# 5
# Desenvolvimento do protótipo

## 5.1
## Visão geral

A construção do protótipo levou em consideração ferramentas de código aberto que pudessem desempenhar alguns dos papéis necessários na arquitetura. Sistemas foram desenvolvidos onde não foram encontradas ferramentas e para realizar a integração dos nós e componentes.

Da arquitetura completa proposta no capítulo anterior, a construção do protótipo teve seu escopo limitado nestes três elementos:

- Canal Principal de Comunicação de Eventos: este elemento da arquitetura foi estudado, instalado e configurado para atender a toda comunicação em grupo necessária entre os nós.
- Nó de primeira interação: neste nó é feita à análise léxica e transformação das linhas de log, uma correlação de eventos simples, e a publicação dos eventos no canal. Desta forma, além do Nó de análise léxica, transformação e publicação, um processamento complexo de eventos também é realizado neste nó.
- Nó de correlação de eventos por consulta: neste nó são recebidos eventos de um grupo do Canal Principal de Comunicação de Eventos e estes fluxos de eventos são inseridos em um DSMS, para análise e correlação através de consultas SQL. Desta forma, trata-se de um Nó de processamento complexo de eventos.

Os demais nós, Nó de conhecimento adaptativo, Nó de visualização em tempo real de eventos de log, Nó de armazenamento de eventos de log e Nó de pesquisa de eventos de logs, não foram implementados.

O protótipo foi desenvolvido e testado em uma máquina virtual, com sistema operacional Linux[1] (kernel[1] 2.6.9-42), mais especificamente com a

[1] Termo utilizado para designar os sistemas operacionais baseados no kernel Linux.

distribuição CentOS[2] versão 4.4. A máquina virtual possui 256 MB de memória RAM e uma CPU com freqüência de 2,33 GHz.

Para o desenvolvimento, foi utilizada a linguagem Perl, instalada na máquina virtual na versão 5.8.5. Também foi preciso instalar API's de acesso ao banco e de comunicação com o Spread 54, como detalhado a seguir na seção 5.2.

## 5.2 Canal Principal de Comunicação de Eventos

O Canal Principal de Comunicação foi criado utilizando-se da ferramenta Spread 54. Trata-se de uma ferramenta de comunicação de grupo, de código aberto, inicialmente desenvolvida por pesquisadores da Universidade Johns Hopkins e da Universidade George Washington. Esta ferramenta possui uma grande comunidade de desenvolvedores e usuários e oferece bastante documentação no seu *Web site*, desde publicações científicas a orientação de uso da API.

O Spread provê funcionalidades essenciais de comunicação em grupo no nível de aplicação. Isto quer dizer que o Spread não é dependente da existência de um protocolo de comunicação em grupo no nível de rede, como o *IP-Multicast*[3] para realizar a comunicação em grupo. Entre as funcionalidades providas pelo Spread estão a gestão de grupos (criação de grupos e associação a grupos) e a entrega de mensagens com diferentes níveis de garantias de entrega e ordem.

A arquitetura 55 do Spread é da forma *client-daemon*. Desse modo, um *daemon*[4] em execução em um servidor local, ou remoto, aceita conexões de aplicações clientes que desejem se comunicar em grupo. Cada *daemon* possui a configuração dos segmentos de rede com os quais deve se comunicar.

A Figura 9 apresenta um exemplo de configuração de um segmento:



---

[1] Kernel corresponde ao núcleo de um sistema operacional.

[2] CentOS é uma distribuição de Linux de código aberto voltada para empresas

[3] *IP-Multicast* é um método de camada de rede para comunicação de grupo.

[4] *Daemon* é um nome dado a processos que tem um longo tempo de duração em termos de execução e que são executados em plano de fundo .

```
Spread_Segment 192.168.1.255:3333 {
   no1 192.168.1.20
   no2 192.168.1.21
   no3 192.168.1.22
   no4 192.168.1.23
   no5 192.168.1.24
}
```

Figura 9– Exemplo de configuração de segmento no Spread

Cada segmento possui o endereço de comunicação do grupo, que pode ser um endereço de *broadcast* da rede, ou um endereço de *IP-Multicast*, e a indicação da porta a ser utilizada pelo *daemon*. No exemplo, é utilizado um endereço de *broadcast* de uma rede privada (192.168.1.255) e indicada a porta 3333. Este endereço de comunicação é utilizado para enviar as mensagens para todos os participantes do segmento.

Deve ser informado também o endereço específico de cada nó do segmento. Este endereço será utilizado para comunicação de mensagens de controle pelos *daemons*.

Um *daemon* pode ter mais de um segmento. Isto acontece quando os *daemons* do Spread estão espalhados por mais de uma rede, seja localmente ou até mesmo pela *Internet.* É importante ressaltar que a configuração de segmentos deve ser a mesma em todos os *daemons* participantes de uma comunicação. Isto é importante, pois se, por exemplo, um dos *daemons* não tiver um dos segmentos configurados em outros *daemons*, ele não enviará e nem receberá mensagens daquele segmento.

Esta configuração de segmentos e nós participantes define a camada de rede do Spread. Com isto, os *daemons* estão prontos para receber conexões de clientes e se comunicarem. A configuração de segmentos e o estabelecimento da comunicação entre os *daemons* representa o estabelecimento da infra-estrutura básica do Canal Principal de Comunicação de eventos deste protótipo.

Os clientes se comunicam com o *daemon* através de uma API. Esta API foi desenvolvida para linguagens como C, C++, Java, Python e Perl. O módulo Perl 57 é de código aberto, está disponível na *Internet* e deve ser instalado em todos os nós da arquitetura, pois todos precisam se comunicar com os *daemons* para enviar mensagens de eventos.

A API de acesso aos *daemons* possui quatro funções básicas: conexão ao *daemon*; associação e criação de grupos; envio de mensagens para um grupo; e

recepção de mensagens de um grupo. Há ainda funções para desconectar de um *daemon*, desassociar de um grupo e para obter informações de um grupo.

As mensagens a serem enviadas pelo Spread podem ser de diferentes tipos. Para este protótipo foi utilizado o tipo “AGREED_MESS”. Este tipo de mensagem é descrita como confiável, ou seja, a mensagem será entregue a todos os nós membros do grupo para qual a mensagem foi enviada, e caso ocorra perdas na rede, o Spread irá recuperar as mensagens. Além disso, o tipo de mensagem escolhido tem outra característica importante quanto à ordem de entrega das mensagens, chamada de “*total order*”. Este tipo de entrega garante que todas as mensagens de todos os nós que publicarem serão entregues na mesma ordem a todos os nós que estiverem consumindo.

Com toda a configuração realizada, o tipo de mensagem definido, e o conhecimento de como acessar um *daemon*, já é possível desenvolver componentes nos nós que sejam capazes de criar grupos, se associar, e publicar e consumir eventos.

## 5.3
## Nó de primeira interação

Neste nó, além de ser feita a análise léxica, transformação, e transmissão de eventos, foi adicionada a funcionalidade de correlação de eventos. Um mesmo servidor pode ter diferentes arquivos de log, que podem ser correlacionados ainda no servidor. Contudo, é preciso atentar para o fato de que uma ferramenta de correlação que consuma mais recursos computacionais pode onerar o próprio servidor.

A primeira função a ser desempenhada neste nó é a análise léxica das linhas de um ou mais arquivos de log e a transformação de cada linha de log no formato de evento exposto no capítulo anterior. Para desempenhar este papel foi escolhida a ferramenta *Log PreProcessor* (LogPP) 58.

Esta ferramenta, também de código aberto e escrita em C, permite que sejam definidas entradas, filtros, saídas e fluxos. As entradas correspondem a arquivos de log que serão monitorados pela ferramenta e a cada linha escrita, processados. Os filtros correspondem a expressões regulares em C que analisam as linhas de log e as transformam segundo um modelo pré-definido, no caso deste protótipo um modelo CSV. A saída corresponde no protótipo a um mecanismo de comunicação entre o LogPP e o próximo componente do Nó, e

pode ser um arquivo, uma FIFO[1] ou ainda o STDIN[2] de um programa. Por fim, um fluxo corresponde à definição de uma seqüência de entradas, filtros e saídas.

Um exemplo de um arquivo de configuração é mostrado na Figura 10. Neste exemplo, a entrada é um arquivo de log de acesso do Apache. O filtro é uma expressão regular que interpreta a linha no padrão *Common Log Format* e a transforma para o padrão CSV da arquitetura. A saída corresponde ao Canal de Local de Comunicação deste componente com o próximo.

```
input input-log {
  file  /usr/local/httpd-2.2/logs/access_log
}
output output-log {
  file ../output_fifo
}

filter http-access {
  regexp ^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}).*\[(.*)\s.*\]\s+"GET\s(.*)HTTP/1\.[0-1]"\s([0-9]{3})\s

  template no1,$2,0,$4,apache|acesso,host:$1-pagina:$3
}

flow get-access {
  input input-log
  filter http-access
  output output-log
}
```

Figura 10 – Configuração LogPP

No exemplo da Figura 10, o Canal Local de Comunicação do LogPP com o SEC, próximo componente a ser apresentado, é uma FIFO 56, um mecanismo de comunicação entre processos disponível em sistemas UNIX. Desta forma,

[1] FIFO, também chamado de *named pipe*, é um tipo de arquivo especial em sistemas UNIX para que processos troquem informação.

[2] STDIN é a entrada padrão (*standard input*) de um programa em ambiente UNIX referenciada por um ponteiro em programas C.

processos sem relação direta - pai e filho, por exemplo - podem se comunicar. Para isso, um processo deve escrever na FIFO, como se fosse um arquivo, e o outro processo ler da FIFO.

É importante destacar, no exemplo de configuração do LogPP da Figura 10, o uso de uma expressão regular, associada ao termo *regexp,* para separar a informação de data e hora na variável *$2*, a informação de código HTTP na variável *$4*, a informação do endereço IP do usuário na variável *$1* e a página acessada na variável *$3*. Essas informações são adequadas ao padrão CSV do sistema, associado ao termo *template*.

O SEC (*Simple Event Correlator*) é uma ferramenta de código aberto, desenvolvida em Perl, fruto de pesquisa desenvolvida sobre eventos de log [17, 18, 30, 31]. Esta ferramenta tornou-se bastante popular entre administradores de sistema, motivando inclusive uma publicação em um congresso de administração de sistemas 59.

Para se utilizar o SEC é necessário especificar qual será a entrada, no caso uma FIFO, onde os eventos de log de diferentes arquivos no servidor são escritos já no formato CSV. Além disso, é preciso passar para o SEC o conjunto de regras utilizado para correlacionar os eventos. Este conjunto de regras pode estar em um ou mais arquivos.

Existem nove tipos de regras no SEC baseadas nas operações de correlação citadas por Jakobson e Weissman [28, 29], já mencionadas no Capítulo 3. Além das regras que representam operações de correlação, é possível criar e excluir contextos operacionais para determinar certas condições sobre quais uma regra pode atuar ou não, criar novos eventos e ainda disparar comandos ou programas a serem executados no sistema operacional.

A documentação completa das regras e de como se utilizar o SEC pode ser encontrada no *Web site* da ferramenta 60. A Figura 11 apresenta um exemplo de regra no SEC que monitora o número de acessos a páginas não encontradas, o que pode caracterizar um possível ataque ou um referência a uma página não existente e que precisa ser consertada. O tipo da regra é *SingleWithThreshold*, o que significa que será identificado um padrão, expresso por uma expressão regular (*pattern*), e que este padrão será contado dentro de uma janela de cinco segundos (*window*). Caso o padrão aconteça mais de cinqüenta vezes dentro da janela de cinco segundos, a ação (*action*) é disparada.

```
type=SingleWithThreshold
ptype=RegExp
pattern=^(.*),(.*),(.*),404,(.*),(.*)
desc=Excessivos erros 404 no periodo
action=shellcmd perl /home/globolab/perl/current/message_to_queue.pl
$1,$2,1,WARN,$4|notfound,$5 1
window=5
thresh=50
```

Figura 11– Exemplo de regra no SEC

A ação no caso corresponde à chamada de um programa desenvolvido em Perl que apenas insere o novo evento criado, indicado pelo algarismo "1" na terceira posição e não mais 0, em uma fila de mensagens de eventos a serem enviadas para o Canal Principal de Comunicação.

Vale ressaltar que as operações de correlação a serem desempenhadas pelo SEC não podem onerar o servidor que está entregando páginas para usuários na *Internet*. Nesse sentido, é preciso que o conjunto de regras não seja nada muito grande e complexo.

A fila de mensagens entre o SEC e o Componente Publicador, que será apresentado mais à frente neste tópico, é outro mecanismo de comunicação entre processos utilizado, ou seja, um Canal Local de Comunicação. Trata-se na verdade de uma *message queue*, mecanismo implementado pelo sistema operacional, para que dois processos possam enviar e receber mensagens. Uma *message queue* 56 é na verdade uma lista encadeada de mensagens armazenada na memória do *kernel* do sistema operacional. Toda *message queue* possui um identificador único, que é utilizado para acessá-la. No Linux os limites relacionados a *message queues* são de 8.192 *bytes* para uma mensagem, o que é bastante confortável para um mensagem de evento de log, que possui aproximadamente 100 *bytes*¸ e de 16.384 *bytes* para o tamanho total da fila, o que significa que aproximadamente 160 mensagens de eventos de log podem enfileirar antes de a fila ficar lotada.

No livro utilizado como referência 56, é feita uma ressalva sobre a utilização de *message queues*, pois estas persistem no sistema operacional, mesmo que nenhum processo as estejam utilizando, até que seja reinicializado o sistema operacional. Desta forma, é preciso ter a atenção de remover uma *message queue* explicitamente, através de comandos, caso não seja mais

utilizada. Na verdade, este comportamento de persistir em memória as mensagens de eventos de log, mesmo que nenhum processo esteja utilizando a *message queue*, é desejado pois, caso aconteça uma falha nos Componentes que utilizam a fila, estes podem ser reiniciados sem que nenhum evento de log seja perdido.

Em termos de implementação, não é preciso instalar nenhum módulo Perl adicional para se trabalhar com *messages queues*, pois o módulo necessário (IPC::SysV) já faz parte dos módulos centrais do Perl. É importante também não se esquecer de exportar as constantes a serem utilizadas na criação de *messages queues* e a função "*ftok*". A função "*ftok*" é utilizada para que um identificador para uma *message queue* seja gerado a partir de um caminho no sistema de arquivos. Desta forma, dois processos podem referenciar uma mesma *message queue* por um identificador gerado através de um mesmo caminho no sistema de arquivos.

O último componente deste nó é o Publicador. O papel do Publicador é monitorar a fila de mensagens alimentada pelos eventos de log enviados pelo SEC e enviar estes eventos para o Canal Principal de Comunicação. Este componente foi desenvolvido em Perl e necessita do módulo do Spread instalado na máquina para poder se comunicar com o *daemon*.

O *daemon* está rodando no nó e o Publicador precisa acessá-lo na porta especificada na configuração do Spread. Após acessar o *daemon*, o Publicador deve se associar a grupos. Neste protótipo apenas um grupo existe no Canal Principal de Comunicação, correspondendo a uma aplicação. Com a associação feita ao grupo, o Publicador pode agora enviar os eventos de log para o canal conforme os vai retirando da fila de mensagens.

A Figura 12 ilustra todos os componentes e Canais Locais de Comunicação do nó de primeira interação.

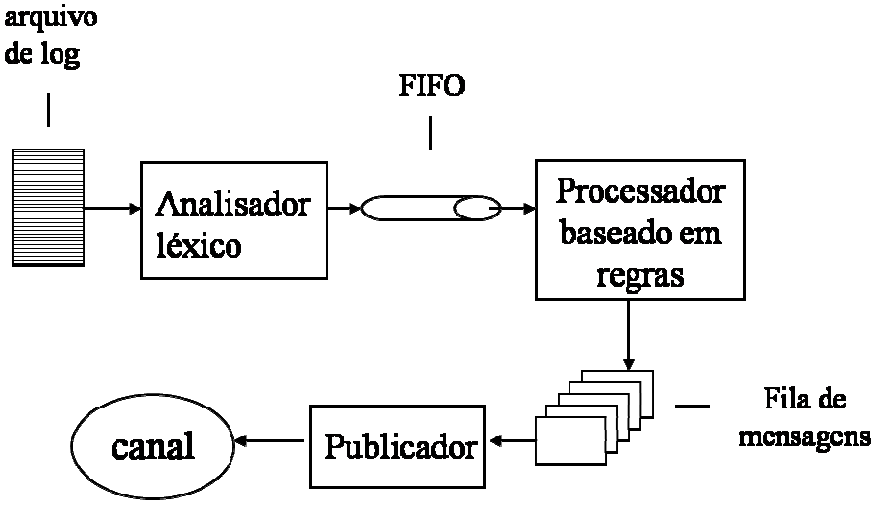


Figura 12 – Componentes do Nó de primeira interação

## 5.4
## Nó de correlação de eventos por consulta

O primeiro componente do nó de correlação de eventos por consulta é o componente responsável por consumir os eventos do canal de comunicação. Este componente, chamado de Consumidor, foi desenvolvido em Perl e faz o papel inverso do componente Publicador apresentado no tópico anterior.

O Consumidor se conecta ao *daemon* do Spread, que também deve estar em execução na máquina do nó de correlação de eventos por consulta. Para que o programa funcione é necessário que o módulo Perl com as funções relativas ao Spread também esteja instalado na máquina. O Consumidor se conecta ao mesmo grupo do Publicador, através do *daemon* do Spread, e passar a receber todos os eventos de log publicados. Na medida em que recebe os eventos de log o Consumidor salva em uma *message queue*.

A motivação em se sempre uma *message queue* entre os componentes que interagem com o Canal Principal de Comunicação é aumentar o desacoplamento entre o nó e o canal. Desta forma, o recebimento de evento de log e o processamento posterior deste evento são atividades assíncronas.

Outro componente desenvolvido em Perl, chamado de *Insersor*, consome a *message queue* e insere os eventos de log no TelegraphCQ, um DSMS que representa o componente de correlação por consulta. Para se conectar com o TelegraphCQ, o Insersor também depende que módulos Perl para conexão com banco de dados sejam instalados na máquina. Uma observação sobre a versão do módulo utilizado será feita depois de explicada a arquitetura do TelegraphCQ.

O TelegraphCQ [61, 62] foi desenvolvido sobre o Postgres 63, um banco de dados de código aberto. Como mencionado no Capítulo 3, o TelegraphCQ foi submetido a testes de desempenho em [48, 49], com resultados bastante positivos. Aliado a isto, foi desenvolvido sobre um banco de dados já estável e amplamente utilizado, o que melhora a sua estabilidade e documentação.

O TelegraphCQ possui dois processos, denominados *back-end* e *front-end*, que compartilham uma área de memória para comunicação. O *front-end* é responsável por receber conexões de clientes e receber consultas. O *back-end* é responsável por executar as (longas) consultas contínuas. Os processos utilizam memória compartilhada para se comunicarem e semáforos, para se sincronizarem.

É preciso ressaltar que para o TelegraphCQ funcionar foram realizados ajustes nos parâmetros padrão do sistema operacional, aumentando o tamanho de um segmento de memória compartilhada, a área total disponível para memória compartilhada e o número máximo de semáforos que podem ser criados no sistema operacional.

Depois de compilado e instalado na sua última versão, versão 2.1, e com o sistema operacional devidamente configurado, o TelegraphCQ finalmente pode ser executado. Diferentemente de um banco de dados convencional, é preciso criar *streams*, ou fluxos, e não tabelas. Um fluxo terá os campos correspondentes ao formato CSV de evento de log. A Figura 13 mostra a descrição de um fluxo criado, obtido pela execução de um comando de descrição executado no TelegraphCQ:

```
log=# \d app1_log.eventstream
 Archived stream "app1_log.eventstream"
  Column   |           Type
-----------+----------------------------
 source    | character varying
 tcqtime   | timestamp without time zone
 hierarchy | character varying
 severity  | character varying
 tags      | character varying
 message   | character varying
```

Figura 13 – Descrição de um fluxo de eventos de log

A Figura 14 exemplifica uma consulta contínua que exibe os acessos bem sucedidos a uma determinada página na *Internet* nos últimos cinco minutos, de cinco em cinco segundos.

```
SELECT * FROM app1_log.eventstream AS ev [RANGE BY '300 seconds'
SLIDE BY '5 second'] WHERE ev.severity='200' AND ev.message LIKE
'%pagina.html%';
```

Figura 14 – Exemplo de consulta contínua

Consultas mais complexas, envolvendo agregação de diferentes fluxos de eventos e até mesmos consultas recursivas, também podem ser expressas no TelegraphCQ.

A Figura 15 representa os componentes do Nó de correlação de eventos por consulta.

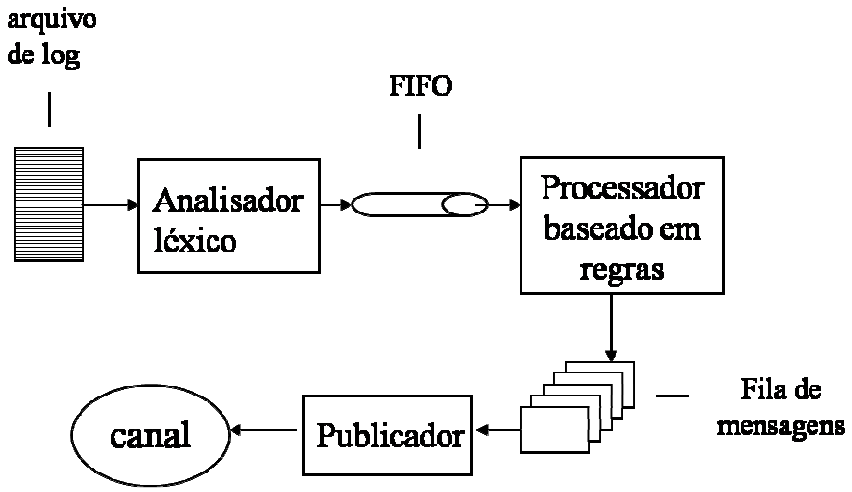


Figura 15 – Componentes do Nó de correlação de eventos por consulta

## 5.5
## Um caso de uso do protótipo

Para uma prova de conceito do protótipo, foi definido um cenário que correspondesse à realidade de muitas empresas de *Internet* e que pudesse, inclusive, ser aplicado na Globo.com.

É necessário, então, apresentar uma das arquiteturas mais comuns de uma aplicação Web no contexto de uma empresa de *Internet* e ressaltar os logs

que são hoje usados no gerenciamento da aplicação. Na verdade, os logs utilizados no prova de conceito são logs reais de um portal da Globo.com.

Nesta arquitetura, no que diz respeito a gerenciamento de logs, existem dois componentes principais, que são o servidor *Web* e o servidor de aplicação. A aplicação responsável pelo papel de servidor *Web* é o Apache 7, um servidor de código aberto e extremamente popular. Já o papel de servidor de aplicação é desempenhado pelo Jboss Web 11, um servidor para aplicações *Web* desenvolvidas em Java.

Em uma aplicação em produção, as requisições dos usuários chegam a um balanceador de carga que irá escolher aquele servidor *Web* que está mais livre para responder. O servidor *Web* recebe a requisição e determina se é capaz de atendê-la ou se deve encaminhar para o servidor de aplicação. Geralmente, as requisições de conteúdo estático - por exemplo, figuras - e de objetos HTML que sejam os mesmos para todos os usuários são respondidas pelo próprio servidor *Web*. Quando a requisição requer um processamento dinâmico, por exemplo, uma votação ou comentário, o servidor *Web* encaminha para o servidor de aplicação que irá processar a requisição utilizando a aplicação que hospeda. Há ainda nesta arquitetura um outro componente, que são os bancos de dados, no caso o MySQL64, consultados sempre que é necessário consultar uma informação ou persistir um dado, como por exemplo os votos da aplicação ou um comentário de uma matéria.

Neste processo de atendimento das requisições dos usuários é gerado um grande volume de logs em cada um dos componentes. No Apache são gerados dois logs principais: o de acesso e o de erro; no servidor JbossWeb são gerados os logs do próprio servidor e os logs gerados pela aplicação. No MySQL são gerados os logs de operações feitas no banco. Cada um desses logs possui diferentes padrões e informações presentes.

Em termos de infra-estrutura, os servidores *Web*, os servidores de aplicação e servidores de banco de dados podem estar fisicamente nos mesmos servidores, ou segregados em servidores dedicados, conforme a demanda. A Figura 16 resume a arquitetura apresentada.

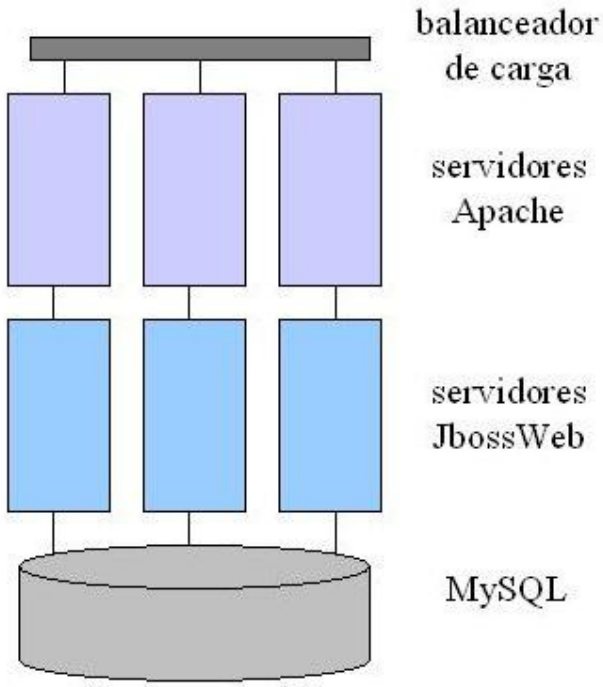


Figura 16 – Arquitetura de uma aplicação *Web*

Com base nesta arquitetura, dois arquivos de log de uma aplicação real foram utilizados para teste: um arquivo de log de acesso do Apache e outro arquivo de log do JbossWeb. O LogPP foi configurado para monitorar estes dois arquivos e duas expressões regulares foram criadas para extrair as informações das linhas de log e transformá-las no padrão CSV do sistema. Figura 17 mostra um trecho de configuração do LogPP com os dois filtros utilizados. Uma necessidade identificada nos testes foi o uso de aspas para cada conteúdo separado por vírgulas, pois muitas páginas na *Internet* possuem vírgula no seu endereço.

```
filter http-access {
  regexp ^([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}).*\[(.*)\s.*\]\s+"GET\s(.*)HTTP/1\.[0-1]"\s([0-9]{3})\s
  template "no1","$2","0","$4","apache|acesso","host:$1|pagina:$3"
}
filter jboss-server{
  regexp ^(.*),[0-9]{1,3}\s(FATAL|ERROR|WARN|INFO|DEBUG)\s\[(.*)\]\s(.*)
  template "no1","$1","0","$2","jboss","classe:$3|mensagem:$4"
}
```

Figura 17 – Filtros configurados no LogPP para prova de conceito

A saída do LogPP foi monitorada pelo SEC, através de três regras. Uma primeira regra procura identificar a ocorrência excessiva de erros de página não encontrada. Uma segunda regra procura por um erro específico na aplicação quando esta não consegue acessar um serviço de *cache* de objetos. Ambas as regras tem como ação a chamada de um programa em Perl que encaminha o evento de log para a fila de mensagens monitorada pelo Publicador. Vale ressaltar, que estas regras elevam a hierarquia do evento de log, de 0 para 1, por se tratar de um padrão reconhecido. Além disso, estas regras acrescentam palavras-chaves aos eventos, completando assim sua classificação. A terceira regra apenas encaminha qualquer evento para o programa que enfileira os eventos de log para o Publicador. É importante notar que o fato do evento já estar em um padrão CSV facilita muito a criação das expressões regulares. A Figura 18 exibe o arquivo de configuração do SEC com as regras.

```
#Regra para identificar excessivos erros 404
type=SingleWithThreshold
ptype=RegExp
pattern=^(.*),(.*),(.*),"404",(.*),(.*)
desc=Excessivos erros 404 no periodo
action=shellcmd sudo perl /home/globolab/perl/current/message_to_queue.pl
'$1,$2,"1","WARN",$4|notfound",$5' '2'
window=5
thresh=50

#Regra para identifiar erro de conexao memcached
type=Single
ptype=RegExp
pattern=^(.*),(.*),(.*),"ERROR",(.*),(.*)MemCached.SockIOPool(.*)
desc=Erro de conexao com memcached
action=shellcmd sudo perl /home/globolab/perl/current/message_to_queue.pl
'$1,$2,"1","ERROR",$4|memcached","Erro memcached"' '2'

#Regra para enviar todos os eventos
type=Single
ptype=RegExp
pattern=^(.*)
desc=$0
action=shellcmd sudo perl /home/globolab/perl/current/message_to_queue.pl '$0'
'1'
```

Figura 18 – Regras do SEC utilizadas no protótipo

Os eventos de log gerados são publicados no canal, utilizando para isso o Spread. Todos os eventos são publicados em um único grupo, nomeado com o nome da aplicação. Os eventos são recebidos pelo Consumidor, que os enfileira na fila de mensagens monitorada pelo componente que irá inserir os eventos no TelegraphCQ. No TelegraphCQ, foram criadas duas consultas contínuas, uma para mostrar todos os acessos bem sucedidos relativos a páginas HTML e outra para mostrar todos os eventos com nível de hierarquia igual a 1. As consultas utilizadas são mostradas na Figura 19.

```
SELECT * FROM app1_log.eventstream AS ev [RANGE BY '5 seconds' SLIDE
BY '1 second'] WHERE ev.hierarchy='1';

SELECT * FROM app1_log.eventstream AS ev [RANGE BY '5 seconds' SLIDE
BY '1 second'] WHERE ev.severity='200' AND ev.message LIKE '%.html%';
```

Figura 19 – Consultas utilizadas no protótipo

Para realizar os testes, os registros reais dos logs foram inseridos nos arquivos que estavam sendo monitorados pelo LogPP. Nesta prova de conceito foi possível identificar os padrões que estavam representados através de regras no SEC e também visualizar os resultados das consultas no TelegraphCQ.

## 5.6 Testes funcionais do protótipo

Os testes do protótipo desenvolvido foram realizados de maneira gradual, inicialmente sobre os componentes separados e, por fim, sobre o sistema como um todo. Na descrição dos testes, os endereços internos da Globo.com foram substituídos por endereços fictícios, por razões de segurança.

Para a realização dos testes foram utilizados dois arquivos de log reais de uma aplicação da Globo.com. O primeiro arquivo corresponde ao log de acesso do servidor Apache, “acess_log” com 3.092 KB e 13.628 linhas e o segundo arquivo corresponde ao log principal do servidor Java, o “*stdout.log*”, com 9.027 KB e 93274 linhas. Estes arquivos foram retirados de um ambiente de homologação e copiados para o ambiente de desenvolvimento do protótipo.

Os dois casos de teste a seguir foram realizado da seguinte forma. O LogPP foi configurado para ter como entrada o arquivo “input.log” e como saída o arquivo “output.log”. Foram então inseridos os 100 últimos eventos de log no arquivo “apache.log” através do comando “*tail -100 access_log >> apache.log*”. Em seguida, foi verificado o arquivo “output.log” para conferir se todos os eventos foram gerados corretamente no padrão esperado.

A Tabela 5 e a Tabela 6 a seguir ilustram os dois casos de teste mostrando exemplos de linhas do log que foram entrada e a saída obtida. O objetivo do primeiro caso de teste era verificar a corretude da expressão regular para o caso de uma URL sem vírgula e o objetivo do segundo caso de teste era tratar os casos onde a linha de log possuía vírgulas na URL.

| **Componente:** | LogPP |
|---|---|
| **Operação:** | Análise léxica e transformação do log de acesso do Apache |
| **Propósito:** | Testar a corretude do padrão de saída gerado |
| **Descrição:** | Este teste verificará a corretude da análise léxica do log de acesso do Apache e sua transformação no padrão CSV esperado |
| **Entrada** | |
| 10.10.1.1 - - [19/Jun/2008:18:25:01 -0300] "GET /vcnofantastico/service/timeOutVideo.ssp HTTP/1.1" 404 5563 "-" "Java/1.5.0_06" | |
| **Saída Esperada** | |
| "no1","19/Jun/2008:18:25:01","0","404","apache\|acesso","host:10.10.1.1\|pagina:/vcnofantastico/service/timeOutVideo.ssp " | |
| **Saída Obtida** | |
| "no1","19/Jun/2008:18:25:01","0","404","apache\|acesso","host:10.10.1.1\|pagina:/vcnofantastico/service/timeOutVideo.ssp " | |
| **Resultado** | |
| Homologado | |

Tabela 5 – Caso de teste Apache e LogPP

| **Componente:** | LogPP |
|---|---|
| **Operação:** | Análise léxica e transformação do log de acesso do Apache |
| **Propósito:** | Testar a corretude do padrão de saída gerado com endereços com vírgula. |
| **Descrição:** | Este teste verificará a corretude da análise léxica do log de acesso do Apache e sua transformação no padrão CSV esperado |
| **Entrada** | |
| 10.10.1.1 - - [19/Jun/2008:17:08:10 -0300] "GET /Gente/Celebridades/0,,9805,00.html HTTP/1.1" 200 16943 "http://www.x.com/Gente/Fotos/0,,GF59513-9801-p-5,00.html" "Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-BR; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14" | |
| **Saída Esperada** | |
| "no1","19/Jun/2008:17:08:10","0","200","apache\|acesso","host:10.10.1.1\|pagina:/Gente/Celebridades/0,,9805,00.html " | |
| **Saída Obtida** | |
| "no1","19/Jun/2008:17:08:10","0","200","apache\|acesso","host:10.10.1.1\|pagina:/Gente/Celebridades/0,,9805,00.html " | |
| **Resultado** | |
| OK | |

Tabela 6 – Caso de teste Apache e LogPP com vírgula

O próximo caso de teste é bastante semelhante ao primeiro, sendo que neste caso as linhas de log a serem inseridas são oriundas do log do servidor Java, o “stdout.log”. Após remover os dados do teste anterior, o mesmo comando foi dado e 100 linhas foram inseridas. Em seguida, foi verificado o

arquivo “output.log” para conferir se todos os eventos foram gerados corretamente no padrão esperado.

A Tabela 7 a seguir resume o caso de teste mostrando uma das linhas do log que foram entrada e a saída obtida.

| | |
|---|---|
| **Componente:** | LogPP |
| **Operação:** | Análise léxica e transformação do log do Jbossweb |
| **Propósito:** | Testar a corretude do padrão de saída gerado o padrão gerado pelo log4j |
| **Descrição:** | Este teste verificará a corretude da análise léxica do log de acesso do Jbossweb e sua transformação no padrão CSV esperado |
| **Entrada** | |
| 2008-06-19 19:25:01,599 ERROR [org.apache.struts.action.RequestProcessor] Invalid path was requested /vcnofantastico/service/timeOutVideo | |
| **Saída Esperada** | |
| "no1","2008-06-19 19:25:01","0","ERROR","jboss", "classe:org.apache.struts.action.RequestProcessor\|mensagem:Invalid path was requested /vcnofantastico/service/timeOutVideo" | |
| **Saída Obtida** | |
| "no1","2008-06-19 19:25:01","0","ERROR","jboss", "classe:org.apache.struts.action.RequestProcessor\|mensagem:Invalid path was requested /vcnofantastico/service/timeOutVideo"" | |
| **Resultado** | |
| OK | |

Tabela 7 – Caso de teste JbossWeb e LogPP

Com o componente LogPP testado, o próximo passo foi fazer com que o SEC passasse a ter como entrada a saída do LogPP para então testá-lo. Primeiramente foi identificado no log um número muito grande de acessos a páginas não encontradas, isto pode significar que um objeto não está onde deveria, ou alguma página está com a referência errada, ou ainda, um possível ataque. Depois foi identificado um segundo caso onde o SEC poderia ser útil, ou seja, na identificação de uma situação específica e grave, no caso a perda de conexão entre o servidor de aplicação e o *daemon* de cache de objetos.

Foram então criadas e testadas duas regras, uma baseada na contagem de mais de 50 eventos de códigos 404 no acesso HTTP em um intervalo de tempo de 5 minutos, e outra que procurar por um erro específico de perda de conexão.

Através de um comando no Linux, as últimas 1000 linhas de cada arquivo de log, do servidor web e do servidor de aplicação, foram inseridas nos arquivos

monitorados pelo LogPP, que converteu para o formato CSV e entregou para o SEC.

Conforme o esperado, o SEC processou o conjunto de eventos e identificou os padrões expressados pelas regras, gerando eventos de nível hierárquico superior que indicam o padrão reconhecido. As tabelas 8 e 9 a seguir resumem os casos de testes relatados.

| | |
|---|---|
| **Componente:** | SEC |
| **Operação:** | Correlação de múltiplos eventos com severidade 404 no log do apache. |
| **Propósito:** | Testar se a regra criada irá identificar o padrão expressado. |
| **Descrição:** | Este teste verificará a corretude da regra criada. Será forçado o acontecimento do padrão a ser reconhecido e espera-se que o SEC o detecte. |
| **Entrada** | |
| Como entrada, foram inseridos em um arquivo que o SEC monitorava mais de cinqüenta eventos com erro 404. Isto em menos de cinco segundos. | |
| **Saída Esperada** | |
| Deve ser criado um evento sintético de maior nível hierárquico para o padrão esperado. | |
| **Saída Obtida** | |
| O evento foi criado, com o nível de hierarquia correta. | |
| **Resultado** | |
| OK | |

Tabela 8 – Caso de teste do SEC com padrão 404

| **Componente:** | SEC |
|---|---|
| **Operação:** | Identificação de um erro no acesso de um serviço de cache de objetos. |
| **Propósito:** | Testar se a regra criada irá identificar o padrão expressado. |
| **Descrição:** | Este teste verificará a corretude da regra criada. Será forçado o acontecimento do padrão a ser reconhecido e espera-se que o SEC o detecte. |
| **Entrada** | |
| Como entrada, foi inserido em um arquivo que o SEC monitorava o evento de erro com a descrição do erro de acesso ao serviço de cache de objetos. | |
| **Saída Esperada** | |
| Deve ser criado um evento sintético de maior nível hierárquico para o padrão esperado. | |
| **Saída Obtida** | |
| O evento foi criado, com o nível de hierarquia correta. | |
| **Resultado** | |
| OK | |

Tabela 9 – Caso de teste do SEC com erro específico

A integração entre o SEC e o Publicador foi feita através de uma fila de mensagens, implementada através de uma *message queue* do próprio sistema operacional. Para testar esta integração uma versão preliminar do Publicador, que apenas lia da fila e escrevia na tela as mensagens recebidas, foi utilizada.

Desta forma, o SEC, através de um script Perl inseria os eventos na fila de mensagens e o Publicador monitorava a fila e escrevia na tela. Foram inseridas 1000 linhas de cada arquivo de log nos arquivos monitorados pelo LogPP, que entregou ao SEC, que por sua vez processou os eventos e os escreveu na fila de mensagens. O Publicador consumiu a fila como esperado, e os eventos foram escritos na tela. A Tabela 10 resume este caso de teste.

| **Componente:** | Fila de mensagens |
|---|---|
| **Operação:** | Envio de mensagens entre o SEC e o Publicador |
| **Propósito:** | Testar se a comunicação entre o SEC e o Publicador através de uma fila de mensagens funcionaria. |
| **Descrição:** | Este teste verificará a corretude da comunicação entre o SEC e o publicador. A funcionalidade de publicação no canal do Publicador foi suprimida para este teste.O Publicador apenas escrevia na tela as mensagens recebidas pela fila de mensagens. |
| **Entrada** | |
| Foram inseridos centenas de eventos de log no arquivo que o SEC monitorava. | |
| **Saída Esperada** | |
| Os eventos de log devem ser listados na tela pelo Publicador. | |
| **Saída Obtida** | |
| Os eventos foram listados. | |
| **Resultado** | |
| OK | |

Tabela 10 – Caso de teste de fila de mensagens

O próximo caso de teste valida o envio de eventos pelo Publicador para o Canal. Para este teste uma versão completa do Publicador foi utilizada, nesta versão as mensagens são lidas da fila e publicadas no grupo. Também foi utilizada a ferramenta “spmonitor” que acompanha a aplicação Spread e permite visualizar todas as mensagens enviadas a um grupo do Spread.

Foram então inseridas novamente 1000 linhas de cada arquivo de log nos arquivos monitorados pelo LogPP e todo o processamento foi feito até que o Publicar consumisse a fila de mensagens e publicasse todos os eventos no Canal. A Tabela 11 resume este caso de teste.

| **Componente:** | Publicador |
|---|---|
| **Operação:** | Envio de mensagens com eventos de log pelo canal. |
| **Propósito:** | Testar se o Publicador está enviando mensagens corretamente para o canal. |
| **Descrição:** | Este teste verificará a corretude da Publicação de mensagens de eventos de log no canal. Para este teste será utilizado uma aplicação do Spread, o "spmonitor", que se conecta a grupos e mostra na tela toda a atividade no segmento, como entrada em grupos e envio de mensagens. |
| **Entrada** | |
| Foram inseridos centenas de eventos de log na fila de mensagens que o Publicador monitora. | |
| **Saída Esperada** | |
| Os eventos de log devem ser listados pelo "spmonitor" como mensagens enviadas para o grupo da aplicação. | |
| **Saída Obtida** | |
| Os eventos foram listados pelo "spmonitor". | |
| **Resultado** | |
| OK | |

Tabela 11 – Caso de teste do Publicador

Para testar o Consumidor, uma versão simplificada do Insersor foi utilizada. Nesta versão, o Insersor apenas mostra na tela os eventos que chegam à fila de mensagens que ele consome. Sendo assim, foram inseridos novamente 1000 linhas de log de cada arquivo no arquivo monitorado pelo LogPP e, depois do processamento pelo SEC e do envio pelo Publicador, os eventos foram recebidos pelo Consumidor que os escreveu na fila de mensagens. O Insersor escreveu na tela os eventos recebidos.

A Tabela 12 mostrada a seguir resume este caso de teste.

| **Componente:** | Consumidor |
|---|---|
| **Operação:** | Recepção de mensagens de eventos de log. |
| **Propósito:** | Testar se o Consumidor está recebendo as mensagens corretamente e inserindo-as na fila de mensagens. |
| **Descrição:** | Este teste verificará a corretude da recepção de mensagens do canal através de um grupo e a inserção dos eventos na fila. Para este teste, o componente Insersor, que monitora a fila de mensagens, foi modificado para apenas mostrar os eventos de log que recebe na fila. |
| **Entrada** | |
| Foram enviados centenas de eventos de log para o canal, através do componente Publicador. | |
| **Saída Esperada** | |
| Os eventos de log devem ser listados pelo Insersor. | |
| **Saída Obtida** | |
| Os eventos foram listados pelo Insersor. | |
| **Resultado** | |
| OK | |

Tabela 12 – Caso de teste do Consumidor

O componente Insersor foi testado novamente em sua versão final, ou seja, lendo da fila de mensagens e inserindo os eventos no TelegraphCQ. Foram inseridos 1000 linhas de cada arquivo de log no arquivo monitorado pelo LogPP e, depois de todo o processo, esses eventos chegaram a fila de mensagens consumida pelo Insersor.

No TelegraphCQ foi criada uma consulta contínua que mostrava todo os eventos do fluxo, e assim, ao mostrar os eventos presentes no fluxo o componente Insersor foi validado. A Tabela 13 resume este caso de teste.

| **Componente:** | Insersor |
|---|---|
| **Operação:** | Inserção de eventos de log no TelegraphCQ |
| **Propósito:** | Testar se o Insersor está consumindo a fila de mensagens corretamente e inserindo-as no TelegraphCQ. |
| **Descrição:** | Este teste verificará a corretude do consumo de uma fila de mensagens com a inserção de eventos de log no TelegraphCQ. |
| **Entrada** | |
| Foram enviados centenas de eventos de log para o canal, através do componente Publicador. O componente Consumidor, insere os eventos na fila de mensagens. | |
| **Saída Esperada** | |
| Os eventos de log devem ser mostrados no TelegraphCQ através de uma consulta contínua que lista todo o conteúdo do fluxo. | |
| **Saída Obtida** | |
| Os eventos foram listados pelo TelegraphCQ. | |
| **Resultado** | |
| OK | |

Tabela 13 – Caso de teste do Insersor

Por fim, foram inseridas as duas consultas contínuas descritas na seção 5.5 no TelegraphCQ e realizado o teste completo do protótipo. Desta forma, novamente 1000 linhas de cada arquivo de log foram inseridas e foram alisadas as saídas das duas consultas contínuas. Este teste valida o caso de uso aprestado na seção 5.5, conforme mostra a Tabela 14.

| **Componente:** | Sistema |
|---|---|
| **Operação:** | Caso de uso descrito na seção |
| **Propósito:** | Testar se todos os componentes do sistema estão se comportando como o esperado e validar o caso de uso descrito na seção 5.5. |
| **Descrição:** | Este teste verificará a corretude do sistema, que deve atender ao caso de uso descrito na seção 5.5. Todos os componentes foram configurados e iniciados. Os registros de logs inseridos nos logs monitorados e a saída vista pelo TelegraphCQ. |
| **Entrada** | |
| Foram inseridos 1000 registros do log do Apache e 1000 registros do log do JbossWeb nos logs monitorados pelo LogPP. Estes registros continham as condições esperadas pelo SEC. | |
| **Saída Esperada** | |
| Os eventos de log devem aparecer no LogPP, segundo as duas consultas contínuas descritas. | |
| **Saída Obtida** | |
| Os eventos foram listados pelo TelegraphCQ. | |
| **Resultado** | |
| OK | |

Tabela 14 – Caso de teste do sistema completo