FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ LUIZ VIEIRA DE OLIVEIRA

RECORD E PLAYBACK PARA APLICAÇÕES WEB

MARÍLIA
2004

JOSÉ LUIZ VIEIRA DE OLIVEIRA

RECORD E PLAYBACK PARA APLICAÇÕES WEB

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação.

Orientador:
Prof. Dr. Márcio Eduardo Delamaro

MARÍLIA
2004

JOSÉ LUIZ VIEIRA DE OLIVEIRA

RECORD E PLAYBACK PARA APLICAÇÕES WEB

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM/F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação.

Resultado: APROVADO

ORIENTADOR: Prof. Dr. Márcio Eduardo Delamaro

1º EXAMINADOR: Prof. Dr. Edmundo Sérgio Spoto

2º EXAMINADOR: Profa. Dra. Simone do Rocio Senger de Souza

Marília, 28 de outubro de 2004.

DEDICATÓRIA

É muito difícil chegar em casa e não te encontrar. É impossível entrar em casa e não lembrar de você. É difícil não ter sua recepção. A falta que você faz dói muito.

Tenho certeza que se você ainda estivesse por aqui, esse tempo de turbulências seria mais fácil de agüentar.

Infelizmente você não esta mais aqui com a gente. E isso dói.

Fico lisonjeado de ter convivido com alguém tão especial como você. Com certeza, você nos ensinou muita coisa. Sua presença nos engrandecia e enaltecia. Você foi e ainda é única.

Desculpe por não poder ter lhe ajudado mais. Gostaria de ter feito mais. Você merecia mais.

Como eu prometi na monografia de graduação, aquele, esse e todos os outros trabalhos e conquistas que ainda estão por vir vou dedicar a você, pois com certeza, você estará sempre presente, mesmo que seja em pensamento e na saudade...

E que saudade...

Saudade de você minha fiel amiga e eterna companheira ***Jolie****.*

AGRADECIMENTOS

Aos meus pais, Irineu e Ester, pelo apoio, estrutura e incentivo;

À Raquel, pela companhia, incentivo e apoio, e também pela paciência por agüentar nesse período de estudos um noivo muitas vezes com pensamento distante, irritado e nervoso;

À minha tia Iza, pela ajuda no inicio do curso;

À Java, pela constante companhia e alegria;

Ao amigo Gustavo Habermann, que muito ajudou no desenvolvimento desse trabalho, sempre disposto a colaborar, meu muito obrigado;

Aos amigos do LOST, Rodrigo Araújo e Gustavo Rondina, pela força e apoio;

Aos amigos e colegas de estudo, entre eles Elvio, Kelton, Luis Alexandre, Fernando, Plínio e tantos outros, pela força, companhia e sempre pronta ajuda;

Aos colegas professores, funcionários e estagiários do laboratório de informática da FAI, pela ajuda quando necessária;

Aos professores e secretarias do Mestrado em Ciência da Computação da Univem;

e

Ao professor Márcio Delamaro, pela orientação e grande ajuda no desenvolvimento desse trabalho.

Obrigado a todos! ☺

OLIVEIRA, José Luiz Vieira de. **Record e Playback para Aplicações WEB**. 2004. 101f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

RESUMO

Com o surgimento e a rápida expansão da Internet, houve a necessidade de adaptar o desenvolvimento de sistemas para que o seu uso aproveitasse os recursos e facilidades do ambiente WWW, como a arquitetura cliente/servidor, protocolos, servidores proxies, entre outros. Surgiu assim, uma nova categoria de software: as aplicações WEB. Com o avanço do processo de desenvolvimento dessas aplicações voltadas para o ambiente WEB e a necessidade de sempre mantê-las atualizadas, novas técnicas e ferramentas para o teste e controle de qualidade dessas aplicações surgiram para auxiliar e automatizar o processo de teste. Neste cenário, uma ferramenta de auxílio à atividade de teste para aplicações WEB é apresentada. Seu objetivo é implementar, através de um servidor proxy facilidades de “Recording & PlayingBack” para as aplicações WEB.

Palavras-Chave: Teste de Software – Proxy – Aplicações WEB

OLIVEIRA, José Luiz Vieira de. **Record e Playback para Aplicações WEB**. 2004. 101f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

ABSTRACT

With the appearance and the fast expansion of the internet, there was necessity to adapt the system's development to make good use of the resource and facility of the WWW, like the construction client/server, protocols, proxies server and others. Then appeared the new software category: WEB applications. With the progress of the development process of these applications turned to the WEB and the necessity of a always keep them up to date, new techniques and tools to the test and quality control of these applications appeared to assist and to automate the test process. In this set, an assistant tool to the test activity to the WEB applications is proposal. Its purpose is to introduce, through the facility proxy attendant of “Recording and Playing Back” to the WEB applications.

Keywords: Software Test – Proxy – WEB Application

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1 – INTRODUÇÃO

Com o surgimento e rápida expansão da Internet, houve a necessidade de adaptar os sistemas desenvolvidos para serem usados em um ambiente local para aproveitar os recursos e facilidades do ambiente WEB (*World Wide Web*), assim, surgiram as aplicações WEB.

O desenvolvimento de aplicações para a WEB está seguindo uma evolução similar àquela observada para sistemas de software: a produção está movendo-se de uma fase doméstica baseada em programadores hábeis para uma fase industrial. Segundo Hassan e Hold (2001), a qualidade é o ponto chave do desenvolvimento, fazendo com que técnicas e ferramentas sejam criadas para que essa qualidade seja alcançada.

Os sistemas baseados na WEB tendem a evoluir rapidamente e submeterem-se às modificações freqüentes devido às novas oportunidades tecnológicas e comerciais (RICCA e TONELLA, 2001), fazendo com que as informações ou recursos disponíveis pela tecnologia WEB sejam facilmente distribuídas e compartilhadas.

As aplicações WEB surgiram para facilitar essa distribuição da informação, utilizando recursos modernos e relativamente fáceis de serem utilizados. No Capítulo 2, comentam-se as características, funcionalidades e arquiteturas que envolvem as aplicações WEB, como arquiteturas Cliente/Servidor e Protocolo HTTP.

Com esse avanço da demanda de software voltado para a WEB, a Engenharia de Software também evoluiu. Novas técnicas para garantir a qualidade de software foram criadas para acompanhar esse desenvolvimento.

Uma das atividades para se garantir a qualidade dos produtos de softwares desenvolvidos é o teste de software. Aspectos relevantes sobre o teste de software são discutidos no Capítulo 3.

No Capítulo 4 discute-se a atividade de teste específica para aplicações WEB, abordando suas características, métodos e ferramentas.

No Capítulo 5, com o propósito de colaborar com a garantia de qualidade dessas aplicações WEB, é apresentado o protótipo de uma ferramenta que em um ambiente proxy, utilize "*Recording & Playing Back*", ou seja, que possa, utilizando-se o protocolo HTTP, capturar e armazenar casos de testes em aplicações WEB e posteriormente executá-las auxiliando assim as atividades de teste, em particular do teste de regressão. Será mostrado os métodos de implementação e sua funcionalidade operacional.

No Capítulo 6, com o propósito de exibir a justificativa e resultados do desenvolvimento do projeto, é mostrado um estudo de caso, sendo um exemplo prático de como utilizar a ferramenta proposta, auxiliando o desenvolvedor a comparar a execução da aplicação WEB em dois momentos, primeiro capturando os casos de testes e depois, executando-os novamente em uma situação diferente.

CAPÍTULO 2 - APLICAÇÕES WEB

Neste Capítulo são apresentados os principais conceitos sobre aplicações WEB e as partes que as constituem. A visão que se tem das aplicações WEB é de que elas representam a última palavra em software para ambiente cliente/servidor que utiliza a Internet. Na Seção 2.1 discute-se o que são aplicações WEB e qual sua utilidade para usuários e negócios que envolvam a Internet.

Na Seção 2.2, é discutida a Arquitetura Cliente/Servidor, e na Seção 2.3 é apresentado o Protocolo HTTP (*Hiper Text Transfer Protocol*), que com o avanço dos recursos WEB, se tornou o protocolo mais importante (no uso da Internet) e que possibilita o uso dos mais variados recursos existentes na rede. Na Seção 2.4 discute-se sobre os WEB Browsers, que agem como intérpretes das regras dos recursos do protocolo HTTP. Na Seção 2.5 são discutidas as características de um servidor proxy, mostrando os recursos e importância de se utilizar um servidor com essas características em aplicações WEB.

2.1 - Conceitos Gerais

O avanço da Internet proporcionou uma nova etapa no desenvolvimento de software e aplicativos com o surgimento do ambiente **WWW** (*World Wide Web*) ou simplesmente WEB. Essa etapa, sem dúvida, marca a história na era dos sistemas de computação. A Internet não só significou a conquista da informação, mas também a revolução dos meios de comunicação.

Acessibilidade, facilidade e rapidez na realização de operações, troca de informações e o desenvolvimento contínuo da rede são predominantes hoje, segundo Ricca e Tonella (2001).

O termo aplicação WEB abrange de uma simples página HTML, fornecendo um simples texto estático ao usuário, até um complexo cálculo de juros bancários realizado em tempo real e exibido via "WEB Browser" a um cliente que consulta sua conta corrente.

Pressman (2002) diz existir pouca discussão sobre o fato de que sistemas e aplicações baseados na WEB são diferentes no seu desenvolvimento e atributos de outros softwares de computador. Em aplicações WEB, podem-se encontrar os seguintes atributos (PRESSMAN, 2002):

- **Redes concentradas ou abertas:** uma aplicação WEB é uma estrutura que, por natureza, reside em uma rede e serve às necessidades de uma comunidade diversificada de clientes. Uma aplicação WEB pode residir na Internet permitindo comunicação aberta ao mundo todo, e alternativamente, também pode ser colocada em uma Intranet, implementando um canal de comunicação dentro de uma organização ou em uma Extranet (comunicação entre redes);
- **Necessidade de segurança**: estando o acesso disponível através de uma rede, é difícil limitar os usuários finais a terem acesso a determinada aplicação, e com isso, fortes medidas de segurança devem ser implementadas;
- **Imediatismo e evolução continuada:** diferentemente do software de aplicação convencional, que evolui através de um cronograma e versões planejadas, as aplicações WEB evoluem continuamente,

especialmente seus conteúdos. As aplicações WEB possuem um imediatismo que não é encontrado em nenhum outro tipo de software, pois um determinado recurso pode ter de ser implementado e disponibilizado em questão de horas apenas;

- **Estética e conteúdo:** a maior atração de uma aplicação WEB é a sua estética e, portanto, deve-se projetar a aplicação visando a um aspecto agradável e aproveitar seus recursos hipermídia para que o objetivo da aplicação seja alcançado.

2.1.1 – Arquitetura das Aplicações WEB

Segundo Conallen (1999), aplicações WEB são softwares que se utilizam da funcionalidade e características disponíveis na WEB. Com o surgimento da Internet, muitas aplicações têm sido desenvolvidas usando tecnologias que exploram vários recursos presentes na WEB como WEB Browsers e servidores de aplicação, segundo Hassan (2002).

Ainda segundo Hassan (2002), um WEB Browser é usado como a interface do usuário para a aplicação e os protocolos da Internet, como por exemplo, o protocolo HTTP, que é usado para comunicação entre clientes e servidores. Usuários seguem ligações (*links*) de navegação para páginas ou recursos que residem no servidor.

Este trabalho, ao utilizar-se o termo aplicações WEB, está se referindo a um sistema computacional modelo cliente/servidor que deve possuir os seguintes componentes:

- **Cliente**: pode ser um browser, ou qualquer outro programa que se conecta a um servidor WEB, via HTTP;
- **Servidor:** é o responsável pelo acesso aos recursos solicitados pelo cliente. O acesso pode ser feito através de páginas HTML estáticas, consultas dinâmicas a banco de dados ou qualquer outro método que permita acesso aos recursos desejados;
- **Protocolo:** utiliza o protocolo HTTP para a realização da comunicação entre o cliente. Possivelmente utiliza o serviço de um proxy.
- **Proxy:** fornece acesso à internet de uma forma rápida e segura para usuários que utilizam de sub-redes.

2.1.2 - O uso das Aplicações WEB

O avanço da Internet ocasionado pelo surgimento da WEB, acelerou o desenvolvimento de novas aplicações, todas voltadas para o uso na própria WEB. Surgiram assim novas modalidades de software, todas usufruindo das características e recursos que o ambiente fornece. Entre eles a estética, melhor comunicação com o usuário, agilidade e segurança. Grandes empresas de desenvolvimento de software migram para essa tendência, e as aplicações WEB fazem parte da prioridade de desenvolvimento de software das empresas.

Há no mercado computacional hoje, uma grande quantidade de programas e aplicações voltadas para a WEB, e muitos deles já fazem parte do dia-a-dia de usuários e se tornam indispensáveis.

Um reflexo disso é o fato de que grandes empresas estão disponibilizando, via aplicações WEB, a possibilidade de o cliente comprar seus produtos através da Internet. O setor de vendas pela WEB no ano de 2002, teve um crescimento de 50% em relação a 2001, quando foram registrados mais de 600 milhões de reais nas transações comerciais na Internet, segundo Manzoni (2004).

Manzoni (2004) diz que no Brasil, o relatório “Comércio Eletrônico no Mercado Brasileiro”, realizado pela Faculdade Getulio Vargas, assinalou que 2,39% do total financeiro movimentado entre empresas é eletrônico, sendo algo entre cinco bilhões de dólares e sete bilhões de dólares.

Hassan (2002) afirma que as aplicações WEB vêm se tornando padrão no desenvolvimento de novos aplicativos pelas seguintes razões:

- **Aplicações WEB são mais acessíveis:** O protocolo HTTP usado em aplicações WEB é um protocolo padrão, que pode se comunicar por barreiras de proteção (*firewall*) incorporadas à rede. Assim, as aplicações são acessíveis a muitos usuários, desde uso doméstico ao uso corporativo. As aplicações tradicionais usam protocolos proprietários que são normalmente bloqueados através das barreiras de proteção, limitando o acesso dos usuários a essas aplicações. Além disso, uma aplicação WEB não requer um cliente especializado. Segundo Hassan e Holt (2001), um WEB Browser é usado como cliente, tornando o recurso desejado acessível a várias plataformas operacionais.
- **Aplicações WEB têm custos baixos de manutenção e desenvolvimento:** Hassan (2002) diz que não há nenhum custo

associado com desenvolvimento do software do cliente, já que para isso, é utilizado o WEB Browser. Manter a aplicação WEB requer modificar apenas o código que reside no servidor. Isto reduz o custo de atualização e desenvolvimento de aplicações WEB comparada a aplicações tradicionais.

2.2 - Arquitetura Cliente / Servidor

Muitos dos recursos computacionais, que são vistos hoje, têm a derivação da arquitetura cliente/servidor. Isso se dá porque um software "hospeda-se" em um computador (servidor), e outro aplicativo (cliente) requer os recursos ou serviços desse software ativando-o remotamente. O software cliente solicita serviços, e o software servidor fornece esses serviços. Segundo Pressman (2002), o avanço das tecnologias, do hardware, software, banco de dados e Internet, contribuem para a evolução da arquitetura cliente/servidor.

A grande vantagem de se utilizar esse tipo de arquitetura é o compartilhamento de recursos, seja ele hardware, ou, no caso da Internet, o software, sendo que os termos "cliente" e "servidor" referem-se aos programas que solicitam e enviam serviços e informações.

Como a arquitetura cliente/servidor possui características interessantes no seu contexto, pode-se ter diferentes implementações, segundo Orfali (1999). Por exemplo, servidores de arquivos, servidores de banco de dados, servidores de transação e servidores de trabalho em grupo (*grupoware*).

Em uma arquitetura cliente/servidor, o aplicativo ou software que necessita de recursos para que seja efetuado algum processamento, é chamado de

cliente. O aplicativo ou software que forneça recursos para que outras aplicações processem informações, é chamado de servidor.

Quando se usa a Internet, aprende-se a usar cada um dos programas clientes. Uma das funções do usuário é iniciar o cliente (browser) e dizer a ele o que fazer. A tarefa do cliente é de conectar-se ao servidor apropriado e cuidar de que seus comandos sejam executados, e assim, esperar a resposta desse servidor.

Muitos dos protocolos usados na Internet são baseados no paradigma de requisição e resposta, como por exemplo, FTP, Telnet ou ainda HTTP (*HiperText Transfer Protocol*), utilizado pelos Browsers.

No caso de se utilizar o protocolo HTTP, um programa requisitante (cliente) estabelece uma conexão com um outro programa receptor (servidor) e envia um pedido de requisição para o servidor contendo, o URI (*Uniform Resouce Identifier*), a versão do protocolo, uma mensagem contendo os modificadores da requisição, informação sobre o cliente, e possivelmente o conteúdo no corpo da mensagem a ser requisitada.

O servidor responde com uma “linha de status” incluindo sua versão de protocolo e um código de operação bem-sucedida ou um código de erro, seguido pelas informações do servidor, meta informações da entidade e possível conteúdo no corpo da mensagem, que deve ser interpretado e devidamente exibido pelo cliente (Browsers).

2.3 - O Protocolo HTTP

A WEB utiliza a Internet como meio de comunicação, devendo obedecer aos protocolos de comunicação próprios da Internet. O protocolo HTTP surgiu

através de um físico chamado Tim Berners-Lee, nos anos de 1989 a 1991 no CERN (*European Center for High-Energy Physics*) em Genebra, na Suíça.

Segundo Wong (2000), a proposta do protocolo HTTP visava facilitar a comunicação e o compartilhamento de informações. Deveria existir uma maneira pelas quais documentos gravados em um computador pudessem ser acessados através de outros computadores, por meio de uma rede, facilitando assim a comunicação entre os pesquisadores. Esses documentos, por sua vez, poderiam fornecer atalhos para outros documentos, através de links, que também poderiam estar localizados em computadores fisicamente distantes.

A principal característica do HTTP estaria na representação dos dados (formato e negociação), não importando como os sistemas foram construídos, permitindo a transferência de qualquer tipo de informação, seja som, texto, vídeo ou imagem. Dessa maneira, o HTTP é o protocolo de rede utilizado para entregar virtualmente todos os arquivos e outros dados (de forma abrangente chamados de recursos) na WEB.

Por essa característica, pode-se dizer que sem o protocolo HTTP, a Internet como é utilizada hoje, não teria tanto sucesso. Foram criadas padronizações de comunicação entre clientes e servidores da WEB, especificando como seria essa comunicação, desde a forma de requisições até as respostas dos recursos pretendidos.

Entende-se, por recurso, segundo Wong (2000), como sendo qualquer trecho de informação que possa ser identificado por um URL (*Uniform Resource Locator*), como por exemplo, um arquivo HTML, ou uma consulta a um banco de dados ou qualquer recurso que esteja disponível no servidor WEB.

2.3.1 - As camadas de Protocolo HTTP

Existem vários protocolos que trabalham em um sistema de comunicação em rede como a Internet e todos eles têm a finalidade de fornecer serviços.

A arquitetura da Internet se caracteriza pelas camadas:

- **Aplicação**: onde se situa o protocolo HTTP;
- **TCP**: com missão de tornar confiável a transferência de informações e
- **Protocolo Internet (IP)**: que recebe ou envia pacotes individuais e os endereça aos seus destinos.

Assim, o Protocolo de Controle de Transmissão (TCP) e o IP fazem com que clientes (Browsers) e servidores se conectem. Através dessa conexão, o protocolo HTTP é utilizado pelos clientes para a requisição de serviços, e ao mesmo tempo define as regras básicas para as transações entre esses clientes e os servidores WEB.

A comunicação da arquitetura cliente/servidor na Internet ocorre geralmente sobre uma conexão TCP/IP, mas isso não impede que o protocolo HTTP seja implementado no topo de qualquer outro protocolo.

Podemos definir uma transação HTTP como:

- Estabelecimento da conexão;
- Requisição: o cliente requisita um recurso ao servidor HTTP;
- Resposta: o servidor HTTP, se não houver nada de errado com a requisição do cliente, retorna o recurso desejado para o cliente;

- Fechamento da conexão: a conexão é encerrada logo após o recurso requisitado chegar ao cliente.

O HTTP é um protocolo *"stateless"*, ou seja, não armazena qualquer informação da conexão entre as transações. Junior (2001) mostra que os tipos de tramissão das mensagens de requisição ou resposta, são semelhantes nas suas estruturas, que consistem em:

- Linha inicial;
- Linhas de cabeçalho, que não são obrigatórias;
- Linha em branco obrigatória e
- Corpo da mensagem, opcional.

2.3.2 - Transações HTTP

A seguir, é mostrado um exemplo comum de uma transação WEB, utilizando o protocolo HTTP entre a requisição de recursos do cliente para o servidor.

Supondo que o recurso desejado fosse o URL "**http://www.fundanet.br:80/teste.htm**", segundo Wong (2000), o WEB Browser interpretaria o URL da seguinte maneira:

- *http://* : use o protocolo HTTP;
- *www.fundanet.br/:* conecte a um computador com nome de host de *www.fundanet.br*;
- *:80*: conecte a esse computador utilizando a porta 80. Segundo Junior (2001), esse número de porta pode ser qualquer número de

porta IP legítimo, podendo variar de 1 até 65535, e, se o cliente omitir esse número da porta, o protocolo HTTP assumirá o valor 80.

O recurso requisitado pelo cliente é uma página HTML chamada "teste.htm" e que, se não houver problemas com a conexão e com o protocolo HTTP, o servidor irá enviá-la como resposta.

Após a conexão com o URL requisitado através da porta 80, usando o protocolo HTTP, a seguinte mensagem exemplo é enviada pelo WEB Browser ao servidor (JUNIOR, 2001):

```
GET /HTTP/1.1
Accept:image/gif,image/x-xbitmap,image/jpeg,image/pjpeg, /*/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Host:fundanet.br/teste.htm
Connection: Keep-Alive
```

Figura 2.1: Mensagem Exemplo enviada pelo WEB Browser ao servidor (JUNIOR, 2001).

Pode-se interpretar a Figura 2.1 da seguinte maneira (WONG 2000):

- A primeira linha (*GET /HTTP/1.1*) requer um recurso de um servidor e indica a versão do protocolo HTTP que vai ser usada na transação;
- A segunda linha indica ao servidor quais tipos de documentos serão aceitos pelo WEB Browser;
- A terceira linha mostra que a linguagem utilizada pelo WEB Browser (cliente) é a Americana;

- A quarta linha aponta que o cliente tem condições de interpretar a resposta dada pelo servidor utilizando algoritmos de descompressão *gzip* ou *deflate*;
- A quinta linha (*User-Agent*) especifica que o cliente é identificado como *"Mozilla 4.0"* utilizando *"Windows NT"* como plataforma operacional, e entre parênteses, o desenvolvedor e a versão do WEB Browser;
- A sexta linha se refere a qual servidor host o cliente estará conectado e qual o recurso solicitado pelo cliente;
- A sétima linha indica ao servidor que a conexão TCP deve permanecer aberta enquanto não seja explícito o pedido para desconexão.

Após o envio da requisição e a conexão ao servidor, uma resposta será enviada ao cliente, como é exibido na Figura 2.2, segundo Wong (2000):

```
HTTP/1.1 200 OK
Date: Mon, 23 Jun 2003 23:54:44 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Fri, 20 Jun 2003 13:06:11 GMT
ETag: "2f5cd-964-381e1bd6"
Accept-Ranges: bytes
Content-length: 327
Connection: close
Content-type: text/html
<title>Exemplo de Teste</title>
<img src="/images/fundanet.gif">
<h1>Bem Vindo</h1>
```

Figura 2.2: Resposta do Servidor ao Cliente (WONG, 2000).

Pode-se entender os códigos exibidos na Figura 2.2 da seguinte maneira, segundo Wong (2000) e Junior (2001):

- A primeira linha mostra a versão do protocolo HTTP que o servidor está utilizando, e segundo Wong (2000), o mais importante, o código de status, onde diz (*código 200*) que o

recurso foi encontrado com sucesso e será transmitido como resposta. Além do código de sucesso *200*, o protocolo HTTP também possui outras famílias de códigos, como Junior (2001) cita:

- *1xx*: Envia informações para o cliente, sendo apenas informativa, indicando que a requisição foi recebida e está sendo processada;
- *2xx:* Indica que a requisição feita pelo cliente foi bem-sucedida;
- *3xx:* Indica que ações adicionais devem ser tomadas antes de a requisição ser completada com sucesso;
- *4xx:* Indica que o navegador fez uma requisição que não pode ser atendida ou não existe;
- *5xx:* Indica um erro no servidor.

- A segunda linha indica a data atual do servidor;
- A terceira linha qual a versão do software servidor que o cliente está conectado;
- A quarta linha especifica a mais recente modificação no recurso requisitado pelo cliente. Este recurso é muito utilizado por ferramentas proxies, como será discutido na Seção 2.5;
- A quinta linha atribui uma identificação WEB única para o cliente que fez a requisição do recurso, evitando o “extravio” do recurso requisitado e também muito utilizado pelos mecanismos de cache;
- A sexta linha aponta que o servidor está habilitado a retornar outras seções de documentos, sempre que ele for requisitado;

- A sétima linha mostra o tamanho em bytes do recurso desejado;
- A oitava linha indica que a conexão será encerrada depois da resposta dada pelo servidor ao cliente;
- A nona linha indica ao WEB Browser qual o tipo de documento será enviado pelo servidor como resposta ao cliente, sendo que no exemplo, é um arquivo no formato HTML, e por fim
- O recurso desejado, que no caso são instruções HTML que exibirão mensagens e uma figura no formato *gif*.

2.4 - WEB Browsers (navegadores)

A maneira mais simples de se requisitar recursos na Internet é por meio dos WEB Browsers, os “navegadores”, pois agem como cliente e intérpretes das regras de requisições e respostas do protocolo HTTP. O primeiro navegador a utilizar os recursos HTTP foi o Mosaic, desenvolvido por Marc Andreesen, na NCSA (*National Center for Supercomputing Applications*), na Universidade de Illinois, Urbana-Champaign, nos Estados Unidos.

O navegador possibilita ao usuário da Internet o acesso aos serviços de uma maneira intuitiva e flexível, independente da arquitetura do computador utilizado para a navegação, como mostra a Figura 2.3.

Além disso, uma aplicação WEB não requer um cliente especializado. Segundo Hassan e Holt (2001), um WEB Browser é usado como cliente, tornando o recurso desejado acessível a várias plataformas operacionais.

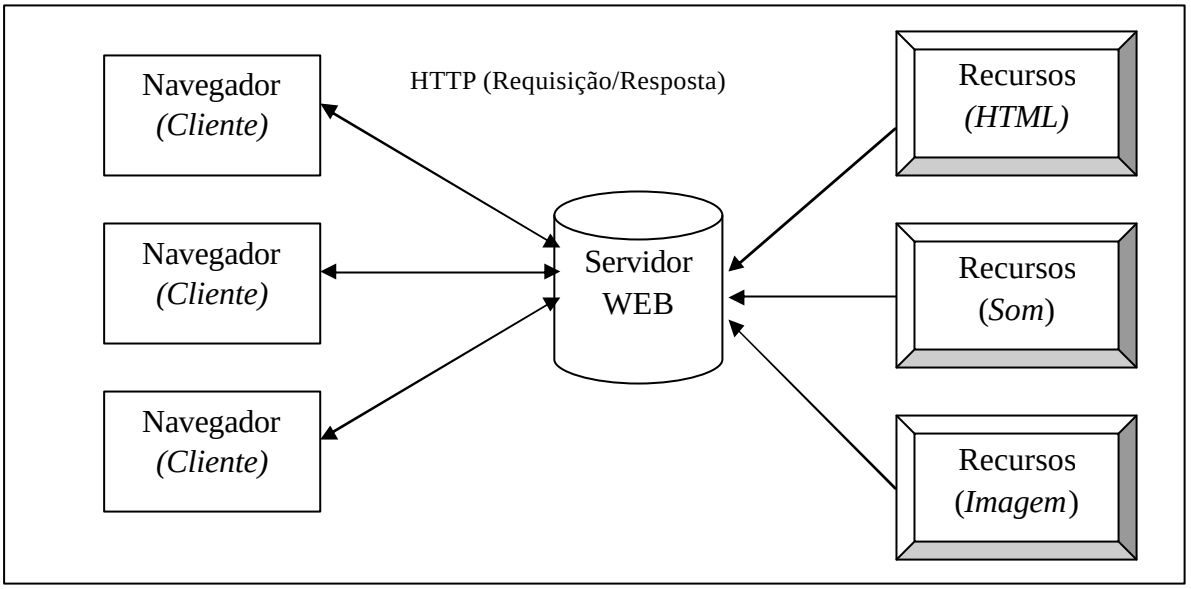

Figura 2.3: Exemplo de comunicação entre o cliente (browser) e o Servidor WEB.

Os browsers fornecem suporte a diversos tipos de plataformas operacionais. A flexibilidade em relação aos tipos de dados a serem enviados ou recebidos também permite fornecer suporte aos vários formatos de recursos, como texto, som, imagem e outros.

Quando um link (ou recurso) é requisitado em um navegador WEB, ele tenta estabelecer uma conexão TCP/IP com um servidor que reside em algum local na rede. Essa conexão normalmente é feita para a porta 80, a porta HTTP.

Em seguida, o navegador envia uma mensagem, denominada “solicitação HTTP”, para o servidor e esse responde com as informações solicitadas. As informações são recebidas pelo navegador e processadas ou exibidas, dependendo de seu tipo, segundo Hatch, Lee e Kurtz (2002).

2.5 - Servidor Proxy

Um servidor proxy é muito utilizado para fornecer acesso à Internet de uma maneira rápida e segura para usuário que se utiliza de sub-redes. Com um proxy, pode-se ainda aumentar o controle de acesso a Internet.

Os servidores proxies podem controlar as atividades de diferentes usuários e fornecer relatórios dessas atividades usando o sistema de cache. Com o cache, o tempo de acesso às informações disponíveis na Internet se torna mais curto após a primeira busca, pois estas ficam armazenadas em um servidor e assim, não é necessária uma nova busca na Internet.

Com o uso de um servidor proxy, usuários podem ter livre acesso a Internet, mas sem criar uma vulnerabilidade na segurança da rede da organização. Com o aumento das redes comerciais, o uso de firewalls passou a ser uma obrigação de segurança para as empresas, o que dificultaria o acesso completo da Internet. Com o avanço da tecnologia de proxies, mesmo que a rede não esteja protegida por um firewall, a possibilidade de uma vulnerabilidade na segurança é reduzida, devido às regras rígidas de acesso.

Segundo Santos (2003), normalmente, o mesmo proxy é usado por todos os clientes em uma sub-rede, fazendo com que todos os usuários tenham acesso aos mesmos documentos e serviços requisitados, aumentando assim, a performance da rede e diminuindo os custos do tráfego da rede significativamente. Em um download, por exemplo, uma vez que o primeiro acesso foi feito para um determinado usuário, os demais acessos são feitos no cache local.

Se a necessidade de acesso a Internet for intensa para um determinado recurso, Chan, Griffith e Iasi (1999) dizem que as performances do servidor e do cliente são prejudicadas, mas se, no caso de uma requisição simples e de ações que não dependam do acesso contínuo à Internet, o proxy não afetará o desempenho da rede.

2.5.1 – Funcionamento do Proxy

Santos (2003) explica que, em uma rede que utiliza proxy, as estações de trabalho não possuem uma conexão direta com a Internet, assim sendo, não se comunicam de maneira direta com a mesma. O servidor proxy faz a ligação entre a estação de trabalho e a Internet. Segundo Chan, Griffith e Iasi (1999), o servidor proxy repete a mensagem recebida de um cliente para outro servidor. Por sua vez, o servidor proxy repete as mensagens que recebe de um servidor para um cliente.

Os servidores de proxy são usados para permitir aos computadores de uma rede interna o acesso à WEB, FTP e outros serviços configurados. O proxy é um servidor especial, que roda em uma máquina que pode agir também como se fosse um Firewall, escondendo os computadores da rede interna (SANTOS, 2003).

Basicamente, ele recebe requisições de máquinas que estão na rede interna, envia aos servidores que estão do lado externo da rede, lê as respostas externas e envia de volta o resultado aos clientes da rede interna.

O servidor proxy permite aos usuários uma navegação mais eficiente. O proxy funciona como uma cópia local de outros sites espalhados pelo mundo.

Toda vez que um usuário (que esteja utilizando o Proxy) acessar um site localizado na Internet, este é copiado para o servidor local (WATANABE, 2004) onde as informações ficam registradas. Depois deste usuário, todos os outros usuários (utilizando o proxy) que tentarem acessar o mesmo site carregarão a cópia que agora está no servidor proxy.

Antes de mostrar um URL-cópia, o servidor Proxy verifica no site original qual a data de última atualização para que não sejam mostrados sites desatualizados.

Normalmente, o mesmo servidor proxy é usado para todos os clientes em uma rede interna, que pode ou não ser constituída de sub-redes, explica Watanabe (2004).

Os tipos de servidores Proxy mais utilizados, segundo Santos (2003), são:

- **Proxies genéricos**, que oferecem serviços de proxy para várias aplicações (por exemplo WEB, Ftp, Gopher e Telnet) em um único servidor;
- **Proxies específicos**, que oferecem serviços de proxy para uma determinada aplicação, como é o caso do WEB Proxy, que é um proxy que tem por finalidade, fazer caching de documentos WEB que foram acessados, reduzindo de forma considerável, o tráfego de acesso à Internet em requisições futuras.

De acordo com Luotonen e Altis (2004), para transações de um cliente com um servidor proxy, esse cliente se utiliza somente do protocolo HTTP,

mesmo quando deseja buscar um recurso através de um outro protocolo, como por exemplo, o FTP. A Figura 2.4 exemplifica essa transação.

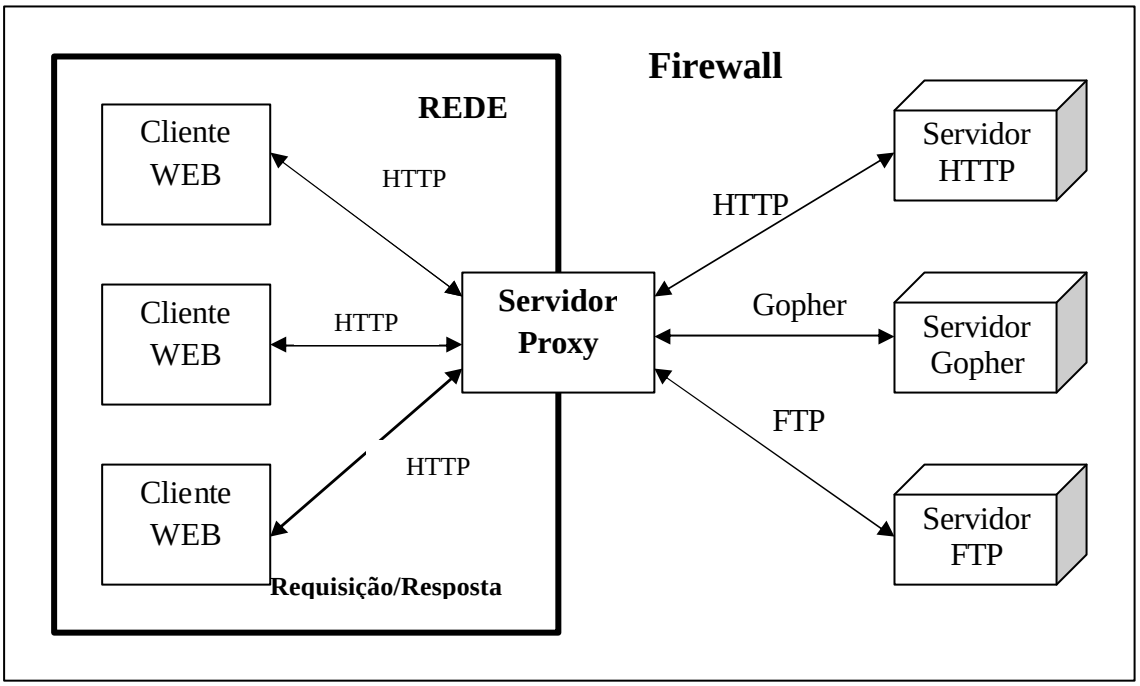

Figura 2.4: Transação dos protocolos utilizando Firewall.

Ao emitir um pedido a um servidor proxy, o URL inteiro é especificado e não apenas o caminho e cabeçalhos opcionais de requisição do HTTP. Neste momento, o proxy começa a funcionar como um cliente para recuperar o recurso original, usando o protocolo apropriado.

Na Figura 2.5, é exibida uma demonstração de como funciona o fluxo dentro de um servidor WEB Proxy, ele recebe as requisições, faz uma análise no cache local, e se o documento estiver no cache, ele responde automaticamente, caso contrário, se o documento não estiver no cache, ou, se ele estiver precisando de atualização, ele vai ao endereço remoto e busca o documento, ou as atualizações e guarda no cache local, para ser usado nas requisições futuras.

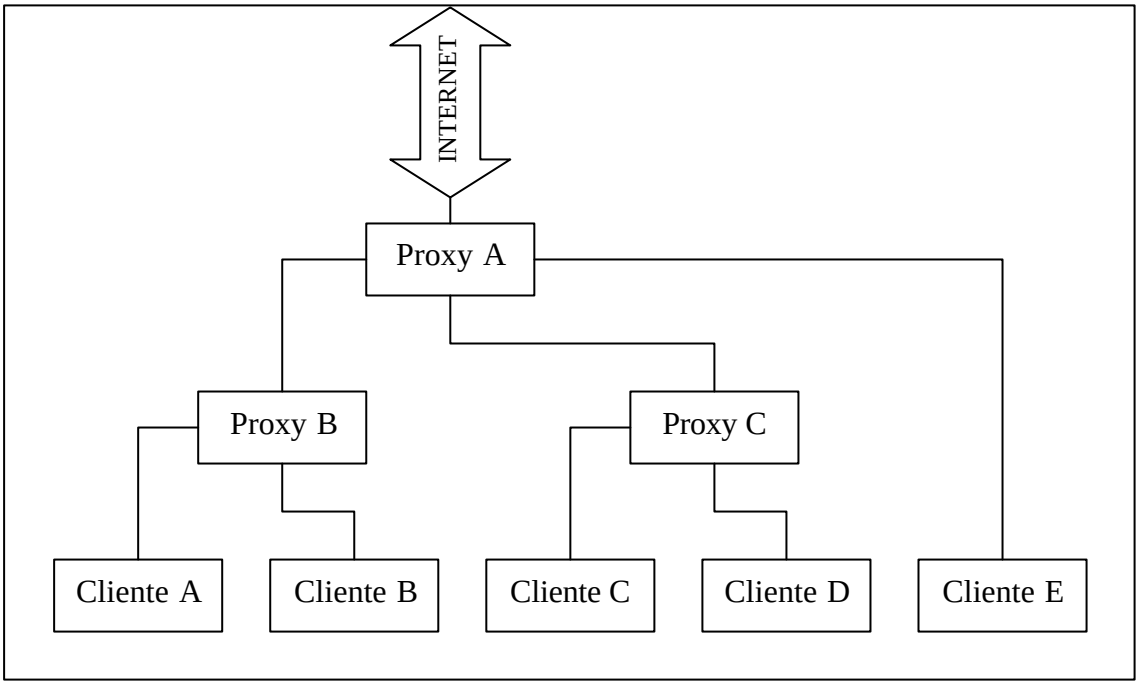

Figura 2.5: Funcionamento de um Proxy.

Um servidor proxy permite também um nível satisfatório de gravação das transações de clientes, incluindo endereço IP, data e hora, URL, contagem de bytes e código de sucesso HTTP. Também é possível fazer a filtragem de transações de clientes no nível do protocolo de aplicação. O proxy pode controlar o acesso a serviços por métodos individuais, servidores e domínios (LUOTONEN E ALTIS, 2004).

2.5.2 - Serviço Cache do Proxy

A aplicação Proxy age como intermediário entre clientes e servidores WEB. O servidor local procura pela página, grava-a no disco e repassa para o usuário. Requisições subseqüentes de outros usuários recuperam a página que está gravada localmente, utilizando assim, de um cache (SANTOS, 2003).

Sem um proxy, clientes fazem conexão diretamente aos servidores WEB. Os servidores proxies são usados por organizações ou provedores que querem

reduzir a quantidade de banda que utilizam. Para aumentar o desempenho, muitos servidores proxy podem fazer parte de uma hierarquia (ver Figura 2.5), na qual há a comunicação entre servidores proxy, reduzindo a necessidade de objetos serem recuperados diretamente.

A Figura 2.6 mostra como funciona o sistema cache do proxy. As requisições efetuadas pelos usuários são representadas pela letra "***A***", já as respostas a essas requisições são representadas pela letra "***B***". A letra "***C***" representa o fluxo de dados dentro do sistema proxy, segundo Santos (2003).

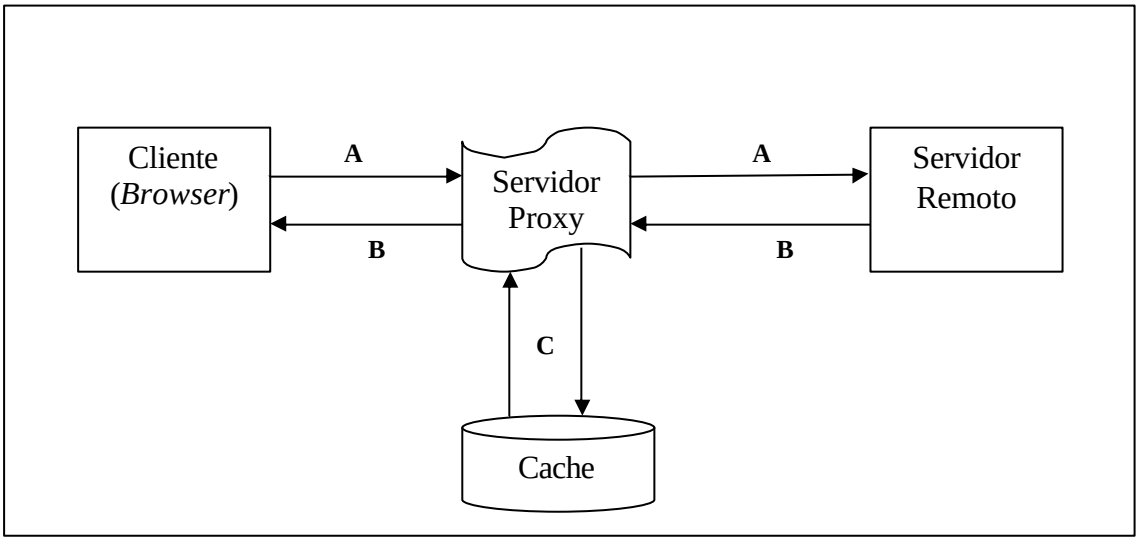

Figura 2.6: Funcionamento do Sistema Cache do Proxy (SANTOS, 2003).

Pode-se citar as seguintes vantagens ao se utilizar cache no sistema proxy:

- *Redução do tráfego:* Menos requisições e respostas precisam trafegar na rede, o objeto é recuperado do servidor somente uma vez, reduzindo a quantidade de banda usada pelo cliente. Pode-se conseguir taxas de acerto de até 60%;
- *Redução de carga dos servidores:* Menos requisições para o servidor *www* atender;

- *Redução da latência:* As respostas de requisições aos objetos cacheados são feitas a partir do cache local, não pelo servidor ***www*** original.

CAPÍTULO 3 – FERRAMENTAS DE TESTE

Durante o processo de desenvolvimento de um programa existem atividades que procuram garantir a qualidade do produto final. Entretanto, apesar dos métodos, técnicas e ferramentas utilizadas, falhas no programa ainda podem ocorrer. Assim, a etapa de teste, uma das atividades de garantia de qualidade, é de grande importância para a identificação e eliminação de defeitos, representando assim uma etapa de grande importância no desenvolvimento do programa (JORGE *et al.*, 2002).

A atividade de teste não pode mostrar a correção de um software, pois o domínio dos dados de entrada normalmente é muito grande, a ponto de ser inviável testar todas as possibilidades de entrada, saída ou manipulação de dados, o que corresponderia ao teste exaustivo, pelo qual a correção poderia ser comprovada (OSTRAND e WEYUKER, 1988). A limitação do tempo destinado à prática do teste também é fato relevante nesse contexto, sendo assim, existem três questões fundamentais:

1. Como devem ser selecionados os dados de teste?
2. Como decidir se um produto foi suficientemente testado?
3. Como indicar se um conjunto de dados de teste é adequado?

A Seção 3.1 apresenta alguns conceitos básicos do teste de software. Na Seção 3.2 e Seção 3.3 são mostradas as ferramentas para aplicação de critérios e automatização de testes, que auxiliam no teste de programas.

Por fim, nas Seções 3.4 e 3.5 serão descritas algumas ferramentas e conceitos relacionados ao teste de aplicações WEB e de Recording & Playing Back.

3.1 - Conceitos Básicos

Segundo Myers (1979), a atividade de teste é o processo de executar um programa com a intenção de encontrar um defeito. Um bom caso de teste é aquele que tem alta probabilidade de revelar a presença de defeitos. Já um teste bem-sucedido é aquele que detecta a presença de um defeito ainda não descoberto.

Uma das maneiras de se prevenir ou detectar erros cometidos ao longo do processo de desenvolvimento é a definição de diferentes níveis de testes em relação às etapas do processo de desenvolvimento do software, enfocando classes distintas de erros.

Pode-se dividir em quatro etapas a atividade de teste de software:

1. Planejamento;
2. Projeto de casos de teste;
3. Execução do teste;
4. Avaliação dos resultados.

Essas etapas devem ser executadas ao longo de todo o processo de desenvolvimento do software. O planejamento da atividade do teste de software deve fazer parte do planejamento global de desenvolvimento do sistema, desde

sua criação e evolução, constituindo assim o teste um dos pontos cruciais no desenvolvimento do software.

É importante ressaltar que a realização de testes pode mostrar se o software contém defeito, mas não provar que o mesmo nunca falhará. Testes devem, então, ser realizados com o intuito de encontrar defeitos, e conseqüentemente, ao encontrá-los, o teste poderá ser considerado como bem-sucedido.

Ntafos (1988) diz que a atividade de teste é cercada por uma série de limitações. Por exemplo, normalmente, os seguintes problemas são impossíveis de se solucionar: dados dois programas, se eles são equivalentes; dadas duas seqüências de caminhos (comandos) de um programa, ou de programas diferentes, se elas processam a mesma função; dado um caminho, se ele é executável ou não, ou seja, de uma maneira geral, se existe um conjunto de dados de entrada que leve à execução desse caminho.

Não existe um procedimento de teste “geral” que possa ser usado para provar se um programa está correto ou não, mas, se os testes forem conduzidos sistemática e criteriosamente, mesmo não podendo afirmar que um programa está correto, contribuem para aumentar a confiança de que o software desempenha as funções específicas para qual foi desenvolvido.

Para auxiliar a atividade de teste, deve-se utilizar uma ferramenta de apoio ao teste. Uma ferramenta de teste reduz a intervenção humana nos resultados obtidos. Essa ferramenta deve produzir casos de teste que mostrem o comportamento errado do programa e revelar os defeitos.

O desenvolvimento de ferramentas de teste para o suporte à atividade de teste propriamente dita também é de fundamental importância uma vez que essa

atividade é muito propensa a erros, além de improdutiva, se aplicada manualmente, cita Jorge *et al.*, (2002).

Jorge *et al.*, (2002), cita ainda que, a disponibilidade de ferramentas de teste oferece recursos para o desenvolvimento de estudos empíricos que visem a avaliar e a comparar os diversos critérios de teste existentes. Ferramentas de teste permitem a transferência de tecnologia para as indústrias e contribui para uma contínua evolução de tais ambientes, fatores indispensáveis para a produção de software de alta qualidade.

Além disso, tais ferramentas auxiliam pesquisadores e alunos de Engenharia de Software a adquirir os conceitos básicos e experiência na comparação, seleção e estabelecimento de estratégias de teste (VINCENZI, 1998).

Segundo Pressman (2002), existem as seguintes categorias de ferramentas de teste de software:

- **Aquisição de Dados:** ferramentas que adquirem dados a serem usados durante o teste;
- **Medição Estática:** ferramentas que analisam o código-fonte sem executar casos de teste. Existem três tipos diferentes de ferramentas de teste estático, sendo ferramentas de teste baseado em código, linguagens especializadas de teste e ferramentas de teste baseado em requisitos;
- **Medição Dinâmica:** ferramentas que analisam o código-fonte durante a execução do programa, ou seja, interagem com o programa em execução. As ferramentas dinâmicas podem ser intrusivas, que modificam o software a ser testado, inserindo

instruções extras, ou não-intrusivas, que usam um processador de hardware separado que é executado em paralelo com o processador contendo o programa que está sendo testado;

- **Simulação:** ferramentas que simulam funções do hardware e outros dispositivos externos;
- **Gestão de Testes:** ferramentas que assistem o planejamento, desenvolvimento e controle de testes, geram e coordenam testes de regressão, realizam comparações que avaliam as diferenças entre as saídas esperada e real;
- **Ferramentas de Cruzamento Funcional:** ferramentas que cruzam as fronteiras das categorias precedentes.

Deve-se notar que muitas ferramentas de teste têm características que abrangem duas ou mais das categorias citadas acima.

3.2 - Ferramentas para Aplicação de Critérios

As atividades de teste de software são bastante custosas, demandando muito tempo e esforço de desenvolvedores, programadores e testadores, como já mencionado. Assim, muitas ferramentas destinadas ao auxílio do teste foram desenvolvidas e assim, auxiliam na atividade de teste.

Chaim (1991), Vincenzi (1998) e Jorge *et al.*, (2002), dizem que, essa situação de grande demanda e custo nos testes, motivou o desenvolvimento de ferramentas de teste para auxiliar na produção de testes efetivos e na análise dos resultados dos testes.

O uso de uma ferramenta de software para auxílio ao teste de programas pode ser vinculado a um critério de teste de duas maneiras. A ferramenta de software pode utilizar o critério de teste como um guia para a geração de casos de teste que satisfaçam o critério; outra possibilidade é a utilização do critério para a análise de cobertura de um conjunto de casos de teste, isto é, verificar se os casos de teste aplicados preencheram os requisitos de critério de teste (CHAIM, 1991).

Segundo Chaim (1991), a partir da década de 70 começaram a surgir as primeiras ferramentas de apoio ao teste estrutural de software, que suportavam principalmente a aplicação de critérios de teste baseados somente no grafo de fluxo de controle do programa a ser testado.

No início dos anos 80, ainda segundo Chaim (1991), surgiram as primeiras ferramentas de teste comerciais, que suportavam basicamente a análise de cobertura dos critérios todos-nós e todos-ramos, mais alguma forma de análise estática do programa fonte para detecção de anomalias de fluxo de dados e verificação dos tipos dos parâmetros utilizados nas chamadas de procedimentos.

Em meados da década de 80, com o surgimento dos critérios baseados em análise de fluxo de dados, começaram a surgir ferramentas que suportam a aplicação desses critérios. Na sua maioria, essas ferramentas foram implementadas pelos criadores dos critérios de teste e visavam mostrar que a aplicação dessa nova classe de critérios era provável de ser utilizada na prática (CHAIM, 1991).

Em meados dos anos 90, com o avanço da Internet e as novas tecnologias voltadas para o desenvolvimento de aplicações WEB, novas ferramentas e técnicas de teste foram criadas para o teste dessa nova categoria de aplicações.

3.2.1 - Ferramenta PROTEUM/IM 2.0

A ferramenta de teste PROTEUM/IM 2.0 apóia o critério Mutação de Interface (DELAMARO, MALDONADO E VINCENZI, 2000), e auxilia a atividade de teste durante o desenvolvimento do software, apoiando o teste de unidade e o teste de integração.

A ferramenta está disponível para os sistemas operacionais SunOS e Linux e configurada para o teste de programas escritos na linguagem C, sendo também uma ferramenta multilinguagem.

Segundo Delamaro, Maldonado e Vincenzi (2000), as operações mínimas suportadas pela ferramenta são:

- Manipulação de casos de teste: execução, inclusão e exclusão;
- Manipulação de mutantes: geração, seleção, execução e análise e
- Análise de adequação: escore de mutação e relatórios estatísticos.

Algumas dessas operações são suportadas pela ferramenta de modo completamente automático, outras, necessitam da intervenção do testador.

A ferramenta permite ao testador avaliar a adequação de um conjunto de casos de teste ***T*** para um determinado programa ***P*** e com o resultado dessa avaliação o testador pode melhorar o conjunto ***T*** visando a satisfazer o critério de Mutação de Interface.

Os programas que compõem a ferramenta PROTEUM/IM estão divididos em dois grupos. O primeiro é composto por programas básicos que agem diretamente na base de teste que caracteriza a sessão. O segundo é composto por programas utilitários que utilizam os programas básicos para realizar algumas operações durante uma sessão de teste. Bianchini *et al.*, (2003) mostra esses comandos na Tabela 3.1.

Programas Básicos	
li	Transforma um programa C numa representação intermediária
instrum	Cria programa instrumentado que produz como saída a lista de nós percorridos por um caso de teste
ptest	Cria e manipula o arquivo de teste, que dá características gerais da base de teste
tcase	Cria e manipula a base de dados dos casos de teste
muta	Cria e manipula a base de dados dos mutantes
opmuta	Aplica os operadores de mutação ao programa original, criando descritores de mutação
exemuta	Constrói os códigos fontes dos mutantes e executa-os; também usado para ativar / desativar mutantes
report	Cria relatório sobre o comportamento dos casos de teste
Programas Utilitários	
test-new	Cria uma nova base de teste
Tcase-add	Inclui um novo caso de teste, interativamente
Muta-gen	Gera descritores de mutação e os inclui na base de dados dos mutantes
Muta-view	Exibe os mutantes para visualização e análise
list-good	Exibe o conjunto de casos de teste em operação
check-equiv	Marca automaticamente os mutantes equivalentes baseando-se na freqüência de execução.

Tabela 3.1: Programas que compõem a ferramenta Proteum/IM 2.0 (BIANCHINI *et al.*, 2003).

A execução desses programas pode ser feita de duas formas: diretamente na linha de comando através de scripts ou de forma transparente para o usuário através da interface gráfica.

A interface gráfica facilita a condução de uma sessão de teste no caso do usuário ser iniciante. Os recursos de visualização dos casos de teste e dos mutantes equivalentes são melhores com a interface gráfica, tornando a condução da sessão de teste mais fácil.

No entanto, esse tipo de interface depende muito da interação do usuário, sendo, assim, menos flexível do que chamar diretamente os programas, segundo Delamaro, Maldonado e Vincenzi (2000). A chamada direta aos programas, na forma de scripts, reduz o número de interações com as ferramentas e possibilita a execução de longas sessões de teste em lote, nas quais o usuário pode construir um programa especificando as tarefas a serem realizadas e a ferramenta executa esse programa, reduzindo, assim, o tempo gasto na atividade de teste.

Entretanto, se o usuário optar por utilizar scripts, a elaboração desses scripts exige um esforço de programação e um completo domínio tanto dos conceitos sobre teste baseado em mutação quanto dos próprios programas que compõem a ferramenta.

A atividade de teste é iniciada com a criação de uma sessão de testes. Uma sessão de teste é caracterizada por um banco de dados criado e administrado pela ferramenta Proteum/IM 2.0 para armazenar a informação necessária sobre o programa a ser testado, seus mutantes e os casos de teste providos pelo testador. Uma sessão de teste pode ser criada, interrompida e reiniciada utilizando o módulo de recuperação de sessões.

Uma vez que a sessão de teste é criada, podem ser executadas duas maneiras de se construir um banco de dados com casos de testes. A primeira maneira é a inserção interativa de casos de teste pelo testador.

A ferramenta Proteum/IM 2.0 também provê um modo para remover um caso de teste logicamente. Nesta situação, o caso de teste não é removido fisicamente do banco de dados de casos de teste, porém não é considerado nas operações futuras da ferramenta. Depois, o caso de teste pode ser re-habilitado e disponível ao banco de dados lógico de casos de testes. Isto permite ao testador usar combinações e situações diferentes de casos de testes disponíveis no banco de dados, cita Delamaro, Maldonado e Vincenzi, (2000).

A principal tarefa da ferramenta é a manipulação de mutantes, e para isso, é necessário escolher que tipo de mutantes será usado no teste. A ferramenta Proteum/IM 2.0 dispõe de um conjunto de 33 operadores de Mutação de Interface, divididos em dois grupos: 24 operadores no Grupo I e 9 operadores no Grupo II. Para o teste de unidade, há um conjunto de 75 operadores. Eles são classificados em operadores de constantes, declaração e operadores de variáveis.

Para cada operador é possível selecionar uma porcentagem de geração e o número máximo de mutantes por ponto de mutação, sendo que esse número pode ser determinado individualmente ou a um grupo específico de cada operador.

Após a criação do conjunto de casos de testes é iniciada a execução dos mutantes. A ferramenta Proteum/IM 2.0 executa cada mutante e compara seu comportamento com o comportamento apresentado pelo programa original.

Como resultado de execução do mutante, Proteum/IM 2.0 marca mutantes com os tipos: “morto” ou “vivo”, e o testador pode analisar os mutantes que

ainda permaneçam “vivo” e decide, se eles são equivalentes ou não. Da mesma maneira que um conjunto de caso de testes pode ser desabilitado e posteriormente re-habilitado, um mutante também pode ser marcado com um tipo e depois, alterado.

Mutantes vivos são os mutantes que não “morreram” com nenhum caso de teste (***t***) do seu conjunto de caso de teste (***T***), ou seja, para todas as entradas (casos de teste) a saída foi a mesma do programa original. Mutantes equivalentes são os mutantes que, qualquer que seja a entrada (caso de teste) a sua saída vai ser idêntica a do programa original. Mutante “morto” é o mutante que obteve a saída diferente do programa original a partir de um determinado caso de teste.

A Figura 3.1 exibe um pequeno exemplo de como a ferramenta Proteum/IM 2.0 faz a alteração do programa original pelo mutante.

```
// Programa Original
if (a != 10)
{
x++;
}

// Programa Mutante
if (a = = 10)
{
x++;
}
```

Figura 3.1: Exemplo de Mutante.

3.2.2 - Ferramenta PokeTool

A ferramenta PokeTool (*Potential Uses Criteria Tool for Program Testing*) (CHAIM, 1991) é uma ferramenta de teste que apóia os critérios estruturais Potenciais-Usos, Todos-Nós e Todos-Arcos e todos aqueles que se baseiam em informações do código do programa, para a seleção de dados de teste para a análise da cobertura atingida (BUENO *et al.*, 1995).

Segundo Martimiano (1999), a versão inicial da ferramenta utiliza unidades de programas escritos na linguagem C, entretanto, a ferramenta possui a característica de multilinguagem.

Por utilizarem informações relativas à estrutura interna do programa, tais critérios são chamados estruturais, e têm um enfoque comum.

Todos os critérios implementados na ferramenta buscam a caracterização de um conjunto de componentes elementares de um programa (chamados elementos requeridos) que devem ser exercitados através dos casos de teste aplicados, durante o processo de teste do programa (BUENO *et al.*, 1995).

Segundo Barbosa *et al.*, (2002), basicamente, a ferramenta PokeTool executa três operações principais:

1. **Criar a sessão de testes:** Uma sessão de teste relaciona-se a algumas atividades que compreendem o teste. Desta maneira, se o teste tiver de ser interrompido, o status do teste intermediário pode ser armazenado e recomeçado em outro momento;
2. **Testar os casos de adequação:** Incluindo a execução e a inclusão/exclusão dos exemplos de teste fornecidos pelo verificador, e
3. **Análise de adequação:** Determina a porcentagem de exigências do teste para um critério específico que seja satisfeito pelo conjunto de casos de teste. Envolve a geração de relatórios estatísticos sobre os testes conduzidos.

A ferramenta PokeTool está disponível para o ambiente DOS e UNIX. Sua versão para o DOS tem uma interface simples, baseada em menus. A versão

para UNIX tem os módulos funcionais que podem ser chamados pela linha de comando ou pela interface gráfica. A relação gráfica permite que o usuário explore e aprenda os conceitos de teste e também compreender melhor a operação da ferramenta (BARBOSA *et al.*, 2002).

Inversamente, para conduzir estudos empíricos, a linha de comandos é mais adequada, uma vez que reduzem a quantidade de interações requeridas pela relação gráfica. Barbosa *et al.*, (2002), também diz que fica claro que os scripts requerem mais esforço de programação, profunda compreensão de conceitos de teste, e informação precisa dos módulos que compõem a ferramenta.

Um outro aspecto relevante, segundo Bueno *et al.*, (1995), consiste na possibilidade de se testar diversas unidades simultaneamente. Considerando como unidade uma função em C que está sendo submetida à PokeTool, pode-se realizar a análise estática de todas unidades de um arquivo, executar casos de teste que as exercitem, e avaliar a cobertura atingida para cada unidade distinta, de acordo com o critério escolhido.

A ferramenta é orientada à sessão de trabalho, na qual o usuário entra com o programa a ser testado, com o conjunto de dados de teste e seleciona um dos critérios suportados (CHAIM, 1991).

3.3 - Automatização

As ferramentas de automatização possuem praticamente as mesmas características das outras ferramentas para teste, com a diferença que essas ferramentas possuem recursos de automatizar o teste, seja pelo fato de produzir casos de testes automaticamente de acordo com o código fonte ou até mesmo

automatizar o processo de teste, fazendo com que o testador tenha o mínimo de intervenção com o teste.

3.3.1 - JUnit

A ferramenta de teste JUnit é um framework de código aberto para o teste automatizado em programas Java, diz Rainsberger (2003). A ferramenta fornece uma maneira muito simples de expressar o modo que o testador pretende trabalhar com o código. Expressando suas intenções no próprio código fonte, o testador pode verificar como o código se comporta dentro de situações determinadas anteriormente.

Segundo Beck e Gamma (2003), é possível com a ferramenta JUnit, eliminar erros de expressões sem a necessidade de recompilar o programa e principalmente, não requer a interpretação do testador.

Rainsberger (2003) cita que são dois os objetivos principais da ferramenta: automatizar o processo de geração de casos de testes, fazendo o papel de “oráculo” no momento de decidir quais os casos de testes serão criados para determinada situação ou tipo de programas orientados a objeto; e automatizar o processo não permitindo que erros inesperados interrompam a sessão de testes, fazendo com que a sessão de testes continue sendo executada sem a necessidade da intervenção do testador.

A ferramenta JUnit possui dois tipos de execuções diferentes:

- **Execução textual:** é mais rápido e deve ser usado quando não houver necessidade de uma indicação gráfica do sucesso do teste;

- **Execução gráfica:** mostra um diálogo gráfico simples para incorporar o teste e para fornecer alguma indicação gráfica do progresso do teste. Por exemplo, pode-se configurar o ambiente de teste para que não seja necessário reiniciar a sessão de testes se o código do programa foi alterado ou não.

A ferramenta executa os seguintes passos na etapa de automatização de teste:

1. Criar objetos;
2. Enviar mensagens a esses objetos;
3. Verificar as asserções definidas.

Beck e Gamma (1998) dizem que erros inesperados podem fazer com que o programa que está sendo testado seja interrompido inesperadamente, fazendo com que o processo de teste seja inútil. A ferramenta JUnit distingue entre falhas e erros. A possibilidade de uma falha é antecipada com o uso das assertivas. Erros são problemas inesperados. A Figura 3.2 mostra como a ferramenta trata e distingue os erros das falhas.

```
public void run(TestResult result) {
    result.startTest(this);
    setUp();
    try {
        runTest();
    }
    catch (AssertionFailedError e) { //(A)
        result.addFailure(this, e);
    }
    catch (Throwable e) {           //(B)
        result.addError(this, e);
    }
    finally {
        tearDown();
    }
}
```

Figura 3.2: Tratamento e distinção de falhas pela ferramenta JUnit (BECK e GAMMA, 1998).

Para distinguir um erro inesperado de uma falha, são coletadas falhas em uma cláusula de captura (***catch***) extra (***A***). A segunda cláusula (***B***) captura todas

as outras exceções e assegura que a sessão de testes continua independendo se houve um erro ou uma falha.

3.3.2 - HttpUnit

A ferramenta HttpUnit é um conjunto de classes em Java para testar aplicações de WEB utlizando-se do protocolo HTTP. Junto com a ferramenta JUnit, HttpUnit é uma ferramenta poderosa por criar sessões de teste para assegurar a funcionalidade das aplicações WEB, diz Scheinblum (2004). Segundo Bhogal (2004), o HttpUnit pode emular vários comportamentos de browser, inclusive requisições, JavaScript, autenticação HTTP e uso de *cookies*.

A principal diferença entre as ferramentas JUnit e HttpUnit está no fato que, enquanto o próprio JUnit testa o código em Java construindo testes para classes individuais, a ferramenta HttpUnit estende as funcionalidades do JUnit para testar a integração da aplicação com a WEB, emulando o modo como um browser trabalha e assim, trabalhando como um servidor de WEB.

Scheinblum (2004) cita que outro grande aspecto da ferramenta HttpUnit é que ela é capaz de testar aplicações WEB inteiras. Isto se deve porque a ferramenta se baseia em *cookies* gerados no uso e na navegação pela aplicação WEB, e assim, se baseando nesses *cookies* gerados, o testador pode gerar casos de testes para uma aplicação inteira. Considerando que os testes são escritos em linguagem Java, não há nenhum limite de como os testes serão detalhados.

Segundo Bhogal (2004), diferentemente como o nome da ferramenta pode insinuar, a HttpUnit não faz teste de unidade, mas sim, o teste funcional ou “caixa-preta”, pois os testes realizados por essa ferramenta examinam

externamente a comunicação entre servidores, analisando as mensagens recebidas e as enviadas, permitindo uma avaliação mais detalhada pelo testador.

3.4 - Record & Playback

Como já foi descrito em outras seções desse trabalho, ferramentas que auxiliem e dinamizem o processo de teste são bem-vindas na hora de testar essas aplicações. As ferramentas de Record e Playback auxiliam no processo de teste, como por exemplo, em um ambiente de teste de regressão, pois automatizam esse processo, não necessitando ao testador refazer todos os casos de testes para que a aplicação seja verificada novamente.

O processo de Record e Playback essencialmente envolve capturar todos os eventos que são gerados durante uma sessão de teste, e reproduzir esses mesmos eventos exatamente na mesma seqüência em que foram gerados, segundo Khetawat, Lavana e Brglez (1997). Evento é a ocorrência de uma interação entre o usuário e um processo do sistema a ser testado.

Khetawat, Lavana e Brglez (1997) dizem que um evento no processo de "*Recording & Playing Back*" pode ser também definido pelas ações do usuário ou testador com a aplicação, como um clique do mouse, o acionamento da tecla "*enter*", o clicar de um botão ou qualquer outro tipo de interação do usuário ou testador com o programa.

Nas seções seguintes, resumidamente, serão discutidas duas ferramentas que realizam o processo de Record e Playback.

3.4.1 - Ferramenta DeJaVu

DeJaVu (*Deterministic Java replay Utility*) é uma ferramenta de teste para *"Record"* e *"Playback"* de programas escritos em Java, principalmente de aplicações desenvolvidas para servidores, segundo Alpern *et al.*, (2000).

A ferramenta DeJaVu faz parte do *"Jalapeño"*, uma máquina virtual Java, desenvolvida pela IBM para servidores de alta performance, segundo Choi e Zeller (2003) e Alpern *et al.*, (2000).

A máquina virtual "Jalapeño" trabalha em sistemas "uniprocessados", onde os programas são executados de acordo com a fatia de tempo que a máquina dispõe a esses programas. Os programas, por sua vez, executam uma quantia de eventos até que sua fatia de tempo acabe ou alguma interrupção finalize sua execução.

A ferramenta DeJaVu divide as operações de uma aplicação e seu tempo de uso em operações determinísticas (como operações de execuções), que necessariamente produzem o mesmo resultado quando se utiliza o Replay, e operações não-determinísticas (como operações casuais), que não produzem o mesmo resultado.

O objetivo da ferramenta DeJaVu é reproduzir erros, ou seja, toda vez que um erro não-determinístico é capturado, pode ser executado de uma forma determinística novamente, cita Alpern *et al.*, (2000).

No modo de Record, DeJaVu ignora operações determinísticas enquanto registra os resultados das operações não-determinísticas. Em modo de Playback, ignora as operações determinísticas enquanto aplica sistematicamente as

operações não-determinísticas previamente gravadas pelo modo de Record, mas aplicadas dessa vez, de forma determinística, diz Alpern *et al.*, (2000).

3.4.2 - Ferramenta TestTube

A ferramenta TestTube, segundo Rosenblum e Rothernel (1997) e Chen, Rosenblum e Vo (1994), se utiliza de *"Recording & Playing Back"*, auxiliando o teste de regressão, baseada na cobertura de todos os caminhos de funções em que os casos de testes são executados. A ferramenta foi desenvolvida para trabalhar com programas escritos na linguagem de programação C.

Na execução da ferramenta TestTube na linguagem C, cada caso de teste é associado com os tipos e variáveis das definições da função que está sendo testada.

Segundo Chen, Rosenblum e Vo (1994), para recolher esta informação da cobertura, são geradas marcas sob o sistema que está sendo testado de todas as funções chamadas durante sua execução. A execução de um caso de teste no sistema gera desse modo uma marca nas funções cobertas por esse caso de teste.

Após a execução do teste de cobertura, uma base de dados é gravada com o código fonte da aplicação testada.

A base de dados grava todas as referências entre as entidades e funções do código fonte do sistema testado. Para cada função coberta por um caso de teste, são computados nas referências da função as variáveis globais, tipos e macros, identificando desse modo as variáveis, os tipos e os macros globais que o caso de teste cobre potencialmente (ROSENBLUM e ROTHERNEL, 1997).

Quando uma nova versão do sistema é criada, uma base de dados de casos de teste do código fonte está pronta para a nova versão. As bases de dados para as novas e velhas versões são comparadas para identificar as entidades e funções que foram mudadas, sendo que esta comparação é baseada nas somas de controle que são computadas sobre as definições de cada entidade, diz Chen, Rosenblum e Vo (1994).

Rosenblum e Rothernel (1997) dizem que a lista de entidades alteradas é comparada então com a informação da cobertura preservada para cada caso do teste, e todos os casos do teste que cobrem as entidades ou funções alteradas são selecionados testando assim, a nova versão.

Chen, Rosenblum e Vo (1994) citam que uma mudança a uma entidade ou função, no novo programa, provoca a seleção de todos os casos do teste que cobrem essa entidade.

Ainda segundo Chen, Rosenblum e Vo (1994) e Rosenblum e Rothernel (1997), a ferramenta TestTube utiliza uma técnica segura para se fazer Teste de Regressão. A única exceção a esta ferramenta é para os sistemas “não-determinísticos”, em que as execuções múltiplas do mesmo caso de teste podem produzir trajetos diferentes na execução do programa.

CAPÍTULO 4 – TESTE DE APLICAÇÕES WEB

Segundo Ricca e Tonella (2001), a relevância econômica das aplicações WEB aumentou a importância de controlar e de melhorar sua qualidade, e com isso, está surgindo uma elevada demanda de metodologias e ferramentas para garantir a qualidade dessas aplicações.

Sendo assim, as técnicas e recursos para o teste de software foram sendo adaptados para esse novo paradigma de desenvolvimento. Será discutido nesse Capítulo os testes voltados para as aplicações WEB, suas técnicas e suas características.

Na Seção 4.1 é apresentado as ferramentas ReWeb e TestWeb, na Seção 4.2 é discutido as técnicas de Recuperação de Arquiteturas de Aplicações WEB e a ferramenta jWebUnit é apresentada na Seção 4.3.

A técnica utilizada para testar aplicações WEB adota os mesmos princípios básicos de todo teste de software, mas aplica também conceitos de táticas e estratégias que foram adquiridas para sistemas orientados a objetos e também desenvolvidos para uma arquitetura cliente / servidor. Pressman (2002) mostra que esses princípios podem ser definidos como:

1. **O modelo de conteúdo das Aplicações WEB é revisto para descobrir erros.** Essa atividade de teste é semelhante em muitos aspectos à revisão de um documento escrito, assim, um site de grande porte da WEB poderia contratar os serviços de um editor profissional de textos para descobrir erros de grafia, erros gramaticais, erros de consistência no conteúdo, erros de representação gráfica e erros de referência cruzada.

2. **O modelo de projeto da Aplicação WEB é revisto para descobrir erros de navegação**. Casos de uso, derivados como parte da atividade de análise, permitem a um engenheiro da WEB exercitar cada cenário de uso, com base no projeto arquitetural e de navegação.
3. **Componentes de processamento selecionados e páginas da WEB são sujeitas à teste de unidade**. Quando as aplicações WEB são consideradas, o conceito de unidade se modifica. Cada página da WEB encapsula conteúdo, ligações de navegação e elementos de processamento. Nem sempre é possível ou prático testar cada uma dessas características individualmente. Em muitos casos, a menor unidade testável é a página da WEB. Diferentemente do teste de unidade de software convencional, que tende a focalizar o detalhe algorítmico de um módulo e os dados que fluem através da interface do módulo, o teste de página para as aplicações WEB é guiado por conteúdo, sendo processado em ligações encapsuladas na página da WEB.
4. **A arquitetura é construída e testes de integração são conduzidos.** A estratégia para teste de integração depende da arquitetura que foi escolhida para a Aplicação WEB. Se a aplicação foi projetada com uma estrutura linear, em malha, ou simplesmente hierárquica, é possível integrar páginas da WEB de um modo muito semelhante ao que usamos para integrar módulos de software convencional, mas, se uma arquitetura de hierarquia mista, ou de WEB é usada, o teste de integração é semelhante à abordagem usada para sistemas orientados a objetos.

5. **A Aplicação WEB desenvolvida é testada quanto à funcionalidade global e à entrega de conteúdo.** Como a validação convencional, a validação de sistemas e aplicações baseadas na WEB focaliza as ações visíveis do usuário e as saídas reconhecidas pelo usuário do sistema. Para ajudar a derivação de testes de validação, o testador deve se apoiar nos casos de uso, que fornece um cenário com grande probabilidade de descobrir erros nos requisitos de interação com o usuário.
6. **A Aplicação WEB é implementada numa variedade de configurações ambientais diferentes e é testada quanto à compatibilidade com cada configuração.** É criada uma matriz de referência cruzada que define todos os prováveis sistemas operacionais, browsers, plataformas de hardware e protocolos de comunicação, e então, são realizados os testes para descobrir erros associados com cada possível configuração.
7. **A Aplicação WEB é testada por uma população de usuários finais controlada e monitorada.** Uma população de usuários que abrange cada característica possível de usuário é escolhida. A Aplicação WEB é exercitada por esses usuários e o resultado de sua interação com o sistema é avaliado quanto a erros de conteúdo e de navegação, preocupações de utilização e preocupações quanto à compatibilidade e à confiabilidade de desempenho da aplicação.

Segundo Wu e Offutt (2002), um aspecto fundamental, e talvez a parte mais complicada de analisar e testar aplicações WEB está no fato de controlar a

natureza dinâmica do software. As características dinâmicas são causadas por incertezas no comportamento do programa, como mudanças nas exigências da aplicação e a rápida evolução da tecnologia WEB.

4.1 - Ferramenta ReWeb e TestWeb

As ferramentas ReWeb e TestWeb foram desenvolvidas para suportar a análise e testar aplicações WEB (RICCA e TONELLA, 2001).

A ferramenta ReWeb armazena e analisa as páginas de uma aplicação ou site WEB com a finalidade de construir um modelo UML dessa aplicação. Já a ferramenta TestWeb gera e executa um conjunto de casos de teste para uma aplicação WEB cujo modelo seja computado pela ReWeb, sendo o processo inteiro semi-automático, e as poucas intervenções necessárias do usuário são indicadas pelos losangos, mostrado na Figura 4.1.

Segundo Ricca e Tonella (2001), são funções das ferramentas:

- **Testar página**: cada página no site é visitada ao menos uma vez em algum caso de teste;
- **Testar hyperlink**: cada hyperlink de página ou recurso no site testado é acessado pelo menos uma vez;
- **Testar Definição-uso**: são exercitados todos os trajetos de navegação onde há o uso da variável ou caso de teste definidos no início do teste;
- **Testar Todos-usos**: é exercitado ao menos um trajeto de navegação onde há o uso de cada variável ou caso de teste definidos no início do teste;

- **Testar Todos-caminhos**: cada trajeto no site é acessado ao menos uma vez.

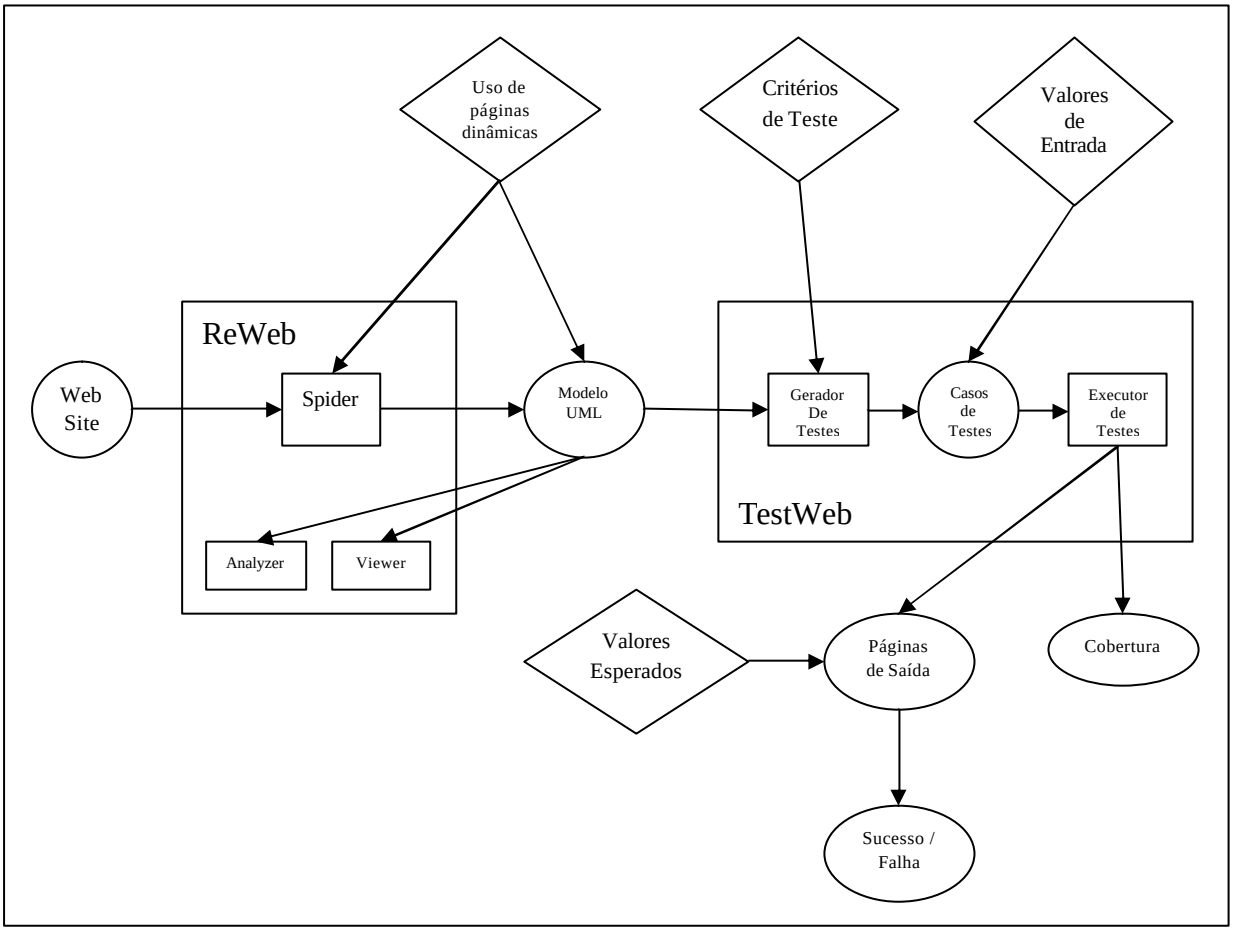

Figura 4.1: Diagrama de funcionamento das ferramentas ReWeb e TestWeb (RICCA e TONELLA, 2001).

A ferramenta ReWeb consiste em três módulos:

1. **Spider**: grava todas as páginas de um WEB site alvo do teste, a partir de um URL. Cada página encontrada no host que corresponde a URL requisitada é gravada com a data de download. As páginas de um host são obtidas emitindo os pedidos associados às requisições feitas pelo testador. O resultado de tais pedidos é sempre uma página no formato

HTML, de modo que não seja possível discriminar entre páginas dinâmicas e estáticas;

2. **Analisador**: utiliza a navegação e interação dos hiperlink's e frames como informação para a análise. São considerados documentos e caminhos acessados pela aplicação testada. A partir dessas informações gravadas a ferramenta analisa os dados e constrói um modelo UML do site que está sendo testado;
3. **View** (**visor**): fornece uma interface de usuário gráfica para indicar o modelo da aplicação WEB. A interface gráfica suporta um grande conjunto de facilidades da navegação e busca incluindo zoom e busca, o foco e código HTML. Entre as visualizações disponíveis, a ferramenta mostra o histórico de visitas aos URL's, a organização das páginas em diretórios e o fluxo de dados dos acessos de leitura e gravação das páginas testadas.

Após a execução, o coordenador de teste intervém para avaliar o resultado de sucesso ou falha de cada caso de teste. Para isso, o testador abre as páginas da saída em um browser e verifica se a saída está correta para cada entrada dada.

Segundo Ricca e Tonella (2001), a ferramenta TestWeb contém um mecanismo de geração de casos de testes capaz de determinar a expressão do trajeto do modelo de uma aplicação WEB, e de gerar casos de teste para essa situação, contanto que um critério para este tipo de teste seja especificado.

Os testes de casos gerados são as seqüências de URL´s que, uma vez que executadas, concedem a cobertura do critério selecionado. Os valores da entrada

em cada seqüência do URL são deixados vazios pelo gerador do teste, e o testador tem de preenchê-los, possivelmente explorando as técnicas de teste usadas no tradicional teste “caixa-preta” (RICCA e TONELLA, 2001).

O mecanismo de teste da ferramenta TestWeb pode fornecer a seqüência de URL de cada caso de teste ao testador, unindo entradas apropriadas a cada formulário requisitado para o teste.

As páginas de saídas produzidas pelo usuário são armazenadas para um exame posterior mais específico.

4.2 - Recuperação de Arquiteturas de Aplicações WEB

Uma outra maneira de se analisar e testar aplicações WEB são através de métodos que auxiliem no desenvolvimento desses sistemas. Segundo Hassan (2002), os desenvolvedores de aplicações WEB necessitam uma melhor compreensão de seus sistemas de software e, com a velocidade do desenvolvimento que esses programas necessitam e a alta taxa de mudança aumentou a necessidade de adotar ferramentas para gerenciar esses projetos de desenvolvimento para WEB. Hassan (2002) diz ainda que mesmo as aplicações WEB sendo uma realidade, é notória a falta de documentação, e quando existem, são complexas.

Hassan e Holt (2001) mostram que os desenvolvedores de software têm necessidades e interesses diferentes quando há a necessidade de se estudar e analisar as aplicações WEB, assim, apresenta um novo modelo para recuperação da arquitetura de uma aplicação WEB.

O modelo apresentado por Hassan (2002) é formado por três camadas:

1. **Entity Level Schema** (*ELS*): camada para o nível de entidade;
2. **Component Level Schema** (*CLS*): camada para o nível de componentes, e
3. **Architecture Level Schema** (*ALS*): camada para o nível de arquitetura da aplicação.

A camada situada em um nível mais baixo está a *ELS*, que descreve todas as relações permitidas entre as entidades "top-level" da aplicação tais como funções, variáveis, os objetos do usuário, tabelas de banco de dados e objetos distribuídos. Segundo Hassan (2002), a camada *ELS* está voltada para as duas das linguagens de programação mais intensamente usadas no desenvolvimento de aplicações WEB: Javascript e VBScript.

A camada *CLS* eleva o nível de abstração ao nível de componente. Descreve todas as relações permitidas entre os diferentes componentes de uma aplicação WEB tal como as páginas ativas do usuário, tabelas de banco de dados, bibliotecas e objetos distribuídos.

Já o nível mais elevado de abstração, ainda segundo Hassan (2002), encontra-se a *ALS*, que descreve todas as relações permitidas entre os elementos da arquitetura tais como subsistemas e componentes.

O processo da extração dessas camadas emprega diferentes tipos de métodos extratores. Cada extrator é invocado somente na seção ou componente apropriado para que foi desenvolvido, havendo assim, extratores para o código HTML, Scripts, acessos ao banco de dados, um extrator de linguagens e um extrator binário.

Os resultados da execução de todos os extratores em todos os componentes da aplicação WEB são combinados em uma única base de dados para posterior avaliação.

Uma vez que é descrito todos os níveis e relações entre todos os componentes da aplicação WEB, um melhor entendimento dessa aplicação é alcançado pelo desenvolvedor, podendo, através dessa estrutura, modificar, corrigir ou expandir a aplicação WEB (HASSAN e HOLT, 2001).

4.3 – jWebUnit

jWebUnit é um framework Java que facilita a criação de testes de para aplicações WEB. Esse framework é uma evolução do projeto HttpUnit e JUnit, utilizados para criar testes de automatização, dizem Weaver e Joiner (2004). Como já descrito anteriormente, as aplicações voltadas para o ambiente WEB sempre estão sendo modificadas ou atualizadas, sendo assim, eram continuamente sendo refeitas, e para que essas aplicações possam ser testadas com segurança e confiabilidade, foi criada a ferramenta jWebUnit.

A ferramenta jWebUnit contém uma API de alto-nível para o teste em uma aplicação WEB, combinando com um conjunto de assertivas para verificar se a aplicação em questão está correta ou não. Isto inclui a navegação por link´s, dados de entrada, validação de tabelas de conteúdo e outras características de aplicações WEB (WEAVER e JOINER, 2004).

A ferramenta jWebUnit utiliza os códigos HttpUnit, mas como se utiliza de métodos de navegação mais simples e intuitivo, as sessões de teste são

criadas de uma maneira mais rápida, comparando ao utilizar as ferramentas JUnit ou HttpUnit, segundo Weaver e Joiner (2004).

A Figura 4.2 mostra o código necessário para se fazer o teste em uma aplicação WEB que faça uma busca no site Google, utilizando-se de algumas palavras-chave, usando a ferramenta JUnit. Os códigos necessários para que o teste seja realizado estão não estão claros, fazendo com que conhecimentos técnicos da linguagem seja de conhecimento do usuário testador.

```
package net.sourceforge.jwebunit.sample;
import junit.framework.TestCase;
import com.meterware.httpunit.WebResponse;
import com.meterware.httpunit.WebConversation;
import com.meterware.httpunit.WebForm;
import com.meterware.httpunit.WebRequest;
public class SearchExample extends TestCase {
public void testSearch() throws Exception {
      WebConversation wc = new WebConversation();
      WebResponse resp = wc.getResponse( "http://www.google.com");
      WebForm form = resp.getForms()[0];
      form.setParameter("q", "HttpUnit");
      WebRequest req = form.getRequest("btnG");
       resp = wc.getResponse(req);
       assertNotNull(resp.getLinkWith("HttpUnit"));
       resp = resp.getLinkWith("HttpUnit").click();
      assertEquals(resp.getTitle(), "HttpUnit");
      assertNotNull(resp.getLinkWith("User's Manual"));
   }
}
```

Figura 4.2: Exemplo de um caso de teste utilizando a ferramenta JUnit (WEAVER e JOINER, 2004).

A Figura 4.3 ilustra o mesmo caso de teste, só que se utilizando da ferramenta jWebUnit. O código utilizado no teste é bem mais simples, onde classes foram criadas, organizando o código fonte testado e facilitando o entendimento do usuário testador e agilizando o processo, tanto na codificação, quanto no entendimento do teste a ser realizado. Com a separação em métodos e classes, a leitura do código da aplicação a ser testada fica mais claro e intuitivo, mesmo para usuários testadores mais inexperientes.

```
package net.sourceforge.jwebunit.sample;
import net.sourceforge.jwebunit.WebTestCase;
public class JWebUnitSearchExample extends WebTestCase {
   public JWebUnitSearchExample(String name) {
      super(name);
   }

   public void setUp() {
      getTestContext().setBaseUrl("http://www.google.com");
   }
   public void testSearch() {
      beginAt("/");
      setFormElement("q", "httpunit");
      submit("btnG");
      clickLinkWithText("HttpUnit");
      assertTitleEquals("HttpUnit");
      assertLinkPresentWithText("User's Manual");
   }
}
```

Figura 4.3: Exemplo de um caso de teste utilizando a ferramenta jWebUnit (WEAVER e JOINER, 2004).

4.4 – FireWeb

A ferramenta FireWeb que visa auxiliar os testadores de aplicações Web na tarefa de selecionar dentre os casos de testes existentes, aqueles que deverão ser reexecutados durante os testes de regressão (FIDELIS e MARTINS, 2004).

A ferramenta procura auxiliar não somente os testes de regressão, mas também a manutenção de sistemas. Fidelis e Martins (2004), dizem que, para isso, a ferramenta mantém algumas rastreabilidades (associações), que permitem "caminhar" entre os diferentes níveis de abstração do sistema, como por exemplo, a partir dos casos de uso, chegar aos componentes que os implementam, o que torna mais fácil o entendimento do sistema.

O número de componentes em um sistema Web tende a ser muito grande, podendo uma única página Web ser formada por vários componentes, entre "*frames*1", *"forms"* (formulários) e arquivos *"include2"*, com isso, o trabalho de manter a associação entre casos de testes e componentes pode ser muito

complicado e trabalhoso. A ferramenta não gera casos de testes, apenas localiza os casos de testes já existentes e que deverão ser reexecutados.

Para se conseguir selecionar os casos de testes que farão parte dos testes de regressão utilizando a ferramenta, três fases deverão ser executadas, que são (FIDELIS e MARTINS, 2004):

- **Preparação da infra-estrutura:** Consiste na criação de uma base de dados constituída pelos casos de uso que compõem o software sob teste, seus componentes, casos de testes e os relacionamentos entre eles.
- **Obtenção do Firewall:** O *Firewall,* composto pelos componentes cujos casos de testes deverão ser reexecutados, é obtido através da execução dos seguintes passos:
    a) Identificação dos componentes que “usam” e “são usados” pelos componentes modificados.
    b) Identificação dos casos de uso que são implementados pelos componentes.
    c) Identificação dos casos de testes que exercitam os casos de uso localizados.
- **Refinamento dos casos de testes:** Uma vez encontrados os casos de testes relacionados ao firewall, os mesmos poderão precisar sofrer atualizações. Nesse ponto deverão ser observados, por exemplo, se os valores de entrada expressos nos casos de testes ainda são compatíveis com os respectivos componentes, sendo que essa etapa deve ser executada manualmente.

Segundo ainda Fidelis e Martins (2004). A ferramenta apresenta algumas limitações no seu uso, pois está preparada para analisar dependências de código apenas entre: páginas WEB desenvolvidas em ASP (*Active Server Pages*), programas COBOL e arquivos “*include*”.

CAPÍTULO 5 – FERRAMENTA PROPOSTA

Este capítulo apresenta o protótipo da ferramenta para Record e Playback desenvolvida durante o projeto, bem como a discussão dos resultados obtidos, benfeitorias e limitações apresentadas pelo protótipo.

A Seção 5.1 mostra o exemplo funcional da ferramenta para Record e Playback proposta, com os respectivos lay-outs de tela e suas funcionalidades.

A Seção 5.2 apresenta os recursos e ferramentas utilizados na implementação da ferramenta, utilizando como base para o desenvolvimento o proxy jHTTPp2.

5.1 – Características Operacionais

Esta seção apresenta as características operacionais da ferramenta de Record e Playback, mostrando em exemplos funcionais, a sua configuração e uso.

5.1.1 – Configuração da Ferramenta de Teste

A configuração do proxy de teste começa com a configuração da porta de comunicação que o browser que está sendo utilizado deve receber e gravar os dados que são transferidos entre o cliente (requisição) e o servidor (resposta). A ferramenta está programada para trabalhar com a porta de comunicação “3000”.

O browser também deve estar preparado para trabalhar como "`localhost`", sendo o seu IP (Internet Protocol) configurado para o número padrão "`127.0.0.1`".

Para que a ferramenta entre em execução, é necessário que se ative o proxy para trabalhar com a comunicação na porta 3000, previamente configurada no browser utilizado. Essa execução deve ser feita em um ambiente Java, sendo necessário o pacote de ferramentas da Sun (www.sun.com) para o desenvolvimento em Java (J2SDK), na versão mínima requerida 1.4.

Após as etapas de configuração do browser e da ferramenta, o proxy já está ativado e a ferramenta fica aguardando os comandos que devem ser digitados na barra de endereços no browser utilizado.

5.1.2 – A Utilização da Ferramenta

A ferramenta apresentada está preparada para realizar testes em aplicações WEB utilizando técnicas de Record e de Playback. Abaixo são descritos os comandos para o uso da ferramenta, sendo utilizado para o exemplo, o URL *www.uol.com.br*:

- **Record**: o comando abaixo, digitado a partir da barra de endereços do navegador, é interceptado pelo proxy e leva à criação de uma nova sessão de testes. Como parâmetro, deve ser fornecido o nome da sessão de testes que se deseja criar logo após a vírgula final:

`http://localhost:3000/admin/jp2-config/test-proxy-config?record,`

A Figura 5.1 mostra o exemplo de criação de uma sessão de testes chamada TESTE1. Logo após a introdução do comando na barra de endereços, o browser exibe uma mensagem dizendo que a sessão em questão foi criada com sucesso, e logo abaixo, o pedido para que um URL seja digitado para que seja gravado como caso de teste “0” (zero).

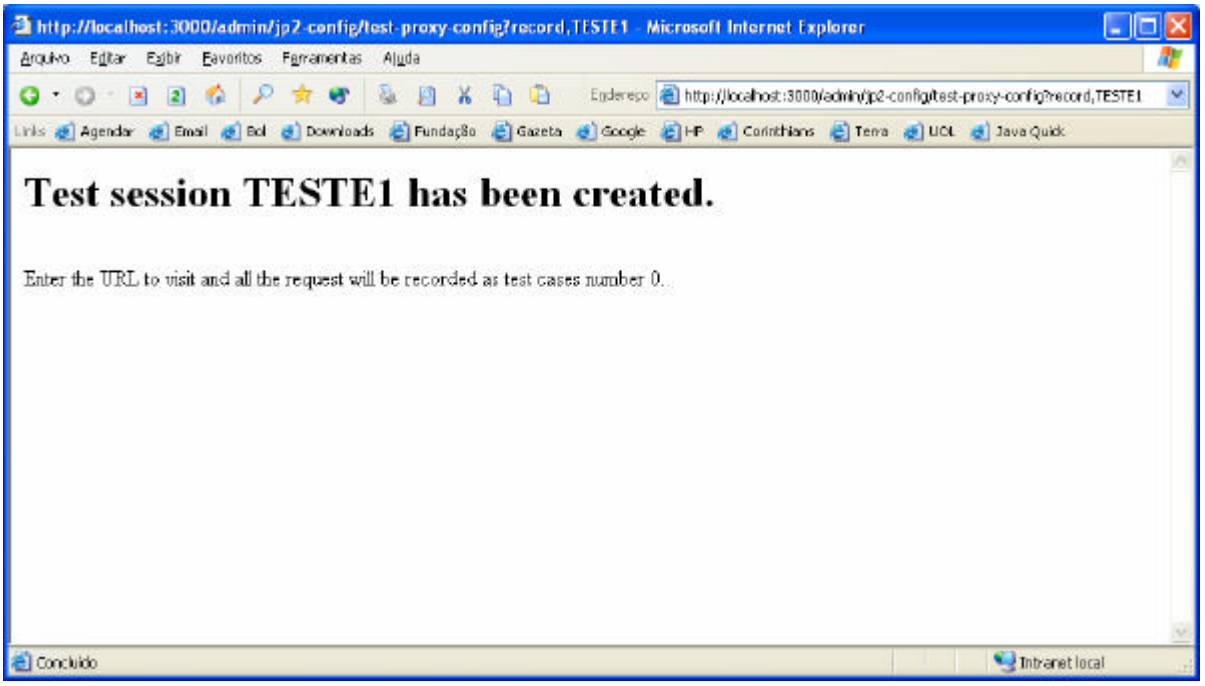

Figura 5.1: Criação de uma sessão de testes chamada TESTE1.

Se uma sessão de teste já foi criada, e o comando acima for executado, o browser exibirá uma mensagem que a sessão de teste foi encontrada, como mostra a Figura 5.2, e os casos de testes serão gravados normalmente, dando seqüência à numeração dos casos de testes já gravados, ou seja, os casos de testes não são sobre gravados pelos novos casos que ainda vão ser requisitados.

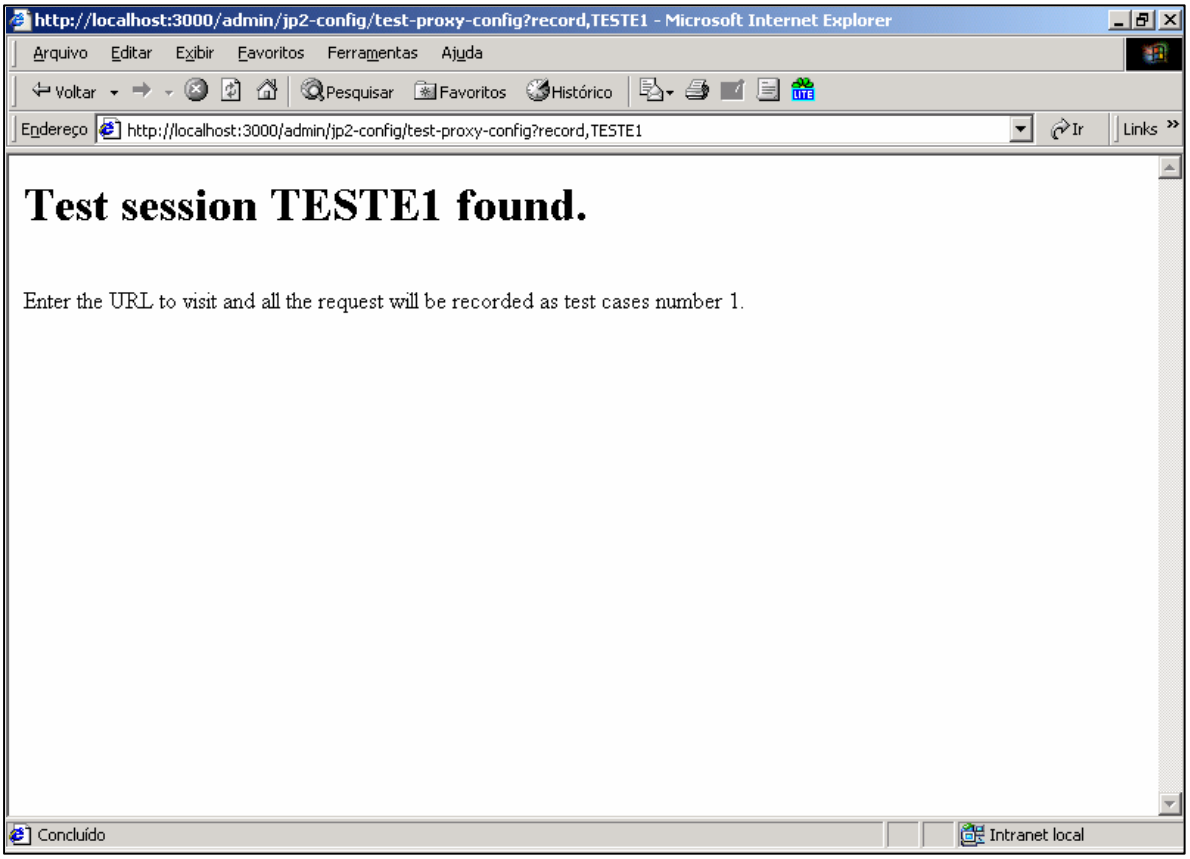

Figura 5.2: Mensagem exibida quando encontrada uma sessão já criada.

Após a criação da sessão, o endereço da aplicação a ser testada é digitado na barra de URL do browser, e assim, exibido como mostra a Figura 5.3. Simultaneamente, a ferramenta vai armazenando todas as requisições feitas pelo cliente e as respostas fornecidas pelo servidor, caracterizando assim, os casos de testes para a sessão.

- **End**: o comando abaixo é utilizado para encerrar a gravação de casos de testes em uma sessão de teste. A ferramenta encerra a gravação das requisições do cliente e respostas fornecidas pelo servidor, finalizando a sessão de testes.

  Para a finalização da gravação, basta fornecer o nome da sessão que se deseja encerrar após a vírgula do comando:

```
http://localhost:3000/admin/jp2-config/test-proxy-config?end,
```

Figura 5.3: Tela exibindo o URL requisitado após ser executado o comando "record".

A Figura 5.4 mostra o exemplo de encerramento da sessão de testes chamada TESTE1. Logo após a digitação do comando na barra de endereços, o browser exibe uma mensagem dizendo que a sessão em questão foi encerrada com sucesso.

- **Info**: o comando "**`info`**", descrito a seguir, exibe para o usuário as informações que foram gravadas na sessão referenciada logo após a vírgula final do comando:

`http://localhost:3000/admin/jp2-config/test-proxy-config?info,`

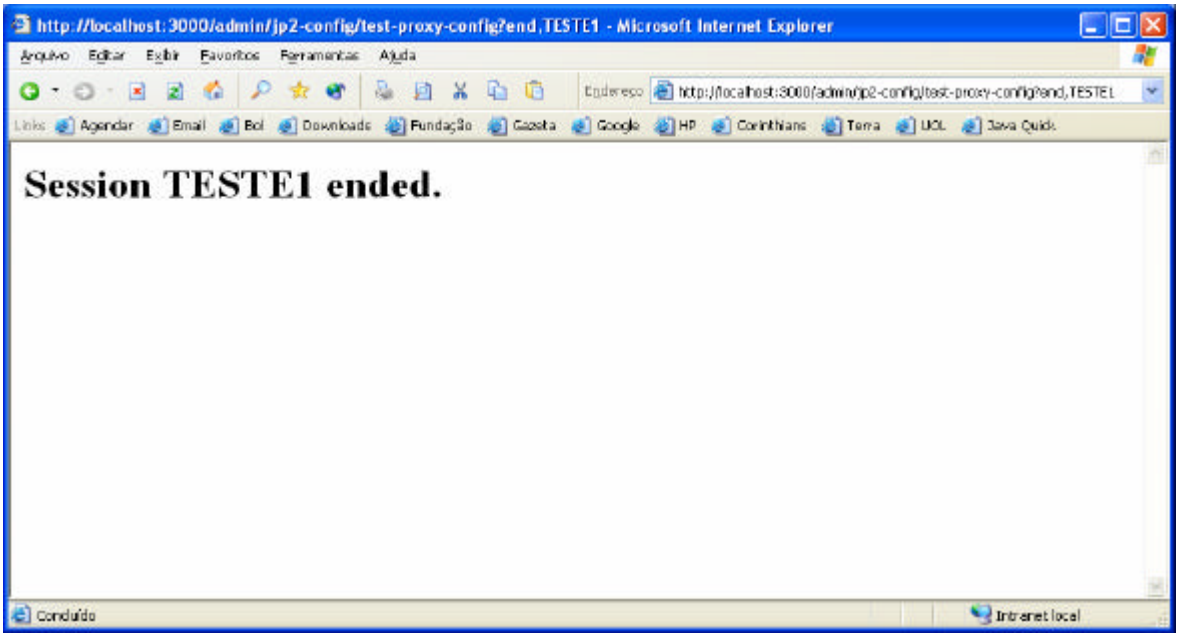

Figura 5.4: Encerramento de uma sessão de testes chamada TESTE1.

As informações exibidas pelo comando "**`info`**" são os casos de testes, numerados sequencialmente, como mostra a Figura 5.5, que apresenta a sessão TESTE1. Ela possui dois casos de testes (casos de testes "0" e "1"). Além dos dois casos de testes da sessão, também são exibidos quais e quantas foram as requisições e respostas efetuadas. Deve-se notar que para cada arquivo com o recurso (*Request+sufixo.txt*) solicitado pelo cliente, é gravado um arquivo com a resposta do servidor (*Reply+sufixo.txt*), além de um arquivo com o URL do recurso requisitado (*URL.txt*).

Como exemplo, as Figuras 5.6, 5.7 e 5.8 mostram o conteúdo, respectivamente, dos arquivos "Request49337.txt", "Reply49336.txt", e o arquivo "URL.txt" da requisição "2" do caso de teste "0". Cada item mostrando na Figura 5.5 é um hiperlink que o usuário pode seguir para visualizar o recurso correspondente. No exemplo, o recurso requisitado e fornecido pelo servidor é uma figura chamada "`sophia.gif`".

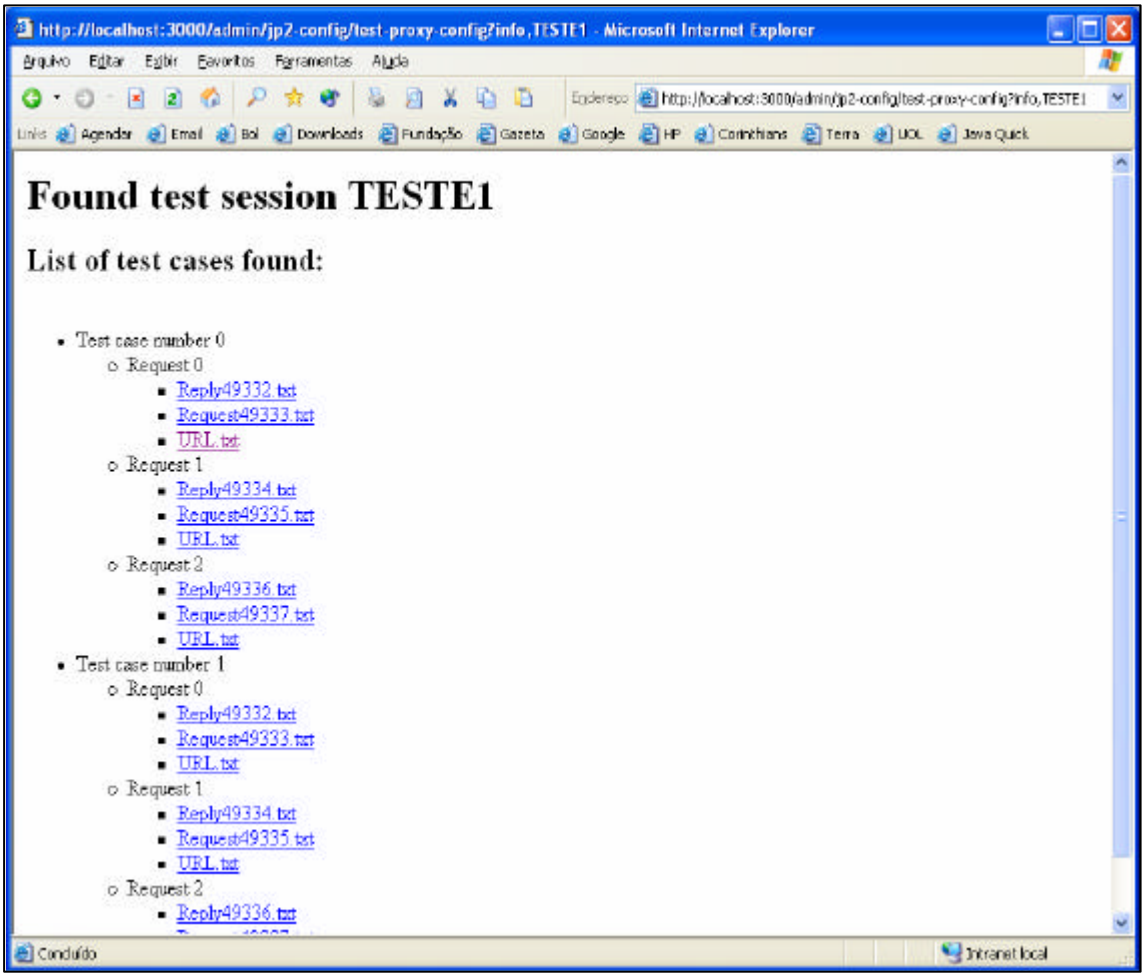

Figura 5.5: Informações gravadas na sessão TESTE1.

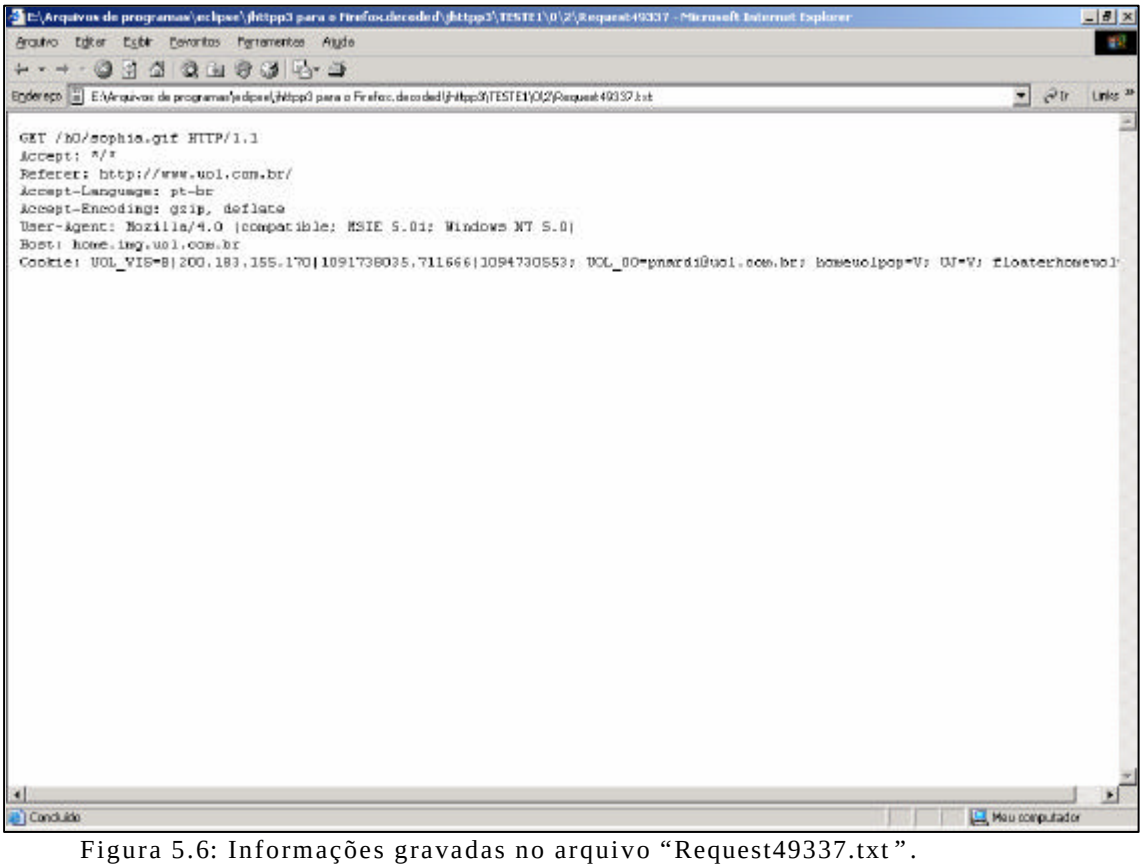

Figura 5.6: Informações gravadas no arquivo “Request49337.txt”.

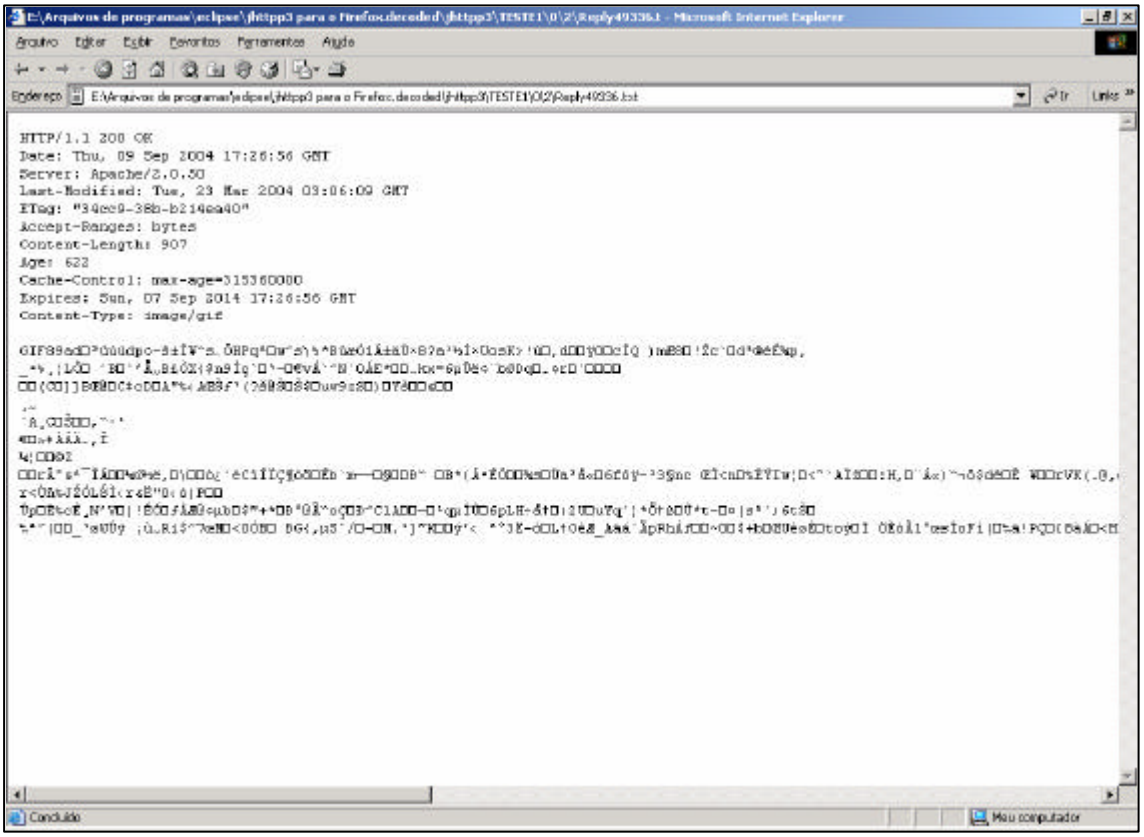

Figura 5.7: Informações gravadas no arquivo “Reply49336.txt”.

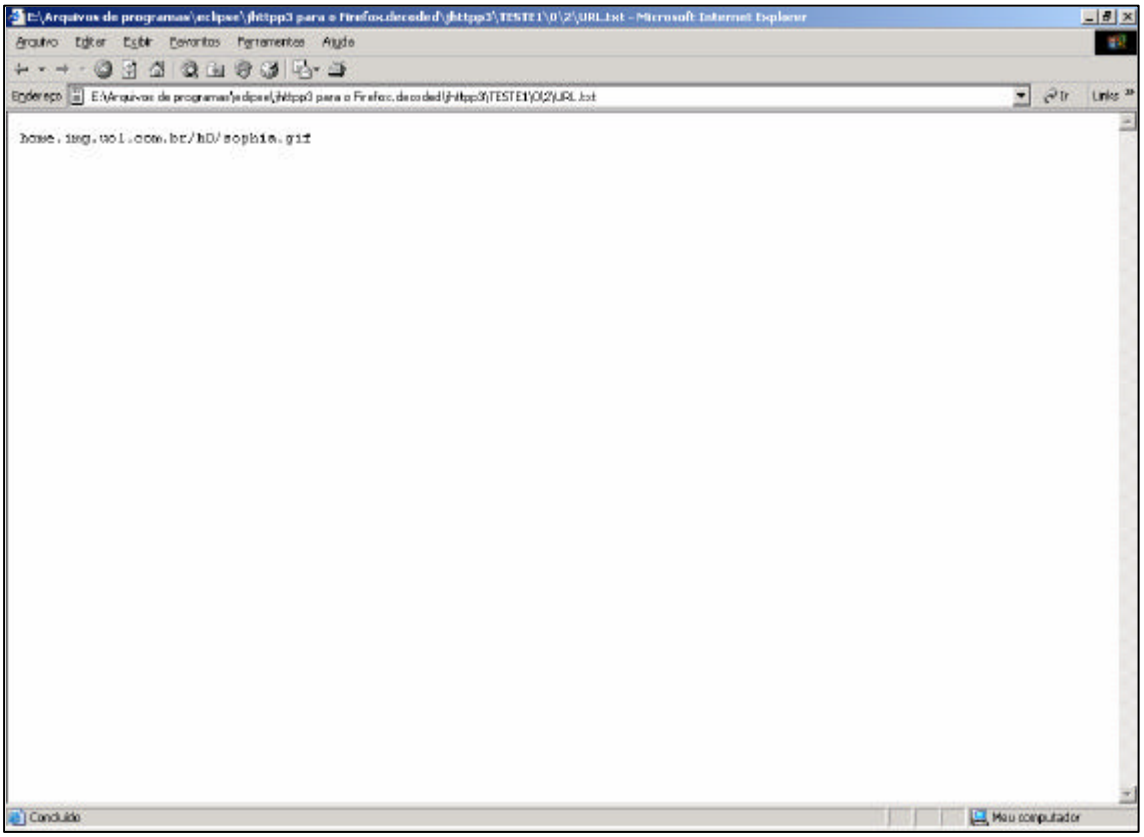

Figura 5.8: Informações gravadas no arquivo “URL.txt”.

- **Replay**: esse comando irá realizar o recurso de “playback” de um caso de testes. Para realizar a operação, basta inserir o nome da sessão e o número do caso de teste que se deseja fazer o “playback” após a vírgula final do comando abaixo, executado a partir da barra de endereços do browser:

`http://localhost:3000/admin/jp2-config/test-proxy-config?replay,`

Após ser fornecido o comando citado acima, uma janela do browser (Figura 5.9) será exibida, mostrando o endereço inicial correspondente ao caso de teste. O usuário deve clicar no link que é exibido pelo browser e assim, na janela principal a ferramenta irá exibir os dados gravados pelo proxy no comando de “**`record`**”.

Em uma outra janela, são exibidas as informações atualizadas que são obtidas diretamente no servidor WEB onde estão gravadas as informações que compõem os recursos do URL requisitado. Essa outra janela exibida pela ferramenta proposta será fornecida pelo WEB Browser Mozilla Firefox, pois com as configurações utilizadas pela ferramenta de teste proxy no browser principal é necessário um browser que seja executado sem essas configurações da ferramenta proxy. O browser Mozilla Firefox não deve estar configurado (portas de comunicação) para trabalhar em conjunto com a ferramenta, para assim, acessar sem a interferência do proxy os recursos disponíveis na WEB.

Para se executar todos os casos de teste gravados, o usuário da ferramenta deve executar o comando “`Replay`” informando todos os casos de teste.

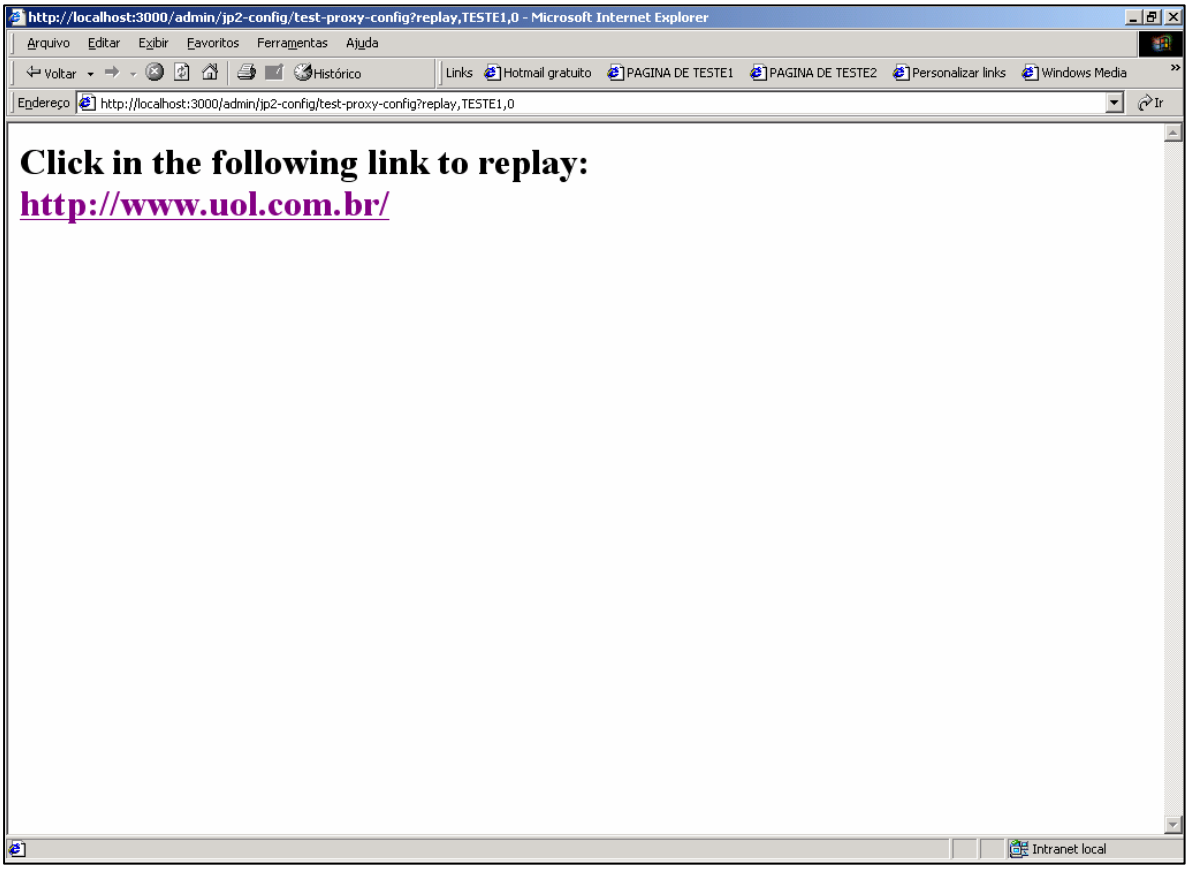

Figura 5.9: Tela do Browser mostrando o link do recurso requisitado para "playback".

A Figura 5.10 mostra o exemplo da ferramenta exibindo as duas janelas, com o recurso gravado pela ferramenta proposta e em outra, o recurso fornecido pelo servidor WEB (internet). Pode-se notar que algumas informações exibidas pelos browsers são diferentes, pois o browser da esquerda está exibindo as informações gravadas pela ferramenta, enquanto o browser da direita mostra as informações atualizadas pelo servidor WEB.

- **Delete**: o comando abaixo pode ser utilizado para apagar uma sessão de testes gravada em disco, bastando digitar o comando a partir da barra de endereços do browser e acrescentar o nome da sessão que se deseja deletar após a vírgula final:

```
http://localhost:3000/admin/jp2-config/test-proxy-config?delete,
```

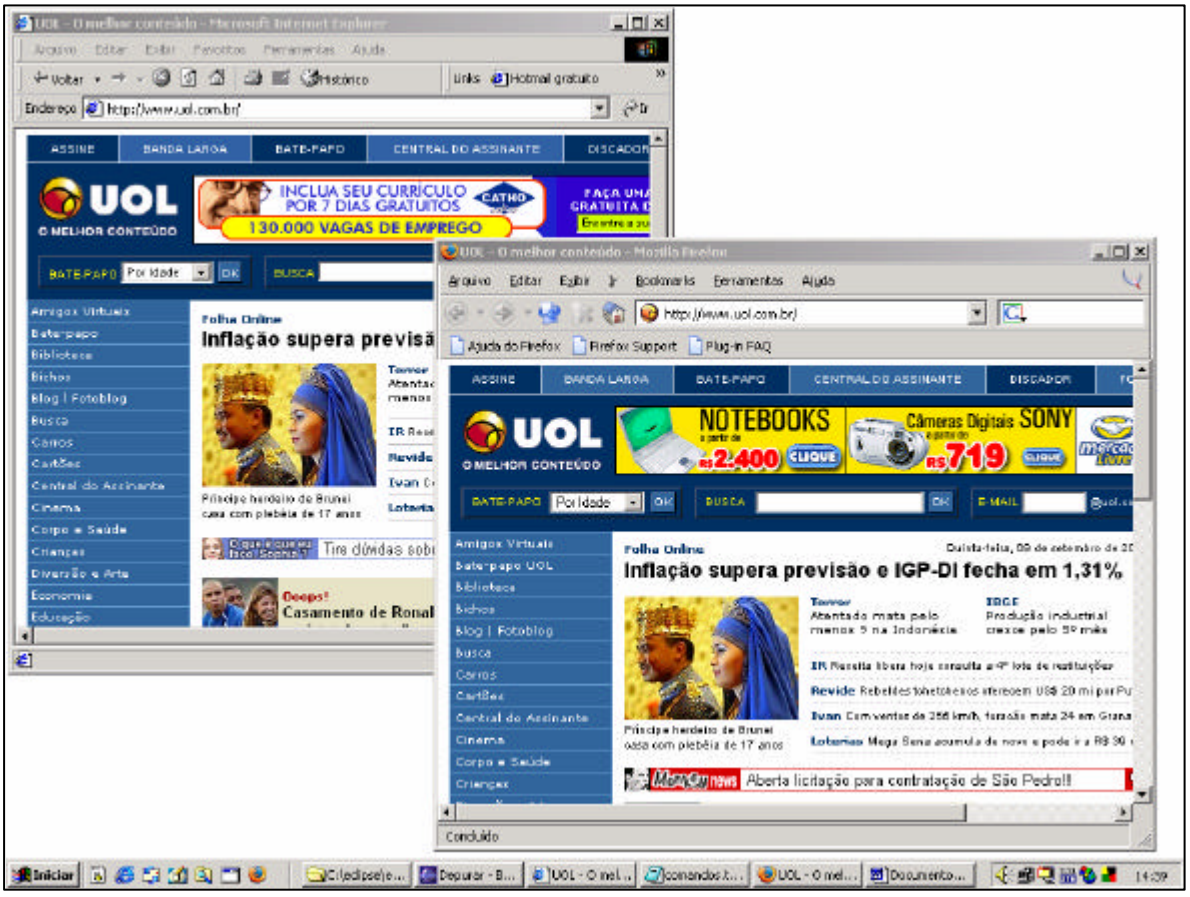

Figura 5.10: Janelas mostrando o recurso recuperado pela ferramenta e o recurso fornecido pelo servidor WEB.

A Figura 5.11 exibe a tela exibida pelo browser após o comando "**`delete`**" ser acionado, apagando uma sessão de teste chamada TESTE1.

Todos os casos de testes gravados na sessão, bem como os arquivos de requisição e resposta, serão excluídos.

? **Comandos Inválidos:** Se o usuário da ferramenta digitar algum comando errado na barra de endereços do browser, a ferramenta irá exibir uma tela com uma mensagem de comando inválido e também uma lista de comandos válidos, como mostra a Figura 5.12. No

exemplo, o comando digitado foi “`reply`”, quando o correto deveria ser “`replay`”.

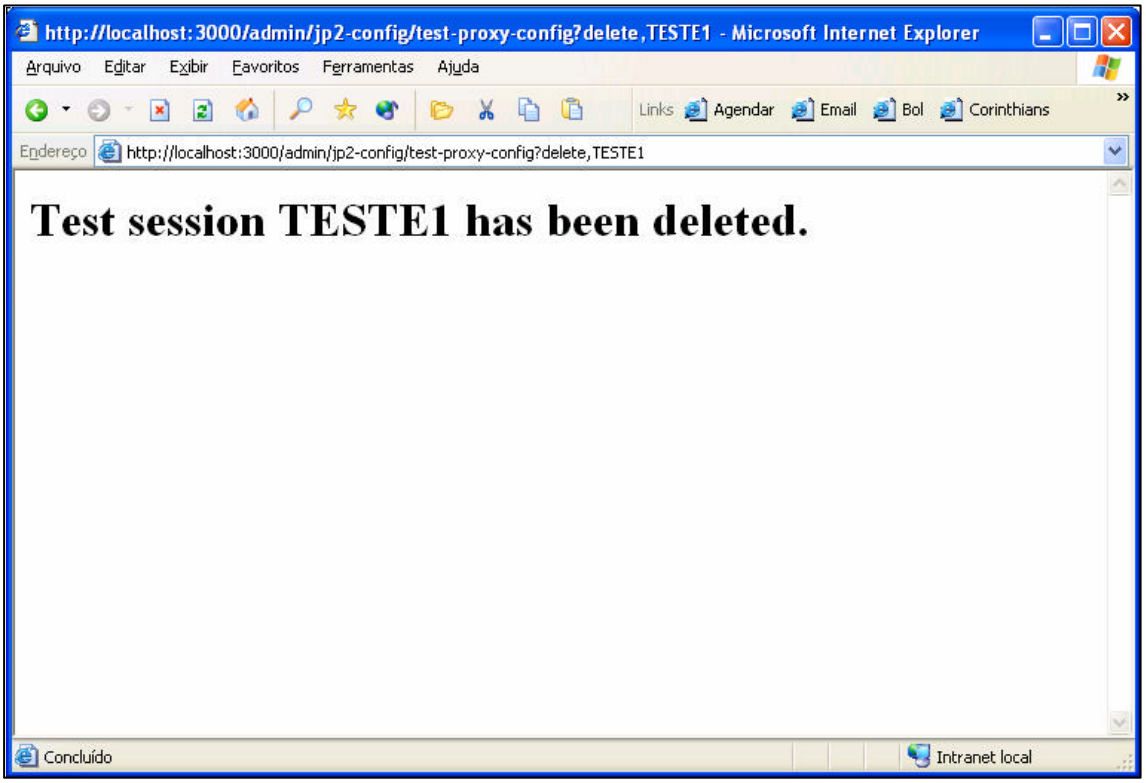

Figura 5.11: Confirmação da exclusão da sessão de testes TESTE1.

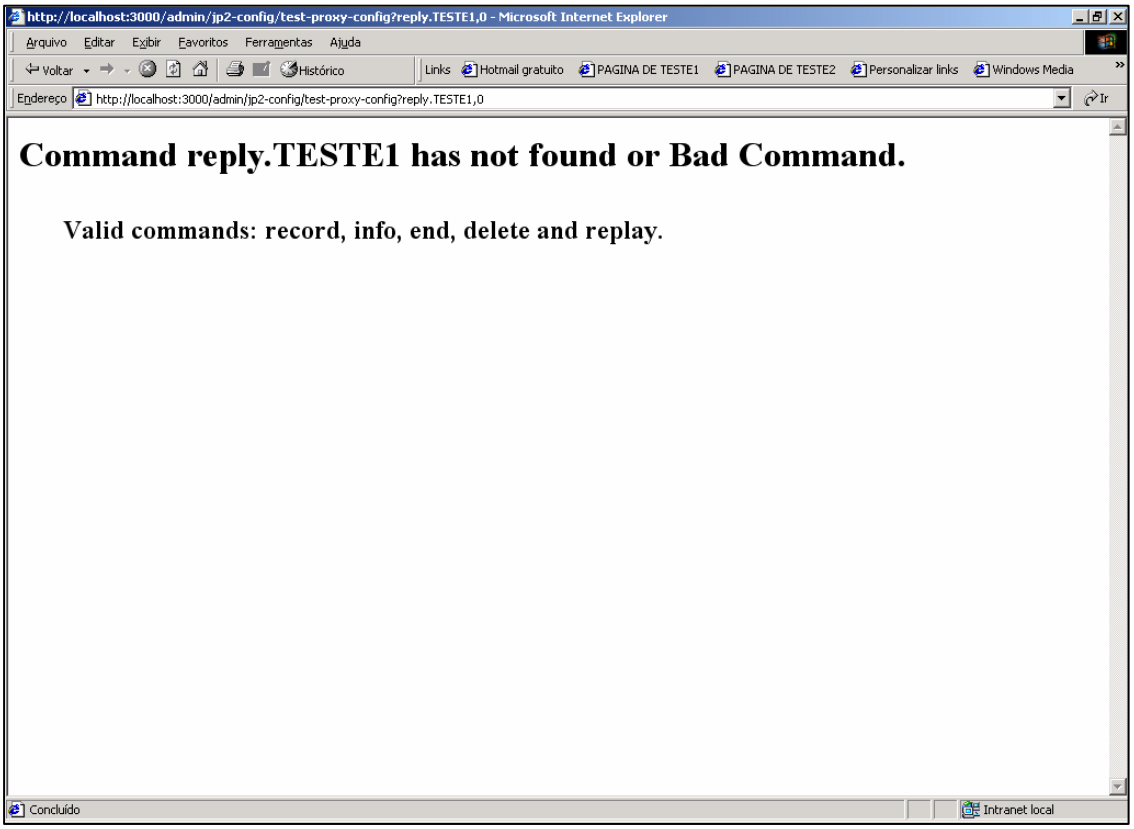

Figura 5.12: Mensagem de erro ao tentar executar um comando que não existe.

5.2 – Implementação

Para a implementação da ferramenta de teste proposta, foi utilizado o proxy jHTTPp2 (KOHL, 2004), modificado e adaptado para atender os requisitos propostos no desenvolvimento do trabalho proposto.

5.2.1 – Proxy jHTTPp2

A ferramenta para Record e Playback para aplicações WEB apresentada nesse trabalho foi baseada no Proxy jHTTPp2 desenvolvido por Benjamin Kohl (KOHL, 2004), que consiste em um servidor proxy baseado no protocolo HTTP desenvolvido na linguagem Java.

O Proxy jHTTPp2 possui as seguintes características, segundo Kohl (2003):

- Suporta as plataformas Windows (9x, 2000 e XP), Linux e Mac Os;
- Permite bloqueio de URL por regras de filtro, sendo um possível uso desta característica no bloqueio de propagandas do tipo “banners” e anúncios “pop-up”;
- Servidor com interface WWW incluso, sendo possível configurar o uso do proxy através do próprio WEB Browser;
- Software de código-fonte aberto (Licença Pública GNU);
- Suporte ao protocolo HTTP/1.1;
- O envio de cookies só é realizado se autorizado pelo usuário, independendo das configurações do browser.

Ainda segundo Kohl (2004), por se tratar de um projeto de código aberto e tratar com aplicações e tecnologia WEB, alguns recursos podem não corresponder ao planejado, causando pequenos defeitos no seu uso.

5.2.2 – Adaptações Realizadas para a Ferramenta Proposta

Foram realizadas alterações no projeto original do proxy jHTTPp2 para que o mesmo adquirisse características de uma ferramenta para Record e Playback, mas mantendo alguns padrões, como a utilização do protocolo HTTP e gravação das informações recebidas pelo proxy. A Figura 5.13 mostra a estrutura do funcionamento da ferramenta de Record e Playback em conjunto com o proxy jHTTPp2.

A inicialização da ferramenta é basicamente formada por duas etapas, sendo a primeira a fixação das propriedades do proxy, onde as configurações do proxy são carregadas do arquivo "`server.properties.txt`", e a segunda, logo após ser configurado, o proxy cria duas conexões, sendo a primeira com o browser e a segunda com o servidor remoto.

O objeto "`Client`" criado na classe "`Jhttpp2Server.class`" é usado para a conexão do browser com o proxy e, o objeto "`Listen`" da mesma classe é usado para a conexão do proxy com a WEB. Esses objetos são criados respectivamente através das API´s "`Socket`" e "`ServerSocket`". A Figura 5.13 demonstra a estrutura das conexões entre o browser, o proxy jHTTPp2 e o servidor remoto, fazendo que a ferramenta fique à espera de algum comando para o iniciar a sessão de testes. Já a Figura 5.14 exibe trechos dos códigos para

criação e instanciação dos objetos para conexão e comunicação da ferramenta com o proxy.

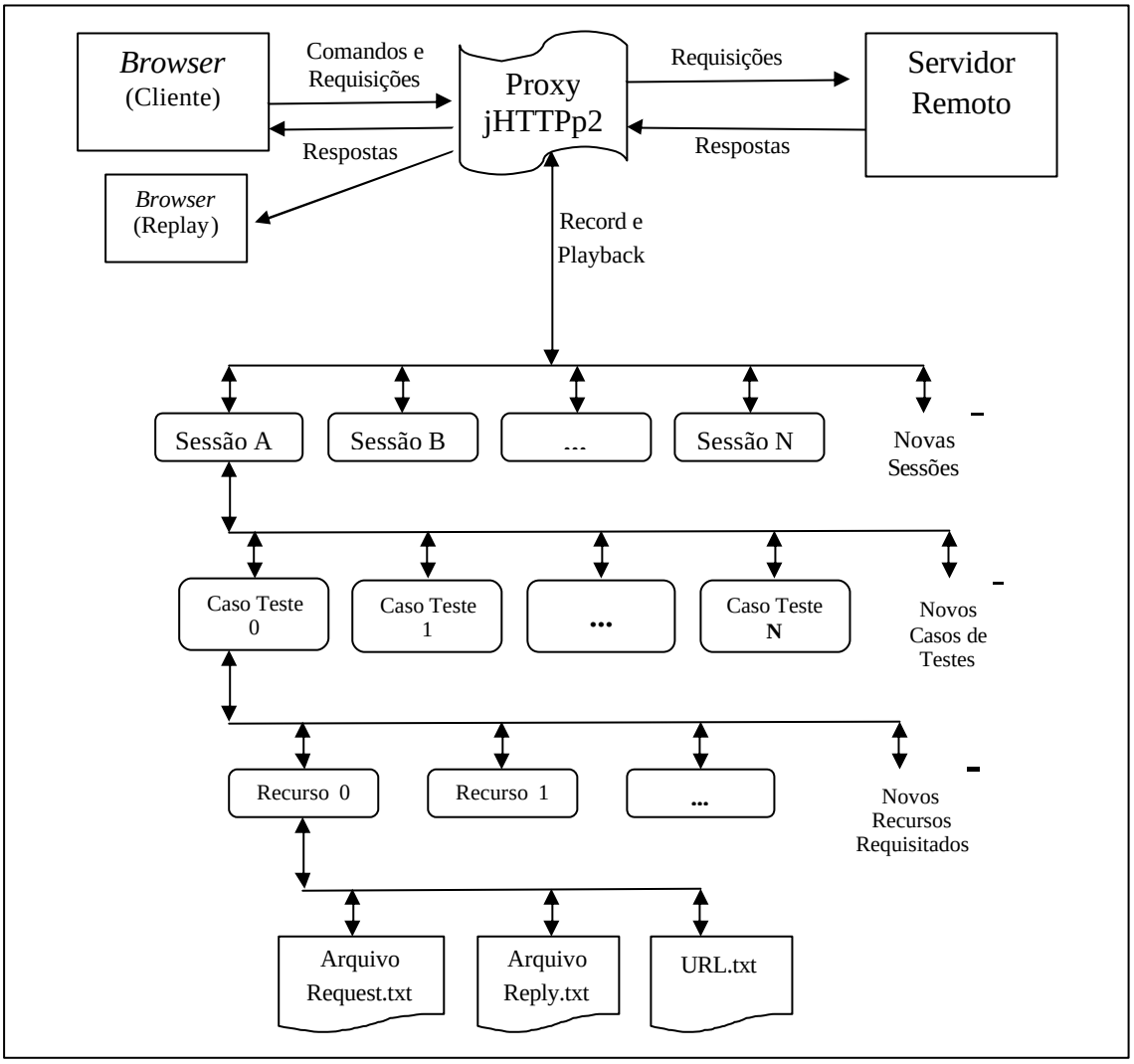

Figura 5.13: Estrutura de funcionamento da ferramenta em conjunto com o proxy.

Em seguida a ferramenta fica à espera dos comandos para o funcionamento da ferramenta via browser através da classe "`Jhttpp2HTTPSession.class`".

Logo após, é criado o objeto a partir da classe "`TestProxyAdmin.class`", sendo essa a classe que possui os métodos de execução dos comandos da ferramenta (`record`, `end`, `info`, `delete` e `replay`).

```
try
        {
            listen = new ServerSocket(port);
        } catch (BindException e_bind_socket)
        {
            setErrorMsg(
                "Socket "
                    + port
                    + " is already in use (Another jHTTPp2 proxy running?) "
                            + e_bind_socket.getMessage());
                                                .
                                                .
                                                .
public void connect(InetAddress host, int port) throws IOException
    {
        HTTP_Socket = new Socket(host, port);
        HTTP_in =
            new Jhttpp2ServerInputStream(
                server,
                this,
                HTTP_Socket.getInputStream(),
                false);
                 HTTP_out = new BufferedOutputStream(HTTP_Socket.getOutputStream());.
                                                .
                                                .
                                                .
```

Figura 5.14: Criação e instanciação dos objetos de comunicação da ferramenta.

Em seguida, é verificado qual comando foi executado pela barra de URL do browser.

Caso o comando seja "`Record`", a ferramenta cria um diretório com o nome da sessão de testes informada pelo browser. Se o diretório da sessão de testes foi criado corretamente, é passado para um objeto tipo "`Server`" a sessão e o número do caso de teste em que será feita a gravação da comunicação entre o browser e a WEB. A Figura 5.15 exibe o trecho de execução do comando "`Record`", verificando a estrutura de diretórios.

Após isso, é criado o diretório do caso de teste que, sendo a sessão nova, se iniciará em zero, senão, será o número seguinte ao último caso de teste. Em seguida serão criados os subdiretórios do caso de teste, onde cada subdiretório possui uma tripla de arquivos, sendo cada um deles "Request+*sufixo*.txt", "Reply+*sufixo*.txt" e "URL.txt". O número desses subdiretórios corresponde ao número de requisições que o browser fez e o servidor respondeu. A Figura 5.13

mostra esse procedimento nos desenhos “Novas Sessões”, “Novos Casos de Testes” e “Novos Recursos Requisitados”.

```
        private void processRecord() {
                if (section == null) {
                        error_msg = ERROR_INVALID_OP;    return;              }
                File dir = new File(section);
                int tcaseNumber = 0;
                if (!dir.isDirectory()) {
                        if (dir.mkdir()) {
                                htmlMsg =  "<h1>Test session "
                                                + section  + " has been created.</h1><BR>";
                        } else {
                                error_msg = "Cannot create directory " + dir.getAbsolutePath();    return;
        }
                } else {
                        htmlMsg = "<h1>Test session " + section + " found.</h1><BR>";
                        File[] tcases = dir.listFiles();
                        if (tcases.length > 0) {
                                int max = 0;
                                for (int i = 0; i < tcases.length; i++) {
                                        try {
                                                String fileName = tcases[i].getName();
                                                int k = Integer.parseInt(fileName);
                                                if (k > max)
                                                        max = k;
                                        } catch (NumberFormatException e) {         ;}
                                }
                                tcaseNumber = max + 1;}         }
                htmlMsg    + = "<p>Enter the URL to visit and all the request will be recorded as test cases
number "   + tcaseNumber         + ".</p>";
```

Figura 5.15: Trecho de código do método “Record”.

Em seguida, quando o browser envia uma requisição ao proxy, ele também cria um número aleatório para que seja usado com sufixo na nomenclatura dos arquivos “Request+*sufixo*.txt” e “Reply+*sufixo*.txt”, após fixado esse número, os próximos arquivos receberão uma numeração na parte do sufixo de forma crescente (*sufixo mais 1, mais 2..... mais N*) na seqüência das requisições necessárias para que o recurso seja exibido, sendo realizado isso na classe “`Jhttpp2Admin.class`”. As Figuras 5.16 e 5.17 mostram, respectivamente, como exemplo, as informações que a ferramenta grava nos arquivos “Request+*sufixo*.txt” e “Reply+*sufixo*.txt”.

```
// Recurso requisitado enviado ao servidor utilizando o protocolo HTTP, no exemplo abaixo, é requerido uma página tipo
// html chamada teste1
GET /informatica/joseluiz/teste1.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, */*
Accept-Language: pt-br
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: planeta.terra.com.br
            Cookie: TERRA=c8b79b8525648108447692300000262c8b003e6
```

Figura 5.16: Exemplo das Informações de requisição gravadas no arquivo "Request+*sufixo*.txt".

```
// Recurso enviado pelo servidor
HTTP/1.1 200 OK
Date: Sat, 19 Jun 2004 21:45:01 GMT
Server: Apache/1.3.27 (Unix)  (Terra 2.2) mod_gzip/1.3.26.1a
Content-Length: 742
Content-Type: text/html

<html>

<head>
<meta http-equiv="Content-Language" content="pt-br">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>PAGINA DE TESTE</title>
</head>

<body>

<p align="center"><font face="Arial">PAGINA DE TESTE</font></p>
<p align="center"><font face="Arial">SOMENTE CODIGOS HTML</font></p>
<p align="center"><font face="Arial">PAGINA SIMPLES</font></p>
<p align="center"> </p>
<p align="center"><font face="Arial">JHTTPP2</font></p>

</body>

</html>

<html>
<img src="http://uv.terra.com.br/UV?c=planeta" width=1 height=1 border=0 alt="" align="left">
            </html>
```

Figura 5.17: Exemplo das Informações de resposta gravadas no arquivo "Reply +*sufixo*.txt".

Para que a gravação dos casos de testes da sessão seja encerrada, o comando "End" foi implementado. Nele são enviados para o objeto "Server" os parâmetros necessários para que a comunicação seja cortada. A Figura 5.18 mostra o trecho de código execução do comando "End".

```
            public void processEnd() {
                        String s = server.getSessionName();
                        if (s != null)
                                    htmlMsg = "<h1>Session " + s + " ended.</h1>";
                        else
                                    htmlMsg = "<h1>Operation performed.</h1>";
                        server.setTcase(0);
                        server.setSession(null);
                        server.setRequest(0);
                        server.setReplyMode(false);
                        }
```

Figura 5.18: Trecho de código do método "End".

Para que o desenvolvedor tenha a possibilidade de saber quais e quantos foram os casos de teste de uma determinada sessão, foi desenvolvido o comando "`Info`", onde são percorridos todos os subdiretórios da sessão e assim, criada uma árvore vertical dos diretórios de casos de teste, subdiretórios de recursos e arquivos. Esses nomes de diretórios e arquivos são mostrados como um link para que possam ser abertos a partir de suas representações no browser. A Figura 5.19 mostra o trecho do código de execução do comando "`Info`".

```
private void processInfo() {
        if (section == null) {
                error_msg = ERROR_INVALID_OP;
                return;    }
        File dir = new File(section);
        if (!dir.isDirectory()) {
                error_msg = "Test session " + section + " has not been created yet";
                return;
        }
        htmlMsg = "<h1>Found test session " + section + "</h1>";
        File[] tcases = dir.listFiles();
        if (tcases.length > 0) {  //verifica se ha casos de testes
                htmlMsg += "<h2>List of test cases found:</h2><BR>" + "\n<UL>";
                        for (int i = 0; i < tcases.length; i++) {
                        String tcaseName = tcases[i].getName();
                        try {
                                int k = Integer.parseInt(tcaseName);
                                htmlMsg += "\n<LI>Test case number " + tcaseName + "<BR>";
                                File[] requests = tcases[i].listFiles();
                                        .
                                        .
                                        .
```

Figura 5.19:Trecho do código do método "Info".

Para realizar a função de Playback, foi implementado o comando "`Replay`", onde a ferramenta irá passar o recurso para o browser (esse recurso é adquirido no arquivo "URL.txt" do subdiretório do caso de teste de menor número). Assim que o usuário clicar nesse URL, exibido como um hyperlink pelo browser, a ferramenta recebe a requisição do browser e ao invés de mandá-la para o servidor de páginas, é utilizada para que seja feita a seleção da resposta "Reply+*sufixo*.txt" correspondente a ela e que foi gravada anteriormente pelo proxy. Ao ser localizado a resposta correspondente à requisição feita, a ferramenta lê o arquivo de resposta e envia seu conteúdo para interpretação do browser, fazendo assim o papel do servidor de páginas. A

Figura 5.20 mostra o trecho do código que faz essa busca pelas requisições e respostas gravadas para o recurso Playback da ferramenta.

```
File tcaseN = new File(dir, tcaseNumber + ""); // tcase directory
if (!tcaseN.exists()) {
        throw new FileNotFoundException(
                "Directory " + tcaseN + " cannot be found.");          }

File[] requestDirs = tcaseN.listFiles();
File requestFile = null;
String url = host.getHostName() + in.url;
// procura em cada diretorio de requisicoes
int i = 0;
for (; i < requestDirs.length; i++) {
        requestFile = new File(requestDirs[i], ID_FILE_NAME);
        if (!requestFile.exists())
                throw new FileNotFoundException(
                "File " + requestFile + " cannot be found.");
        FileReader fis = new FileReader(requestFile);
        BufferedReader bir = new BufferedReader(fis);
        String s = bir.readLine();
        if (url.equalsIgnoreCase(s)) {
                requestFile = requestDirs[i];  break; } }
if (i >= requestDirs.length)
        throw new FileNotFoundException(
                "File for the url " + url + " cannot be found.");
File reFiles[] = requestFile.listFiles();
i = 0;
for (; i < reFiles.length; i++) {
        if (reFiles[i].getName().startsWith(REPLY_PREFIX))      break;      }
if (i >= reFiles.length) {
        throw new FileNotFoundException(
                "Reply file in directory " + requestFile + " cannot be found.");
```

Figura 5.20: Trecho do código para leitura das informações na estrutura de diretórios para o recurso de Playback.

Ao mesmo tempo em que isso acontece, o primeiro recurso (na forma de URL) recuperado do arquivo “URL.txt” é passado como parâmetro para o objeto “`AbreBrowser`” que irá se utilizar desse endereço para inicializar um segundo browser já configurado para conectar-se com a WEB sem utilizar as configurações proxy, como mostra o trecho de código na Figura 5.21.

Para haver a possibilidade de se apagar a sessão de teste criada, o comando “`Delete`” foi implementado, se executado, é feita a procura do diretório da sessão especificada, e se achado o diretório, ele é excluído. A Figura 5.22 mostra o método utilizado para execução do comando “`Delete`”.

```
public class AbreBrowser{
Runtime E = Runtime.getRuntime();
  public AbreBrowser(String URLAlvo) {
    try {
         E.exec("C:\\Arquivos de programas\\Mozilla Firefox\\firefox.exe"+" "+URLAlvo);
     }
     catch (Exception ex) {

          System.err.println(ex);
          } } }
```

Figura 5.21: Trecho do código para abrir o segundo browser com o URL recuperado.

```
          private void processDelete()
            {
            if (section == null)
            {         error_msg = ERROR_INVALID_OP;    return;   }
            File dir = new File(section);
            if (dir.isDirectory())
            {
                      if (deleteDir(dir))
                      {   htmlMsg = "<h1>Test session "          + section
                                                  + " has been deleted.</h1><BR>";
                      } else
                      {   error_msg = "Cannot delete directory " + dir.getAbsolutePath();
                            return;       }   }   }


         public static boolean deleteDir(File dir) {
                   if (dir.isDirectory()) {
                             String[] children = dir.list();
                             for (int i=0; i<children.length; i++) {
                               boolean success = deleteDir(new File(dir, children[i]));
                                        if (!success) { return false; } } }
                   // Sessao Deletada
                   return dir.delete();}
```

Figura 5.22: Trecho do código do comando “Delete”.

CAPÍTULO 6 – EXEMPLO DE UTILIZAÇÃO

Este capítulo irá mostrar um exemplo prático da utilização da ferramenta proposta.

Para essa etapa de caso de testes, foi utilizada a aplicação WEB desenvolvida pela funcionária do Univem, Ana Paula Medeiros de Lima. A aplicação consiste em um software desenvolvido na linguagem PHP para cadastro e consulta de professores, bem como projetos de estudos, orientações, etc. O professor cadastrado pode acessá-lo usando a internet e, através de um nome de usuário e senha, tem acesso aos recursos do software via WEB.

A Figura 6.1 exibe as informações cadastradas para exemplificar o uso da ferramenta proposta.

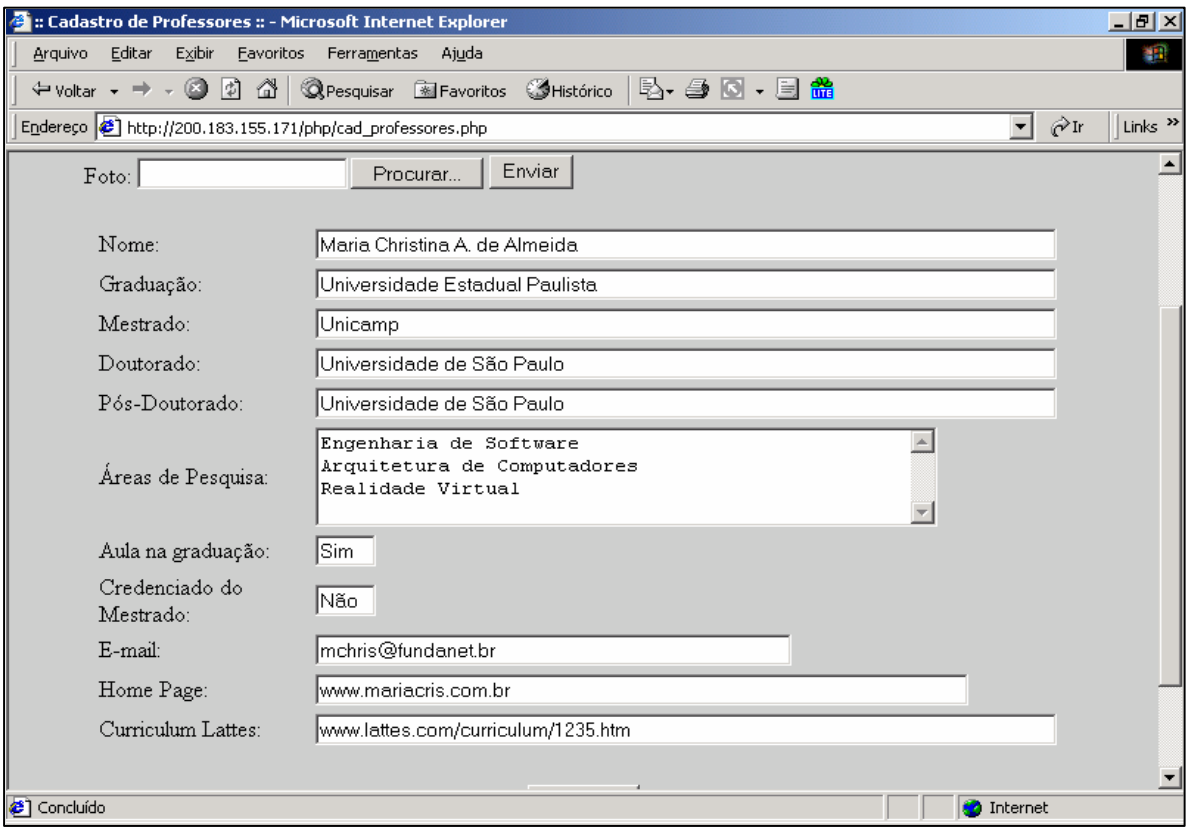

Figura 6.1: Preenchimento dos dados para cadastro de um novo professor.

6.1 – Criação da Sessão de Testes – Modo "*Recording*"

Com a ferramenta de teste proposta devidamente configurada e em estado de execução, foi criada uma sessão de testes chamada "`casoteste`", através do comando "**`record`**".

Em seguida, o endereço da aplicação WEB foi digitado na barra de URL do browser, para que a ferramenta proxy iniciasse a sessão de testes. A aplicação WEB testada exibe uma listagem dos professores cadastrados, conforme a Figura 6.2. Deve ser notado que o campo "Graduação" não exibe corretamente a informação que foi cadastrada (ver Figura 6.1), exibindo ao invés do título de graduação do professor cadastrado, uma coluna em branco.

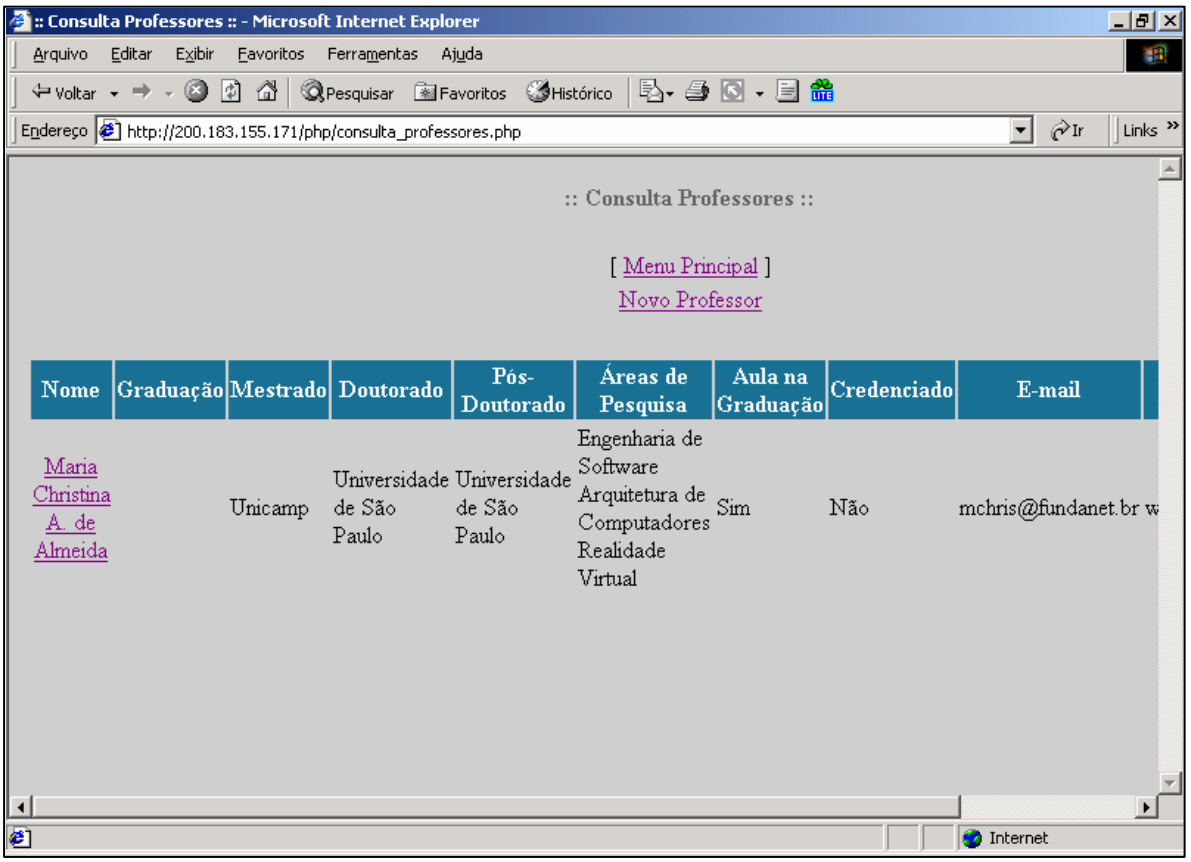

Figura 6.2: Exibindo os dados cadastrados com erro no campo "Graduação".

Como todos os passos estão sendo gravados pela ferramenta proxy, o erro detectado pode ser corrigido, e na sessão de Playback, comparar a execução da aplicação e dos resultados obtidos após a correção dos erros.

Logo após ser detectado o erro de exibição, a sessão de testes é finalizada, como mostra a Figura 6.3.

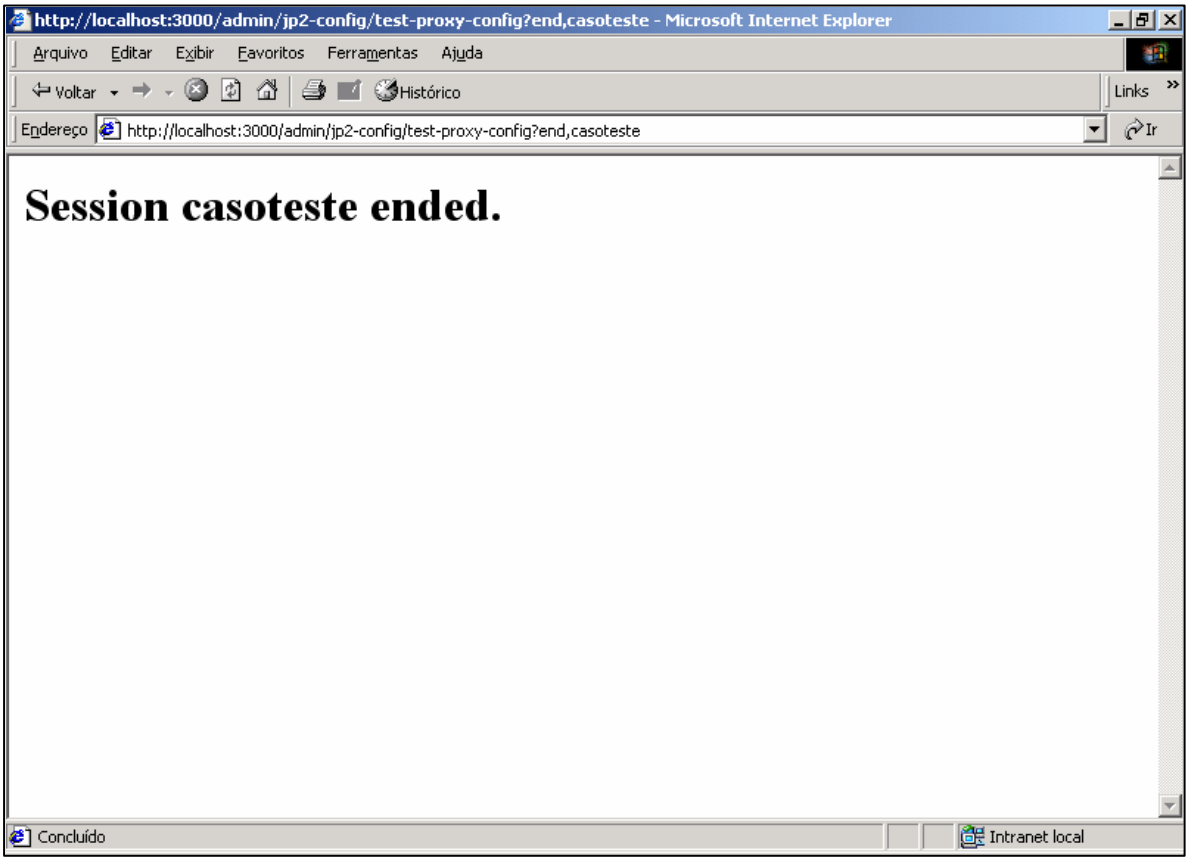

Figura 6.3: Sessão de Teste finalizada.

6.2 – Utilizando os Casos de Testes – Modo "*PlayBack*"

Na execução da aplicação na sessão de testes foi detectado um erro de exibição de informação em um campo do cadastro de docentes.

Como o propósito da ferramenta é fornecer um ambiente de testes onde os casos de testes sejam armazenados para que, se erros forem detectados e posteriormente, corrigidos, a aplicação possa ser testada com esses mesmo casos de testes, ou seja, a mesma seqüência de teste poder ser refeita.

O erro detectado na sessão de testes citado na Seção 6.1 se deu pelo fato do programador escrever o nome de um componente de forma errada, ou seja, um erro de digitação, como é mostrado na Figura 6.4, na parte em destaque do trecho do programa.

Com a correção do código da aplicação (de "*graduacacao*" para "*graduacao*") o usuário da ferramenta de teste pode repetir a execução da aplicação, utilizando o mesmo caso de testes usados na sessão anterior, verificando assim, o comportamento da aplicação após a correção do erro e verificando se nada mais de errado ocorreu.

```
        .
        .
        .
</font></div></td>
 <td><font color="#000000">
<?=$res['graduacacao'];?>   //erro de digitação
 </font></td>
<td><font color="#000000">
<?=$res['mestrado'];?>
</font></td>
        .
        .
        .
```

Figura 6.4: Trecho do programa com o erro detectado.

Com o comando "**replay**", o usuário refaz o caminho de recursos que foi percorrido na sessão de capturas de casos de testes, e assim, compara os dois momentos de execução da aplicação testada. A Figura 6.5 exibe a tela fornecida pela ferramenta de teste proposta, para o usuário iniciar a sessão de Playback.

Nota-se que na barra de URL´s do navegador é exibido, ao final do comando “**`replay`**”, o nome da sessão de testes (*casoteste*) e o número do caso de teste que se deseja testar, no caso, o caso de teste “0” (zero).

Após a confirmação do caso de teste clicando-se no link fornecido, a ferramenta irá abrir em dois browsers as informações do recurso de Playback. A Figura 6.6 exibe no browser da esquerda a página da aplicação WEB com o erro detectado, gravado e recuperado pela ferramenta proxy. No browser da direita, é exibido o mesmo recurso (ou URL), já alterado e agora, correto. A ferramenta proposta busca o recurso desejado para o Playback no servidor, garantindo que as informações estão atualizadas.

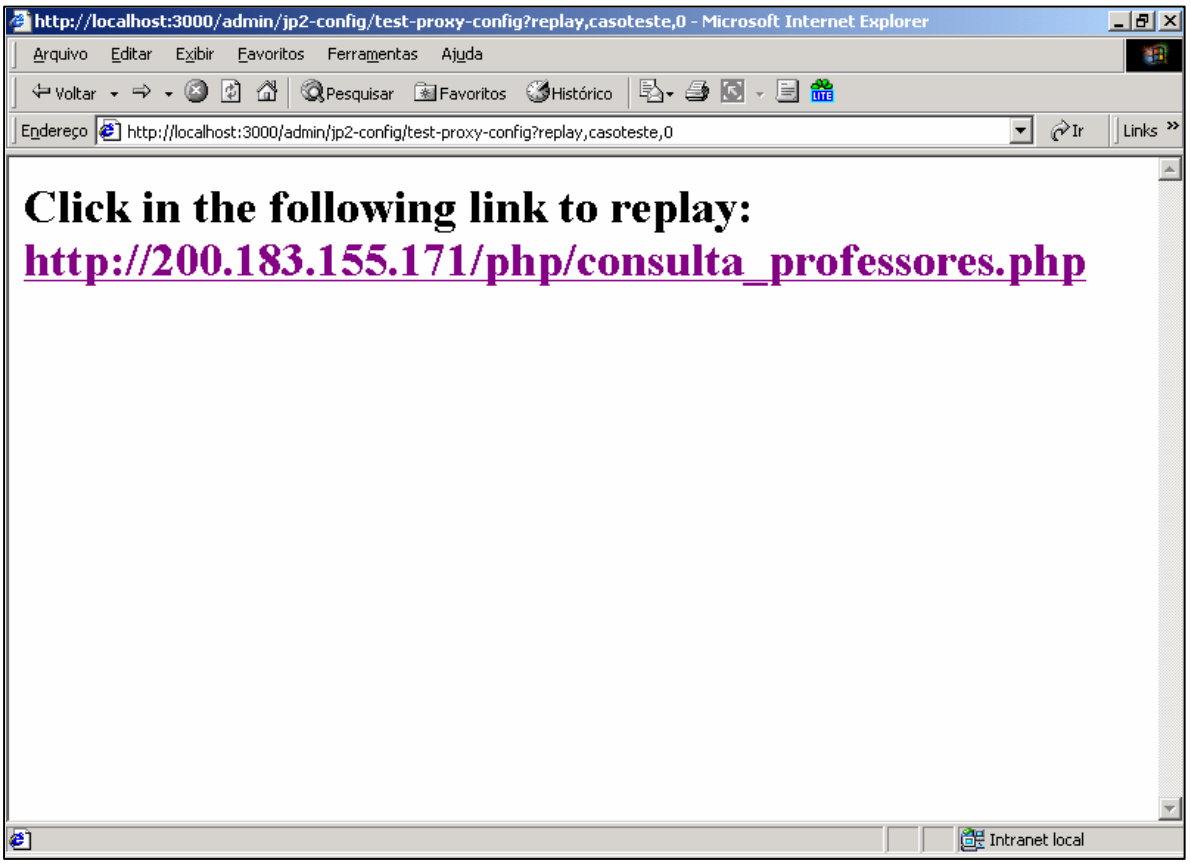

Figura 6.5: Exibição do URL para a execução do modo “Replay”.

Assim, o usuário tem a possibilidade de verificar se a aplicação WEB corresponde às especificações para que foi desenvolvida, e se houver

necessidade de se refazer o teste, por qualquer outro motivo, os casos de testes estarão gravados na sessão, bastando executar novamente o comando “**`replay`**”.

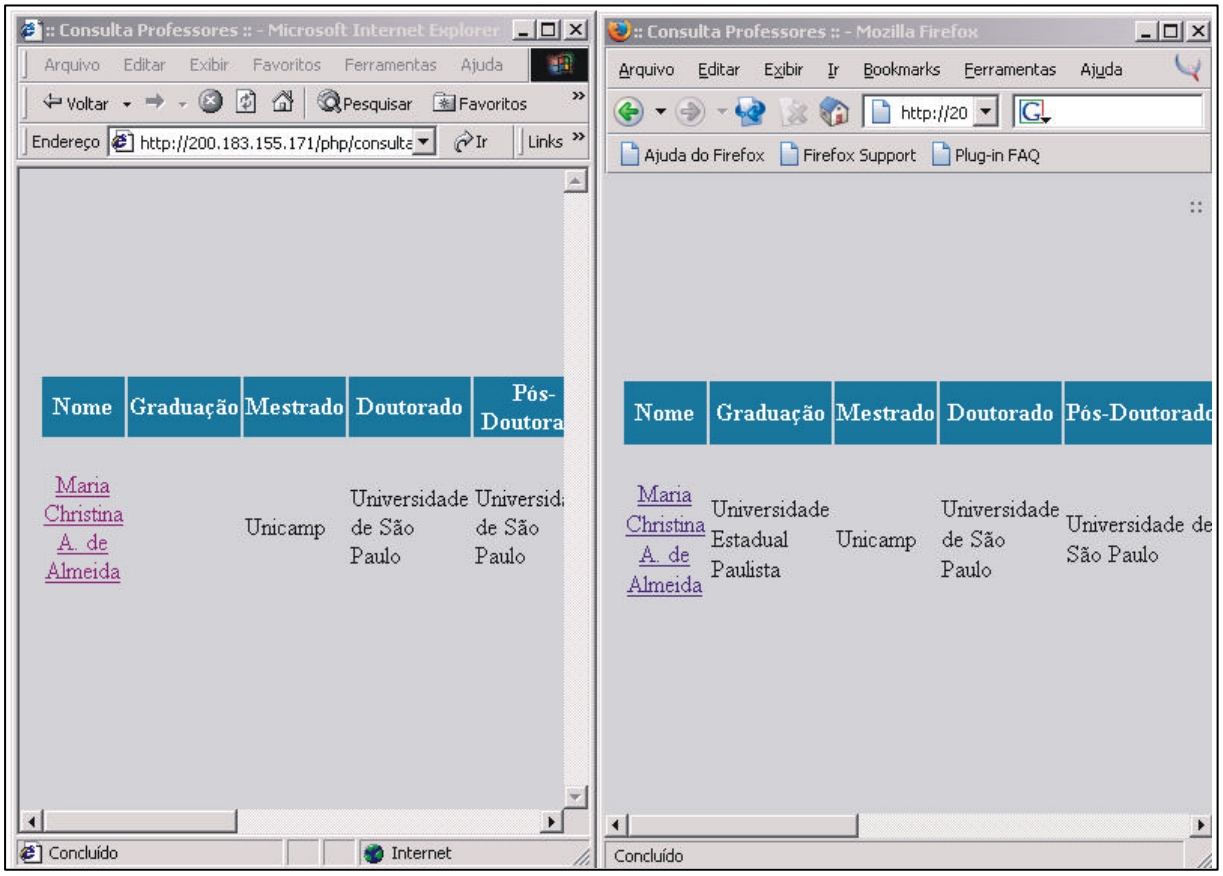

Figura 6.6: Exibição em duas janelas para comparação das execuções da aplicação.

Se o usuário requisitar um recurso que não foi exercido durante a fase de recording para a sessão de teste, uma mensagem de erro é exibida. No caso da aplicação WEB testada, foi clicado no link “Menu Principal”. Como a ferramenta proposta não possui gravado aquele caminho requisitado, uma mensagem de erro é exibida. A Figura 6.7 apresenta a mensagem de erro exibida ao se requisitar um recurso não gravado pelo recurso “Record”.

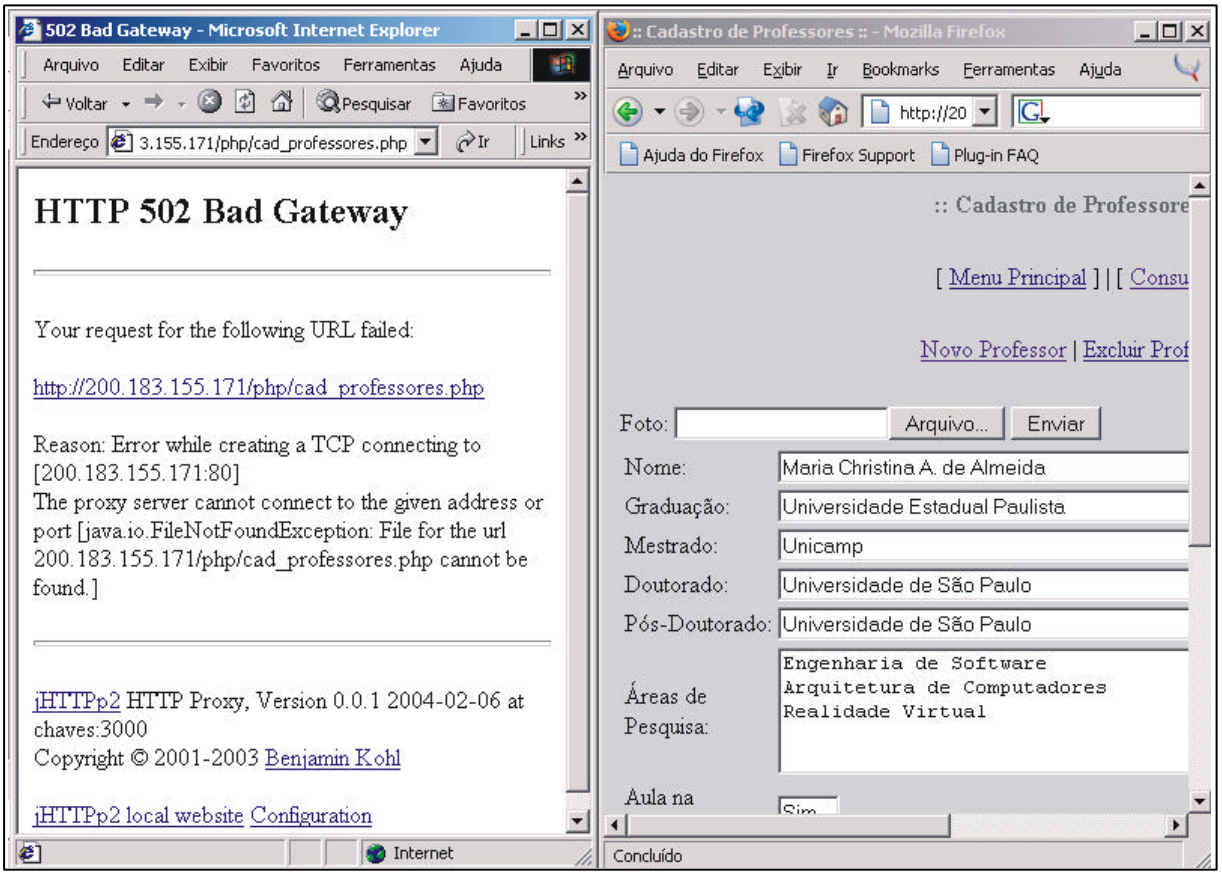

Figura 6.7: Exibição de erro ao se requisitar um recurso que não foi programado no modo “Record”.

CONCLUSÃO

O uso de ferramentas que realizam o teste de software é cada vez mais necessário para automatizar e auxiliar o processo de teste, sendo capaz de reduzir a intervenção humana nos resultados obtidos, possibilitando assim, resultados mais precisos.

Diante dessa situação, o desenvolvimento da ferramenta proposta para o teste de aplicações WEB se mostra bastante útil no processo de teste em uma área em que há necessidade de evolução da maioria dos sistemas computacionais existentes para o ambiente WEB.

A ferramenta de teste para Record e Playback apresentada auxilia o processo de teste em aplicações WEB em um ambiente de teste de regressão, pois possibilita ao desenvolvedor a gravar as sessões e casos de teste necessários, executá-las e compará-las em uma situação posterior, de uma maneira rápida e fácil, desenvolvendo assim, aplicações confiáveis e de alta qualidade.

Pode-se definir para contribuições futuras da ferramenta apresentada, como sendo em estudos para a integração do seu uso em um ambiente que se utilize somente um browser, além da melhoria da sua interface com o usuário testador e de emissão de relatórios exibindo as diferenças e possíveis erros na execução do método de “PlayBack”, possibilitando assim, um aproveitamento melhor dos seus recursos e funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

Alpern, B., Choi J., Ngo T., Sridharan M., Vlissides J.; **A Perturbation-Free Replay Platform for Cross-Optimized Multithreaded Applications**, Divisão de Desenvolvimento IBM Thomas J. Watson, Instituto Massachusetts de Tecnologia de Cambridge, 2000.

Barbosa, E. F., Adriano, C. M., Maldonado, J. C., Ricarte, I. L. M., Jino M.; **Fostering Theoretical, Empirical and Tool Specific - Knowledge in a Software Testing** Learning Scenari. IEEE, 2002.

Beck, K., Gamma, E.; **Test Infected: Programmers Love Writing Tests**, Java Report, Volume 3, Número 7, 1998.

Bianchini, S. L., Vincenzi, A. M. R., Delamaro, M. E., Simão, A. S., Barbosa, E. F., Maldonado, J. C.; **Proteum/IM Version 2.0C - User's Guide**, ICMC, São Carlos, 2003.

Bhogal, K. S.; **IBM Software Services for WebSphere Consultant, Fort Worth, TX.** WEB site disponível em http://www-106.ibm.com/developerworks/websphere/library/techarticles/0303_bhogal/bhogal.html, acesso em 10/05/2004.

Bueno, P. M. S., Chain, M. L., Jino, M., Maldonado, J. C., Vilela, P. R.; **POKE-TOOL 2.0 - Manual do Usuário**, Universidade de Campinas, 1995.

Chaim, M. L.; **POKETOOL Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados**, Dissertação de Mestrado, DCA/FEEC/UNICAMP, 1991.

Chan, M. C., Griffith S. W, Iasi A. F.; **JAVA - 1001 Dicas de Programação**; Tradução de Miguel Cabrera Fernandes; Makron Books, São Paulo, 1999.

Chen, Y. F., Rosenblum, D., Vo, K. P.; **TestTube: A system for selective regression testing**. 16th Conferencia International de Engenharia de Software, página 220, Maio, 1994.

Choi, J., Zeller A.; **Isolating FailureInducing Thread Schedules**. WEB site disponível em http://www.research.ibm.com/jalapeno/dejavu, acesso em 27/06/2003.

Conallen, J. **Building WEB Applications with UML. object technology**. Addison-Wesley Longman, Reading, Massachusetts, USA, Primeira Edição, Dezembro, 1999.

Delamaro, M. E. Proteum - **Um ambiente de teste baseado na análise de mutantes**. Dissertação de Mestrado, ICMC/USP. São Carlos, SP, 1993.

Delamaro, M. E., Maldonado, J. C.; Interface Mutation : A Case Study, Workshop do Projeto Validação de Teste de Sistemas de Operação, Janeiro, 1997.

Delamaro, M. E., Maldonado, J. C, Vincenzi, A. M. R.; **Proteum/{IM} 2.0: An Integrated Mutation Testing Environment**, Simpósio de Mutação 2000, San Jose, CA, 2000.

Delamaro, M. E.; **Mutação de Interface: Um Critério de Adequação Interprocedural para o Teste de Integração**, Tese de Doutorado, IFSC/USP, 1997.

DeMillo, R. A.; Lipton, R. J.; Saywaod, F. G. **Hinte on Tool Data Selection**, Help for the Practicing Programmer. IEEE, 1978.

DeMillo, R. A.; Offutt A. Jefferson. **Constraint-Basead Automatic Test Data Generation**. IEEE, 1991.

Fidelis, W. I. O., Martins, Elaine; FireWeb: **Uma Ferramenta de Suporte aos Testes de Regressão de Aplicações WEB**, Teste de Doutorado, Instituto de Computação – Universidade Estadual de Campinas, 2004.

Hassan, A. E., Holt, Richard C.; **Architecture Recovery of WEB Applications**, Software Architecture Group (SWAG), Department of Computer Science. University of Waterloo, Waterloo, Canada, 2001.

Hassan, A. E.; **Architecture Recovery of WEB Applications**, Tese de Doutorado, Universidade de Waterloo. Waterloo, Ontário. Canada, 2002.

Hatch, B., Lee, J., Kurtz, G.; **Hackers Expostos – Segredos e Soluções para a Segurança Linux.** Pearson Education do Brasil, São Paulo, 2002.

Herman P. M. **A Data Flow Analysis Approach to Program Testing**. Australian Computer Journal, 8:3, 1976.

Jorge, R. F., Vincenzi, A. M. R., Delamaro, M. E., Maldonado, J. C.; **Relatório dos Operadores de Mutação implementados nas ferramentas Proteum e PROTEUM/IM**, ICMC - Usp, São Carlos, SP, 2002.

Junior, D.; **Guia de Consulta Rápida HTTP** - 1.ed. - São Paulo: Novatec, 2001.

Khetawat, A., Lavana, H., Brglez F.; **Recording and Playback of Collaborative Desktops on the Internet**, CBL, Departament of Computer Science, North Carolina State University, 1997.

Kohl, B.; **An Open Source HTTP Proxy Server.** WEB site disponível em http://jhttp2.sourceforge.net/, acesso em 17/06/2004.

Kohl, B.; **User Manual Release 0.4.62.** WEB site disponível em http://jhttp2.sourceforge.net/jp2-user-manual.pdf, 2003, acesso em 15/10/2004.

Lehman, M. M.; **Process Improvement - The Way Forward**, Anais do X Simpósio Brasileiro de Engenharia de Software, Outubro, 1996.

Luotonen, A., Altis, K.; **World-Wide WEB Proxies**. WEB site disponível em http://www.w3.org/History/1994/WWW/Proxies, acesso em 30/08/2004.

Manzoni, R. Jr.; **O Retorno do e-business**, **IDG Now!.** WEB site disponível em http://www.informal.com.br/artigos/a15082003_001.htm, acesso em 24/08/2004.

Maldonado, J. C.; **Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Programas de Software.** Tese de Doutorado, DCA/FEE/UNICAMP, Campinas, SP, 1991.

Martimiano, L. A. F.; **Estudo de Técnicas de Teste de Regressão baseado em Mutação Seletiva**, Dissertação de Mestrado, ICMC-USP, Novembro, 1999.

Myers, G. J.; **The Art of Software Testing**, John Wiley & Sons, 1979.

Ntafos S. C. **A Comparasion of some strctural Testing Strategies**. IEEE Transactions on Software Engineering, 4:6, July, 1998.

Orfali, R., D. Harkey, e J. Edwards, **Essential Client/Server Suvival Guide**, 3.ed. - Wiley, 1999.

Ostrand T. J., Weyuker E. J.; **Using Data Flow Analysis for Regression Testing**, Proceedings of the 6th Annual Pacific Northwest Software Quality Conference, Setembro, 1988.

Powell, T. A.; **WEB Site Engineer**, Prentice-Hall, 1998.

Pressman, R. S.; S**oftware Engineering - A Practitioner's Approach**, 3ª Edição, McGraw Hill, 1992.

Pressman, R. S. **Engenharia de Software** - 5.ed. - Rio de Janeiro: McGraw-Hill, 2002.

Rainsberger, J. P.; **JUnit: A Starter Guide**, WEBsite disponível em http://www.diasparsoftware.com/articles/JUnit/jUnitStarterGuide.html, acesso em 18/06/03.

Ricca, F., Tonella P.; **Analysis and Testing of WEB Applications**, Proceedings of the 23rd International Conference on Software Engineering (ICSE'01). IEEE, 2001.

Rosenblum, D., Rothernel G.; A **comparative study of regression test selection techniques**, In Proc. of the 2nd Int'l. Workshop on Empirics Studies of Software Maintence., Bari, Italia, 1997.

Rothermel, G., Harrold, M. J.; **Analyzing Regression Test Selection Techniques**, IEEE Transactions on Software Engineering, Agosto, 1996.

Santos, L. C,; **Para que serve o Proxy? (Rede)**. WEB site disponível em http://www.clubedasredes.eti.br/rede0002.htm, acesso em 19/06/2003.

Scheinblum, J.; **Test entire WEB applications with HttpUnit**. Builder.com. WEB site disponível em http://www.builderau.com.au/program/java/0.htm, acesso em 05/05/2004.

Vincenzi, A. M. R.; **Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação**, Dissertação de Mestrado, ICMC-USP, Novembro, 1998.

Watanabe, C. S.; **Introdução ao Cache de WEB.** Rede Nacional de Ensino e Pesquisa. WEB site disponível em http://www.rnp.br/newsgen/0003/cache.html, acesso em 30/07/2004.

Weaver, J., Joiner W.; **Introduction of jWebUnit**, SourceForge.com. WEB site disponível em http://jwebunit.sourceforge.net/index.html, acesso em 13/06/2004.

Wong, C. **HTTP Pocket Reference** - 1.ed. - Sebastopol, CA: O'Reilly & Associates, Inc, 2000.

Wong, W. E., Horgan, J. R., London, S., Agrawal, H.; **A Study of Effective Regression Testing in Practice**, Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering, Novembro, 1997.

Wong, W. E., Maldonado, J. C., Delamaro, M. E.; **Reduncing the Cost of Regression Testing by Using Selective Mutation**, Anais VIII Conferência Internacional de Tecnologia de Software: Qualidade de Software, Junho, 1997.

Wu, Y., Offutt, J., **Modeling and Testing WEB-based Applications**, Departamento de Engenharia de Informação e Software, University George Mason. 2002.