginas Web do sistema a ser testado.

A seguir apresentamos as construções que permitem descrever a estrutura da GUI do sistema a ser testado, a parte estática de um programa WSat. Depois mostramos as construções da parte dinâmica de WSat, responsável por simular a execução do sistema e testar o conteúdo dinâmico gerado pelo mesmo.

2.1. Descrevendo a Estrutura da GUI do Sistema

Em WSat, a estrutura desejada da GUI de um sistema Web é descrita através da definição de tipos. Cada tipo WSat definido denota um conjunto de componentes Web (como páginas Web, formulários HTML, etc.) que satisfazem determinadas características. Estas características são determinadas pelas propriedades dos tipos, como veremos mais adiante. Podemos então verificar se um determinado componente Web pertence ao conjunto denotado por este tipo comparando as características deste componente com as propriedades do tipo WSat. Tipos também auxiliam, como mostrado mais adiante, a simular o uso do sistema e assim testar a funcionalidade e GUI do mesmo.

Novos tipos WSat são definidos a partir de tipos especiais existentes na linguagem, denotando conjuntos de componentes Web específicos. Por exemplo, os tipos `WebPage` e `WebImage` denotam, respectivamente, o conjunto de todas as páginas Web e o conjunto de todas as imagens. Já o tipo `EditBox` representa campos de texto de formulários HTML, correspondendo ao parâmetro `text` de HTML [Powell, 2000]. Assim, para testar o espaço para preenchimento de palavras a serem procuradas pelo sistema de busca a partir da página vista na Figura 1, podemos definir o tipo `Palavras`:

```
EditBox Palavras {
    name = "palavras";
    value = "";
}
```

Propriedades de um tipo WSat definem as restrições sobre o conjunto denotado por este tipo. Na definição do tipo `Palavras`, por exemplo, podemos ver o uso das propriedades `name` e `value` herdadas do tipo `EditBox`, supertipo de `Palavras`. Estas propriedades indicam respectivamente o nome e o valor deste parâmetro de formulário.

Caso a definição de um tipo WSat não atribua um valor a uma determinada propriedade herdada, esta propriedade não é utilizada para restringir os componentes que pertencem ao conjunto denotado por este tipo. No caso do tipo `Palavras`, por exemplo, a propriedade `maxLength` herdada de `EditBox` indica o tamanho máximo que o valor do parâmetro pode ter. Como esta propriedade não está sendo inicializada pelo tipo `Palavras`, seu valor não é utilizada para restringir os componentes que pertencem ao conjunto denotado por este tipo.

Componentes Web podem naturalmente conter outros componentes. Páginas Web, por exemplo, podem conter *links*, textos, imagens, entre outros. Em WSat, representamos este relacionamento através da definição de propriedades. Para testar o formulário do sistema de busca, por exemplo, podemos definir o tipo `FormBusca` da página inicial:

```
WebForm FormBusca {
    name = "formBusca";
    action = "pagResposta.jsp";
    method = "POST";
    Palavras palavras;  ...
}
```

Note que formulários HTML possuem, entre outras, propriedades que indicam o nome do formulário, a URL para submissão dos seus parâmetros e o método de envio do protocolo HTTP. Além disto, o tipo `Palavras` foi utilizado para definir uma nova propriedade de nome `palavras`, no tipo `FormBusca`. Esta propriedade indica que os componentes denotados pelo tipo `FormBusca` possuem um outro componente que pertence ao conjunto denotado pelo tipo `Palavras`. De fato, `FormBusca` representa formulários HTML que, além de outras coisas, possuem um campo texto com as características definidas pelas propriedades do tipo `Palavras`.

Um dos principais componentes de sistemas Web, as páginas Web são utilizadas para apresentar toda GUI do sistema através dos navegadores Web. Para testar as páginas Web de um sistema através de programas WSat, definimos subtipos de `WebPage`. Desta

forma, para testar as características que a página inicial do sistema de busca deve satisfazer, definimos o tipo `PagInicial` como visto a seguir:

```
WebPage PagInicial {
    code = 200;
    FormBusca formBusca;    ...
}
```

onde a propriedade `code`, herdada do tipo `WebPage`, registra o código de retorno da página Web definido pelo protocolo HTTP [Consortium, 2000]. Este código indica, por exemplo, se a página foi retornada com sucesso (código `200`) ou não foi encontrada (código `404`) pelo servidor Web. Desta forma, o tipo definido denota o conjunto de todas as páginas Web que, entre outras coisas, possui código de retorno `200` e um formulário com as características definidas pelo tipo `FormBusca`.

Como podemos notar, através da definição de tipos WSat, é possível representar a estrutura estática da GUI do sistema Web testado. Em outras linguagens como as de programação, por exemplo, a estrutura da GUI não seria representada explicitamente reduzindo, assim, o nível de abstração dos programas. Arquivos HTML, assim como os tipos WSat, representam a estrutura da GUI. Entretanto, o código escrito para os tipos WSat é mais abstrato e é utilizado, como mostramos na Seção 2.2., para simular a execução do sistema Web. A definição de tipos WSat que denotam outros tipos de componentes Web como por exemplo imagens, textos e *links* é feita de forma similar, a partir de supertipos como `WebImage`, `Text` e `WebLink`.

Programas WSat podem testar propriedades de componentes Web invalidadas por pequenas mudanças nas páginas Web do sistema. Um exemplo disto é o teste dos nomes dos parâmetros de formulários HTML. Entretanto, a mudança de informações como o nome de parâmetros de formulários implicam em modificações não só dos programas de teste como também do código do sistema. Já os trechos de código HTML sobre a programação visual, frequentemente modificados nas páginas Web sem implicar no funcionamento incorreto do sistema, não são testados por programas WSat.

Além do alto nível de abstração alcançado, a definição de tipos em WSat possibilita que parte do código de programas WSat e do próprio sistema a ser testado sejam gerados automaticamente. Implementamos, por exemplo, um gerador de código de teste que gera, a partir de protótipos de tela HTML do sistema, o esqueleto dos tipos WSat necessários para testar os componentes Web do sistema. Também implementamos um gerador de código de sistema capaz de gerar um Servlet [Hunter and Crawford, 2001] para cada subtipo de `WebPage` definido para testar uma página Web do sistema.

2.2. Descrevendo o Comportamento do Sistema

A partir da representação em WSat da estrutura desejada da GUI do sistema, é possível definir casos de teste para verificar se um determinado sistema satisfaz tal estrutura. Para verificar esta estrutura, é necessário que os casos de teste explorem o sistema testado através da simulação da interação entre o sistema e seus usuários. Desta forma, os casos de teste acabam por validar não só a estrutura da GUI de um sistema como também o seu comportamento. Durante esta seção, são mostrados os comandos WSat que podem ser usados na definição de casos de teste.

2.2.1. Variáveis

Em WSat, variáveis podem ser declaradas como de um tipo WSat, pré-definidos ou não, do tipo `String` ou dos tipos primitivos suportados pela linguagem Java [Goslin and Steele, 2000]. Podemos também ter declarações de variáveis como `arrays` dos tipos citados. Suas declarações são feitas como em Java:

```
String url = "http://www.sistemadebusca.com.br/";
```

2.2.2. Comando `getWebPage`

Para a realização da operação de requisição de uma página Web a partir de sua URL, podemos utilizar o comando `getWebPage`. Este comando recebe como parâmetro uma `String` representando a URL da página desejada e seu valor de retorno é um componente do tipo `WebPage`, representando a página de URL indicada. A seguir podemos ver um exemplo de sua utilização:

```
WebPage indexPage = getWebPage(url);
```

onde utilizamos o comando `getWebPage` para acessar o componente Web que representa a página inicial do sistema de busca cuja URL está armazenada na variável `url`.

2.2.3. Operador de Transformação de Tipo

Para realizar, por exemplo, o teste de que a página de acesso do sistema de busca referenciada pela variável `indexPage` possui as características definidas pelo tipo `PagInicial`, fazemos uso da operação de transformação de tipo como visto a seguir.

```
PagInicial pag = [PagInicial] indexPage;
```

Esta operação verifica se a página Web referenciada pela variável `indexPage` tem código de retorno `200` e possui um formulário de busca. Podemos interpretar a operação de transformação de tipo de um componente Web como a verificação de que este componente pertence ao conjunto denotado pelo tipo final da transformação. Uma vez realizada a transformação de tipo, o componente testado pode ser manipulado pelo programa WSat a partir de variáveis do tipo final da transformação.

Os tipos de componentes envolvidos na operação de transformação de tipo devem ser tipos que representem componentes Web. Além disto, o tipo atual do componente a ser transformado deve ser um supertipo do tipo final da transformação. No exemplo mostrado anteriormente, por exemplo, o tipo atual do componente é `WebPage` e o tipo final é seu subtipo `PagInicial`. Durante a transformação de tipo, as propriedades definidas pelo tipo final são verificadas no componente Web operado. Caso alguma propriedade verificada não seja satisfeita por este componente, o caso de teste em execução é interrompido por motivo de falha na execução, ou seja, o teste realizado não foi satisfeito.

2.2.4. Acesso a Propriedades e Serviços de Componentes Web

Através de variáveis, podemos acessar as propriedades e serviços de seus componentes referenciados. Por exemplo, podemos acessar os parâmetros do formulário de busca da página inicial do sistema através da variável `pag`, como mostrado no código a seguir:

```
FormBusca form = pag.formBusca;
form.palavras.value = "ufpe";
WebPage resp = form.submit();
```

No segundo comando, podemos ver a atribuição de um valor ao parâmetro `palavras` do formulário de busca através de uma sequência de acessos a propriedades, começando a partir da propriedade `formBusca`, definida no tipo `PagInicial` e que representa o formulário de busca. Por fim, podemos ver a invocação do serviço `submit` do componente que representa o formulário de busca, referenciado pela propriedade `busca` e acessada através da variável `pag`. Serviços de componentes Web são utilizados, em geral, para a simulação da execução do sistema testado. Desta forma, o código mostrado simula o preenchimento e submissão do formulário de busca, armazenando na variável `resp` o resultado da busca.

2.2.5. Teste do Conteúdo Dinâmico de Componentes Web

Como visto na Seção 2.1., os tipos WSat são utilizados para descrever propriedades que componentes Web devem satisfazer. Quando os componentes Web são estáticos, como páginas HTML ou imagens escritas em arquivos estáticos, é possível descrever detalhadamente o conteúdo que eles possuem em tipos WSat. No caso de componentes dinâmicos como por exemplo páginas Web geradas a partir de Servlets ou *scripts* Perl, partes importantes de seu conteúdo são criadas dinamicamente e não podem ser testados de forma eficiente com os recursos vistos até agora. De fato, uma mesma página Web dinâmica pode apresentar informações diferentes dependendo dos valores de entrada passados para a mesma. Um exemplo disto é a página de resposta do sistema de busca, que apresentam *links* para outras páginas dependendo das palavras chaves utilizadas na busca.

Para testar o conteúdo de páginas dinâmicas como a de resposta do sistema de busca, WSat fornece serviços especiais para testes em componentes Web dinâmicos. Podemos, por exemplo, acessar o primeiro *link* retornado pelo sistema de busca:

```
WebLink link = resp.findWebLinkByName("resp1");
```

O código mostrado utiliza o serviço `findWebLinkByName`, definido no tipo `WebPage`, para testar e acessar o componente *link* de nome "`resp1`". Este exemplo considera que os *links* retornados pelo sistema de busca possuem nomes como `resp1`, `resp2`, `resp3`, etc.

De fato, os tipos predefinidos em WSat como o tipo `WebPage` e outros, definem serviços similares aos mostrados. Estes serviços procuram componentes Web através dos valores de suas propriedades. Exemplos destes são os serviços `findWebImageByName` e `findWebLinkByTarget` que procuram, respectivamente, componentes imagem a partir do seu nome e componentes *link* a partir do seu *target*.

Uma vez com o acesso ao componente *link*, podemos utilizar o serviço `click`, definido pelo tipo `WebLink` e que dá acesso à página Web indicada na resposta do sistema:

```
WebPage resp1 = link.click();
```

Podemos então utilizar o comando `assert` para verificar se a página referenciada pela variável `resp1` possui o código de retorno `200`. Este código, visto a seguir, testa se o *link* de resposta analisado não está quebrado.

```
assert("Link de resposta quebrado.", resp1.code, 200);
```

Como podemos notar, este comando recebe três parâmetros. O primeiro deles, do tipo `String`, é uma mensagem a ser utilizada no caso de falha do teste. Os parâmetros seguintes representam respectivamente o valor esperado e o valor encontrado de uma propriedade testada. Outra forma de realizar o mesmo teste é através do comando `fail`, como visto no código a seguir.

```
if (paginaEncontrada.codigo != 200) {
    fail("Link de resposta quebrado.");
}
```

O comando `fail` causa uma falha na execução do caso de teste. Este comando recebe como parâmetro uma `String` contendo o motivo da falha de execução. O comando `if` mostrado possui em WSat a mesma sintaxe e semântica que a definida em Java.

Para a realização de testes mais elaborados, podemos fazer uso de outros serviços e propriedades definidas nos tipos pré-definidos de WSat. Por estar fora do escopo deste artigo, a lista exaustiva de serviços e propriedades existentes não é aqui apresentada.

2.2.6. Casos de Testes

A execução de programas WSat é iniciada a partir de casos de teste. Para a definição de casos de teste em WSat, fazemos uso da cláusula `testCase`. Para testar o sistema de busca, por exemplo, definimos o caso de teste `buscar` como visto no código a seguir.

```
testCase buscar {
    String url = "http://www.sistemadebusca.com.br/";
    WebPage indexPage = getWebPage(url);  ...
}
```

O corpo da definição de um caso de teste é composto pelos comandos WSat que implementam o teste. O caso de teste `buscar`, por exemplo, é formado pelos comandos mostrados ao longo desta seção.

Além de casos de teste, programas WSat podem ser estruturados através de funções de teste. Estas funções recebem parâmetros e possuem retorno, facilitando assim o reuso de código. Como a sintaxe e semântica das mesmas é similar a de métodos estáticos em Java, omitimos aqui os seus detalhes.

2.2.7. A Gramática de WSat

Visando um entendimento melhor de WSat, apresentamos a seguir a sua gramática resumida, mostrando algumas de suas principais construções.

```
WSatProgram = (TestComponentDec)+
TestComponentDec = WSatTypeDec | TestCaseDec | ...
WSatTypeDec = WebPageDec | WebFormDec | ...
TestCaseDec = "testCase" <IDENTIF> "{" Statement* "}"
Statement = AssertStat | WebServiceCall | ...
...
```

Como podemos observar, um programa WSat é formado por declarações de componentes de teste como tipos WSat e casos de teste. Os tipos WSat declarados podem ser subtipos de tipos como `WebPage` e `WebForm`. Casos de teste, por sua vez, são formados por comandos como os de atribuição e invocação de serviços Web.

3. Ambiente de Execução

Para a implementação de WSat, optamos pela compilação de programas WSat para Java. A motivação principal desta abordagem é compilar trivialmente os comandos e expressões de WSat que têm similares na linguagem de programação Java, como atribuições, declarações de variáveis e de tipos. Com isto, a execução destes códigos é implementada pelo próprio ambiente de execução Java, facilitando a nossa implementação, reusando recursos disponíveis em Java e aproveitando a portabilidade de Java. Esta abordagem foi escolhida por se mostrar a mais viável, já que o esforço necessário para implementação do ambiente de execução de WSat através desta técnica é menor do que o esforço encontrado na implementação de um interpretador completo.

Os comandos e expressões de WSat específicos para a realização de testes de sistemas Web, como por exemplo a requisição de páginas Web e a simulação de cliques em *links* HTML, não podem ser copiados diretamente para código Java, já que não são suportados diretamente pela linguagem. Entretanto, a tradução destes comandos pelo ambiente de execução de WSat pode gerar código Java que use APIs externas e que implementam estas funcionalidades.

3.1. Um *Parser* para WSat

O primeiro passo realizado para a implementação do ambiente de execução de WSat foi a criação de um *parser* para a linguagem. Este *parser* gera uma árvore sintática que representa um dado programa WSat. Para a implementação do parser, foi utilizada a ferramenta para geração de compiladores JavaCC [Microsystems, 2000] com o pré-processador JJTree que acompanha a ferramenta JavaCC. Este pré-processador permite definir mais facilmente os nós da árvore sintática a serem criados para cada termo da gramática da linguagem.

Todas as classes que formam a árvore sintática de WSat herdam da classe `TestComponent`. Esta classe estrutura os nós da árvore seguindo o padrão de projeto *Visitor* [Gamma et al., 1994] que permite separar em diferentes classes (*visitors*) os códigos das operações que manipulam (visitam) as árvores sintáticas. Nas classes da árvore em si, ficam as operações que implementam o comportamento básico de uma árvore sintática. Esta separação é muito importante para a implementação do *parser* e de trabalhos relacionados a este, considerando o grande número de operações definidas que atuam sobre a árvore sintática de WSat.

Algumas das funcionalidades deste ambiente de execução não foram totalmente implementadas. A análise semântica da linguagem, por exemplo, é feita de forma parcial pela linguagem destino do compilador implementado. Entretanto, isto não foi prejudicial à pesquisa por não influenciar na validação de WSat. Além disto, estas atividades já são bem conhecidas e estudadas pela comunidade científica.

3.2. Compilando Código WSat para Código Java

Para a implementação do compilador de WSat para Java, foi criado um *visitor* que percorre a árvore sintática de um programa WSat criando classes Java que implementam os casos de teste encontrados neste programa. Inicialmente são gerados os arquivos de extensão "`.java`" com código Java. Em seguida, estes arquivos são compilados pelo próprio ambiente de Java para arquivos de extensão "`.class`" com *bytecodes* executáveis [Goslin and Steele, 2000]. Nesta seção, apresentamos a geração do código Java a partir das principais construções de WSat.

Assumimos que, para a compilação de comandos WSat que possuem sintaxe e semântica equivalentes em Java como, por exemplo, atribuições, declarações de variáveis e chamadas de métodos, podemos fazer simplesmente uma cópia do código WSat para o código Java.

3.2.1. Tipos Definidos em WSat

Em Java, representamos os tipos WSat definidos através de classes Java. Desta forma, para cada definição de tipo WSat é criada uma classe Java de mesmo nome. A compilação do tipo WSat `Palavras` cria a classe Java `Palavras` mostrada a seguir:

```
public class Palavras extends EditBox {
    ...
}
```

Como podemos ver, a classe Java gerada herda da classe Java `EditBox` que representa o tipo WSat `EditBox`, supertipo do tipo WSat `Palavras`.

As propriedades dos tipos WSat são representadas por atributos públicos de mesmo nome e tipo em suas respectivas classes Java. Já os serviços dos tipos WSat são representados por métodos públicos na classe Java. Isto facilita a compilação de comandos e expressões WSat. Por exemplo, o acesso a propriedades e a invocação de serviços são simplesmente copiados para o código Java.

O código de inicialização das propriedades de tipos primitivos ou `String` definidos pelos tipos especiais predefinidos em WSat são compilados gerando-se o método `initExpectedPropertiesValues`. A seguir podemos ver a classe Java gerada para o tipo WSat `Palavras` com a definição deste método de inicialização.

```
public class Palavras extends EditBox {
    protected void initExpectedPropertiesValues() {
        name = "palavras";
        value = "";
    } ...
}
```

Note que o código de inicialização das propriedades herdadas do tipo WSat `EditBox` são copiados para o corpo do método `initExpectedPropertiesValues`. Este método é invocado durante a execução do construtor desta classe.

Já durante a compilação de tipos WSat definidos pelo testador que possuem novas propriedades declaradas, estas propriedades são representadas através de atributos públicos de mesmo nome e tipo nas classes Java geradas para estes tipos. A seguir, vemos a classe Java gerada pela declaração do tipo WSat `FormBusca`:

```
public class FormBusca extends WebForm {
    public Palavras palavras;   ...
}
```

A inicialização destes atributos é realizada por comandos WSat como, por exemplo, a transformação de tipo.

3.2.2. Comando `getWebPage`

Para compilar comandos `getWebPage` para Java, faz-se necessário a implementação em Java da funcionalidade deste comando. Para isto, utilizamos a API HttpUnit [Gold, 2000] que fornece a implementação necessária para a simulação das ações realizadas por usuários de sistemas Web. Para estruturar o código utilizado para implementar o comando `getWebPage`, definimos a classe `getWebPage` como visto a seguir:

```
public class getWebPage {
    public static WebPage execute(String url) {
        WebPage pag;   ...
        return pag;
    }
}
```

Os detalhes sobre a implementação deste método são aqui omitidos devido ao curto espaço disponível. A implementação de outros comandos para execução dos testes como, por exemplo, a operação de transformação de tipo, também foram omitidas aqui pelo mesmo motivo. Através da implementação em Java apresentada para o comando `getWebPage`, invocações deste comando podem ser traduzidas como mostrado no código a seguir:

```
getWebPage(url)             // em WSat
getWebPage.execute(url)     // em Java
```

De fato, invocações do comando WSat `getWebPage` são compilados simplesmente para invocações do método `execute` da classe Java `getWebPage`.

3.2.3. Casos de Teste

A compilação de um caso de teste WSat para Java gera uma classe Java de nome igual a do caso de teste. A seguir mostramos a classe gerada para a definição do caso de teste `buscar`, visto anteriomente:

```
public class buscar {
    public static void main(String[] args) {
        String url = "http://www.sistbusca.com.br/";
        WebPage indexPage = getWebPage.execute(url);
        ...
    }
}
```

Como podemos ver, as classes geradas a partir de definições de casos de testes possuem o método `main`. Este método, com a assinatura vista, é utilizado em Java para iniciar a execução de um programa Java. Desta forma, podemos iniciar a execução dos casos de teste de forma independente, inclusive com relação a máquina virtual Java utilizada.

Embora não mostrada aqui por questões de espaço limitado, a operação de transformação de tipo foi a construção de WSat que requisitou o maior esforço de implementação. Esta operação é responsável por instanciar a classes Java que representam o tipo WSat final da operação. Durante a instanciação, as informações sobre o componente Web são armazenadas em um contexto. Estas informações são representadas por classes da API HttpUnit que dão acesso, por exemplo, ao código fonte HTML da página Web em que o componente se encontra.

Ao se criar um objeto que represente um componente Web, suas características acessadas através do seu contexto são testadas de acordo com as propriedades do seu supertipo WSat. Entretanto, ao se declarar um novo tipo WSat, podemos definir novas propriedades representando a existência de subcomponentes. Para cada uma destas propriedades, um novo objeto é criado e testado a partir de um subcontexto do contexto original.

4. Conclusões

A principal contribuição deste trabalho é a definição de uma nova linguagem para descrição de casos de teste de aceitação de sistemas Web. Contrastando com outras abordagens, esta linguagem permite a escrita de programas de teste com um alto nível de abstração através da definição de tipos que representam componentes Web.

Outras linguagens de teste como, por exemplo, as fornecidas pelas ferramentas JXWeb [la Forge, 2002] e Canoo Web Test [Engineering, 2002], são baseadas em XML [McLaughlin, 2000]. O uso de XML facilita o seu aprendizado devido ao grande número de conhecedores desta linguagem, mas a sintax de XML prejudica a legibilidade do código. Já a linguagem TestTalk [Liu and Richardson, 1999b] visa a portabilidade dos testes, sendo independente de linguagens de programação e de ambiente de desenvolvimento. Entretanto, a definição de tipos para representar a estrutura da GUI dos sistemas testados e a manipulação destes tipos para simular o comportamento do sistema permite um nível de abstração em programas WSat superior aos alcançados por estas outras linguagens. Por outro lado, a definição de tipos WSat causa, em geral, um aumento no número de linhas de código de teste escritas.

O ambiente foi implementado em Java através da compilação de código WSat para código Java, o que facilitou bastante a implementação e trouxe a portabilidade de Java.

Além disto, foi possível utilizar a API HttpUnit para implementar construções específicas de teste. Para validar a definição de WSat, implementamos um ambiente de execução para a linguagem, usando técnicas de compilação conhecidas e usando padrões de projeto para a estruturação e manutenção de árvores sintáticas.

Com a definição de WSat, foi possível implementar dois geradores de código que visam diminuir o impacto causado na produtividade, a curto prazo, pela programação dos testes de aceitação. O primeiro gerador é utilizado para gerar código WSat a partir de protótipos de telas HTML, criados geralmente para validar os requisitos elicitados. A partir do código HTML destes protótipos, é possível extrair informações sobre a estrutura estática do sistema testado. Estas informações são utilizadas para criar a definição inicial dos tipos WSat que são usados para testar o sistema. Assim, reduz-se boa parte do esforço de definição dos tipos, podendo o programador de teste se concentrar na parte dinâmica do sistema.

O segundo gerador implementado é utilizado para gerar, entre outras coisas, trechos de código Java do sistema testado a partir de programas WSat. Este gerador é capaz de gerar, por exemplo, *Servlets* [Hunter and Crawford, 2001] para atender as requisições Web, seus arquivos de configuração e os do servidor Web. Além de aumentar a produtividade, estes geradores de código fornecem suporte à prática *Test First Design*, onde a criação dos testes é feita antes mesmo da implementação do próprio sistema a ser testado. Esta prática é utilizada, por exemplo, pela metodologia *Extreme Programming*.

Foi realizado um estudo de caso para validar a definição de WSat e dos geradores de código citados anteriormente. Foram construídos programas WSat para um sistema Web real de pequeno porte, com operações básicas de cadastro e de consulta. Dois programadores experientes pertencentes a equipe de desenvolvimento do sistema Web testado participaram do estudo de caso e foram acompanhados durante a criação dos programas WSat. Foi notado através destes programadores que a linguagem WSat é de fácil aprendizado, possuindo as características objetivadas, o alto nível de abstração e de reuso. Defeitos no sistema foram detectados pelos testes, mostrando a eficácia da linguagem. A estensibilidade da linguagem também foi validada através da estensão dos casos de testes criados. Observamos também que o número de tipos WSat definidos tende a crescer dependendo do nível de detalhes da GUI testados, fato amenizado pelo uso do gerador de código de teste WSat. Por fim, a facilidade de programação da linguagem, assim como o uso dos geradores, mostraram contribuir para o aumento da produtividade do desenvolvimento.

Como trabalhos futuros, sugerimos estender WSat com mecanismos com alto nível de abstração para tarefas comuns na criação dos testes como, por exemplo, a sua parametrização, o acesso a banco de dados e a verificação da performance e da execução concorrente do sistema testado. Atualmente estas tarefas são implementadas através da utilização de código Java embutido em programas WSat. A possibilidade de embutir código Java nos permite utilizar todo o poder da linguagem Java, mas o uso deste tipo de código prejudica a legibilidade e abstração da linguagem.

Agradecimentos

Os autores agradecem os comentários recebidos dos julgadores deste artigo, o que contribuiu para melhorar a qualidade deste trabalho.

Referências

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Consortium, W. W. W. (2000). HTTP - Hypertext Transfer Protocol. http://www.w3.org/Protocols.

Engineering, C. (2002). Canoo WebTest. http://webtest.canoo.com.

Feathers, M. (2000). Test First Design. http://www.xprogramming.com/xpmag/test_first_intro.htm.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gold, R. (2000). HttpUnit. http://httpunit.sourceforge.net.

Goslin, B. and Steele, G. (2000). *The Java Language Specification*. Addison-Wesley, second edition.

Hunter, J. and Crawford, W. (2001). *Java Servlet Programming*. O'Reilly & Associates, second edition.

Jacobson, I. (1999). *The Unified Software Development Process*. Addison–Wesley, first edition.

la Forge, B. (2002). JXWeb. http://sourceforge.net/projects/jxweb.

Liu, C. and Richardson, D. J. (1999a). Programming Languages Considered Harmful in Writing Automated Software Tests. Technical Report 99-09, Information & Computer Science, University of California, Irvine.

Liu, C. and Richardson, D. J. (1999b). TestTalk: A Comprehensive Testing Language. In *the 14th IEEE International Conference on Automated Software Engineering, Doctoral Symposium*, Cocoa Beach, Florida, USA.

McLaughlin, B. (2000). *Java and XML*. O'Reilly & Associates.

Microsystems, S. (2000). Java Compiler Compiler (JavaCC) - The Java Parser Generator. http://www.webgain.com/products/java_cc.

Nguyen (2000). *Testing Applications on the Web: Planning for Internet Based Systems*. John Wiley.

Powell, T. (2000). *HTML: The Complete Reference*. McGraw Hill, third edition.