MiddLog: Uma infra-estrutura de serviços de log de aplicações baseada em tecnologias de middleware

Marcelo Pitanga Alves, Paulo F. Pires,
Flávia C. Delicato, Maria Luiza M. Campos

Departamento de Ciência da Computação (DCC/IM) - Núcleo de Computação Eletrônica (NCE) - Universidade Federal do Rio de Janeiro (UFRJ)

Bloco C – Cidade Universitária – Ilha do Fundão – Rio de Janeiro – RJ – Brasil

mpitanga@domain.com.br, {paulopires,fdelicato,mluiza}@nce.ufrj.br

Abstract. MiddLog is an extensible and configurable logging infrastructure based on the middleware technology. It includes a set of classes and components that are deployed on an application server, performing analysis and recordings of application events at a log file, in a dynamic and transparent way. MiddLog's extension capability enables its users to increase its services by aggregating new features or creating new components and inserting them into the infrastructure. This paper focuses on three major parts of MiddLog: interception layer, integration layer and processing messages layer.

Resumo. MiddLog é uma infra-estrutura extensível e configurável de serviços de log de aplicações, baseada em tecnologias de middleware. Ela é formada por um conjunto de classes e componentes que são instalados em um servidor de aplicações e executam, de forma dinâmica e transparente, a análise e registro dos eventos das aplicações no arquivo de log. Sua capacidade de extensão permite a seus usuários ampliar seus serviços, agregando novas funcionalidades ou criando novos componentes e incorporando-os à infra-estrutura. Este artigo concentra-se em três partes fundamentais do MiddLog: camada de interceptação, camada de integração e camada de processamento das mensagens.

1. Introdução

Ao longo dos anos o *log* tem sido um grande aliado dos profissionais de TI para depurar e documentar as versões dos sistemas informatizados, tornando-se um importante instrumento no monitoramento do ciclo de vida de sistemas. O avanço tecnológico, aliado a uma crescente complexidade das aplicações, fizeram com que os projetistas de sistemas se preocupassem cada vez mais com a eficiência dos serviços prestados. Acompanhando esta evolução, o *log* tornou-se um importante mecanismo de registro de eventos e detalhes de funcionamento das aplicações, podendo ser adotado como fonte de dados para ambientes analíticos mais sofisticados [Kimball et al. 1998, Kimball e Merz 2000, Bonchi et al. 2001, Cruz et al. 2003].

Implementar mecanismos de gerenciamento de *log* constitui uma tarefa complexa. Esta implementação é usualmente realizada de maneira proprietária, uma vez que os comandos para criar e gerenciar as saídas no *log* são incluídos diretamente no

código fonte da aplicação, aumentando, assim, o tempo de desenvolvimento e a complexidade do código gerado. Outro problema é a falta de padrões para o conteúdo e formato do arquivo de *log* gerado, dificultando o seu uso por aplicações de análises de *log*.

Alguns trabalhos têm sido conduzidos visando facilitar a criação, padronização e incorporação de mecanismos de *log* de aplicações. Tais trabalhos seguem a idéia de propor uma infra-estrutura de desenvolvimento [LogKit 2003, Log4j 2003, Nate 2001, RP 2003, JLog 2003, JSDK 2003] que retire dos desenvolvedores das aplicações a responsabilidade de desenvolver o complexo mecanismo de *log*, deixando que os mesmos se preocupem somente com a lógica de negócio de suas aplicações. Uma infra-estrutura de desenvolvimento de *log* oferece um conjunto de classes que o desenvolvedor inclui no código fonte de sua aplicação e arquivos de configuração para habilitar a aplicação desenvolvida a gerar arquivos de *log*. Desta forma, a alteração do código fonte e recompilação da aplicação são inevitáveis sempre que o desenvolvedor desejar incluir ou desativar a geração do *log* de sua aplicação, uma vez que a infra-estrutura de desenvolvimento não atua como uma camada de serviço externa à aplicação, como acontece, por exemplo, com os serviços de controle de transações em servidores de aplicações.

Neste artigo são apresentadas a arquitetura e a implementação da MiddLog, uma infra-estrutura extensível e dinamicamente configurável de serviços de *log*.

A MiddLog é uma infra-estrutura de serviços em camadas que atende as necessidades atuais dos desenvolvedores de aplicações através de um mecanismo de *logging* transparente, que executa independente da aplicação monitorada e que permite a configuração, de forma dinâmica, da geração do *log* da aplicação. A MiddLog atende tais necessidades através da implementação dos seguintes requisitos: monitoramento e captura transparente das informações das aplicações em execução; formatação dos dados capturados a partir de um modelo de dados canônico; retirada do código da aplicação dos comandos de geração de *log*, permitindo que esta tarefa seja feita através de arquivos de configuração; disponibilidade e execução de seus serviços de *log* de forma independente da construção da aplicação que utilizará o *log*; e minimização do impacto do serviço de *log* no desempenho das aplicações. Para atingir estes requisitos, a infra-estrutura do MiddLog utiliza como base tecnologias de *middleware* [OMG 2003, Chung et al. 1998, COM+ 2004] e a programação por aspectos AOP (Aspect-Oriented Programming) [Gregor et al. 1997, Shukla et al. 2002].

Sistemas de *middleware* consistem em componentes de software que fornecem uma infra-estrutura de serviços (transações, mensagens, segurança, conexões a banco de dados, entre outros) para desenvolvimento de aplicações distribuídas. A tecnologia de *middleware* é utilizada na composição das camadas de serviços da infra-estrutura da MiddLog através da implementação de componentes. Os componentes são instalados e executados por um servidor de aplicações permitindo, assim, que os serviços oferecidos pela MiddLog estejam sempre disponíveis, a despeito do fato de uma aplicação usar ou não os serviços de *log*.

A AOP foi desenvolvida nos laboratórios da XEROX PARC na década de 90 [Gregor et al. 1997], como uma técnica para permitir que o desenvolvimento de tarefas como operações matemáticas, tratamento de exceções, logging, etc., possa ser realizado separadamente da lógica da aplicação. A técnica de AOP é utilizada na MiddLog na

construção da camada de componentes interceptadores que monitoram a execução da aplicação para capturar as informações sobre seu contexto de execução, eliminando da aplicação os comandos para geração de *log* e permitindo que o desempenho da aplicação monitorada não seja afetado.

O *log* é um serviço complexo e atualmente um instrumento altamente requisitado pelos desenvolvedores para promover a qualidade dos serviços oferecidos pelas suas aplicações. O serviço de *logging*, como propõe a AOP, pode ser oferecido como um serviço ortogonal ao desenvolvimento da aplicação e é neste contexto que a infra-estrutura MiddLog se diferencia das atuais infra-estruturas de log, fornecendo a lógica para geração de *log* como um serviço de middleware.

Este artigo está estruturado em 5 seções. Na Seção 2 são apresentados os trabalhos relacionados. Na Seção 3, descreve-se a arquitetura geral da MiddLog e, a seguir, detalha-se a implementação de cada camada e os seus componentes. Na Seção 4, é apresentado um cenário que demonstra o funcionamento geral da infra-estrutura. Finalmente na Seção 5, são apresentadas a conclusão e considerações finais do trabalho.

2. Trabalhos relacionados

A MiddLog é um trabalho multidisciplinar que está relacionado com outras áreas como infra-estruturas de desenvolvimento de *log* e AOP. Muitos trabalhos têm sido propostos nestas duas áreas com o objetivo de facilitar a incorporação do processo de *logging* nas aplicações com o mínimo de esforço. Por exemplo, na área de infra-estruturas de *log* existem diversos trabalhos [LogKit 2003, Log4j 2003, Nate 2001, RP 2003, JLog 2003, JSDK 2003] que fornecem o suporte necessário à criação de mecanismos de *log* para aplicações através de um conjunto de classes, as quais possuem as operações básicas para instanciar, configurar, formatar as saídas e envio das mensagens para o mecanismo de *log*. Entretanto, estas infra-estruturas não fornecem recursos de captura automática de informações das aplicações, ficando essa responsabilidade a cargo do desenvolvedor da aplicação.

A AOP [Gregor et al. 1997, Shukla et al. 2002] propõe técnicas que permitem que uma aplicação tenha sua execução inspecionada a cada objeto, método ou campo acessado. Nesta área, existem diversos trabalhos [AOSD 2004] que fornecem o suporte necessário para implementar aspectos com um mínimo de esforço. Em [Ramnivas 2004] o autor mostra como usar a técnica de AOP na criação de um aspecto, *interceptador*, que utiliza uma infra-estrutura de desenvolvimento de *log* para criar o mecanismo de *log* e registrar todos os passos de execução da aplicação. Esta é uma solução amplamente utilizada que, apesar de facilitar a captura de informações, deixa ainda sob a responsabilidade do desenvolvedor da aplicação a formatação dos dados capturados e seu envio à infra-estrutura de *log*.

3. Infra-estrutura de serviços MiddLog

A arquitetura da MiddLog, apresentada na Figura 1, é composta de diversos serviços, os quais são decompostos em 3 camadas básicas: *camada de integração, camada de interceptação* e *camada de processamento de mensagens*. Os serviços são implementados através de componentes de tecnologias de *middleware* (Figura 2), permitindo a sua integração com servidores de aplicações. Isto permite aos serviços do

MiddLog serem oferecidos como parte dos serviços do servidor de aplicações, facilitando sua utilização por parte das aplicações.

A camada de integração de serviços é a responsável por oferecer os serviços básicos da MiddLog para as demais camadas, permitindo um maior grau de abstração e de extensibilidade da arquitetura, uma vez que novos serviços de interceptação poderão ser criados e incorporados à MiddLog.

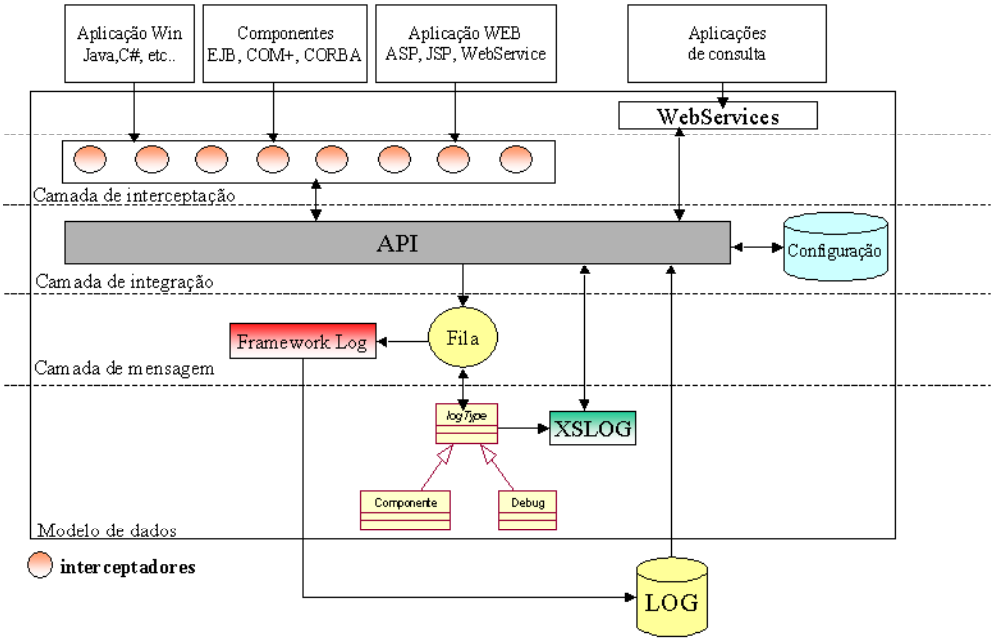

Figura 1. Arquitetura de camadas da MiddLog

A camada de interceptação é constituída de componentes responsáveis pela captura das informações da aplicação em execução e envio destas informações para a camada de processamento de mensagens. Os componentes da camada de interceptação são implementados através de *advices* da AOP. Os *advices* são módulos de programas que são executados quando um determinado evento da aplicação é iniciado. Na MiddLog os *advices* são implementados através de componentes *interceptadores* [Lowy 2003, AspectJ 2003, Shukla et al. 2002]. Os *interceptadores* são associados à aplicação a ser monitorada através de um arquivo de configuração gerenciado pela camada de integração, que contém as classes e métodos para monitoramento. As informações capturadas das aplicações são organizadas de acordo com o modelo de dados especificado na infra-estrutura, gerando uma mensagem no formato texto ou um objeto, o qual é, então, enviado para a fila da MiddLog na camada de processamento de mensagens.

A camada de processamento de mensagens é responsável pelo gerenciamento da fila de mensagens do *log*, que é alimentada pelos *interceptadores*. Após uma mensagem entrar na fila, um componente entra em execução retirando esta mensagem da fila, abrindo o pacote da mensagem, extraindo e redirecionando os dados para a infra-estrutura de *log*, que é o mecanismo responsável pelo registro físico destas informações em um meio de armazenamento previamente configurado.

A Figura 2 ilustra a interação entre estas camadas, que serão descritas com maior detalhe nas próximas seções.

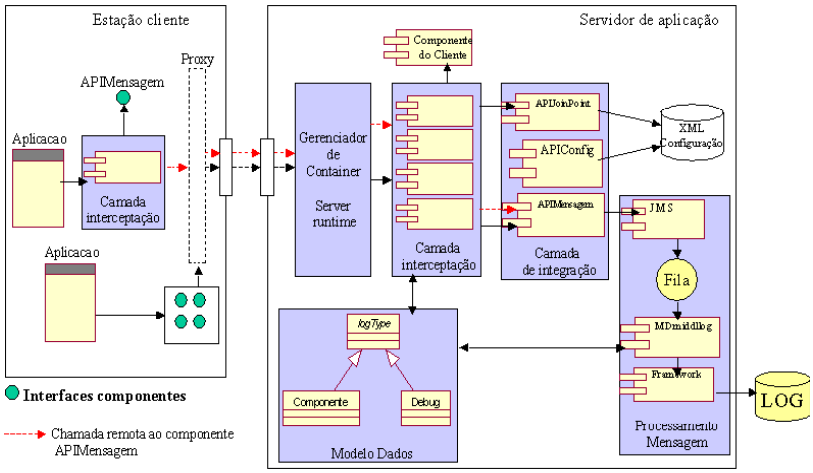

Figura 2. Arquitetura de implementação

3.1.1. Serviço de mensagens

É o serviço responsável por oferecer as interfaces necessárias para que os interceptadores possam enviar os dados capturados da aplicação à fila de mensagens da MiddLog. O serviço de mensagens é implementado pelo componente *APIMensagem*.

APIMensagem. O serviço de mensagens da MiddLog utiliza a infra-estrutura de mensagem do servidor de aplicações no qual a MiddLog está incorporada. O objetivo do componente APIMensagem é encapsular o acesso à infra-estrutura de mensagem do servidor de aplicação, implementando uma interface de alto nível (Figura 3), a qual é utilizada pelos componentes interceptadores no acesso e envio das informações capturadas das aplicações para a fila de mensagem da MiddLog. A Figura 2 mostra o componente APIMensagem invocando o componente JMS (Java Message Service), da arquitetura J2EE [Sun MicroSystem 2004], para enviar a mensagem.

```
public interface APIMensagem extends EJBObject
{
        public void abrefila() throws JMSException, NamingException, RemoteException;
        public void fecharfila() throws RemoteException;
        public void enviarMensagem(String id, String msg) throws JMSException, NamingException, RemoteException;
        public void enviarMensagem( String id, logType objmsg ) throws JMSException, NamingException,
RemoteException;
}
```

Figura 3. Interface para APIMensagem

3.1.2. Serviço de configuração

O serviço de configuração é responsável por oferecer as interfaces necessárias para acesso aos arquivos de configuração. O serviço de configuração atua em tempo de execução permitindo assim que a qualquer momento o desenvolvedor possa incluir novas aplicações, classes e métodos a serem monitorados ou interromper o monitoramento simplesmente eliminando as configurações da aplicação dos arquivos de configuração. O serviço de configuração é implementado pelos componentes: *APIJoinPoint* e *APIConfig*.

APIJoinPoint. O componente APIJoinPoint (Figura 4) é responsável por oferecer os serviços necessários para que os *interceptadores* possam acessar o arquivo de configuração que contém as aplicações a serem monitoradas. Este arquivo de

configuração é a ligação entre a aplicação a ser monitorada e os *interceptadores*. Os *interceptadores* utilizam os serviços do componente APIJoinPoint para carregar o arquivo de configuração e navegar por suas configurações a fim de identificar quais classes e métodos de uma aplicação serão monitorados.

```
public interface APIJoinPoint extends EJBObject {
        public boolean aplicacaoExiste(String tagAplicacao) throws RemoteException;
        public boolean classeExiste(String tagAplicacao,String tagClasse) throws RemoteException;
        public boolean metodoExiste(String tagAplicacao, String tagClasse, String tagMetodo) throws RemoteException;
}
```

Figura 4. Interface para APIJointPoint

APIConfig. O componente *APIConfig* (Figura 5) é o responsável por fornecer ao desenvolvedor da aplicação os serviços necessários para que ele possa incluir sua aplicação para ser monitorada pela MiddLog. Os serviços do componente APIConfig retiram do desenvolvedor a responsabilidade de edição dos arquivos de configuração, garantindo ao MiddLog a integridade das informações nos arquivos de configuração.

```
public interface APIConfig extends EJBObject {
        public void JoinPointInterceptor(String aplicacao, Hashtable classe, Hashtable metodo) throws RemoteException;
        public void log(String aplicacao, String arquivoLog) throws RemoteException;
        public void excluirlog(String aplicacao) throws RemoteException;
}
```

Figura 5. Interface para APIConfig

3.2. Camada de interceptação

A camada de interceptação é responsável, junto com os serviços de configuração, por eliminar a necessidade de inclusão de comandos dentro da aplicação, seja ela uma aplicação que execute no lado cliente (java, C/C++, C#) ou no lado servidor (componentes EJB, Servlets, etc.). Tratar o processo de *logging* como um aspecto [Gregor et al. 1997] permitiu utilizar técnicas de AOP como a base da solução da camada de interceptação. A técnica de AOP trata uma aplicação em termos dos seus *joinpoints* [AspectJ 2003], que são pontos na execução da aplicação onde um determinado aspecto pode ser aplicado. Assim, é possível realizar o acompanhamento do fluxo de execução de uma aplicação quando ele transita de um método para outro. O código original da aplicação permanece inalterado, e a ativação/desativação do processo de *logging* depende da inclusão/exclusão do aspecto relacionado na aplicação.

Os aspectos são criados através de instância da classe *advices* [AspectJ 2003] da AOP. Os *advices* são implementados na MiddLog através de componentes da arquitetura de Middleware chamados *interceptadores*. Estes *interceptadores* contêm as alterações que devem ser aplicadas ortogonalmente à aplicação, neste caso o processo de *logging,* e uma especificação que identifica em quais *joinpoints* o aspecto se aplica. Os *joinpoints* são definidos através de arquivo de configuração, mantido pela camada de integração, conforme apresentado na Seção 3.1.2.

Interceptadores do lado servidor. A interceptação no lado servidor busca monitorar as aplicações (componentes, serviços web, etc.) que o desenvolvedor instala no servidor de aplicações. O *interceptador* é implementado na forma de um componente de middleware e é "plugado" na arquitetura do servidor de aplicações, atuando como mais um serviço no ambiente de execução destas aplicações, ou seja, estendendo o *container* do servidor de aplicações com as funcionalidades de captura de informações de execução dos seus serviços (Figura 6).

Durante a invocação de um componente por uma aplicação cliente, o servidor de aplicações cria um contexto de execução, associa o componente *interceptador* da MiddLog ao contexto e carrega o componente para este contexto a fim de ser executado. A partir deste momento todas as invocações aos componentes passam agora pelo *interceptador,* que por sua vez deve saber quais *joinpoints* devem ser tratados.

```
public interface Interceptor extends ContainerPlugin
{
        ...
        Interceptor getNext();
        Object invokeHome(Invocation mi) throws Exception;
        Object invoke(Invocation mi) throws Exception;
}
```

Figura 6. Interface para um interceptador de container para componentes EJB

```
...
if (ejbcfg == null) {
     ejbcfgHome = (APIJoinPointHome)context.lookup("APIJoinPoint");
    ejbcfg = ejbcfgHome.create();
}
if (log == null) {
    msgHome = (APIMensagemHome) context.lookup("APIMensagem");
    log = msgHome.create();
}
for (int i = (filaCopia.length-1); i >= 0; --i)
{
    Componente comp = (Componente) array[i];
   if (ejbcfg.metodoExiste(comp.getidApp(), comp.getObjeto(), comp.getMetodoEvento())) {
        System.out.println("Middllog: EJBInterceptor enviando mensagem...");
        log.enviarMensagem(comp.getidApp(),comp);
   }
}
log.fecharfila();
...
```

Figura 7. Verificação dos joinpoints e envio da mensagem à fila da MiddLog

Na Figura 7, observa-se um exemplo de código que realiza a verificação dinâmica dos *joinpoints*. O *interceptador* realiza esta verificação através da invocação de um serviço do componente *APIJoinPoint*. Quando um *joinpoint* é identificado, o *interceptador* envia os dados coletados do componente em execução para a fila da MiddLog na camada de processamento de mensagens através da invocação de um serviço do componente *APIMensagem*.

Retardo na interceptação e verificação do *joinpoint*. Apesar da MiddLog utilizar uma fila para despachar os dados coletados para a camada de processamento, um problema enfrentado na criação deste tipo de interceptador foi não inserir retardos na execução do componente no servidor de aplicações durante o processo de interceptação.

A solução deste problema foi implementar o processo de captura e envio das informações coletadas pelo *interceptador* através de comunicação assíncrona. O processo de captura é implementado com o auxílio de uma fila interna no *interceptador* e o controle da fila é feito através de um *Thread*. Por exemplo, quando a chamada de um componente é interceptada, os dados são capturados e formatados neste contexto, segundo o modelo de dados adotado. Os dados formatados são adicionados na fila interna do *interceptador*, liberando, logo em seguida, o componente cliente para continuar o fluxo normal de processamento. Sempre que o *Thread* entra em execução, é feita uma cópia desta fila para processamento, para em seguida a fila ser novamente esvaziada e habilitada a receber novas mensagens. Paralelamente, o *Thread* retira cada mensagem desta cópia da fila para verificação dos *joinpoints* e envio das mensagens para a fila da MiddLog.

Interceptadores do lado cliente. A interceptação no lado cliente busca monitorar aplicações que são executadas na estação cliente. O objetivo é permitir que o desenvolvedor da aplicação possa realizar o monitoramento de seu aplicativo, mesmo que este não esteja rodando em um servidor de aplicações. A grande dificuldade existente neste ambiente é o seu contexto de execução. Diferentemente do ambiente de execução do servidor de aplicações, que em sua própria arquitetura já oferece os recursos para interceptação, no ambiente cliente a execução da aplicação é feita diretamente pelo sistema operacional, no caso de aplicações escritas, por exemplo, em *C/C++*, ou através de uma máquina virtual (*virtual machine*), no caso de aplicações *java (JVM)* e *.Net (CLR)*.

Neste contexto, o uso das técnicas da AOP torna-se mais uma vez um elemento fundamental na infra-estrutura proposta. Importantes trabalhos [AOSD 2004] têm sido conduzidos nesta área, como *AspectJ*, *AspectC#*, *JBossAOP Framework*, *AspectWerkz* entre outros, que facilitam a construção dos aspectos e definição dos pontos de interceptação a serem aplicados na aplicação cliente. Nestes ambientes, ao contrário do ambiente do lado servidor, a aplicação cliente passa por um processo de pré-compilação onde seus códigos binários (*byte-codes*) são manipulados para serem ligados aos aspectos, seguindo as definições configuradas pelo desenvolvedor do aspecto no seu arquivo descritor.

O aspecto no lado cliente oferecido pela MiddLog (Figura 8) é implementado por um *interceptador* baseado no JBossAOP Framework [JBOSS 2004]. A escolha desta infra-estrutura foi motivada por alguns de seus requisitos, em especial: possuir o suporte de *middleware* do servidor de aplicações JBoss; permitir que os pontos de interceptação sejam definidos por arquivos de descrição; permitir que os *advices* sejam implementados como interceptadores e possuir implementação independente do servidor de aplicações.

```
public interface Interceptor
{
  public String getName();
  public Object invoke(Invocation invocation)
 throws Throwable;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <bind pointcut="all(teste)">
     <interceptor factory="middllog.interceptor.TraceInterceptorFactory"/>
  </bind>
</aop>
```

Figura 8. Interface e arquivo descritor para um interceptador no JBossAOP

Ao se implementar este *interceptador* deve haver uma preocupação com a captura, identificação e formatação dos dados gerados durante o período de interceptação da aplicação, para envio à fila da MiddLog. Assim, da mesma forma que no *interceptador* do lado servidor, o envio do pacote de dados é feito através da invocação remota de um serviço do componente *APIMensagem* da camada de integração, conforme apresentado na Figura 9. O uso do componente *APIMensagem* traz, entre outros benefícios, o seu uso por um *interceptador* criado pelo desenvolvedor da aplicação e envio das mensagens à fila da MiddLog através de uma invocação local ou remota, permitindo assim manter a extensibilidade da infra-estrutura.

Retardo na interceptação e verificação do *joinpoint*. Diferentemente dos *interceptadores* no lado servidor, na implementação do lado cliente não há a preocupação de incluir a programação necessária para verificação dos *joinpoint*s onde o aspecto (*interceptador*) irá atuar. Como visto nos parágrafos anteriores, a verificação é feita em tempo de compilação da aplicação. Esta técnica traz vantagens e desvantagens

para o processo. A vantagem é o ganho de desempenho do *interceptador,* porque ele já sabe aonde atuar, não perdendo tempo na verificação dos *joinpoints*. A desvantagem é que sempre que o desenvolvedor desejar incluir novos pontos de interceptação terá que incluí-los no arquivo descritor e gerar um novo código objeto de sua aplicação.

```
...
if (log == null) {
        ...
        lookup = context.lookup("APIMensagem");
        msgHome = (APIMensagemHome)PortableRemoteObject.narrow(lookup, APIMensagemHome.class);
        log = msgHome.create();
}
log.enviarMensagem(debug.getidApp(),debug);
log.fecharfila();
...
```

Figura 9. Invocação remota do componente APIMensagem e envio dos dados à fila

Uso do JBossAOP no lado servidor. A Infra-estrutura JBossAOP pode ser utilizada também para a criação de interceptadores no lado servidor. Porém, o seu uso nesta camada da infra-estrutura proposta coloca a cargo do desenvolvedor da aplicação a obrigatoriedade de compilar a sua aplicação para inclusão ou exclusão do *interceptador*. Com isto a infra-estrutura perderia a transparência e o dinamismo no processamento do lado servidor, uma vez que a implementação do *interceptador* do lado servidor, baseada no *container,* não necessita de recompilação da aplicação cliente, sendo todo o processo baseado em arquivo de configuração, conforme visto na seção anterior .

3.3. Camada de processamento de mensagens

A camada de processamento de mensagens, apresentada na Figura 2, é responsável pelo recebimento e tratamento de todas as mensagens enviadas pelos *interceptadores,* antes de serem armazenadas no arquivo de *log*. Esta camada é formada por 2 (dois) serviços principais que são: *fila de mensagens* e *infra-estrutura de log*.

Fila de mensagens. A fila de mensagens da MiddLog é um importante instrumento para permitir que o processo de captura e envio de dados pela camada de interceptação seja assíncrono. Esta fila é criada no provedor de mensagens do servidor de aplicações e é utilizada pelos componentes *interceptadores*, ou diretamente pela aplicação, via componente *APIMensagem,* para que possam enviar as informações coletadas das aplicações para o *log*.

```
...
  if (id != null) {
        String apl = "middlog.mensagem.MDmiddog."+id;
        String appenderName = "middlog."+id;
        System.out.println("Middlog: Recebendo mensagem da aplicacao.");
        Category log = Category.getInstance(apl);
        if (log.getAppender(appenderName) == null) {
                System.out.println("Middlog: Carregando configuracao de log...");
                DOMConfigurator.configure("/middlog/app/log.xml");
        }
        log.debug(strMsg);
        System.out.println("Middlog: Mensagem gravada.");
}...
```

Figura 10. Envio das informações para a infra-estrutura de log

O processamento das mensagens desta fila é realizado através de um componente orientado a mensagens, *MDmiddlog*, o qual é associado à fila criada no servidor de aplicações. O componente *MDmiddlog* é responsável por receber cada mensagem da fila, realizar a verificação do tipo de mensagem recebida, extrair da mensagem a identificação da aplicação e as informações para serem enviadas à infra-

estrutura de *log*. A identificação da aplicação (*ID*) é uma informação importante porque é através dela que se associa uma instância da infra-estrutura de *log* à aplicação sendo monitorada e também ao *appender*, que é o componente da infra-estrutura de *log* responsável por processar as saídas do *log*.

Infra-estrutura de log. Na MiddLog usa-se uma infra-estrutura de *log* de mais baixo nível na implementação do mecanismo de *log*. Essa infra-estrutura é responsável por gerenciar o recebimento e envio, para um dispositivo previamente configurado, das informações recebidas do componente *MDmiddlog* referentes as aplicações que estão sendo monitoradas. Dentre as iniciativas de infra-estrutura de *log* disponíveis para uso [LogKit 2003, Log4j 2003, Nate 2001, RP 2003, JLog 2003, JSDK 2003], a Log4J, foi a escolhida para fazer parte da MiddLog pela sua flexibilidade, portabilidade e popularidade na comunidade de desenvolvimento de software.

3.4. Modelo de dados

A MiddLog conta com uma proposta de modelo de dados extensível que busca como objetivo principal a padronização do conteúdo e a exposição das informações coletadas das aplicações. A seguir são descritos as principais vantagens e benefícios do modelo proposto.

Extensibilidade. A extensibilidade deste modelo é um dos pontos fortes da proposta. Usando técnica de Orientação a Objetos, novos tipos podem ser criados a partir de um tipo base ou de subtipos, garantindo a flexibilidade e reutilização do modelo sem afetar os componentes e aplicações já desenvolvidas.

O tipo base do modelo proposto é o *logType*. Este tipo abstrato fornece as informações mínimas para a geração do *log* e um método obrigatório chamado *getXML*, que são utilizados na criação dos subtipos. O método abstrato *getXML* é utilizado para que os desenvolvedores formatem as suas saídas para o *log* (Figura 11).

```
public abstract class logType implements java.io.Serializable {
. . . public abstract String getXML();
}
public class Componente  extends logType  implements  java.io.Serializable {
        . . .
        public String getXML(){
                String strXML = "";
                strXML += "<xslog:componente idApp="'+getidApp()+'"";
                strXML += " descricao="'+getDescricao()+'"";
                . . .
                strXML += "/>";
                return strXML;
```

Figura 11. Fragmento de código do modelo em uma classe java

Padronização e exposição das informações. A MiddLog implementa uma linguagem de descrição baseada na linguagem XML [W3C 2004], para representar, validar e expor o modelo de dados proposto, denominada *XSLOG* (Figura 12). A linguagem XSLOG estende os tipos básicos oferecidos pela XML, fornecendo um conjunto bem organizado de tipos e estruturas para que sejam utilizados na criação dos arquivos de *log*. Através da capacidade de extensão da XML foi possível criar um esquema extensível que representa o modelo de dados proposto e que atende as necessidades de extensibilidade da MiddLog e de validação dos arquivos de *log* com conteúdo variável.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.middlog.com.br" ... ">
   <xs:element name="logEntry" type="xslog:logType" abstract="true"/>
   <xs:element name="log">
   <xs:complexType> <xs:sequence>        <xs:element ref="xslog:logEntry" maxOccurs="unbounded"/>     </xs:sequence>
   </xs:complexType>
   </xs:element>
   <xs:element name="componente" type="xslog:componenteType"  substitutionGroup="xslog:logEntry"/>
   <xs:complexType name="logType">
            <xs:attribute name="idApp" type="xs:string" use="required"/>  ...
   </xs:complexType>
   <xs:complexType name="componenteType">
            <xs:complexContent>
               <xs:extension base="xslog:logType">
                  <xs:attribute name="objeto" type="xs:string" use="required"/>
                        ...
   </xs:complexType>
...
```

Figura 12. Fragmento do esquema XML da XSLOG

4 . Cenário de uso

Nesta seção é apresentado um experimento que ilustra o funcionamento e características do processo de *logging* da infra-estrutura MiddLog.

A aplicação foi desenvolvida utilizando-se componentes distribuídos, cuja relativa simplicidade para testar o sistema proposto não invalida o resultado do experimento. A aplicação tem como objetivo receber o nome de uma pessoa e exibir a palavra "*Olá",* juntamente com o nome da pessoa, na tela do usuário. A aplicação possui duas partes: um componente do lado servidor implementado através de um Bean de sessão (*session bean)* [Sun MicroSystem 2004] e uma aplicação cliente que utiliza o serviço.

```
public class HelloWorldBean implements SessionBean
{
            private String nome;
            public String getHelloWorld() { return "Ola, "+nome; }
            public void setname(String name) { this.nome = name; }
            ...
}
```

Figura 13. Fragmento do código do componente HelloWorldBean

O componente *HelloWorldBean* (Figura 13) representa uma classe de negócio que expõe seus serviços para a aplicação cliente. Os serviços oferecidos são: *setname(String name),* que recebe como parâmetro o nome de uma pessoa e *getHelloWorld(),* que retorna a mensagem "*Olá <nome_da_pessoa>"* para a aplicação cliente.

```
...
public class Client {
public static void main(String[] args) {
try {       InitialContext ic = new InitialContext();
            Object lookup = ic.lookup("HelloWorld");
            HelloWorldHome home = (HelloWorldHome)PortableRemoteObject.narrow(lookup, HelloWorldHome.class);
            HelloWorld hello = home.create();
            hello.setname("Marcelo Pitanga");
            System.out.println(hello.getHelloWorld());
            hello.remove();
            hello = null;    }
catch ...
```

Figura 14. Fragmento do código da classe cliente que invoca o componente

A aplicação cliente é composta de uma classe principal chamada *Client* (Figura 14) que é responsável pela invocação do componente no servidor de aplicações e invocação dos serviços.

Pode-se verificar que, tanto no código do componente (Figura 13), como no código da aplicação cliente (Figura 14), não existe nenhum código que gere informações de *log*. Para gerar o *log* da aplicação através da MiddLog, deve-se registrar a aplicação na lista de *joinpoint* para que a MiddLog inicie o processo de monitoramento e geração do arquivo de *log* (Figura 15). Ainda, é necessário que a MiddLog seja instalada como um serviço no servidor de aplicação. Primeiro, realiza-se a implantação da MiddLog no servidor. Segundo, cria-se a fila de mensagens no provedor de mensagens do servidor e por último, associa-se cada componente interceptador ao *container* específico no servidor de aplicações.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuracao . . . >
  <aplicacao nome="HelloWorld">
        <classe nome="*">
              <metodo nome="*"/>
        </classe>
  </aplicacao>
</configuracao>
```

```
. . .
<xslog:componente idApp='HelloWorld' descricao='' Owner='EjbModule'
objeto='HelloWorldHome' idObjeto='7298031' interface='HelloWorld'
metodoEvento='create' argsmetodo='() return (interface HelloWorld)'
dataEvento='14/10/2004' horaInicioEvento='16:47:15'
horaTerminoEvento='16:47:15'/>
. . .
```

Figura 15. Arquivo de configuração de joinpoint e saída gerada no log

O objetivo principal deste experimento é mostrar que o processo de *logging* da MiddLog pode ser facilmente utilizado nas aplicações executadas em servidores de aplicações sem que o desenvolvedor necessite alterar a sua aplicação. Ainda, neste experimento procurou-se avaliar o impacto da MiddLog sobre a execução das aplicações clientes no servidor, através de uma coleta de tempos efetuada antes e após a aplicação do processo de *logging* sobre a aplicação monitorada. A Tabela 1 descreve o ambiente de hardware utilizado para rodar este experimento. Na Tabela 2, é apresentado o resultado do experimento. Pode-se observar que não houve degradação de desempenho na execução da aplicação com a utilização do serviço de *log* da MiddLog.

Tabela 1. A máquina servidora e software utilizados nos testes

Pentium4 de 1Ghz com 256MB de RAM e HD de 80GB com 64GB livres
Sistema operacional Windows2000 com SP4 instalado
Servidor de Aplicações: JBOSS 4.00DR4
J2SDK versão 1.4.2_05
Jakarta Log4J versão 1.2.8

Tabela 2. Comparativo do tempo de execução da aplicação (15 chamadas do serviço)

Aplicação	Usa MiddLog	Hora inicio	Hora término	Duração
EJB	Não	18:31:13,406	18:31:35,421	00:00:22,015
EJB	Sim	18:34:51,15	18:35:13,828	00:00:22,7

5. Conclusão

As atuais infra-estruturas de *log* permitem que programadores possam incluir logs em suas aplicações sem que eles tenham que se preocupar em desenvolver os complexos códigos de gerenciamento do arquivo de *log*. Porém, cabe ainda ao desenvolvedor escolher os pontos de monitoramento de sua aplicação e incluir no seu código chamadas de *API's*, para enviar as informações coletadas para o arquivo de *log*.

A MiddLog é uma infra-estrutura que estende a atual tecnologia de infra-estrutura de *log*, agregando novas funcionalidades que oferecem ao desenvolvedor um conjunto completo e flexível de serviços de *logging*. A MiddLog eleva as funcionalidades de gerenciamento de *log* ao nível de um serviço de middleware, permitindo, desta forma, que desenvolvedores de aplicações não tenham que se

preocupar com os problemas associados a geração de logs. Tendo como base da arquitetura as tecnologias de *middleware,* as camadas de serviços de *log* puderam ser implementadas por componentes, facilitando a sua instalação e execução em servidores de aplicação. Combinando a flexibilidade das tecnologias de desenvolvimento baseado em componentes com o uso das técnicas de *AOP*, foi possível a implementação de componentes, *interceptadores*, que fazem o monitoramento da aplicação e realizam dinamicamente a verificação dos *joinpoints*, através de arquivos de configuração. Através do experimento apresentado na Seção 4, verifica-se que essa combinação permite ao desenvolvedor incluir/excluir pontos de monitoramento na aplicação a qualquer momento sem a necessidade de reiniciar o servidor de aplicação, ou a MiddLog, e sem comprometer o tempo de execução da aplicação monitorada. Cabe ressaltar que, devido a camada de integração disponibilizada pela MiddLog, aplicações cliente desenvolvidas em qualquer tecnologia podem fazer uso das funcionalidades de infra-estrutura. Ainda, as características arquiteturais da MiddLog permitem que a mesma possa ser facilmente integrada a qualquer servidor de aplicação.

Novos trabalhos podem ser desenvolvidos explorando os recursos da infra-estrutura proposta, permitindo adequá-la aos novos paradigmas do desenvolvimento de sistemas, principalmente na técnica de interceptação do lado cliente e na extensão do modelo de dados para melhor adequá-la aos diversos tipos de domínios de aplicações.

Referências

AspectJ (2003), a Java implementation of AOP. Disponível em: http://www.parc.com/research/csl/projects/aspectj/default.html. Último acesso: Dezembro/2003.

AOSD (2004), Aspect-Oriented Software Development, Tools for practitioners. Disponível em: http://www.aosd.net/technology/practitioners.php. Último acesso: Dezembro/2004.

Bonchi, F., Giannotti, f., Gozzi, C., Manco, G., Nanni, M., Pedreschi, D., Renso, C., Ruggieri, S. (2001), Web Log data Warehousing and Mining for Intelligent Web Caching. Elsevier, Data & Knowledge Engineering 39, 165-189.

COM+ (2004). Disponível em: http://msdn.microsoft.com/library/en-us/dnanchor/html/complus_anchor.asp. Último acesso: Janeiro/2004.

Cruz, S. M. S., Campos, L. M., Campos, M. L. M., Pires, P. F. (2003), A Data Mart Approach for Monitoring Web Services Usage and Evaluating Quality of Services. Publicado nos Anais do 18º Simpósio Brasileiro de Banco de Dados (SBBD2003).

DCOM and CORBA Side by Side, Step by Step, and Layer by Layer de P. E. Chung, Y. Huang, S. Yajnik, D. Liang, J. C. Shih, C.-Y. Wang, and Y. M. Wang (1998). Disponível em: http://www.research.microsoft.com/~ymwang/papers/html/dcomncorba/s.html. Último acesso: Abril/2003.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, John Irwin (1997), Aspect-Oriented Programming. Published In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlarg LNCS 1241.

JBOSS (2004), Aspect Oriented Programming. Disponível em: http://www.jboss.org/products/aop. Último acesso: Novembro/2004.

JLog (2003), A Java Logging Framework. Disponível em: http://www.adtmag.com/java/articleold.asp?id=966&mon=8&yr=2000. Último acesso: Julho/2003.

JSDK Logging (2003), Sun Microsystems Inc. Disponível em: http://java.sun.com/ j2se/1.4/docs/guide/util/logging/index.html. Último acesso: Julho/2003.

Kimball, R.; Reeves,L.; Ross,M.; Thornthwaite,W (1998), The Data Warehouse Lifecycle Toolkit: expert methods for designing, developing, and deploying data warehouses. New York, John Wiley & Sons.

Kimball, R., Merz, R., (2000) The Data Webhouse Toolkit:Building the Web-Enabled Data Warehouse, 1 ed., New York, USA, John Wiley & Sons.

Log4j Project (2003). Disponível em: http://logging.apache.org/log4j/docs/index.html. Último acesso: Maio/2003.

LogKit (2003), Apache Software Foundation. Disponível em: http://avalon.apache.org/logkit. Último acesso: Maio/2003.

Lowy J. (2003), Contexts in .NET: Decouple Components by Injecting Custom Services into Your Object's Interception Chain. Disponível em: http://msdn.microsoft.com/msdnmag/issues/03/03/contextsinnet/default.aspx. Último acesso: Janeiro/2004.

Nate Sammons (2001), Protomatter Syslog Whitepaper. Disponível em: http://protomatter.sourceforge.net. Último acesso: Maio/2003.

OMG (2003), Object Managment Group - CORBA. Disponível em: http://www.omg.org. Último acesso: Maio/2003.

Ramnivas Laddad (2004), Simplify your logging with AspectJ. Disponível em: http://www.developer.com/java/other/article.php/10936_3109831_4. Último acesso em: Julho/2004.

RP Logging Framework (2003). Disponível em: http://www.richardsonpublications.com/ rplog/index.jsp. Último acesso: Maio/2003.

Shukla D., Fell S. and Sells C. (2002), AOP - Aspect-Oriented Programming Enables Better Code Encapsulation and Reuse. Disponível em: http://msdn.microsoft.com/msdnmag/issues/02/03/AOP/default.aspx. Último acesso em: Dezembro/2003.

Sun MicroSystem (2004), Java 2 Platform Enterprise Edition. Disponível em: http://java.sun.com/j2ee/1.4/docs/index.html. Último acesso: Junho/2004.

World Wide Web Consortium (W3C) – XML (2004). Disponível em: http://www.w3.org/TR/2004/REC-xml-20040204/. Último acesso: Outubro/2004.

Word Wide Web Consortium (W3C). XML Schema (2004). Disponível em: http://www.w3.org/xm/schema. Último acesso: Outubro/2004.