UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIAS, GERÊNCIA E SEGURANÇA DE REDES DE COMPUTADORES

HEINI THOMAS GEIB

Estudo e Análise de Registros do Servidor Squid

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do grau de Especialista

Prof. Dr. Sérgio Luis Cechin
Orientador

Prof. Dr. Sérgio Luis Cechin
Prof. Dr. Luciano Paschoal Gaspary
Coordenadores do Curso

Porto Alegre, dezembro de 2008.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade.
AFS	Andrew File System.
AJAX	Asynchronous Javascript and XML.
API	Application Programming Interface.
ASCII	American Standard Code for Information Interchange.
CERN	Conseil Européen pour la Recherche Nucléaire.
CLF	Common Log Format.
CSS	Cascading Style Sheets.
CSV	Comma Separated Values.
DNS	Domain Name System.
DOM	Document Object Model.
FTP	File Transfer Protocol.
GIF	Graphics Interchange Format.
GNU	GNU Is Not Unix.
GPL	General Public License.
HTML	Hypertext Markup Language.
HTTP	Hypertext Transfer Protocol.
HTTPS	Secure Hypertext Transfer Protocol.
IETF	Internet Engineering Task Force.
IP	Internet Protocol.
JPEG	Joint Photographic Experts Group.
KB	Kilobyte.
MIME	Multipurpose Internet Mail Extensions.
MVC	Model View Controller.
NFS	Network File System.
OSI	Open Systems Interconnection.
PDF	Portable Document Format.
PHP	PHP Hypertext Preprocessor.

PNG	Portable Network Graphics.
RFC	Request For Comment.
RTP	Real-time Transport Protocol
SGBD	Sistema de Gerenciamento de Banco de Dados.
SMTP	Simple Mail Transfer Protocol.
SSL	Secure Sockets Layer.
TB	Terabyte.
TCP	Transmission Control Protocol.
TCP/IP	Transmission Control Protocol / Internet Protocol.
TLS	Transport Layer Security.
URI	Universal Resource Identifier.
URL	Universal Resource Locator.
W3C	World Wide Web Consortium.
XHTML	Extensible Hypertext Markup Language.
XML	Extensible Markup Language.
XSLT	Extensible Stylesheet Language Transformations.

LISTA DE FIGURAS

RESUMO

Hoje a maioria das organizações possui conexão à Internet. A boa utilização da Internet é um diferencial competitivo para as empresas, e este diferencial é levado em conta em algum nível no planejamento estratégico das organizações mais desenvolvidas. Esta boa utilização é caracterizada por garantir que as pessoas da organização tenham acesso adequado ao conteúdo que lhes é útil. Outro dado importante é que o preço pago pelo acesso à Internet no Brasil ainda é bastante alto. Estes fatores fazem com que a conectividade à Internet seja tratada como um valioso recurso institucional, que precisa ser bem gerenciado.

Uma forma de as organizações otimizarem a utilização da conectividade à Internet é através de serviços de proxy HTTP. Estes serviços são caracterizados por atuarem como intermediários na conexão entre os computadores internos da organização e os servidores web situados na Internet. A consulta ao servidor da Internet passa então a ser realizada pelo serviço proxy, que obtém o conteúdo e o encaminha para o computador solicitante. Com isso, o serviço de proxy pode realizar cache e controle de acesso deste tráfego.

O serviço de proxy chamado Squid Web Proxy Cache registra em arquivos de log cada requisição efetuada pelos clientes. Este trabalho se concentra no estudo dos conceitos relacionados a web, proxy, cache e do próprio Squid, bem como seus registros. Finalmente, será desenvolvida uma aplicação que leia os dados contidos nos arquivos de log e forneça informações úteis aos gestores de redes que utilizam este serviço.

Palavras-Chave: proxy, cache, Squid, registros, log, análise.

Squid´s Log Files Study and Analysis

ABSTRACT

Today most organizations have Internet connection. The Internet proper use is a competitive differential for businesses, and this differential is taken into account at some level in more developed organizations strategic planning. This proper use is characterized by ensuring that people's organization has adequate access to content that is useful to them. Another important factor is that the price paid for Internet access in Brazil is still quite high. These factors mean that the connectivity to the Internet is treated as a valuable resource institutions, which must be well managed.

One way for organizations to optimize the Internet connectivity use is through HTTP proxy services. These services are characterized by acting as intermediaries in the connection between the organization internal computers and the web servers located on the Internet. The query to Internet server is then being held by the proxy service, which gets the content and send it to the requesting computer. Therefore, the proxy service can perform cache and access control of this traffic.

The proxy service called Squid Web Proxy Cache records in the log files every request made by clients. This academic work focuses on the study of the web concepts, proxy, cache and the Squid itself, as well as their records. Finally, will be developed an application which read the data contained in the log files and provide useful information to network administrators who use this service.

Keywords: proxy, cache, Squid, log files, log, analysis.

1 INTRODUÇÃO

Este trabalho pretende esclarecer diversos conceitos relacionados à web e componentes de sua arquitetura, como *proxy* e *cache*. Será desenvolvido, com ferramentas de software livre, um aplicativo que, a partir de arquivos de *log* do servidor de *proxy* Squid, gere informações úteis aos gestores de redes que utilizam este serviço.

Atualmente quase todas as organizações possuem conexão à Internet, por diversos motivos, indo desde pesquisa e obtenção de conhecimento até relacionamento com clientes e parceiros. Porém, assim como a Internet oferece cada vez mais oportunidades de bom proveito, crescem na mesma proporção os conteúdos inadequados. Sabe-se que a boa utilização da Internet, tanto para fornecer quanto para obter serviços, é também um diferencial competitivo. Este diferencial é levado em conta pelas organizações mais desenvolvidas em seus planejamentos estratégicos, que por sua vez irão exigir o bom uso da Internet. Este bom uso da Internet é caracterizado por garantir que as pessoas da organização tenham acesso adequado ao conteúdo que lhes é útil em suas atividades, ao mesmo tempo em que evita o acesso a conteúdos que fujam do escopo de utilidade.

Estes fatores, se somados ao dado de Venter (2003), de que o preço pago pelo acesso à Internet em países como o Brasil ainda é muito alto quando comparado aos Estados Unidos e à Europa, torna a conectividade à Internet um valioso recurso institucional, e que por este motivo precisa ser bem gerenciado.

Uma das formas mais efetivas de as organizações aproveitarem melhor o seu acesso à Internet é através da utilização de serviços de *proxy* HTTP. Estes serviços são caracterizados por atuarem como intermediários na conexão entre os computadores internos da organização e os servidores web situados na Internet.

O serviço de *proxy* HTTP é fornecido por um ou mais servidores instalados na rede da instituição. Os navegadores dos micros internos são configurados, de forma automática ou manual, para que utilizem o *proxy*, ou então é configurado o *firewall* da organização para que direcione ao serviço de proxy as requisições HTTP efetuadas pelos micros internos a servidores externos. A consulta ao servidor da Internet passa então a ser realizada pelo serviço *proxy*, que obtém o conteúdo e o encaminha para o computador solicitante. Com esta estrutura, exemplificada na Figura 1.1, obtemos duas grandes vantagens. Primeiro, torna-se possível armazenar no serviço *proxy* um *cache* dos conteúdos acessados, de forma que, quando outro computador interno efetuar a mesma requisição, o *proxy* já possui o conteúdo e não precisa buscá-lo na Internet, bastando apenas encaminhá-lo ao computador que solicita. A outra grande vantagem é a possibilidade de implementação de controles de acesso, permitindo negar requisições de acesso a conteúdos considerados inadequados à organização. Estes dois fatores fazem com que o serviço de *proxy* seja um grande aliado na otimização da conexão à Internet,

diminuindo o consumo de largura de banda através das suas características de *cache* e controle de acesso, e por isso tornando esta conexão de certa forma mais eficiente.

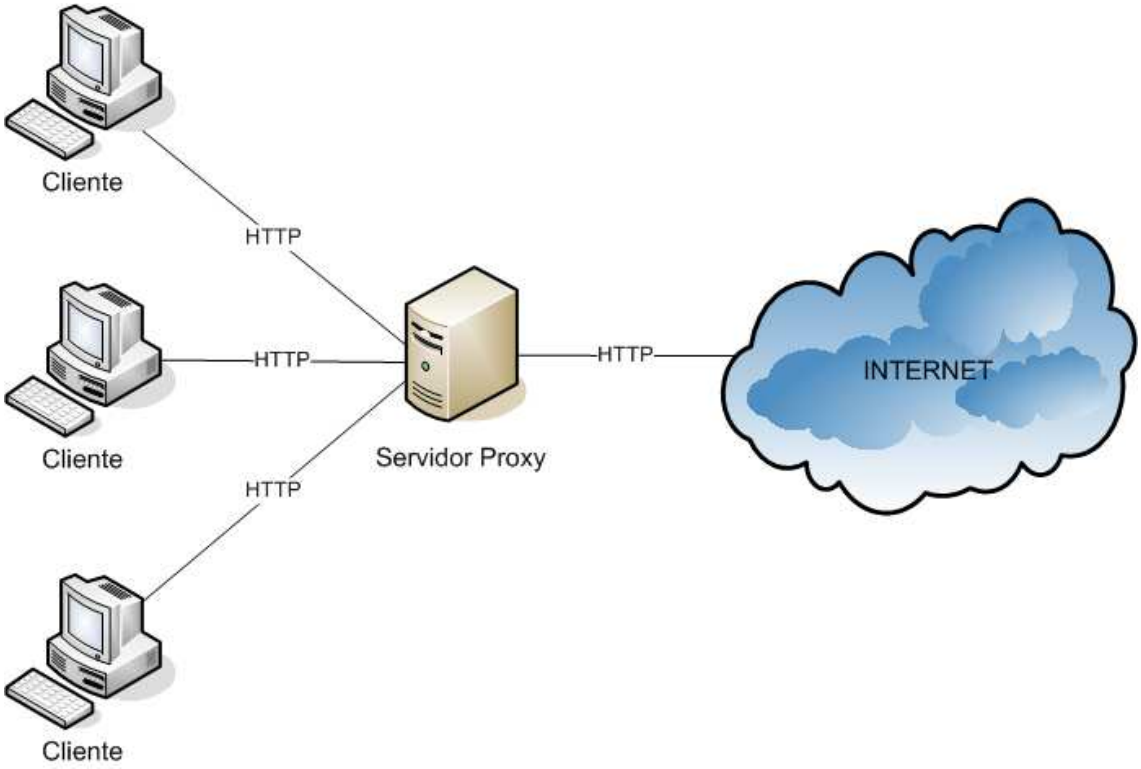

Figura 1.1: Estrutura Clientes-Proxy-Internet

Este trabalho irá enfatizar uma terceira grande vantagem do serviço de *proxy*. Este serviço pode ser configurado para registrar, em arquivos de *log*, todas as requisições de acesso efetuadas pelos computadores clientes. Com isso, temos dados interessantes que indicam o que as pessoas da organização estão buscando na Internet. Pelos fatores mencionados no início desta introdução percebemos que a conectividade à Internet é um recurso institucional que precisa ser bem gerenciado. Para realizar este gerenciamento de forma eficiente, necessitamos de informações sobre como está sendo utilizada esta conectividade. Estas informações podem ser extraídas dos arquivos de *log* do serviço de *proxy*.

Será abordada neste trabalho a análise de *logs* no formato gerado pelo serviço de *proxy* Squid, que é provavelmente o mais utilizado na Internet. É um software livre, de código fonte aberto e gratuito para utilização em qualquer tipo de ambiente. Existem atualmente diversas ferramentas gratuitas que efetuam este tipo de análise, porém todas deixam a desejar em critérios que envolvem análises mais complexas ou em níveis gerenciais, ou seja, menos técnicas e mais claras para os administradores da organização.

Os objetivos deste trabalho são os seguintes:

- Estudar a arquitetura e o funcionamento da web, seus protocolos, *cache* e *proxy*;
- Estudar o formato de *log* nativo do servidor de *proxy* Squid e extrair informações úteis a partir dos seus dados;

- Desenvolver uma aplicação capaz de fornecer informações úteis aos administradores de rede e aos gestores de uma organização usuária do serviço de proxy Squid.

2 CONCEITOS INICIAIS

Nesse capítulo serão abordados os conceitos principais relacionados ao trabalho, como *cache*, web e protocolos utilizados.

2.1 Cache

De acordo com Wessels (2001), o termo *cache* possui raízes francesas, e significa, literalmente, armazenar. Na informática, *cache* refere-se ao armazenamento de informações buscadas recentemente, para uso futuro. A informação armazenada pode ou não ser utilizada novamente, com isso um *cache* só é benéfico quando o custo para armazenar a informação é menor que o custo para buscar ou computar novamente tal informação.

O conceito de *caching* permeia quase todos os aspectos da computação e dos sistemas em rede. Os processadores possuem *caches* para dados e para instruções. Os sistemas operacionais fazem *cache* dos *filesystems*. Sistemas de arquivos distribuídos, como NFS e AFS, dependem de *cache* para uma boa performance. Roteadores fazem *cache* de rotas recentemente usadas. Servidores DNS fazem *cache* de consultas.

Os *caches* funcionam bem devido ao princípio conhecido como “local de referência”. Existem dois tipos de local: temporal e espacial. O **local temporal** significa que alguns dados são mais populares que outros, ou seja, recebem mais requisições em um determinado período de tempo. O **local espacial** significa que requisições por determinados dados ocorrem geralmente juntas, por exemplo a requisição de uma página normalmente é seguida pelas requisições das imagens da página. Os *caches* usam o princípio de “local de referência” para prever acessos futuros baseados nos prévios. Quando a previsão é correta, há uma melhoria significante de performance. Na prática, estas técnicas funcionam tão bem que os computadores seriam muito lentos sem os *caches* de memória e disco.

Quando um dado requisitado é encontrado no *cache*, denomina-se ***hit***. Da mesma forma, quando um dado requisitado não é encontrado no *cache* denomina-se ***miss***. A melhoria de performance que um *cache* fornece é baseada principalmente na diferença entre o tempo gasto em *hit* comparado ao tempo gasto em *miss*. A porcentagem de *hit* em relação ao total de requisições é denominada *hit ratio*.

Qualquer sistema que utilize *cache* precisa ter mecanismos para manter a consistência do *cache*. Este é o processo pelo qual as cópias em *cache* são mantidas atualizadas com as originais. Dados em *cache* podem ser considerados ***fresh*** ou ***stale***. O *cache* pode utilizar um dado *fresh* imediatamente, porém um dado *stale* requer validação antes de ser utilizado.

Os algoritmos que mantém a consistência podem ser fracos ou fortes. Consistência fraca significa que o *cache* às vezes retorna informações desatualizadas. Consistência forte significa que os dados em *cache* são sempre validados antes da utilização. Os *caches* de processador e sistema de arquivos necessitam consistência forte. No entanto, alguns tipos de *cache*, como aqueles em roteadores e servidores DNS, são efetivos mesmo quando retornam informações desatualizadas.

2.2 Arquitetura da Web

2.2.1 Clientes e Servidores

As peças fundamentais da web são clientes e servidores. Um servidor web gerencia e provê acesso a um conjunto de recursos. Estes recursos podem ser arquivos de texto simples e imagens, ou coisas mais complexas, como bancos de dados relacionais. Os clientes, também conhecidos como *user agents*, iniciam uma transação através do envio de uma requisição a um servidor. O servidor então processa a requisição e envia uma resposta de volta ao cliente.

Na web, a maioria das transações são operações de ***download***; o cliente "baixa" alguma informação do servidor. Nestes casos, a requisição em si é pequena (em torno de 200 bytes) e contém o nome do recurso, mais uma pequena quantidade de informações sobre o cliente. A informação retornada pelo servidor normalmente é uma imagem ou arquivo de texto, com tamanho médio bastante superior. Normalmente as taxas de transferência de recebimento são muito maiores que as taxas de envio.

Uma pequena porcentagem de transações web são melhor caracterizadas como sendo operações de ***upload***. Nestes casos, as requisições são relativamente grandes e as respostas são pequenas. Um exemplo de *upload* é a transferência de um arquivo de imagem do cliente para o servidor.

Os clientes web mais comuns são os *browsers*, ou navegadores. Sua função é montar o conteúdo web para que o usuário possa ver e interagir.

Há diversos softwares servidores web. O Apache HTTP Server é bastante popular e disponível gratuitamente. A maioria dos provedores de conteúdo preocupam-se com a performance dos seus servidores, já que o acesso aos mesmos pode ser tal que exige múltiplos servidores rodando em paralelo para suportar a carga.

2.2.2 Proxies

O *proxy* é um intermediário em uma transação web. É uma aplicação que localiza-se em algum ponto entre o cliente e o servidor de origem. São utilizados freqüentemente em *firewalls* para prover segurança. Eles encaminham requisições da rede interna para a Internet. Podem ser utilizados para diversas tarefas, como registro, controle de acesso, filtragem, tradução, varredura antivírus e *cache*.

Um *proxy* comporta-se simultaneamente como cliente e servidor. Atua como servidor para os clientes, e como cliente para os servidores. Ele recebe e processa requisições dos clientes, e então as encaminha para os servidores de origem. Os *proxies* também são conhecidos como "*application layer gateways*", ou "gateways de camada de aplicação". Esta denominação reflete o fato de que o *proxy* situa-se na camada de aplicação do modelo de referência OSI, assim como os clientes e os servidores. Uma

característica importante de um “gateway de camada de aplicação” é que ele utiliza duas conexões TCP: uma para o cliente e outra para o servidor.

2.2.3 Objetos Web

Objeto é o termo que se refere à entidade trocada entre um cliente e um servidor. Também pode ser chamado de documento ou página, porém estes termos podem levar à idéia de que trata-se apenas de texto ou uma coleção de texto e imagens. Objeto é um termo genérico e descreve melhor os diferentes tipos de conteúdo retornado dos servidores. Objetos web possuem algumas características importantes, como tamanho (número de bytes), tipo (HTML, imagem, áudio, etc.), hora de criação e hora da última modificação.

Em termos gerais, os recursos web podem ser considerados dinâmicos ou estáticos. Respostas de recursos dinâmicos são geradas sob demanda no momento em que a requisição é feita. Respostas estáticas são geradas previamente, independentemente das requisições de clientes. A distinção entre conteúdo dinâmico e conteúdo estático não é bem definida. Diversos recursos são atualizados em vários intervalos, não sendo gerados uma única vez e nem sob demanda.

2.2.4 Identificadores de Recursos

Identificadores de recursos, ou *resource identifiers*, são uma parte fundamental da arquitetura da web. São os nomes e endereços de objetos web. Oficialmente, são chamados de *Universal Resource Identifier* (URI). São utilizados pelos caches web para identificar e indexar os objetos armazenados. O tipo mais comum de URI em uso atualmente é o *Uniform Resource Locator* (URL).

2.3 Protocolos de Transporte da Web

Clientes e servidores utilizam diversos protocolos de transporte para trocar informações. Estes protocolos, construídos sobre o TCP/IP, suportam a maior parte do tráfego da Internet atualmente. O HTTP é o mais comum, devido ao fato de ter sido desenvolvido especialmente para a web. Também são utilizados FTP, SSL e RTP.

2.3.1 HTTP

O *Hypertext Transfer Protocol* foi criado originalmente por Tim Berners-Lee, e atualmente é mantido pelo Grupo de Trabalho sobre HTTP da IETF.

As transações HTTP utilizam uma mensagem com estrutura bem definida. Esta mensagem, que pode ser tanto uma requisição quanto uma resposta, possui duas partes: os **cabeçalhos** e o **corpo**. Os cabeçalhos estão sempre presente, porém o corpo é opcional. Cada cabeçalho é uma linha de caracteres ASCII. Nos cabeçalhos são encontradas informações e diretivas úteis ao *cache*. Uma linha em branco define o fim dos cabeçalhos e o início do corpo. O corpo das mensagens é tratado como dado binário. A maioria das mensagens de requisição não possui corpo, ao contrário da maioria das mensagens de resposta.

Um cabeçalho HTTP consiste de um nome seguido de dois-pontos e então um ou mais valores, separados por vírgula. Existem quatro tipos de cabeçalhos: entidade, requisição, resposta e geral. Cabeçalhos de **entidade** descrevem algo sobre os dados do corpo da mensagem. Por exemplo, *Content-length* é um cabeçalho de entidade, e

descreve o tamanho do corpo. Cabeçalhos de **requisição** devem aparecer apenas em requisições, e não têm significado para respostas. *Host* e *If-modified-since* são cabeçalhos de requisição. Cabeçalhos de **resposta** aplicam-se apenas a respostas. *Age* é um cabeçalho deste tipo. Cabeçalhos **gerais** são utilizados tanto em requisições quanto em respostas. *Cache-control* é um cabeçalho geral.

A primeira linha de uma mensagem HTTP é especial. Em requisições, é chamada *request lin*e (linha de requisição), e contém o *request method* (método de requisição), um URI e a versão do protocolo HTTP. Em respostas, a primeira linha é chamada de *status line* (linha de status), e inclui a versão do HTTP e um *status code* (código de status), que indica sucesso ou falha da requisição.

Os principais métodos de requisição são os seguintes:

- GET: requisita a informação identificada pela URI da requisição.
- HEAD: idêntico ao GET, porém a resposta não deverá conter o corpo da mensagem.
- POST: requisita que o servidor processe os dados presentes no corpo da mensagem.
- CONNECT: utilizado para criar um túnel para certos protocolos, como SSL, através de um *proxy*.

Seguem abaixo alguns exemplos de mensagens com requisições e respostas HTTP. Na Figura 2.1, exemplo de requisição GET.

```
GET /index.html HTTP/1.1
Host: www.exemplo.org
Accept: */*
```

Figura 2.1: Requisição HTTP GET

Na Figura 2.2, exemplo de requisição POST simples, com corpo da mensagem.

```
POST /cgi-bin/teste.php HTTP/1.1
Host: www.exemplo.org
Accept: */*
Content-Length: 16

args=abc+xyz&w=3
```

Figura 2.2: Requisição HTTP POST simples com corpo

Na Figura 2.3, exemplo de resposta com sucesso.

```
HTTP/1.1 200 Ok
Date: Sat, 15 Nov 2008 13:42:10 GMT
Last-Modified: Fri, 14 Nov 2008 10:12:50 GMT
Server: Apache/2.2.10
Content-Length: 13
Content-Type: text/plain

Hello, world.
```

Figura 2.3: Resposta HTTP com sucesso

Um exemplo de resposta de erro HTTP aparece na Figura 2.4.

```
HTTP/1.0 404 Not Found
Date: Sat, 15 Nov 2008 13:48:31 GMT
Server: Apache/2.2.10
content-Type: text/html

<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
</HEAD><BODY>
<H1>File Not Found</H1>
The requested URL /teste.html was not found on this server.<P>
</BODY></HTML>
```

Figura 2.4: Resposta de erro HTTP

Aplicações podem ter seus próprios métodos, porém não é obrigatório aos servidores *proxy* interpretá-los. Se um *proxy* recebe uma requisição com um método desconhecido ou não suportado, deve responder com uma mensagem *"405 – Method Not Allowed"*.

2.3.2 FTP

O *File Transfer Protocol* (Protocolo de Transferência de Arquivos) é utilizado desde os primórdios da Internet em 1971. Uma sessão FTP é um pouco mais complicada que uma transação HTTP, já que utiliza um canal de controle, para comandos e respostas, e um canal de dados, para a transferência propriamente dita.

2.3.3 SSL/TLS

A empresa Netscape inventou o protocolo SSL em 1994, para fomentar as aplicações de comércio eletrônico na Internet. O SSL provê **criptografia** ponta a ponta entre clientes e servidores. O desenvolvimento e padronização do SSL passou para a IETF, onde agora é chamado TLS. O protocolo TLS não é restrito ao HTTP e à web, podendo ser utilizado em outras aplicações, como e-mail (SMTP).

Os *proxies* interagem com tráfego HTTP/TLS de duas maneiras: ou como uma ponta final da conexão ou como um dispositivo intermediário. Se o *proxy* for uma ponta final, ele criptografa e descriptografa o tráfego HTTP, e neste caso pode armazenar e reutilizar respostas. Se o *proxy* for um dispositivo intermediário, ele faz um túnel para o tráfego passar de uma ponta para a outra. Como esse tráfego permanece criptografado, não é possível armazenar e reutilizar as respostas.

2.4 Tipos de Cache Web

Conteúdos da web podem ser armazenados em diversos locais diferentes ao longo do caminho entre um cliente e um servidor de origem.

2.4.1 Cache do Navegador

Os navegadores possuem um *cache* interno, e normalmente permitem a configuração de quanto espaço em disco deve ser alocado ao *cache*. Este *cache* é limitado a apenas um usuário ou navegador, portanto só realiza um *hit* quando o usuário revisita um recurso. Por outro lado, o *cache* do navegador pode armazenar respostas privadas, ao contrário dos demais locais de *cache*.

2.4.2 Caching Proxies

Os *caching proxies*, ou "*proxies* que realizam cache", servem a diferentes usuários simultaneamente, ao contrário dos *caches* de navegador. Como vários usuários diferentes visitam os mesmos sites, os *caching proxies* normalmente possuem *hit ratios* mais altos que os *caches* de navegador. De acordo com Duska (1997), conforme aumenta o número de usuários, também aumenta o *hit ratio*.

Os *caching proxies* são serviços essenciais em diversas organizações, como provedores de internet, grandes corporações e Universidades. Normalmente eles rodam em hardware dedicado, que pode ser um *appliance* ou um servidor comum.

Um *proxy* deste tipo localiza-se entre os clientes e os servidores. Diferentemente dos *caches* de navegador, um caching *proxy* altera o caminho dos pacotes, e divide a transação web em duas conexões, uma para o cliente e outra para o servidor. Como é o *proxy* que encaminha a requisição ao servidor, ele esconde o endereço de rede do cliente.

2.4.3 Surrogates

Surrogates também são chamados de *proxy* **reverso** e de acelerador de servidor web. Ele trabalha em conjunto com um ou mais servidores web, e responde às requisições de clientes em nome do servidor web que lhe delegou esta autoridade. As respostas às requisições dos clientes são normalmente buscadas no *cache* do *surrogate*.

Os *surrogates* são úteis em diversas situações. Redes de distribuição de conteúdo os utilizam para replicar informações a vários locais diferentes, posicionando vários *surrogates* em pontos mais próximos aos clientes. Os clientes então são direcionados ao *surrogate* mais próximo.

Outro uso comum de *surrogates* é na aceleração de servidores web, o que ocorre pelo simples fato de realizar *cache* das respostas do servidor. Alguns servidores web tornam-se lentos porque geram todo conteúdo de forma dinâmica. Se uma página é a mesma para todos os clientes que a visitam, um *surrogate* irá acelerar o servidor. Por outro lado, se as páginas são personalizadas para cada cliente, um *surrogate* não fará grande diferença.

Os *surrogates* também podem ser utilizados para descriptografar conexões HTTP/TLS. Como isso exige um certo poder de processamento, ao invés de colocar essa carga no servidor web, um *surrogate* realiza a tarefa. A comunicação entre o *surrogate* e o servidor de origem ocorrerá sem criptografia, porém nestes casos eles ficam próximos e isto não gera maiores problemas.

2.5 Características dos Caching Proxies

A característica chave de um *caching proxy* é a sua habilidade de armazenar respostas para uso futuro. É isso que economiza tempo e largura de banda. No entanto, os *caching proxies* possuem mais recursos interessantes, sendo que a maioria deles são possíveis apenas em um *proxy*, porém têm pouco a ver com o *cache*.

2.5.1 Autenticação

Um *proxy* pode solicitar aos usuários que se autentiquem antes que ele processe qualquer requisição. Dessa forma, apenas usuários autorizados, com nome de usuário e **senha**, podem navegar na Internet a partir da rede da organização. Esta característica é interessante para auditorias em caso de problemas.

2.5.2 Filtragem de Requisições

Organizações normalmente possuem políticas que proíbem certos conteúdos, como pornografia. Para auxiliar na aplicação da política, o *proxy* pode ser configurado para negar requisições a sites de pornografia conhecidos. A filtragem de requisições é de certa forma polêmica. Algumas pessoas a comparam à censura e afirmam corretamente que os filtros não são perfeitos.

2.5.3 Filtragem de Respostas

A filtragem de respostas envolve a verificação do conteúdo de um objeto antes de encaminhá-lo ao cliente. Por exemplo, pode ser feita uma varredura antivírus.

2.5.4 Prefetching

Prefetching é o processo de buscar dados antes de serem requisitados. Discos e memória normalmente utilizam *prefetching*, também conhecido como "*read ahead*". Para a web, *prefetching* envolve a busca de imagens e páginas referenciadas em um arquivo HTML.

2.5.5 Tradução e Transcodificação

Ambos os termos referem-se a processos que alteram o conteúdo de algo sem alterar o seu significado ou aparência. Por exemplo, pode-se imaginar uma aplicação que traduza páginas de texto de Inglês para Português enquanto estão passando pelo *proxy*.

A transcodificação normalmente refere-se a mudanças de baixo nível em dados ao invés de mudanças de alto nível em linguagem humana. Alterar o formato de arquivo de uma imagem de GIF para JPEG é um exemplo. Outro exemplo seria um par de *proxies* compactando os dados transferidos entre eles e descompactando-os antes de serem entregues aos clientes.

2.5.6 Traffic Shaping

Traffic shaping significa "modelagem de tráfego", e é utilizado principalmente para controle de largura de banda. O *proxy* pode ser configurado para limitar a largura de banda de determinados tráfegos baseando-se no cliente ou no tipo de arquivo, por exemplo.

3 DESCRIÇÃO DO SQUID WEB PROXY CACHE

Neste capítulo é feita uma breve descrição do serviço de *proxy* denominado Squid Web Proxy Cache.

3.1 Principais Características

O serviço de *proxy* Squid é um software muito popular, de código fonte aberto e gratuito para utilização em qualquer ambiente, comercial ou não. Roda sobre os sistemas operacionais Linux, FreeBSD, OpenBSD, NetBSD e Windows. De acordo com Wessels (2004), o Squid permite:

- Utilizar menos largura de banda na sua conexão à Internet quando navegar na web;
- Reduzir o tempo que as páginas levam para carregar;
- Proteger os *hosts* da rede interna intermediando o seu tráfego web;
- Coletar estatísticas sobre o tráfego web na sua rede interna;
- Evitar que os usuários visitem sites inapropriados;
- Garantir que apenas usuários autorizados possam navegar na Internet;
- Melhorar a privacidade dos usuários através da filtragem de informações sensíveis em requisições HTTP;
- Reduzir a carga no seu servidor web;
- Converter requisições HTTPS em HTTP.

O Squid não realiza serviço de *firewall*, sua função é ser ***proxy*** HTTP e ***cache***. Como um *proxy*, o Squid é um intermediário em uma transação web. Ele aceita uma requisição de um cliente, processa esta requisição, e então a encaminha ao servidor web. A requisição pode ser registrada, rejeitada, e até mesmo modificada antes do encaminhamento. Como um *cache*, o Squid armazena conteúdos buscados recentemente para um possível reuso posterior. As próximas solicitações do mesmo conteúdo poderão ser servidas a partir do *cache*, ao invés de ter de contatar o servidor web novamente. A parte de *cache* do Squid pode ser desabilitada, porém a parte de *proxy* é essencial.

3.2 Breve História do Squid

No ínicio havia o servidor HTTP CERN. Além de funcionar como servidor HTTP, ele também foi o primeiro *proxy* e *cache*. O módulo de *cache* foi escrito por Ari Luotonen em 1994.

No mesmo ano, o *Internet Research Task Force Group on Resource Discovery* (IRTF-RD) iniciou o projeto Harvest. Ele era um conjunto integrado de ferramentas para coletar, extrair, organizar, localizar, fazer *cache* e replicar informações da Internet. Duane Wessels juntou-se ao projeto Harvest no final de 1994. Apesar de o Harvest estar sendo mais utilizado como mecanismo de busca, o componente Object Cache também era bastante popular. O cache Harvest forneceu três melhorias importantes em relação ao cache CERN: maior velocidade no uso do sistema de arquivos, design de processo único, e hierarquias de cache através do *Internet Cache Protocol*.

No final do ano de 1995, diversos membros da equipe do Harvest abandonaram o projeto para dedicarem-se à empresas relacionadas a Internet. Os autores originais do código do *cache* do Harvest, Peter Danzig e Anawat Chankhunthod, tornaram-no em um produto comercial. Sua empresa foi posteriormente adquirida pela Network Appliance. No início de 1996, Duane Wessels integra-se ao *National Laboratory for Applied Network Research* (NLANR) para trabalhar no projeto *Information Resource Cache* (IRCache), financiado pela *National Science Foundation*. Neste projeto, o código do *cache* do Harvest foi renomeado para Squid, e liberado sob a licença GNU GPL.

Desde aquele momento o Squid tem crescido em tamanho e funcionalidades. Agora suporta funções como redirecionamento de URL, *traffic shaping*, controles de acesso sofisticados, diversos módulos de autenticação, opções de armazenamento avançadas, interceptação HTTP e modo *surrogate*.

O financiamento ao projeto IRCache terminou em julho de 2000. Hoje voluntários continuam a desenvolver o Squid. Ocasionalmente recebem suporte financeiro ou outras formas de auxílio de empresas que utilizam o Squid.

3.3 Formato do Arquivo de Log do Squid

O Squid possui dois formatos de *log*: nativo e *common log file* (CLF). O formato CLF é definido pelo *CERN web daemon* – também conhecido como W3C httpd – e é utilizado por diversos servidores HTTP. O padrão do Squid é utilizar o formato nativo, que possui mais informações interessantes à administração do servidor.

O Squid registra em seu *log* uma linha para cada objeto requisitado pelos clientes. Na Figura 3.1 há um trecho de um arquivo access.log, para demonstrar o seu formato.

```
116553498.320    419 192.168.97.127 TCP_HIT/200 40511 GET
http://us.js2.yimg.com/us.js.yimg.com/lib/pim/r/medici/15_6/mail/m
ailcommonlib.js - NONE/- application/x-javascript
116553498.320    419 192.168.96.110 TCP_MISS/200 1612 GET
http://br.adserver.yahoo.com/a? - DIRECT/200.152.161.101
application/x-javascript
116553498.320     24 192.168.94.86 TCP_HIT/200 1096 GET
http://images.meebo.com/image/skin/beta/sound/sound.swf - NONE/-
application/x-shockwave-flash
116553498.321   3194 192.168.96.110 TCP_MISS/200 9985 POST
http://br.f325.mail.yahoo.com/ym/Compose? - DIRECT/68.142.207.33
text/html
116553498.343    217 192.168.98.202 TCP_MISS/200 1195 GET
http://br.adserver.yahoo.com/a? - DIRECT/200.152.161.101
application/x-javascript
116553498.343    442 192.168.97.127 TCP_HIT/200 46026 GET
http://us.js2.yimg.com/us.js.yimg.com/lib/pim/r/medici/15_6/mail/u
s/mail_blue_all.css - NONE/- text/css
```

Figura 3.1: Trecho de arquivo access.log

O formato de *log* nativo possui as seguintes informações:

3.3.1 Hora (*time*)

Timestamp Unix, marcando o número de segundos desde 01 de janeiro de 1970, com resolução de milissegundos.

3.3.2 Duração (*duration*)

Tempo decorrido, considerando quantos milissegundos a transação utilizou o *cache*.

3.3.3 Endereço do cliente (*client address*)

Endereço IP do cliente que está requisitando algo ao *proxy*.

3.3.4 Códigos de resultado (*result codes*)

Contém o resultado da transação, e é composta de duas informações separadas por uma barra: o código de resultado da requisição ao cache, e o código de resultado HTTP, com algumas extensões específicas do Squid. Os principais códigos de resultado da requisição ao *cache* são os seguintes:

- TCP_HIT: uma cópia válida do objeto foi encontrada no *cache*.
- TCP_MISS: não há cópia do objeto requisitado no *cache*.
- TCP_REFRESH_HIT: o objeto requisitado estava em *cache* mas era considerado vencido. A requisição ao objeto para o servidor de origem revelou que ele não foi modificado.
- TCP_REF_FAIL_HIT: o objeto requisitado estava em *cache* mas era considerado vencido. A requisição ao objeto para o servidor de origem falhou, e foi entregue ao cliente a cópia vencida.

- TCP_REFRESH_MISS: o objeto requisitado estava em *cache* mas era considerado vencido. A requisição ao objeto para o servidor de origem revelou que ele foi modificado, então foi entregue ao cliente o objeto atualizado.
- TCP_CLIENT_REFRESH_MISS: o cliente utilizou na requisição um código *no-cache* ou semelhante, então o *cache* buscou novamente o objeto no servidor de origem.
- TCP_MEM_HIT: uma cópia válida do objeto requisitado estava no *cache* e na memória, o que evita leitura de disco.
- TCP_DENIED: o acesso foi negado para esta requisição.

3.3.5 Tamanho (*bytes*)

Quantidade de bytes de dados enviados pelo *proxy* ao cliente.

3.3.6 Método de requisição (*request method*)

Método de requisição utilizado para obter o objeto, conforme definido na RFC 2616 e na RFC 2518.

3.3.7 URL

Endereço web do objeto requisitado.

3.3.8 RFC931

Esta informação refere-se à consulta *ident* que pode ser feita contra o cliente requisitando algo. Por padrão tal consulta é desativada, devido ao impacto negativo na performance do *proxy*, e esta informação é preenchida com um hífen.

3.3.9 Código de hierarquia (*hierarchy code*)

Contém informações sobre a hierarquia de servidores *proxy*, sobre como a requisição foi resolvida, e o endereço IP ou *hostname* do servidor que enviou o objeto ao *proxy*. Para servidores *proxy* sem estrutura hierárquica, os códigos que informam como a requisição foi resolvida são:

- NONE: ocorreu um resultado TCP_HIT ou então uma falha.
- DIRECT: o objeto foi buscado no servidor de origem.

3.3.10 Tipo (*type*)

Tipo de conteúdo, conforme informado no cabeçalho da resposta HTTP.

3.4 Conclusão

O Squid Web Proxy Server é um software consolidado e com um histórico longo e de evolução constante. É hoje cada vez mais utilizado para auxílio na redução de custo e controle de navegação em grandes redes. Seu arquivo de *log* é bem estruturado e padronizado, o que facilita a extração de dados para posterior análise.

4 PRINCIPAIS FERRAMENTAS DE ANÁLISE DE LOG DO SQUID

4.1 SARG – Squid Analysis Report Generator

O Squid Analysis Report Generator, conhecido como SARG, foi criado e é mantido por Pedro Lineu Orso. Desenvolvido em linguagem C, roda em sistemas operacionais Unix-like e é software livre. De acordo com seu criador, "é uma ferramenta que permite ver onde seu usuário está indo na Internet". O SARG suporta arquivos de *log* do Squid, do Microsoft ISA e do Novell Border Manager. Está disponível em diversos idiomas, inclusive Português.

Os relatórios são gerados em arquivos HTML. São separados por períodos de datas, e possuem informações como nome de usuário ou *hostname*/endereço IP do computador cliente do *proxy*, número de conexões, volume de tráfego, sites acessados, data e hora. Também há relatórios que mostram quais os sites mais acessados e quais os sites que tiveram o acesso proibido pelo Squid. O único tipo de gráfico que há demonstra o volume de tráfego, em bytes, em cada dia do período analisado. Estes gráficos são individuais por usuário ou computador.

4.2 Internet Access Monitor

O Internet Access Monitor é um software desenvolvido pela Red Line Software, e roda em sistema operacional Windows. Suporta os arquivos de log de diversos serviços de *proxy*, inclusive do Squid. Possui suporte a diversos idiomas, porém não Português. Não é software livre, e exige pagamento de licença para sua utilização.

Este software oferece relatórios bastante aprimorados e flexíveis. Os relatórios são montados e visualizados no próprio programa, e pode-se exportá-los para diversos formatos. É possível filtrar e visualizar os dados por data, horário, distribuição de tráfego por usuários, endereços IP, serviços, protocolos, tipos de conteúdo e sites visitados, entre outros. Também é flexível a criação de gráficos, que podem ser gerados a partir de diversos dados.

4.3 Sawmill

O Sawmill, descrito como "*Universal Log File Analysis & Reporting*", é desenvolvido pela empresa Flowerfire. Não é software livre, sendo necessário o pagamento de licença para sua utilização. Roda nos sistemas operacionais Windows, Apple, Linux, Solaris, FreeBSD e OpenBSD. Conforme consta na sua descrição, o

Sawmill é um analisador de *logs* genérico, suportando centenas de tipos de arquivos de *log*.

O Sawmill roda no micro como um serviço, sendo acessível via navegador. Sua utilização é baseada em perfis, que são criados com base no tipo de arquivo de *log* que será analisado. Analisando arquivos de *log* do Squid, são extraídos diversos dados incluindo distribuição de acessos por faixa de horário e por data, endereços IP dos clientes, tipos de arquivos, tipos de resposta dos servidores, ações HTTP e operações HTTP. Uma característica interessante deste software é que os relatórios são sumarizados. Por exemplo, na listagem de endereços IP dos clientes, são relacionados por padrão os dez endereços que mais realizaram pedidos, cabendo ao usuário optar pela visualização de mais itens, se desejado. Os relatórios gerados pelo Sawmill podem ser exportados para o formato CSV.

4.4 Conclusão

Analisando estas ferramentas, verifica-se que realmente existe a carência de recursos em software livre para análise de log do Squid. As ferramentas proprietárias possuem várias funcionalidades inexistentes em forma gratuita.

5 ANÁLISE E MODELAGEM DO SOFTWARE

5.1 Descrição Narrativa

O software, que irá se chamar Squid Web Stats, servirá como uma ferramenta para auxiliar na análise de arquivos de *log* do Squid. O *log* analisado será o access.log, que é o registro de todas as solicitações dos clientes do *proxy*.

Os registros do arquivo de *log* serão importados em um banco de dados, para maior velocidade de extração e combinação destes dados. Os arquivos de *log* importados serão agregados em um perfil, e poderão existir vários perfis, para separar conjuntos de *log* ou situações diferentes.

Os relatórios serão criados a partir dos dados de um perfil no banco de dados. Existirão relatórios gráficos e textuais. Os relatórios textuais fornecem maior quantidade de informações, enquanto que os gráficos fornecem uma visualização mais intuitiva e impactante.

As informações dos relatórios são criadas a partir dos dados existentes no arquivo de *log*, que fora importado ao banco de dados. Estas informações são geradas a partir da necessidade imposta pelos relatórios e são limitadas pelos dados contidos no *log*.

5.2 Levantamento de Requisitos

Conforme Silva (2005), os requisitos são uma coleção de sentenças que devem descrever de modo claro, sem ambigüidades, conciso e consistente todos os aspectos significativos do sistema proposto. Eles devem conter informações suficientes para permitir que os implementadores construam um sistema que satisfaça os requerentes, e nada mais.

Sommerville (2003), afirma que um requisito é tratado como funcional quando descreve um serviço ou função que o sistema deve realizar. Paralelamente pode haver requisitos não-funcionais, que são restrições impostas tanto ao sistema quanto ao seu desenvolvimento.

5.2.1 Requisitos funcionais

O software desenvolvido deverá:

- Ser acessível via web;
- Restringir acesso através de usuário e senha;
- Efetuar a leitura de arquivos de *log* no formato nativo do Squid;

- Criar relatórios contendo informações como:
    - Nome de usuário;
    - *Hostname*/endereço IP do cliente;
    - Sites acessados;
    - Volume de tráfego;
    - Tipos de arquivos acessados;
    - Horário e faixas de horário.
- Organizar os relatórios cronologicamente;
- Permitir a ordenação das informações do relatório em forma ascendente e descendente dos diferentes dados;
- Criar gráficos para os principais dados dos relatórios;
- Permitir a filtragem de dados do relatório com base em critérios como:
    - Nome de usuário;
    - *Hostname*/endereço IP do cliente;
    - Site acessado;
    - Tipos de arquivos acessados;
    - Faixa de horário.
- Permitir a exportação dos relatórios para o formato PDF.

5.2.2 Requisitos não-funcionais

- Segurança e privacidade
    - O sistema não deverá fornecer quaisquer dados relativos ao *log* sem antes realizar a autenticação do usuário;
    - Deverá existir uma opção para desativar a exibição de nomes de usuário, no caso de análise de *log* de *proxy* autenticado.
- Tecnologia
    - O sistema deverá funcionar corretamente em hardware plataforma x86, sistema operacional Linux, servidor web Apache 2 com suporte a PHP 5, e banco de dados MySQL 5.
- Padrões
    - O sistema deverá funcionar corretamente em navegadores atuais, seguindo os padrões correntes de desenvolvimento de software para web, como o padrão XHTML do W3C.

5.3 Casos de Uso

Para Rumbaugh (1998), os casos de uso modelam a funcionalidade do sistema como percebida pelos usuários externos, chamados atores. Um caso de uso é uma unidade coerente de funcionalidades visíveis externamente, fornecidas por uma unidade do

sistema, e expressada por uma seqüência de mensagens trocadas entre a unidade do sistema e um ou mais atores da unidade do sistema. O propósito de um caso de uso é definir uma peça de comportamento coerente, sem revelar a estrutura interna do sistema. Ainda conforme Rumbaugh, um caso de uso é uma descrição lógica de uma parte de funcionalidade do sistema.

Seguem abaixo o diagrama geral de casos de uso e os casos de uso específicos com seus respectivos fluxos.

5.3.1 Diagrama geral de casos de uso

Segue na Figura 5.1 o diagrama geral de casos de uso.

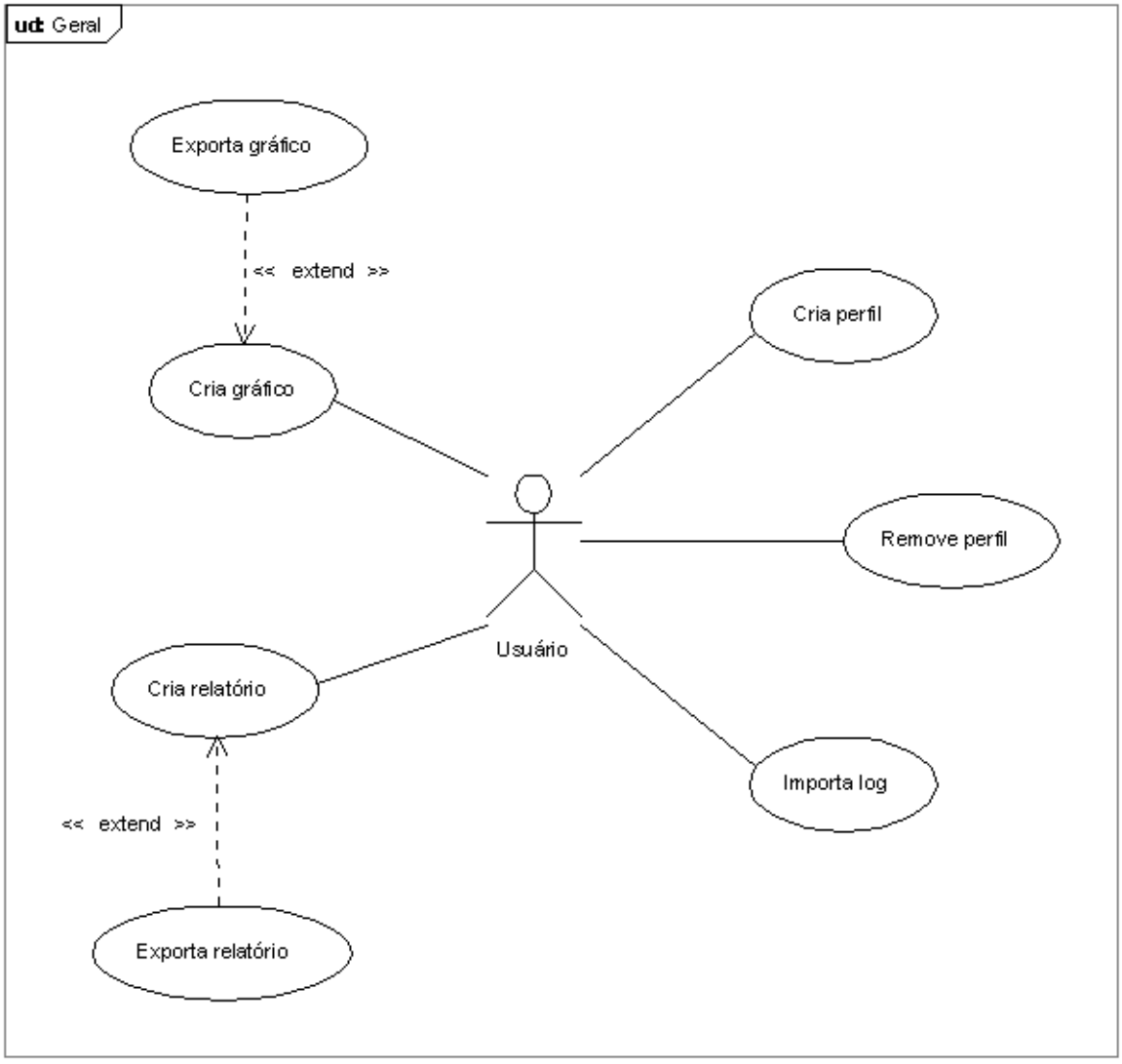

Figura 5.1: Diagrama geral de casos de uso

5.3.2 Criar perfil

A Figura 5.2 contém o diagrama de caso de uso relativo à operação de criação de um novo perfil.

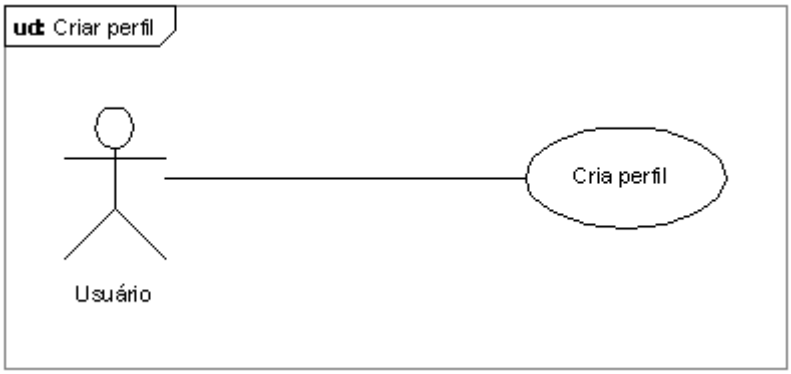

Figura 5.2: Diagrama de caso de uso - Criar perfil

Pré-condições

a) Usuário deverá ter acessado o sistema por meio de *login* e senha.

Fluxo

a) Usuário clica em 'Criar novo perfil';
b) Preenche o nome do novo perfil;
c) Clica em 'Ok'.

5.3.3 Remover perfil

Na Figura 5.3, diagrama de caso de uso relativo à remoção de perfil.

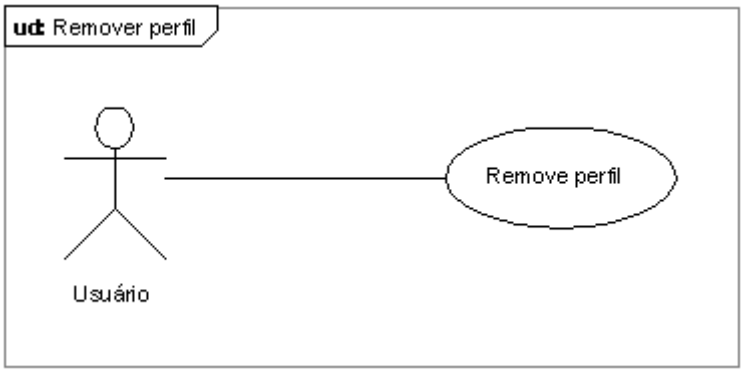

Figura 5.3: Diagrama de caso de uso - Remover perfil

Pré-condições

a) Usuário deverá ter acessado o sistema por meio de *login* e senha.

Fluxo

a) Usuário seleciona o perfil desejado;
b) Clica em 'Remover este perfil';
c) Clica em 'Ok'.

5.3.4 Importar log

Segue na Figura 5.4 o diagrama de caso de uso relativo à importação de *log*.

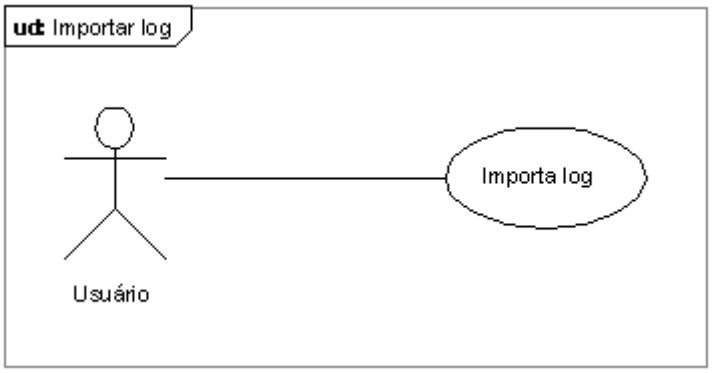

Figura 5.4: Diagrama de caso de uso - Importar log

Pré-condições

a) Usuário deverá ter acessado o sistema por meio de *login* e senha.

Fluxo

a) Usuário seleciona o perfil desejado;
b) Clica em 'Importar log';
c) Especifica o caminho do arquivo de *log*;
d) Clica em 'Ok'.

5.3.5 Criar relatório

Na Figura 5.5, diagrama de caso de uso sobre a operação de criação de relatório.

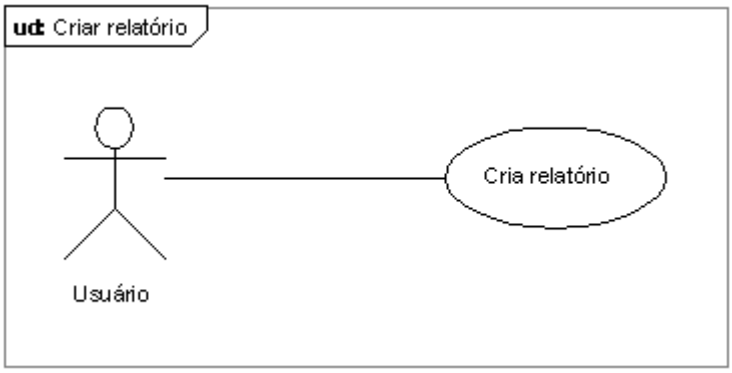

Figura 5.5: Diagrama de caso de uso - Criar relatório

Pré-condições

a) Usuário deverá ter acessado o sistema por meio de *login* e senha.

Fluxo

a) Usuário seleciona o perfil desejado;

b) Clica em 'Relatório';

c) Define o início e o final do período desejado;

d) Seleciona os dados que deverão constar no relatório;

e) Clica em 'Ok'.

5.3.6 Criar gráfico

O diagrama de caso de uso relativo à criação de gráfico aparece na Figura 5.6.

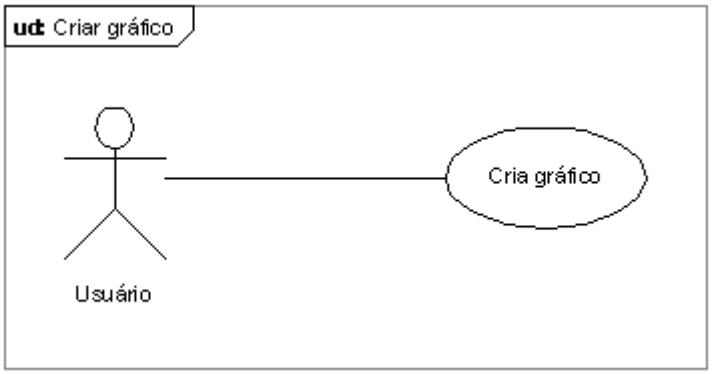

Figura 5.6: Diagrama de caso de uso - Criar gráfico

Pré-condições

a) Usuário deverá ter acessado o sistema por meio de *login* e senha.

Fluxo

a) Usuário seleciona o perfil desejado;

b) Clica em 'Gráfico';

c) Define o início e o final do período desejado;

d) Seleciona os dados que deverão constar no gráfico;

e) Clica em 'Ok'.

5.4 Diagrama de Classes

Encarar o desafio de modelar um sistema fundamentando-se nos conceitos de orientação a objetos parece até fácil em um primeiro momento, afinal, são objetos do mundo real que serão descritos em uma linguagem diagramática. As relações entre os objetos e a estruturação de um modelo organizado, onde os objetos fluam entre as classes através de métodos adequados e consistentes foi, este sim, o primeiro desafio.

A percepção de que havia uma categoria de métodos que servia exclusivamente para a construção de interfaces, outra que fazia o acesso ao banco, e outra que executava operações diversas com objetos, foi o grande ponto de partida para a solução do problema inicial da construção do diagrama de classes do sistema.

Pesquisando por metodologias de modelagem que atendessem a esta nova visão, ainda não totalmente amadurecida, que se tinha do sistema, foi encontrado o design pattern **MVC**, *Model-View-Controller*, ou Modelo-Visão-Controle, que possui como objetivo básico, separar de maneira clara e independente o Modelo, que representa os objetos de negócio (*Model*); a camada de apresentação, que representa a interface com o usuário ou outro sistema (*View*); e o controle de fluxo da aplicação (*Controller*). Assim, a aplicação se divide em três camadas independentes, com classes e métodos bem definidos.

Outras vantagens de uso do MVC são: como o modelo MVC gerencia múltiplas visões usando o mesmo modelo é fácil manter, testar e atualizar sistemas múltiplos; é muito simples incluir novos clientes apenas incluindo suas visões e controles; torna a aplicação escalável; é possível ter desenvolvimento em paralelo para o modelo, para a visão e para o controle, pois são independentes.

A Figura 5.7 mostra o relacionamento básico do usuário com o padrão MVC.

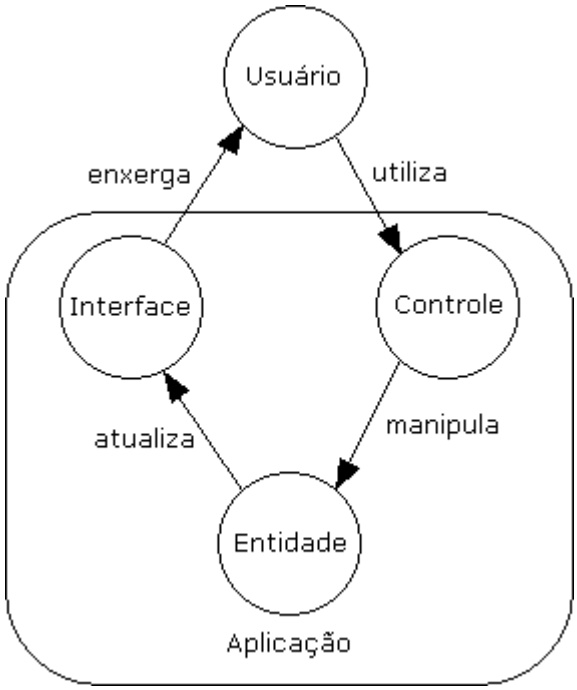

Figura 5.7: Relacionamento básico MVC

A aplicação foi projetada buscando uma aproximação ao padrão MVC, conforme mostra a Figura 5.8.

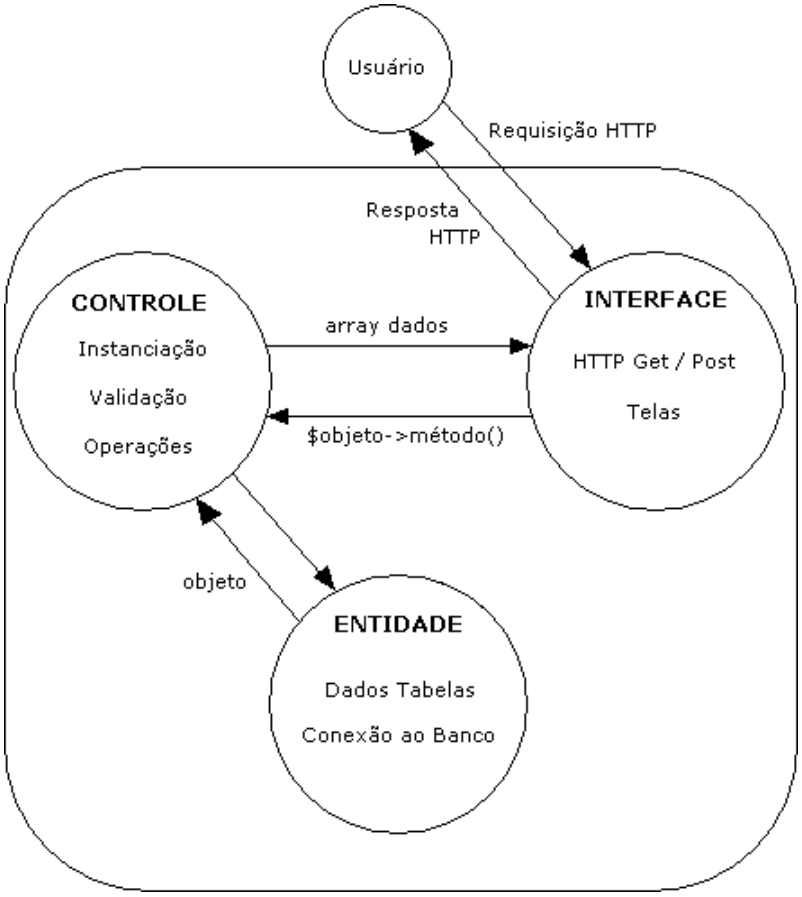

Figura 5.8: Organização do sistema proposto

A Figura 5.9 exibe a mesma estruturação do sistema, vista de outra perspectiva.

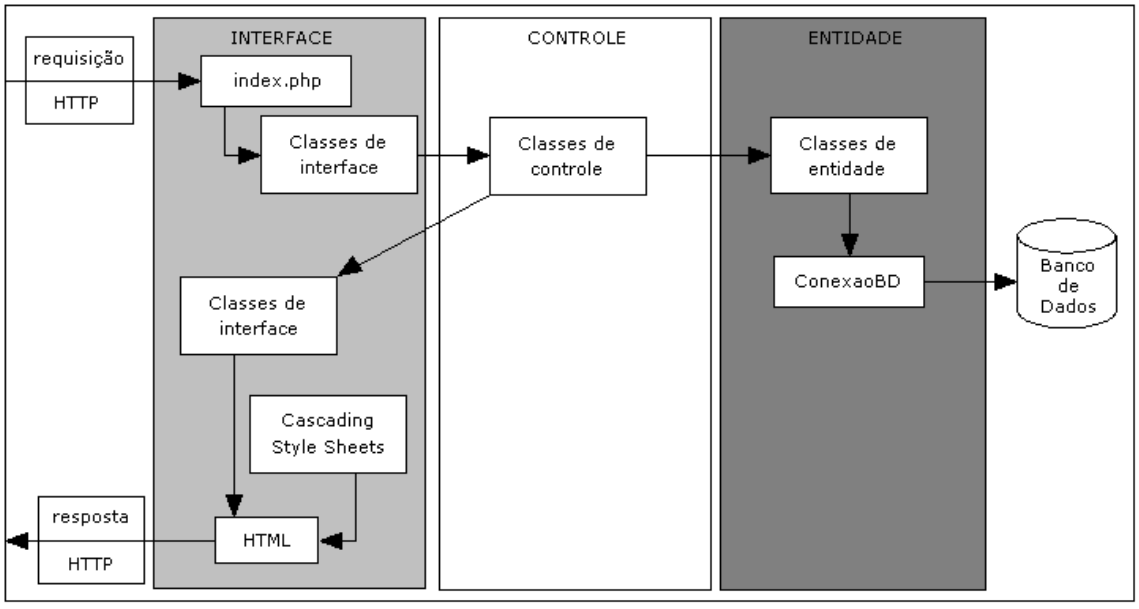

Figura 5.9: Organização do sistema proposto

5.4.1 Diagrama de classes

Na Figura 5.10 encontra-se o diagrama de classes.

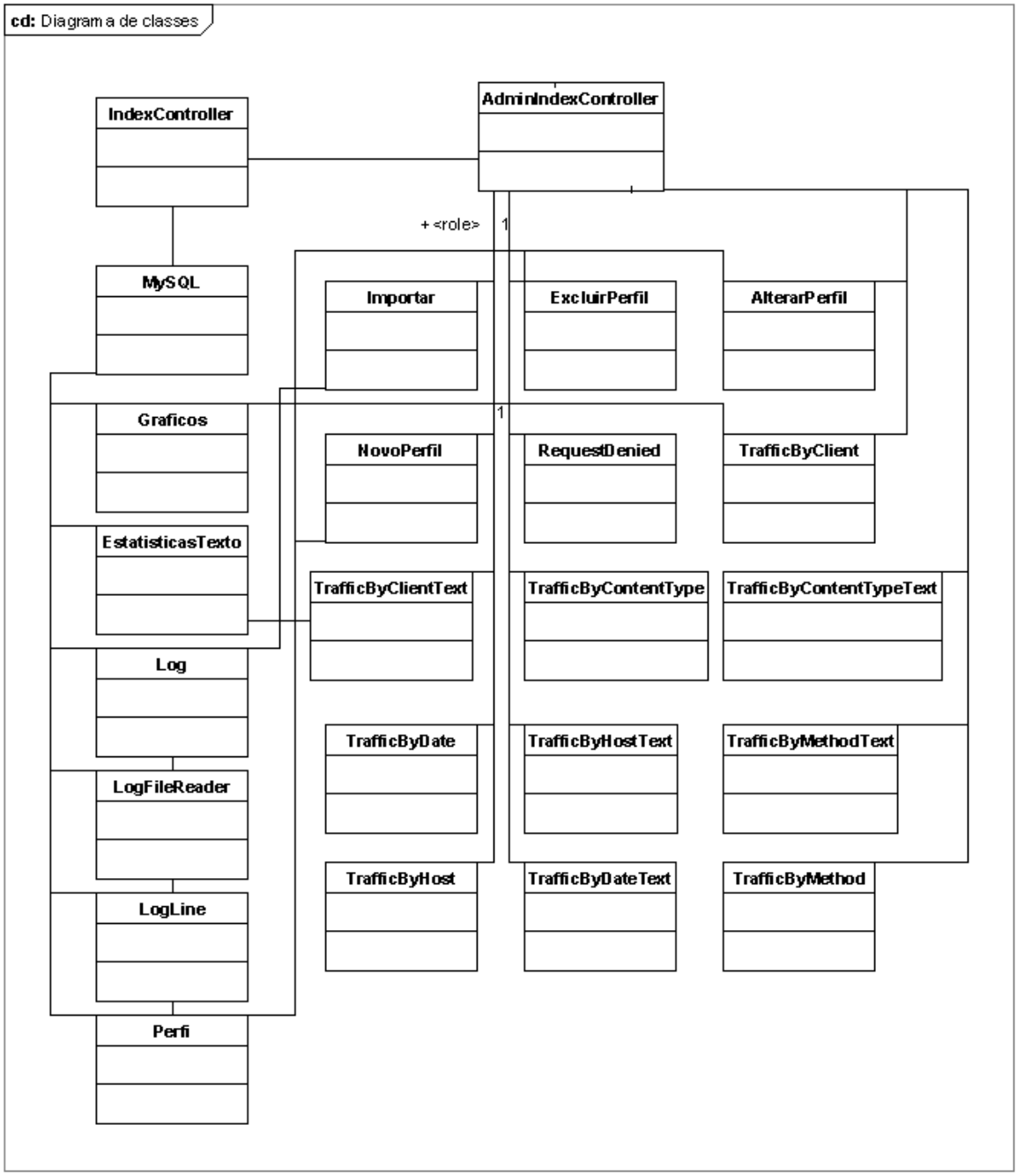

Figura 5.10: Diagrama de classes

5.5 Diagramas de Atividade

O diagrama de atividade mostra o fluxo de controle de uma atividade para outra, representando um processamento. Seguem diagramas de atividade para exemplificar como ocorre o fluxo em duas operações do sistema: efetuar login e criar perfil.

5.5.1 Efetuar login

A Figura 5.11 representa o fluxo do processamento quando o usuário efetua login no sistema.

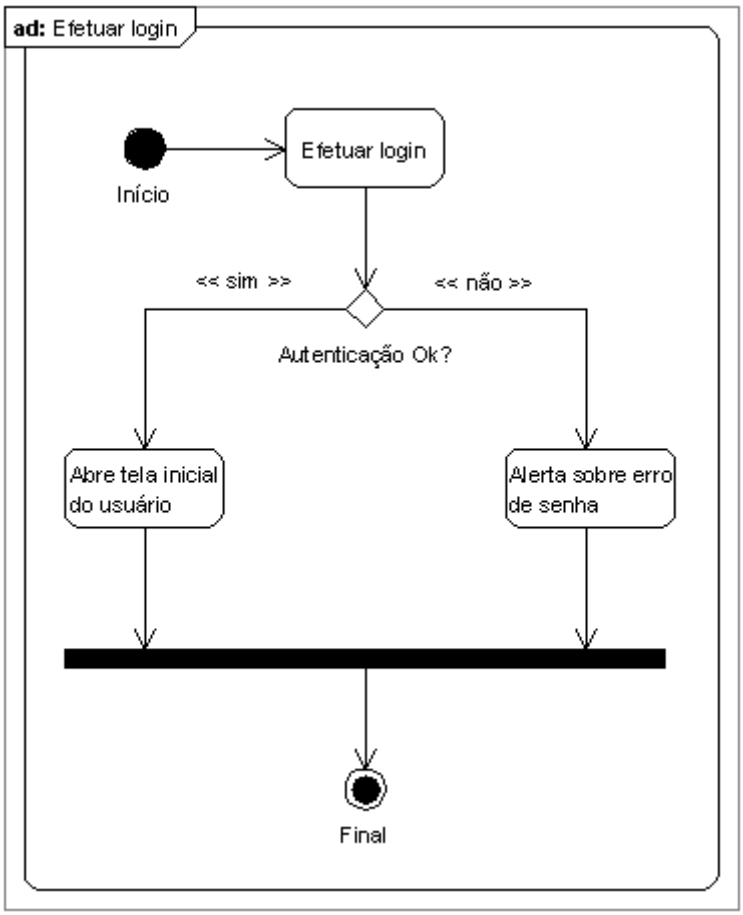

Figura 5.11: Diagrama de atividade - Efetuar login

5.5.2 Criar perfil

O diagrama da Figura 5.12 representa o fluxo do processamento quando o usuário cria um novo perfil no sistema.

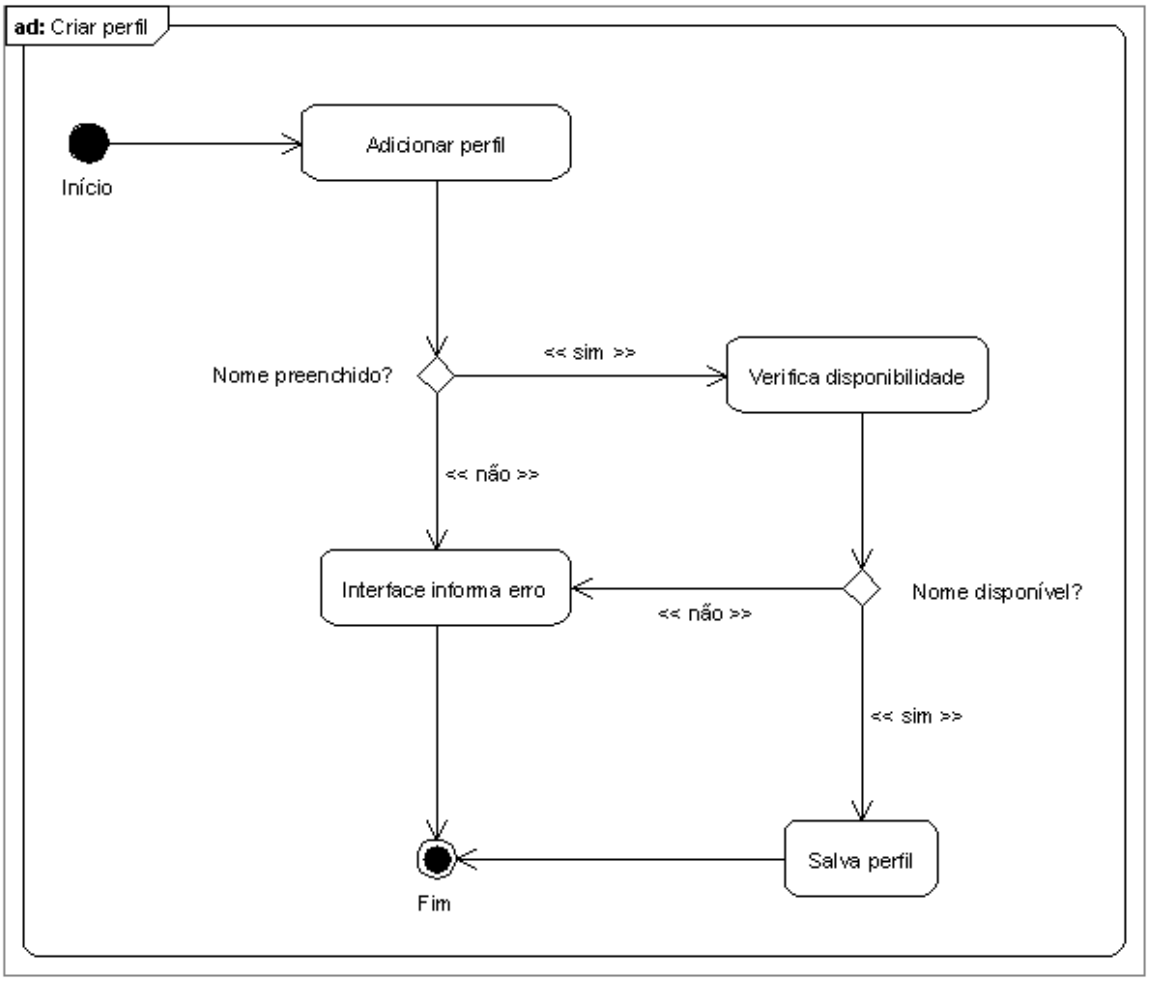

Figura 5.12: Diagrama de atividade - Criar perfil

5.6 Diagrama de Execução

Neste momento o sistema está sendo projetado para ser executado em apenas um servidor, com os clientes acessando via rede. Neste mesmo servidor rodam o Apache HTTP Server, com suporte a PHP versão 5, e também o SGBD MySQL. Na Figura 5.13 o diagrama de execução.

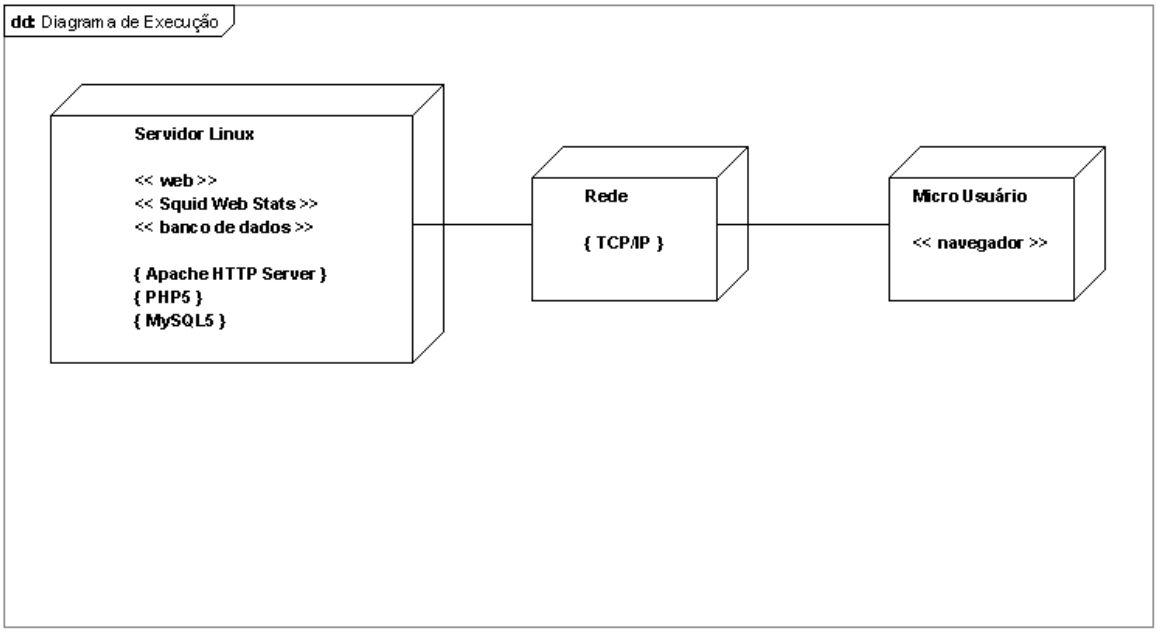

Figura 5.13: Diagrama de execução

6 DESENVOLVIMENTO

6.1 Banco de Dados

O banco de dados criado para esta aplicação é bastante simples. Uma tabela, chamada “profiles”, armazena as descrições e os identificadores únicos de cada perfil. A outra tabela, chamada “access_log”, reflete a estrutura do arquivo de *log* nativo do Squid, possuindo uma coluna para cada informação que consta no arquivo, além de colunas para identificação única e para chave estrangeira em relacionamento com a tabela “profiles”. A Figura 6.1 mostra a estrutura do banco.

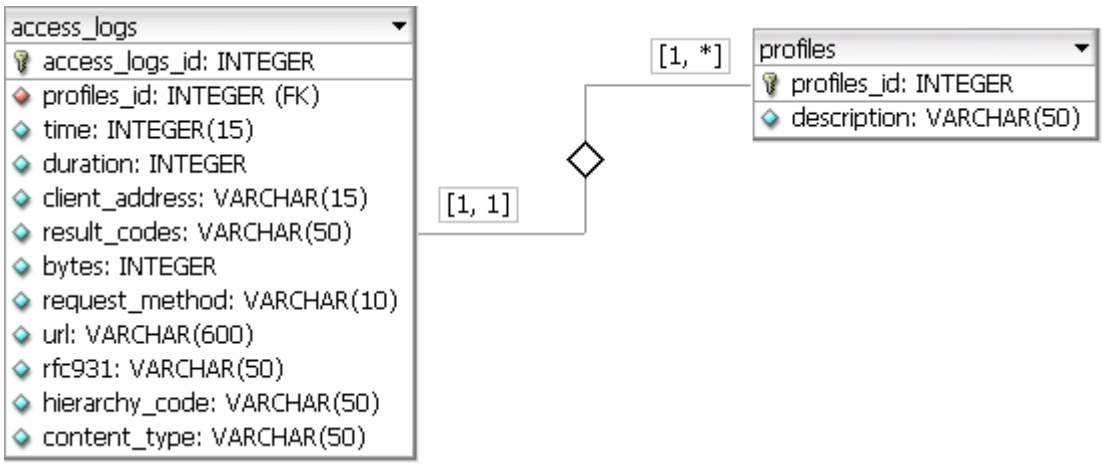

Figura 6.1: Estrutura do banco de dados

Ao implementar o banco de dados da aplicação, houve a preocupação de garantir a integridade referencial dos dados entre as tabelas relacionadas. Assim, buscou-se saber qual a maneira mais adequada de fazer esta implementação sobre o SGBD MySQL, o qual é um requisito funcional do sistema. O tipo de tabela padrão do MySQL é o MyISAM, que não suporta integridade referencial, e se forem criadas tabelas com *Foreign Keys* (chaves estrangeiras) e *Constraints* (restrições no decorrer de uma transação), isto será ignorado. Dessa forma optou-se por buscar um meio de controlar a integridade, mesmo utilizando MySQL. Pesquisando um pouco chegou-se ao InnoDB, um tipo de tabela que pode ser utilizado no MySQL, a partir de sua versão 4.0. O InnoDB é um mecanismo de armazenamento com capacidade de operações como *commit* e *rollback*, suportando controle transacional. Características importantes do InnoDB:

- Suporte para transações ACID;

- Suporta *Foreign Keys* e integridade referencial com implementação dos constraints *Set Null*, *Set Default*, *Restrict* e *Cascade*;
- Alto desempenho com grandes volumes de dados e um número elevado de concorrência entre leitura e escrita;
- Implementa bloqueio a nível de registro, o que fornece um melhor gerenciamento da concorrência, assim como fazem DB2 e Oracle.

O mecanismo InnoDB é utilizado em produção em diversos bancos de dados que exigem grande performance, como o site Slashdot.org e a empresa Mytrix, Inc. Esta última armazena mais de 1 TB de dados no InnoDB.

Apesar de tudo isso, acabou sendo utilizado o mecanismo MyISAM, devido à diferença de performance percebida na prática. Nos testes executados, extraindo dados do arquivo de *log* e os inserindo em uma tabela, o InnoDB obteve performance máxima de 110 linhas por segundo, enquanto que o MyISAM atingiu 4.544 linhas por segundo. A performance do InnoDB manteve-se assim mesmo com a integridade referencial desativada. Tendo em vista que o arquivo de *log* a ser importado pode possuir milhões de linhas, é muito importante a velocidade de inserção no banco. Portanto, foi utilizado o tipo MyISAM para as tabelas.

6.1.1 Extração de informações da tabela access_log

A extração das informações que irão constar nos relatórios é feita através de consultas SQL ao banco de dados. Para exemplificar essa extração, seguem abaixo dois exemplos de consultas, com comentários.

6.1.1.1 Tráfego por cliente

A consulta da Figura 6.2 é utilizada para o relatório gráfico de tráfego por cliente. Os passos desta consulta são:

a) Encontrar endereços IP de todos os clientes;

b) Para cada endereço IP de cliente, verificar todas as linhas onde ele ocorre e somar o campo bytes;

c) Apresentar o endereço IP do cliente e o somatório de bytes dividido por 1024, para conversão em KB;

d) Colocar em ordem decrescente de tráfego e mostrar os 10 primeiros.

```
SELECT client_address, ROUND((SUM(bytes)/1024)) as kbytes
  FROM access_log
  WHERE profile_id = 1
  GROUP BY client_address
  ORDER BY kbytes desc
  LIMIT 10;
```

Figura 6.2: Consulta SQL para tráfego por cliente

6.1.1.2 Requisições negadas por cliente

A consulta utilizada no relatório de requisições negadas por cliente aparece na Figura 6.3, e segue os seguintes passos:

a) Extrair todos os endereços IP de clientes únicos, e apresentar em ordem crescente;

b) Quando um endereço IP de cliente for clicado, exibir os URL das linhas onde o código de resultado for igual a “TCP_DENIED/403”.

```
SELECT client_address
  FROM access_log
  WHERE profile_id = 1
    AND result_codes = "TCP_DENIED/403"
  GROUP BY client_address
  ORDER BY client_address;

SELECT url
  FROM access_log
  WHERE result_codes = "TCP_DENIED/403"
    AND client_address = "192.168.96.51";
```

Figura 6.3: Consultas SQL para requisições negadas por cliente

6.2 Srad Framework

Para desenvolvimento da aplicação foi utilizado o Srad framework, desenvolvido por Anderson Meggiolaro.

O Srad Framework é um *framework* escrito em PHP5, para desenvolvimento rápido de aplicações web. Atualmente está em estado *beta*. É orientado a objetos e baseado no padrão MVC. As aplicações também são desenvolvidas orientadas a objetos.

Utiliza o Smarty Template Engine como gerenciador de *templates* e a biblioteca ADOdb para gerenciamento do acesso à camada de banco de dados. Tem suporte nativo a AJAX, permitindo o desenvolvimento rápido de aplicações sem ter que escrever muitas linhas de código javascript.

6.3 Smarty Template Engine

O Smarty é um gerenciador de templates – modelos de apresentação – para PHP. Ele fornece um meio bem estruturado para separar o conteúdo e as regras de negócios da apresentação.

Algumas das características do Smarty, segundo Ohrt (2005):

- É extremamente rápido;
- É eficiente visto que o interpretador do PHP faz o trabalho mais pesado;
- Sem elevadas interpretações de template, apenas compila uma vez;
- Está atento para só recompilar os arquivos de template que foram mudados;

- Podem ser criadas funções customizadas e modificadores de variáveis customizados, de modo que a linguagem de template é extremamente extensível;
- Delimitadores de tag configuráveis, sendo assim você pode usar {}, {{}}, <!--{}-->, etc;
- Os construtores if/elseif/else/endif são passados para o interpretador de PHP, assim a sintaxe de expressão {if ...} pode ser tanto simples quanto complexa, conforme for necessário;
- É permitido o aninhamento ilimitado de sections, ifs, etc;
- É possível embutir o código PHP diretamente nos arquivos de template, apesar de que isto pode não ser necessário (não recomendado) visto que a ferramenta é tão customizável;
- Suporte a *caching* embutido;
- Fontes de *template* arbitrários;
- Funções de manipulação de *cache* customizadas;
- Arquitetura de *plugin.*

6.4 ADOdb

ADOdb é uma biblioteca de abstração de banco de dados para PHP. Possui suporte a PHP5 e a diversos bancos de dados, entre eles MySQL, PostgreSQL, Firebird, Informix, Oracle, MS SQL e DB2.

Conforme Lim (2006), as funções do PHP para acesso a banco de dados não são padronizadas. Isso cria a necessidade de uma biblioteca com classes que encapsulam as diferenças entre as várias API's de bancos de dados, permitindo a padronização dos códigos e também a troca facilitada de SGBD quando necessário.

6.5 AJAX

De acordo com Garret (2005), AJAX é o uso sistemático de javascript e XML (e derivados) para tornar o navegador mais interativo com o usuário, utilizando-se de solicitações assíncronas de informações. AJAX não é somente um novo modelo, é também uma iniciativa na construção de aplicações web mais dinâmicas e criativas. AJAX não é uma tecnologia, são realmente várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades. AJAX incorpora em seu modelo:

- Apresentação baseada em padrões, usando XHTML e CSS;
- Exposição e interação dinâmica usando o DOM;
- Intercâmbio e manipulação de dados usando XML e XSLT;
- Recuperação assíncrona de dados usando o objeto XMLHttpRequest;
- Javascript.

6.6 GD Graphics Library

GD é uma biblioteca de código livre para a criação dinâmica de imagens. É escrita em C, e existem *wrappers* para PHP, Perl e outras linguagens. A biblioteca GD gera gráficos nos formatos PNG, JPEG e GIF, entre outros. É utilizada para geração de imagens em tempo de execução, sendo que a grande maioria das aplicações que a utilizam são websites.

A biblioteca GD foi utilizada no Squid Web Stats, encapsulada pela biblioteca JpGraph, para gerar os gráficos.

6.7 Biblioteca JpGraph

A JpGraph é uma biblioteca orientada a objetos utilizada para geração de gráficos no PHP, e funciona como um *wrapper* para a biblioteca GD. Ela é escrita totalmente em PHP, e foi utilizada no Squid Web Stats para gerar os relatórios gráficos.

6.8 Conclusão

O desenvolvimento de um software como este é bastante agilizado com a utilização de um *framework* que se adeque à proposta. O Srad Framework serviu muito bem, abstraindo diversos conceitos que seriam trabalhosos ao desenvolver um sistema a partir do zero. O Srad também utiliza tecnologias atuais em sua estrutura, como ADOdb e AJAX.

A biblioteca GD, utilizada através do *wrapper* JpGraph, mostrou-se bastante estável e flexível para a configuração necessária aos relatórios gráficos.

O SGBD MySQL confirmou sua excelente performance, porém apenas com o tipo de tabelas MyISAM, que teve performance de inserção 40 vezes superior ao tipo InnoDB. No entanto é importante salientar que esta comparação é válida apenas neste caso específico, não podendo ser generalizada.

7 TESTES

Após concluída a etapa de desenvolvimento do sistema, o mesmo foi testado com arquivos de *log* reais, simulando sua utilização em um ambiente de produção. Neste capítulo são apresentadas capturas de telas e observações sobre o funcionamento de algumas partes do sistema.

7.1 Acesso

Para acesso ao sistema é necessário nome de usuário e senha. A interface inicial de *login* é apresentada na Figura 7.1.

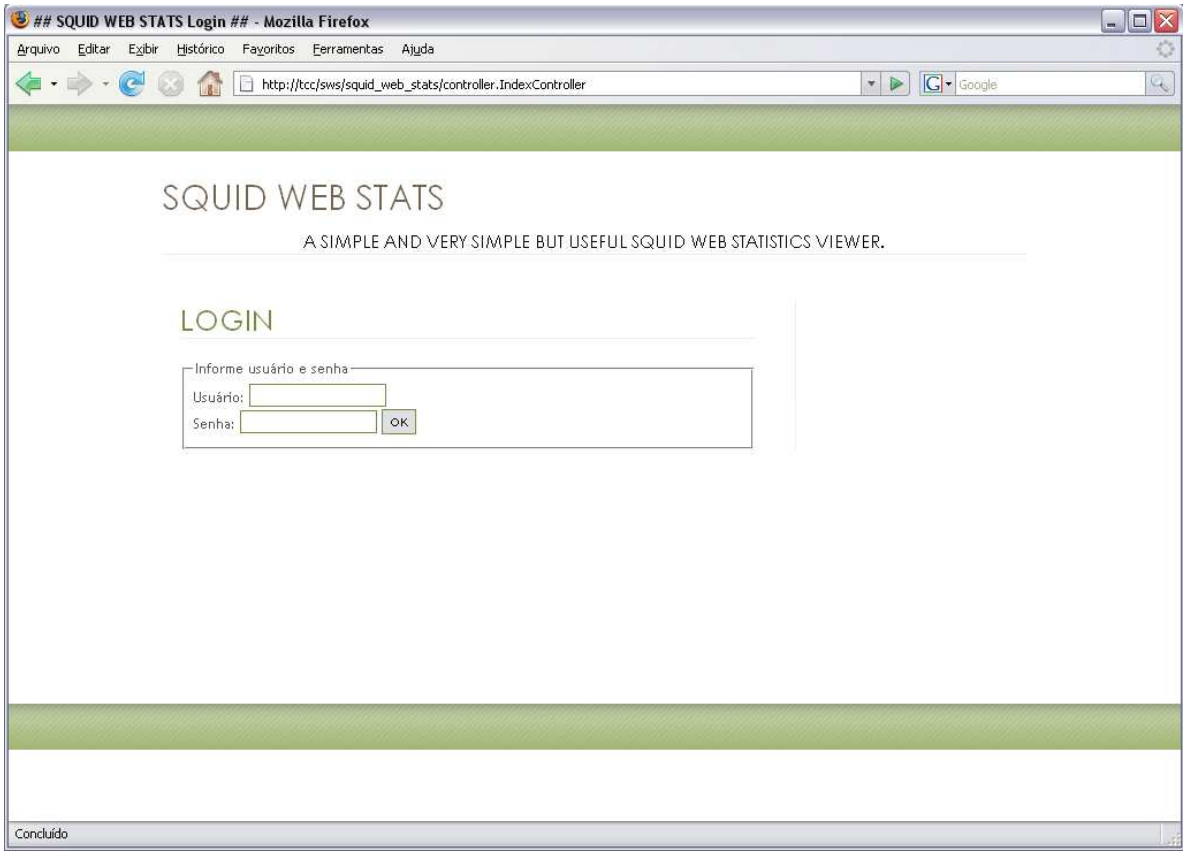

Figura 7.1: Interface de acesso

7.2 Importar Log

Na Figura 7.2 é apresentada a interface utilizada para realizar a importação de arquivos de *log*. Os parâmetros para importação de arquivo de *log* são Tempo máximo de espera, Perfil e Diretório, para os *logs* importados diretamente do disco do servidor, ou Upload, para os *logs* enviados através do navegador.

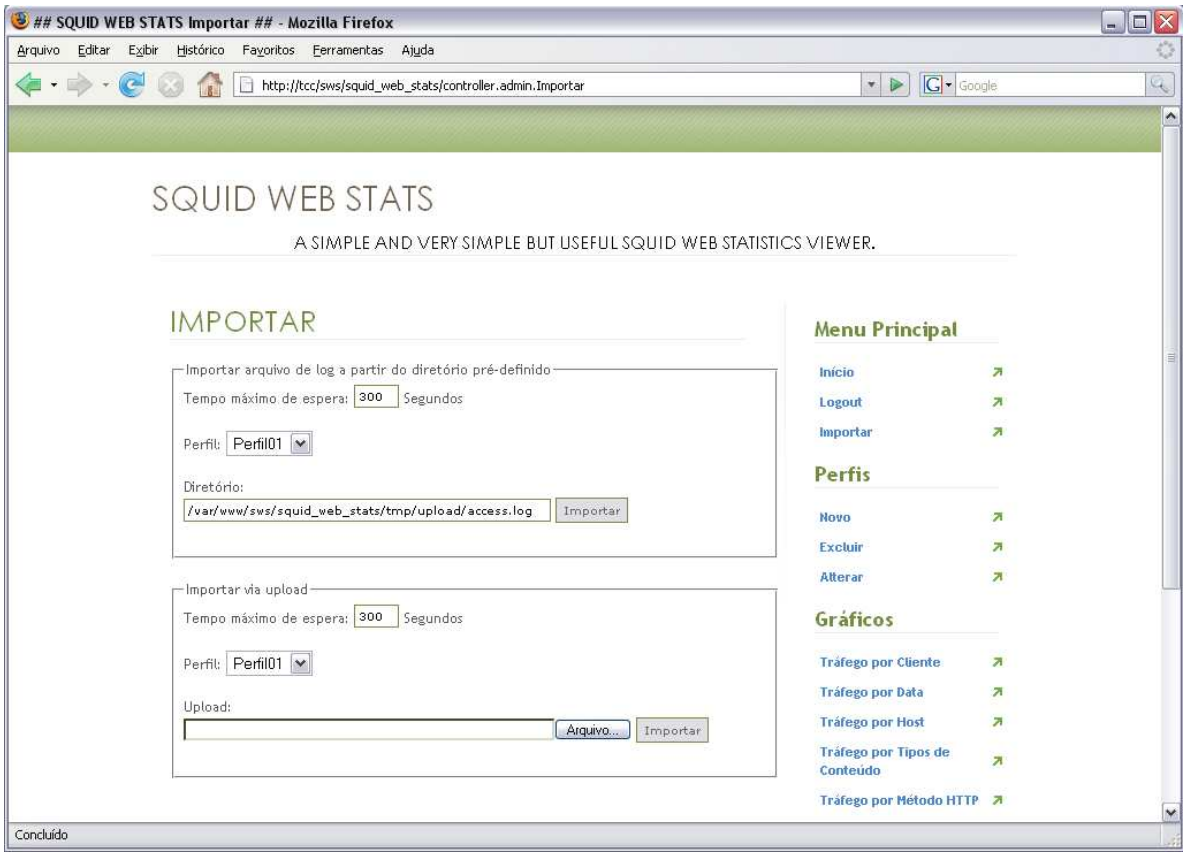

Figura 7.2: Interface de importação de log

7.3 Relatório Gráfico de Tráfego por Cliente

Na Figura 7.3 encontra-se o gráfico de tráfego por cliente, onde aparece o endereço IP dos dez clientes que geraram mais tráfego no Squid. Também é mostrado, em KB o total de tráfego para cada um destes clientes.

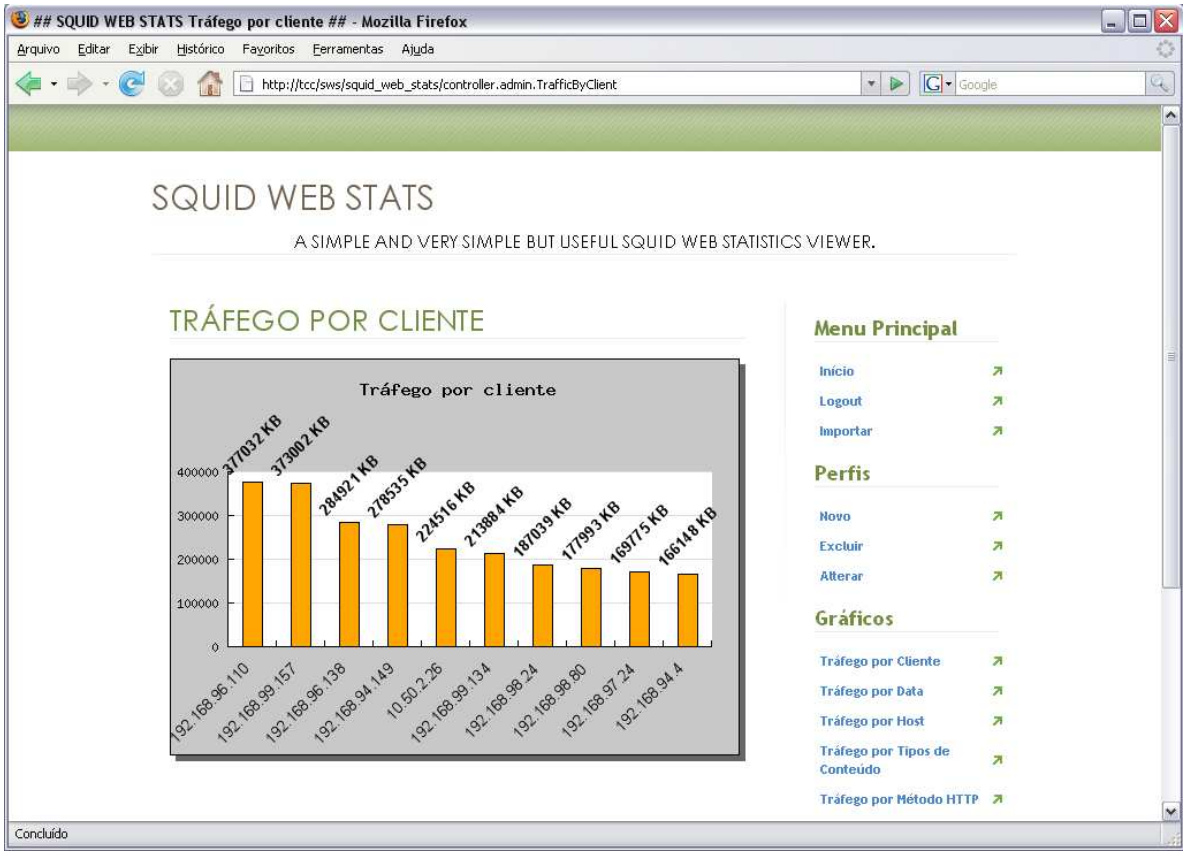

Figura 7.3: Relatório gráfico de tráfego por cliente

7.4 Relatório Gráfico de Tráfego por Host

Este gráfico, exibido na Figura 7.4, mostra o total de tráfego individual, em KB, trocado com os dez *hosts* – servidores web – com maior tráfego.

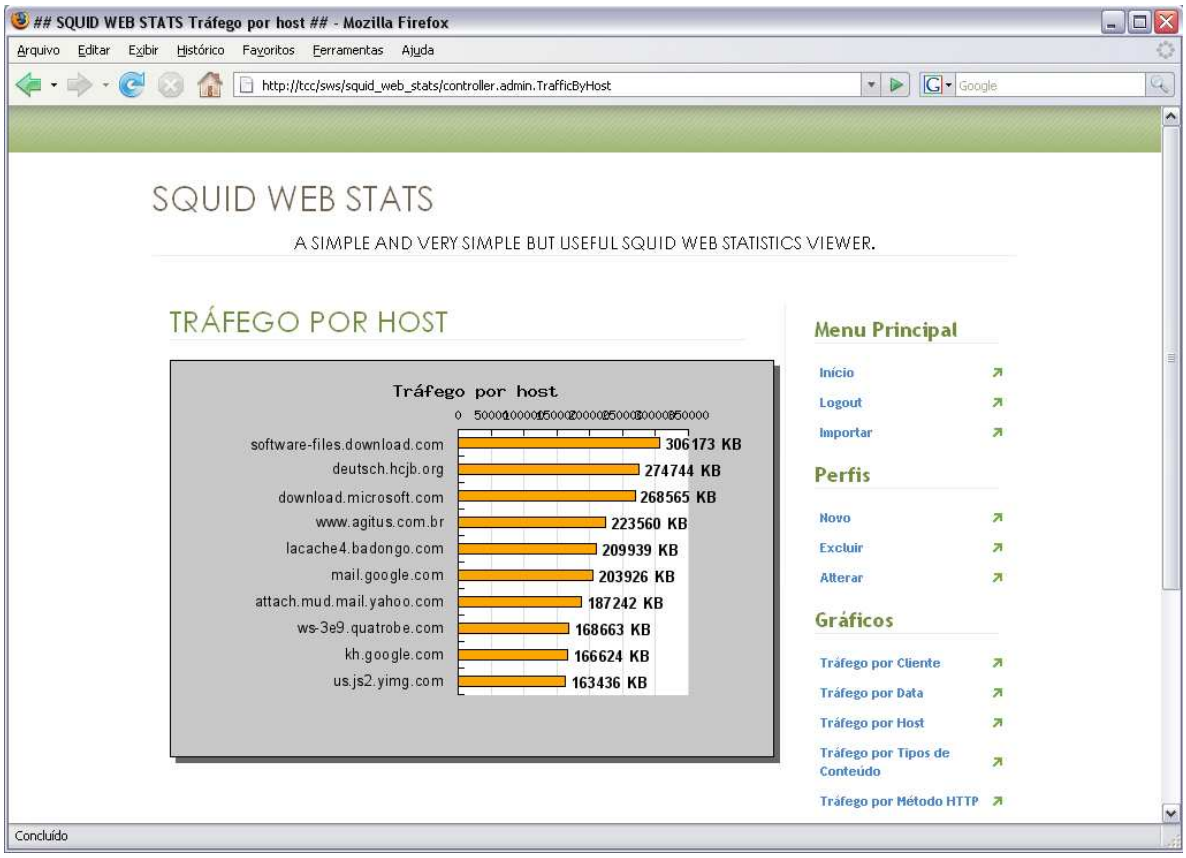

Figura 7.4: Relatório gráfico de tráfego por host

7.5 Relatório Gráfico de Tráfego por Tipo de Conteúdo

Na Figura 7.5 está o gráfico de tráfego por tipo de conteúdo, onde aparecem os dez tipos MIME com maior tráfego.

Figura 7.5: Relatório gráfico de tráfego por tipos de conteúdo

7.6 Relatório Gráfico de Tráfego por Método HTTP

O gráfico de tráfego por método HTTP, exibido na Figura 7.6, exibe os métodos e o somatório de tráfego relativo a cada método.

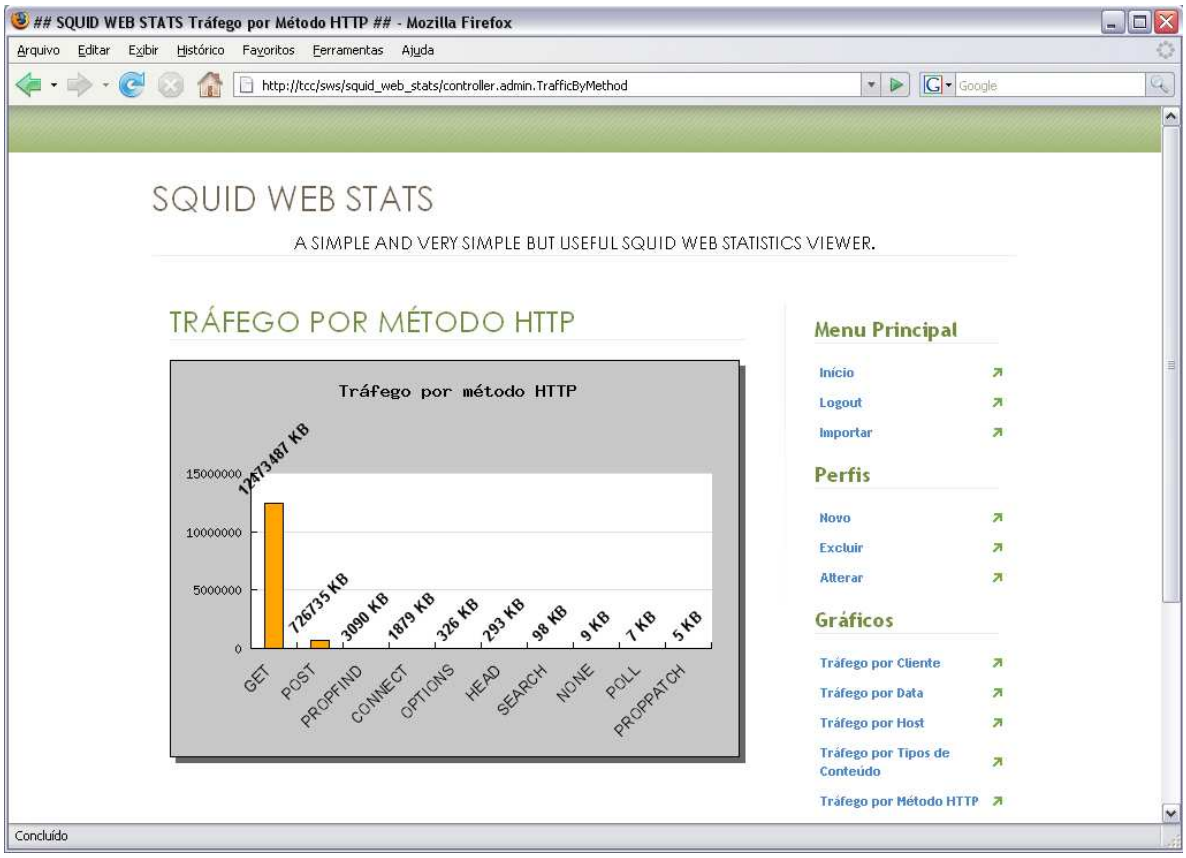

Figura 7.6: Relatório gráfico de tráfego por método HTTP

7.7 Relatório Textual de Requisições Negadas por Cliente

Este relatório exibe os endereços IP de clientes que tiveram requisições negadas em virtude de bloqueio de conteúdo no Squid. Em cada endereço IP é possível clicar e visualizar os endereços dos sites que foram bloqueados. Na Figura 7.7 há um exemplo, mostrando as requisições negadas apenas de um dos clientes.

Figura 7.7: Relatório textual de requisições negadas por cliente

7.8 Demais Relatórios Textuais

Para cada relatório gráfico existe também a versão texto, em formato tabular. Isto permite que sejam exibidos mais dados, além dos contidos nos gráficos. Na Figura 7.8 há um trecho do relatório textual de tráfego por *host*.

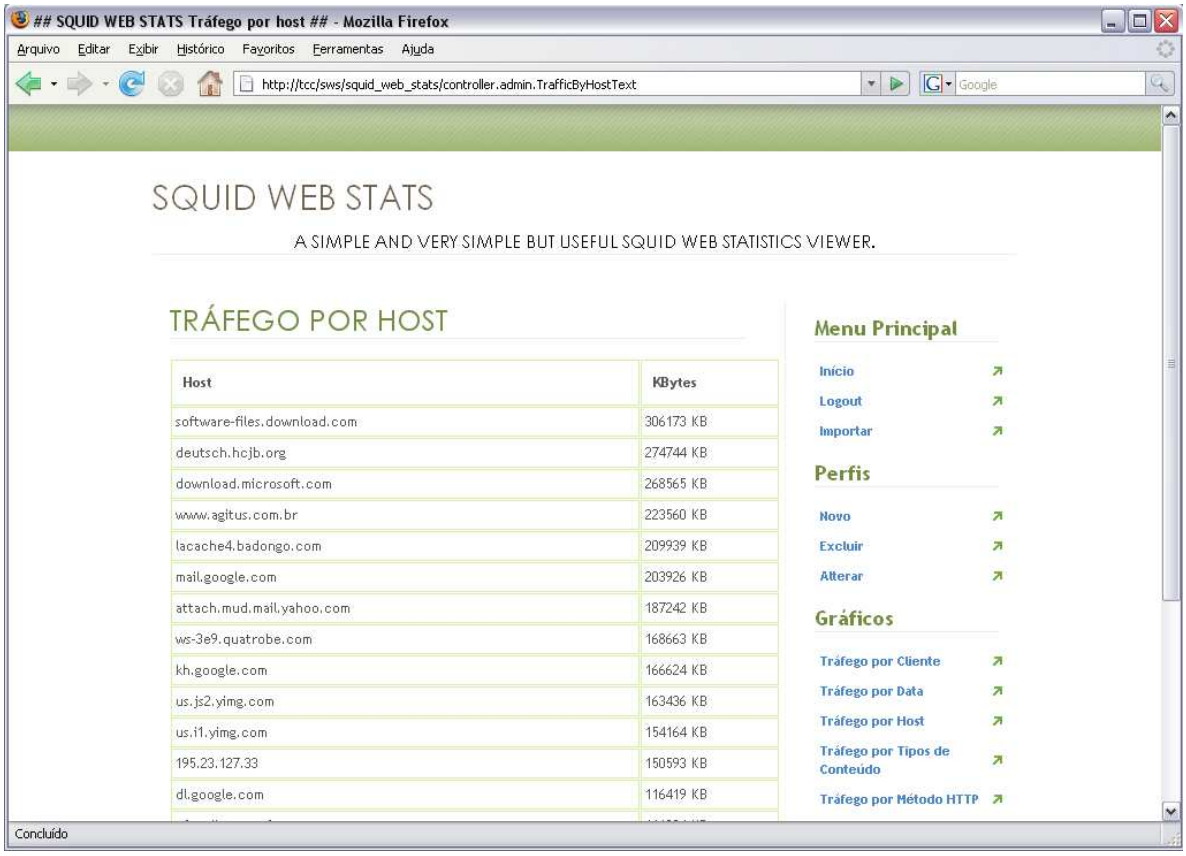

Figura 7.8: Relatório textual de tráfego por host

7.9 Conclusão

Estes testes revelaram um bom funcionamento do sistema. Sua interface simples e agradável ao usuário é bastante intuitiva. A velocidade das operações é boa, em especial as que envolvem consulta ao banco de dados e geração de gráficos. As informações são úteis para o gerenciamento da rede.

8 CONCLUSÃO

Os objetivos aos quais este trabalho foi proposto foram atingidos. Foi realizado o estudo da arquitetura da web e dos principais componentes envolvidos em sua estrutura. Também demonstrou-se que é viável a construção de uma ferramenta software livre que forneça informações úteis aos gestores de redes que utilizam o servidor de *proxy* Squid, a partir dos arquivos de *log* do mesmo. A aplicação foi desenvolvida e mostrou-se bastante funcional, prestando informações claras. Estas informações podem ser utilizadas para tomada de decisão no âmbito de gerenciamento da rede, objetivando otimizar a utilização deste valioso recurso que é a conectividade à Internet.

Ainda assim, o principal produto obtido no decorrer da atividade foi o conhecimento proporcionado pelo estudo das áreas envolvidas no trabalho e pela aplicação prática de conhecimentos adquiridos ao longo do curso. O aprendizado, como processo contínuo que é, mostra que é impossível dominar totalmente qualquer área do conhecimento e assim esgotar suas possibilidades.

A continuidade deste trabalho pode ocorrer de várias formas diferentes. Dentre elas, sugiro: a criação de funcionalidades, na interface de análise dos logs, que interajam com a configuração do servidor de proxy, para que o administrador da rede possa realizar ajustes básicos de forma extremamente simples e intuitiva; e a criação de relatórios dinâmicos, onde o usuário possa selecionar quais informações deseja cruzar.

REFERÊNCIAS

DUSKA, B.; MARWOOD, D.; FEELY, M. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. In: USENIX SYMPOSIUM ON INTERNET TECHNOLOGIES AND SYSTEMS, 1997**,** Monterey. **Proceedings…** Disponível em: <www.usenix.org/publications/library/proceedings/usits97/full_papers/duska/duska.pdf >. Acesso em: set. 2008.

GARRETT, J. J. **Ajax**: A New Approach to Web Applications: Adaptive Path Website. 2005. Disponível em: <http://adaptivepath.com/publications/essays/archives/000385.php>. Acesso em: nov. 2008.

LIM, J. **ADOdb Manual**: ADOdb Database Abstraction Library for PHP. 2008. Disponível em: <http://phplens.com/lens/adodb/docs-adodb.htm>. Acesso em: set. 2008.

OHRT, M. et al. **Smarty Manual**. [S.l.]: New Digital Group, 2007. Disponível em: <http://www.smarty.net/manual/pt_BR>. Acesso em: set. 2008.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual**. Massachusetts: Addison-Wesley, 1998.

SILVA, S. M. A. da; BONIN, M. R.; PALUDO, M. A. Levantamento de Requisitos Segundo o Método Volere. **Revista Negócios e Tecnologia da Informação**, [S.l.], ano 1, n. 1, 2005. Disponível em: <http://rnti.fesppr.br/include/getdoc.php?id=94&article=17&mode=p>. Acesso em nov. 2008.

SOMMERVILLE, I. **Engenharia de Software**. [S.l.]: Addison-Wesley, 2003.

VENTER, G. Optimising Internet Bandwidth in Developing Country Higher Education. In: INTERNATIONAL NETWORK FOR NETWORK FOR THE AVAILABILITY OF SCIENTIFIC PUBLICATIONS**,** 2003. **Proceedings…** Disponível em: <http://www.inasp.info/pubs/bandwidth>. Acesso em: set. 2008.

WESSELS, D. **Squid**: The Definitive Guide. [S.l.]: O'Reilly, 2004.

WESSELS, D. **Web Caching**. [S.l]: O'Reilly, 2001.