UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO
MESTRADO EM SISTEMAS E COMPUTAÇÃO

IVO KENJI KOGA

UM ARCABOUCO PARA APLICAÇÕES DE ACESSO A SERVIÇOS DE MONITORAMENTO MULTI-DOMÍNIO

Salvador
2007

IVO KENJI KOGA

UM ARCABOUCO PARA APLICAÇÕES DE ACESSO A SERVIÇOS DE MONITORAMENTO MULTI-DOMÍNIO

Dissertação apresentada ao curso de Mestrado Profissional em Sistemas e Computação, Universidade Salvador - UNIFACS, como requisito parcial para obtenção do grau de Mestre.

Orientador: Prof. Dr. José A. Suruagy Monteiro
Co-orientador: Prof. Msc. Leobino Sampaio

Salvador
2007

Ficha Catalográfica
(Elaborada pelo Sistema de Bibliotecas da Universidade Salvador - UNIFACS)

Koga, Ivo Kenji

Um arcabouço para aplicações de acesso a serviços de monitoramento multi-domínio. / Ivo Kenji Koga. – Salvador, 2007.

66 p. : il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS, como requisito parcial para a obtenção do grau de Mestre.
Orientador: Prof. Dr. José A. Suruagy Monteiro.
Co-orientador: Prof. Msc. Leobino Sampaio.

1. Monitoramento de Redes. I. Monteiro, José A. Suruagy, orient. II. Sampaio, Leobino, co-orient.III. Universidade Salvador – Unifacs. IV. Título.

CDD: 004

TERMO DE APROVAÇÃO

IVO KENJI KOGA

UM ARCABOUCO PARA APLICAÇÕES DE ACESSO A SERVIÇOS DE MONITORAMENTO MULTI-DOMÍNIO

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre , Universidade Salvador - UNIFACS, pela seguinte banca examinadora:

José A. Suruagy Monteiro - Orientador
Livre Docente e Phd em Computer Science pela University Of California Los Angeles
Universidade Salvador - UNIFACS

Carlos André Guimarães Ferraz
Phd em Computer Science pela University of Kent at Canterbury
Universidade Federal de Pernambuco (UFPE)

Manoel Gomes de Mendonça Neto
Phd em Computer Science pela Universidade de Maryland Em College Park
Universidade Salvador - UNIFACS

Salvador, 19 de dezembro de 2007

Dedico este trabalho a minha mãe, meu irmão e a
todos os que me apoiaram para que este
trabalho fosse desenvolvido.

AGRADECIMENTOS

Tenho muito a agradecer a todos que colaboraram para a realização deste trabalho. Foram inúmeras as pessoas que ao longo desta etapa me apoiaram e me ajudaram para que este trabalho fosse possível.

Agradeço inicialmente a Deus por tudo de bom que aconteceu em toda minha vida durante esses anos. Por ter me dado saúde, força e ter me cercado de pessoas que me ajudaram nessa minha trajetória.

A minha família, por ter me apoiado em tudo e sempre que precisei. Sem eles não teria conseguido chegar até aqui.

Agradecimentos ao meu orientador Prof. Suruagy por toda confiança, paciência, apoio, conselhos e orientações prestadas ao longo desta jornada.

Ao Prof. André Santanchè pela amizade e incentivo durante todos esses anos e por ter me apresentado ao mundo da pesquisa.

Ao Prof. Leobino Sampaio pelas oportunidades e ajuda nos aspectos tecnológicos.

Aos meus colegas Herbert e Rafael pela amizade e companheirismo.

A toda equipe do NUPERC que me forneceu a infra-estrutura que foi fundamental para que este trabalho fosse realizado.

À RNP pela oportunidade de trabalho contínuo ao longo destes anos.

Agradeço também a todos que não foram citados, mas que colaboraram para que este meu sonho se tornasse realidade.

A honra não consiste em não cair nunca,
mas em levantar cada vez que se cai.

— CONFÚCIO

RESUMO

Medidas de desempenho de redes que atravessam diferentes domínios são difíceis de serem obtidas por questões de políticas de uso, de segurança, etc. Por este motivo algumas redes nacionais fizeram esforços no sentido de possibilitar o acesso a estes tipos de medidas, dentre elas a Internet2, Géant2 e RNP. Diante destes esforços, foi criado um documento que definiu e possibilitou a criação de um protótipo de um ambiente de monitoração multi-domínio denominado perfSONAR. Este protótipo forneceu as prerrogativas necessárias para que houvesse uma maior facilidade de acesso às medidas de desempenho de redes de diferentes domínios administrativos. Para possibilitar a utilização das medidas disponibilizadas, diversas ferramentas de acesso e visualização foram adaptadas ou construídas para acessar o perfSONAR. Apesar disto, poucas destas ferramentas oferecem recursos de adaptação e reuso no seu desenvolvimento, e as que ofereceram o fizeram sem a utilização de um padrão bem aceito de reuso de software, o que dificulta a sua adaptação ao longo do tempo e reutilização do código desenvolvido. É de longa data a existência do desenvolvimento baseado em componentes de software. Através deles é possível o desenvolvimento de "peças de software"' que podem ser encaixadas e compostas de diversas maneiras. Ainda assim, este evolui para uma abordagem distribuída o que facilitou sua distribuição e implantação em um ambiente de rede distribuído. Diversos são os modelos de componentes que seguem esta linha dentre eles: EJB, CORBA, COM e OSGi. Este trabalho propõe um arcabouço baseado em componentes OSGi para permitir adaptação dinâmica e reuso mais facilitado em ferramentas de acesso a medidas de desempenho de redes. Ele está sendo utilizado na ferramenta de visualização de dados da rede da RNP denominada *Internet Computer network Eye* (ICE), desenvolvida pelo Grupo de Trabalho de Medições (GT-Medições) da RNP e também no Ambiente de Gerência de Vídeo desenvolvido pelo Grupo de Trabalho de Gerência de Vídeo (GTGV) também da RNP.

Palavras-chave: Medições em Redes. Ambientes de Monitoramento Multi-Domínio. PerfSONAR. Componentes de Software. OSGi.

ABSTRACT

Network performance measurements that cross different administrative domains are difficult to be obtained, due to use policy issues, security, etc. For these reasons, some national networks started efforts to allow the access to these types of measures, among them the Internet2, Géant2 and RNP. Given these efforts, it was created a document that defined and enabled the development of a prototype of a multi-domain network monitoring environment called perfSONAR. This prototype provided the necessary prerogatives to a broader access to network performance measurements of different administrative domains. To allow the use of the available measurements, many tools for access and visualization of network measurements were adapted or built to access perfSONAR data. However, few of these tools offer adaptability and reuse resources in their development, and those which did, made it without the use of a well accepted software reuse standard, that hamper the adaptation along the time and developed code reuse. Component-based software development is a well known discipline for a long time. It is based on the development of pieces of software that can be put together and composed to build applications. This discipline evolved to a distributed approach, which facilitates their distribution and deployment in a distributed network environment. There are many component models which follow this idea, like: EJB, CORBA, COM and OSGi. This work proposes a framework based on OSGi components to allow dynamic adaptation and easily reuse in network performance measurement access tools. It is being used in a network data visualization tool from RNP called Internet Computer network Eye (ICE), developed by the Measurement Working Group (GT-Medições) from RNP and in a video management environment developed by the Video Management Working Group (GT-GV), also from RNP.

Keywords: Network Measurement. Multi-domain monitoring environments. PerfSONAR. Software Components. OSGi.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Diversos utensílios do nosso dia-a-dia possuem indicadores que nos mostram o seu estado. Os automóveis, por exemplo, possuem dispositivos que indicam a velocidade, quantidade de combustível, distância percorrida, a rotação do motor, temperatura, nível de óleo, entre outros. Todos estes informam o estado atual e algumas outras informações que remetem ao estado passado do automóvel. Estas informações servem para indicar o quanto ainda é possível percorrer, qual a distância percorrida, além de problemas como o fato da temperatura do motor atingir um certo limiar indicando que o sistema de resfriamento está com defeito.

Com este mesmo intuito as redes de computadores são monitoradas para conhecer qual é a sua capacidade, quais são os seus tipos de aplicativos e quando eles são executados, o atraso que se obtém, além de conhecer e identificar os problemas que se sofre, etc. Esta atividade de monitoramento deve fornecer informações da rede de forma a beneficiar todos os seus usuários. Para isto deve tornar disponíveis os dados de desempenho de redes de uma forma fácil e completa, ou seja, incluindo informações de todos os nós intermediários do caminho entre os usuários.

Uma das dificuldades existentes nesse sentido é o acesso às medidas de desempenho fim-a-fim que atravessam diferentes domínios administrativos. Isso ocorre porque cada domínio administrativo possui diferentes políticas de medições, acesso e segurança com relação aos dados de suas redes, entre outras questões. Para resolver esse problema, alguns esforços foram iniciados com o objetivo de construir infra-estruturas de medição capazes de fornecer acesso padrão a medidas de desempenho de redes sem violar as políticas internas de cada domínio. Alguns desses esforços são a Iniciativa de Desempenho Fim a Fim (*End to End performance initiative* --- E2Epi) (INTERNET2, 2007a) da Internet2, a Atividade 1 de Pesquisa Conjunta da Géant2 (*Géant 2 Joint Research Activity 1* --- JRA1) (GÉANT2, 2007) e o Grupo de Trabalho de Medições (GT-Medições) (MONTEIRO, 2005) da Rede Nacional de Ensino e Pesquisa (RNP) (RNP, 2007).

A Internet2 propôs e iniciou o desenvolvimento em 2003 de um sistema escalável e distribuído para monitoramento e testes de desempenho fim a fim, chamado piPEs (*E2Epi Performance Evaluation system*) (INTERNET2, 2003). Ele tinha o objetivo de permitir aos operadores e usuários finais verificar o desempenho da rede bem como identificar problemas e encontrar os responsáveis pela resolução dos mesmos no caminho fim a fim da rede. Além

disso, tinha intenção de permitir o disparo de testes remotos e também ser interoperável com outros ambientes de medição de desempenho.

Como o piPEs se encontrava em desenvolvimento e certamente não estaria completamente implementado até a implantação do piloto do GT-Medições da RNP, foi proposto um ambiente baseado, mas não dependente do piPEs, chamado de piPEs-BR (SAMPAIO e outros, 2006). Esse ambiente é um sistema baseado nas idéias do piPEs e que tem a intenção de incorporar novas funcionalidades sob demanda, através da integração de novas ferramentas que viessem a ser desenvolvidas.

Para a gerência do ambiente foi desenvolvida uma Interface de Administração e Gerência do piPEs-BR (PAMI) que adotava utilitários do RRDtool (RRDTOOL, 2007) para gerar gráficos das medidas e permitia o controle dos pontos de medição cadastrados. Além dessa iniciativa, foram desenvolvidas outras interfaces que permitiam o acesso ao piPEs-BR como, por exemplo, a interface criada com a utilização da técnica de Mapas em árvore denominada Treemaps RNP e a utilização da interface gráfica com o usuário da ferramenta MonALISA (*Monitoring Agents in A Large Integrated Services Architecture*) (LEGRAND e outros, 2004).

Após a iniciativa do piPEs-BR, em meados de 2005, a Internet2 e a Géant2 iniciaram um trabalho conjunto em que foi elaborado um documento que especificou o projeto de uma arquitetura para uma infra-estrutura de medição orientada a serviços que ficou conhecido como *General Framework Design* (GFD) (GEANT2, 2005). O principal objetivo desse documento foi formalizar os conceitos necessários para a execução de medições entre diferentes domínios de rede, tornando possível a troca de informações de medição através de serviços padronizados.

A arquitetura definida no GFD prevê uma camada de serviços que abstrai a forma como as medições são realizadas em cada domínio de rede, possibilitando aos clientes um acesso a estes serviços de forma padronizada. Uma visão geral do escopo da infra-estrutura proposta pode ser vista na Figura 1, em que pontos de medições avaliam determinados domínios, exportam ou são acessados por serviços de monitoramento que fornecem funcionalidades podendo ser acessadas pelos clientes de visualização do(s) usuário(s).

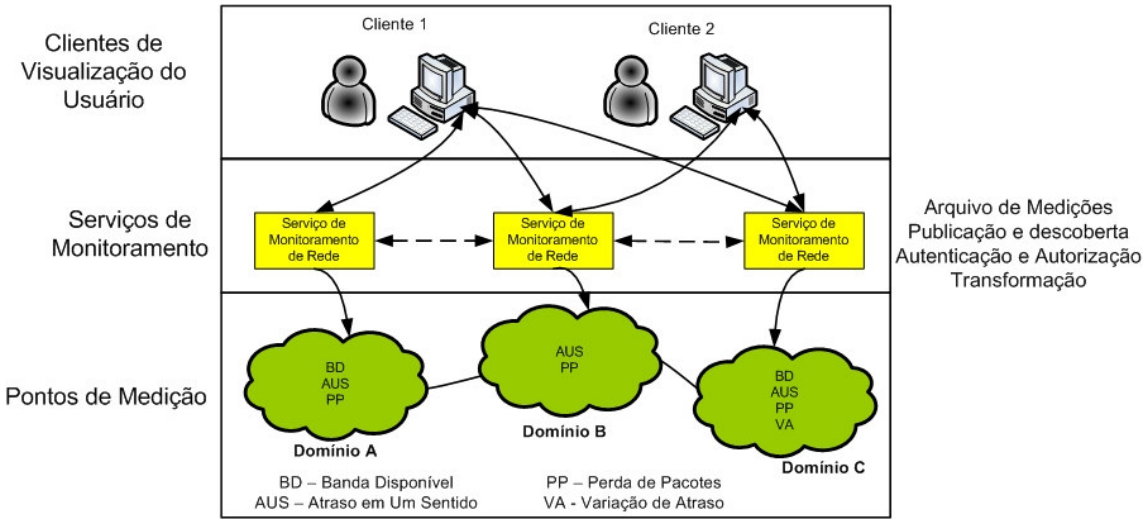

Figura 1 - Arquitetura proposta no General Framework Design (GFD)

Fonte: GÉANT2 (2005)

Os serviços definidos no GFD são:

a) ponto de medição (MP — *Measurement Point*) — usado para coletar dados de medição;
b) serviço de transformação (TS — *Transformation Service*) — usado para encaminhar e transformar dados dos serviços da infra-estrutura;
c) arquivo de medições (MA — *Measurement Archive*) — armazena dados de medi¸ao coletados pelos MPs ou transformados pelo TS;
d) serviço de descoberta (LS — *Lookup Service*) — usado para descobrir e publicar serviços, permitindo a inserção e consulta dos serviços do ambiente;
e) serviço de autenticação (AS — *Authentication Service*) — permite a autenticação e autorização no ambiente e fornece atributos de decisão para o quê pode ser acessado de um determinado recurso ou serviço;
f) serviço de topologia (TopS — *Topology Service*) — fornece informação sobre a topologia da rede.

O desenvolvimento de um protótipo batizado de perfSONAR (*Performance focused Service Oriented Network monitoring ARchitecture*) (HANEMANN e outros, 2005) foi iniciado seguindo as definições do GFD e uma abordagem consistente que considera os aspectos da organização multi-domínio da rede, além dos requisitos dos usuários (BOOTE e outros, 2005). Ele também serve para a validação e teste do GFD e teve como objetivo inicial

o desenvolvimento dos serviços básicos e mais simples do GFD tais como o MP, o MA e o LS.

Com o advento do perfSONAR houve a motivação para que redes nacionais criassem ou adaptassem suas ferramentas de acesso às medidas de desempenho de redes para acessar os serviços de monitoramento deste ambiente. Um destes exemplos é a adaptação do *Customer Network Management* (CNM) (CNM, 2006) desenvolvido pela Rede Nacional de Ensino e Pesquisa Alemã (DFN) (DFN, 2006), que tinha a intenção de fornecer às universidades e instituições de pesquisa alemãs informações sobre seu *backbone* de pesquisa nacional (HANEMANN e outros, 2006) e está sendo adaptado desde o início do sub-projeto JRA1 em Setembro de 2004, para acessar os serviços disponibilizados pelo perfSONAR.

Um outro exemplo de adaptação é o *Network Monitor System* (NEMO) (NEMO, 2006) que é um sistema utilizado pelas redes nacional de ensino e pesquisa da Noruega – UNINETT (UNINETT, 2006) e da Suécia – *Swedish University Computer Network* (SUNET) (SUNET, 2006). Ele tem como objetivo o fornecimento de visualizações para os usuários mostrando como eles estão conectados à rede e suas estatísticas relativas à qualidade de serviço (QoS) (HANEMANN e outros, 2006).

Algumas ferramentas também foram desenvolvidas como a VisualperfSONAR (VISUALPERFSONAR, 2006) que teve sua primeira versão em janeiro de 2006 e é uma aplicação *Web* que possibilita ao usuário visualizar as estatísticas de todas as interfaces de um dado caminho que são recuperadas através do fornecimento da saída do comando de acompanhamento de rota (*traceroute*). Ao usuário é apresentado um mapa que utiliza o serviço do Google maps (GOOGLE, 2007) com a visualização do caminho e os dados de utilização das interfaces recuperados através da requisição aos MAs disponíveis.

O perfSONARUI (JELIAZKOVA; ILIEV; JELIAZKOV, 2006) é outro exemplo de ferramenta que começou a ser desenvolvida no final de 2005 pelo reconhecimento da necessidade de uma interface gráfica *stand-alone*, livre, acessível publicamente, fácil de utilizar, que possibilitasse o acesso aos diversos serviços disponíveis no perfSONAR (HANEMANN e outros, 2006).

Apesar de todas estas aplicações sofrerem adaptações para acessar os serviços do perfSONAR, a única que está utilizando uma abordagem que possibilita o desenvolvimento reusável de código é a perfSONARUI. Esta possibilita atualmente o desenvolvimento de *plugins* que podem ser acoplados a ela através do desenvolvimento padronizado e específico dela. Desta forma, estes *plugins* não podem ser reutilizados em outras ferramentas, o que limita esta abordagem.

Os esforços de desenvolvimento de ambientes de monitoramento multi-domínio tiveram o objetivo de fornecer dados de monitoramento de redes para os usuários. A infra-estrutura dos ambientes construídos já fornece uma camada de serviços que usa a arquitetura orientada a serviços (*Service Oriented Architecture* – SOA) (PAPAZOGLOU, 2003) que encapsula as características internas de cada componente. Isto é, abstrai a forma de interação e disponibilidade dos dados fornecidos. Porém, a adaptação das aplicações de acesso às medidas de desempenho de redes é realizada à medida que os serviços fornecidos por estes ambientes se tornam disponíveis, o que dificulta o planejamento no desenvolvimento destes aplicativos.

Assim, a cada mudança nos parâmetros de acesso ou a cada serviço desenvolvido, as aplicações existentes nas redes nacionais de ensino e pesquisa (NRENs) têm que ser modificadas para continuar acessando os serviços. Portanto, existe uma carência de uma abordagem que facilite a adaptação destas ferramentas de acesso. Além do mais, as visualizações que estas aplicações fornecem devem ser passíveis de reestruturação, adaptações para diferentes públicos e pode ser adequado adicionar ou retirar certos detalhes, além de unir diferentes métricas de redes na mesma visualização. Desta forma, é importante permitir o desenvolvimento de novas visualizações dentro destas aplicações. Estas visualizações devem possuir meios de adaptação facilitados e que permitam o seu uso em outras aplicações.

Já é conhecido que existem tecnologias que fornecem suporte ao desenvolvimento reusável de software como o uso de componentes de software (SZYPERSKI; GRUNTZ; MURER, 2002) e *frameworks* (JOHNSON, 1997). Neste sentido a maior motivação deste trabalho é a possibilidade de reuso e adaptação dinâmica de aplicações de visualização a dados de monitoramento de redes.

Em função deste questionamento, são levantadas as seguintes hipóteses:

a) a utilização de uma abordagem de componentes de software é adequada para o desenvolvimento de software reusável de acesso às medidas de desempenho de redes;
b) um arcabouço para o desenvolvimento de componentes de software flexíveis e extensíveis é suficiente para melhorar a qualidade e possibilitar a reutilização de software em aplicações de acesso aos serviços do perfSONAR.

O objetivo principal deste trabalho foi, portanto, construir um arcabouço que facilitasse a adaptação dinâmica e reuso em aplicações de acesso aos serviços de monitoramento do perfSONAR.

Este arcabouço denominado FLAVOR (KOGA e outros, 2007b) é constituído de interfaces Java e *bundles* OSGi (MARPLES; KRIENS, 2001) que permitem o reuso e a adaptação dinâmica em aplicações que desejam utilizar os dados de monitoramento do perfSONAR. Ele foi utilizado em dois casos de uso: em uma ferramenta de visualização de dados de desempenho de redes denominada ICE e no ambiente de gerência de vídeo da RNP que serão descritos no capítulo cinco desta dissertação.

O restante desta dissertação está organizada da forma apresentada a seguir.

No capítulo dois, **Acesso a dados e Reuso em Monitoramento de Redes**, são apresentados os trabalhos que tem relação com esta iniciativa.

No capítulo três, ***Componentes e Frameworks***, são apresentados os conceitos de componentes de software e componentes de software distribuídos e algumas tecnologias baseadas nestes conceitos.

No capítulo quatro é apresentado o **FLAVOR** (***Framework Layer for Access and Visualization Of network measurement Resources***), que é a principal contribuição deste trabalho, e os detalhes de sua implementação.

No capítulo cinco, **Aplicações do FLAVOR**, são apresentados dois cenários de uso do FLAVOR, sendo o primeiro ocorrido como parte do desenvolvimento da ferramenta de visualização desenvolvida pelo GT-Medições, denominada ICE (*Internet Computer network Eye*), e o segundo ocorreu na integração entre os grupos de trabalho da RNP: Medições e Gerência de Vídeo onde foi utilizado o FLAVOR para acesso a medidas disponibilizadas pelos serviços do perfSONAR em um ambiente implantado na RNP.

Finalmente, o capítulo seis, **Conclusões**, apresenta as conclusões do trabalho, bem como os resultados obtidos e perspectivas de trabalhos futuros.

2 ACESSO A DADOS E REUSO EM MONITORAMENTO DE REDES

O acesso a dados de monitoramento de redes se tornou mais facilitado através do advento de infra-estruturas como a do perfSONAR. Também o reuso é abordado em algumas ferramentas de visualização de dados de monitoração de redes como poderá ser visto a seguir.

2.1 PERFSONAR

O desenvolvimento do perfSONAR permitiu a troca de dados de monitoração entre redes através de serviços padronizados, tornando mais fácil a resolução de problemas de desempenho fim-a-fim. Ele contém um conjunto de serviços que disponibilizam medições agindo como uma camada intermediária entre as ferramentas de medição e as aplicações de visualização. Essa camada tem o objetivo de possibilitar a troca de informações entre redes, usando mensagens padronizadas.

Para isto, os serviços do perfSONAR expõem suas funcionalidades através de Serviços Web que podem ser acessados via mensagens XML padronizadas pelo *Network Measurement Working Group* (NMWG) (GGF, 2005) do Global Grid Forum (GGF).

Essas mensagens padronizadas são divididas em duas partes chamadas de: *metadata* e *data*, como pode ser visto na Figura 2. O *metadata* descreve o tipo de dado e os parâmetros necessários para sua recuperação, enquanto o *data* é a parte que representa os dados retornados, ou seja, os resultados de medição em conjunto com o tempo que a medição foi realizada. Em uma requisição a um determinado serviço, somente o *metadada* é preenchido, sendo o *data* preenchido na resposta do serviço que fornece os dados de medição.

Figura 2 - Mensagem NMWG
Fonte: PERFSONAR (2007*)*

Essa mensagem, no entanto, é encapsulada em uma mensagem SOAP que é empacotada em uma mensagem HTTP e assim por diante. Essa cadeia de empacotamento de protocolos é mostrada na Figura 3.

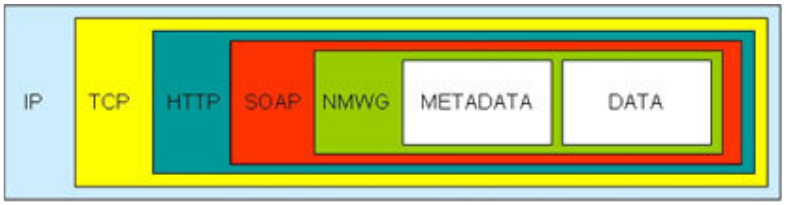

Figura 3 - Cadeia de protocolos que empacotam uma mensagem de acesso a serviços do perfSONAR

Fonte: PERFSONAR (2007)

Um exemplo de mensagem pode ser visto na Figura 4 que descreve a mensagem de requisição de dados de medição ao serviço de *Ping* do *Command Line Measurement Point* (CL MP) (MONTEIRO, 2006b). Este serviço é uma das implementações disponíveis de um MP do perfSONAR.

```
<?xml version='1.0' encoding='UTF-8'?>

<nmwg:message type="MeasurementRequest"
    id="msg1"
    xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/"
    xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/"
    xmlns:ping="http://ggf.org/ns/nmwg/tools/ping/2.0/">

<nmwg:metadata id="meta1">
    <ping:subject id="sub1">
        <nmwgt:endPointPair>
            <nmwgt:src type="ipv4" value="200.237.192.73"/>
            <nmwgt:dst type="ipv4" value="200.237.192.254"/>
        </nmwgt:endPointPair>
    </ping:subject>

    <nmwg:eventType>ping</nmwg:eventType>

    <ping:parameters id="param1">
        <nmwg:parameter name="count">10</nmwg:parameter>
        <nmwg:parameter name="interval">1</nmwg:parameter>
        <nmwg:parameter name="packetSize">56</nmwg:parameter>
        <nmwg:parameter name="ttl">10</nmwg:parameter>
    </ping:parameters>
</nmwg:metadata>

<nmwg:data id="1" metadataIdRef="meta1" />

</nmwg:message>
```

Figura 4 - Exemplo de Mensagem de requisição de dados de medição ao serviço de Ping do CLMP

Nas linhas de 3 a 7 são definidos o cabeçalho da mensagem, os *namespaces* do NMWG, NMWG *Topology* (NMWGT) e PING além do tipo da requisição *MeasurementRequest* (requisição de medição) e seu identificador (id). Após esta definição, o *metadata* é inserido nas linhas 9 a 25, onde são descritos o par de pontos origem e destino da medição, o tipo de evento (*eventType*) e os parâmetros de requisição específicos do serviço como intervalo (*interval*), contador (*count*), dentre outros. Na linha 27 é inserido o campo data vazio, já que é uma mensagem de requisição e este será preenchido na resposta do serviço.

Para cada tipo de serviço são definidas mensagens NMWG, ou seja, toda mensagem de interação com os serviços do perfSONAR deve se submeter à padronização NMWG. Atualmente estão definidas mensagens para os serviços disponíveis, tais como:

a) RRDMA – arquivo de medições de base de dados Round-Robin (RRD);
b) SQLMA – arquivo de medições que disponibiliza os serviços de uma base de dados relacional;
c) *Lookup Service* (LS) – serviço de publicação e descoberta de serviços do perfSONAR;
d) *SSH/Telnet MP* – ponto de medição que permite a execução de comandos via SSH/Telnet em dispositivos remotos;
e) BWCTL MP – ponto de medição que permite o acesso às medidas do BWCTL (INTERNET2, 2007);
f) *E2Emon MP* – ponto de medição de acesso aos dados de monitoramento do sistema de monitoramento E2E;
g) CL MP – *Command Line Measurement Point* – ponto de medição que realiza medições mediante o acesso pelo usuário às ferramentas que funcionam no modo linha de comando.

Ainda existem outros serviços em definição e outros que ainda poderão vir a surgir. A cada novo serviço é necessário produzir uma nova mensagem. Ou seja, para cada serviço disponibilizado, há um esforço na definição das mensagens de requisição e respostas a este serviço.

Sendo assim, um cliente de visualização que queira requisitar dados deste novo serviço deve implementar suporte a estas novas mensagens XML padronizadas. A Figura 5 ilustra como é realizado o suporte de uma ferramenta de visualização a um determinado

serviço, onde cada ferramenta deve implementar o código de requisição e resposta do serviço de acordo com as mensagens definidas.

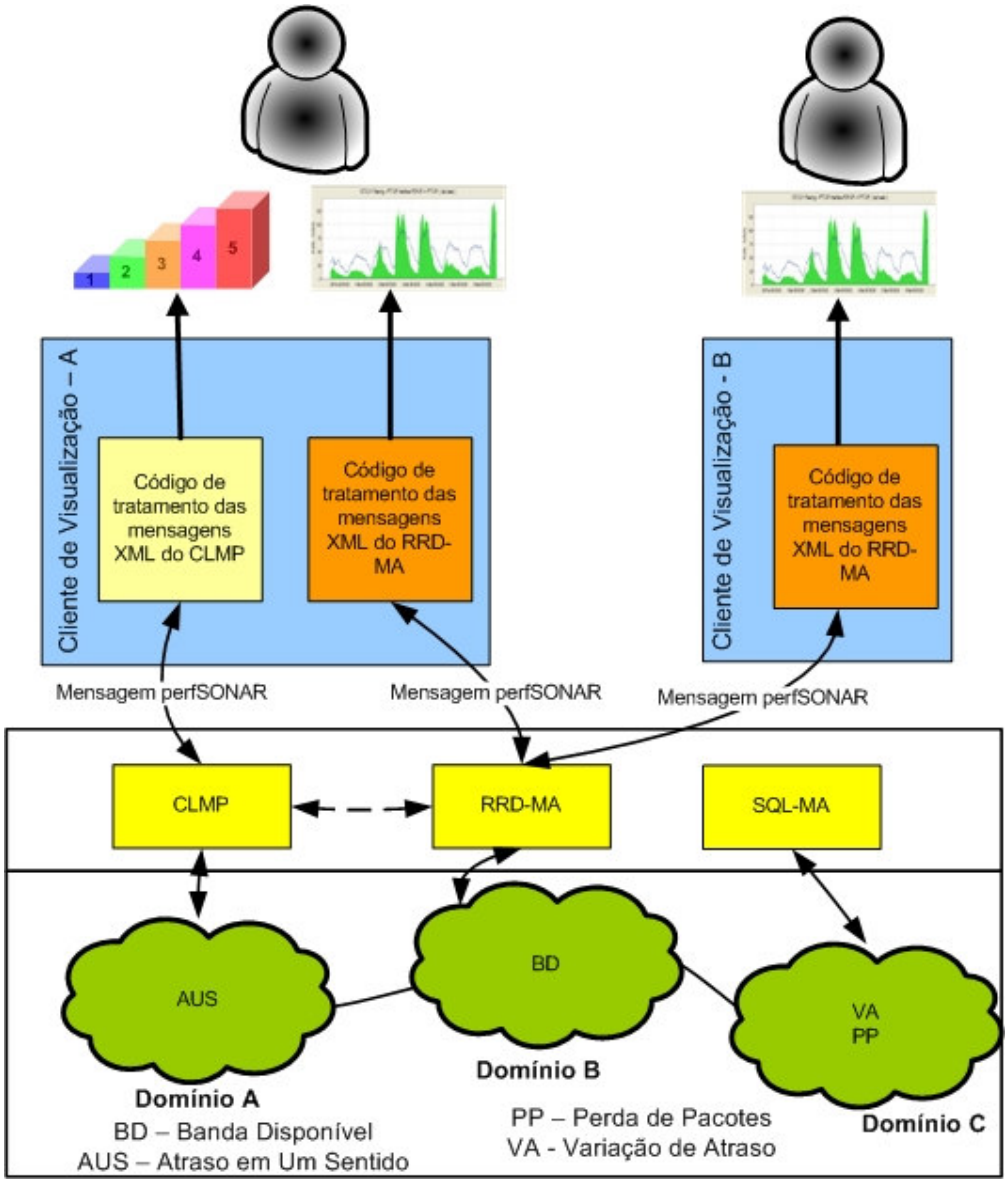

Figura 5 - Acesso aos serviços do perfSONAR através das mensagens NMWG padronizadas

Caso haja alguma alteração nas mensagens deste serviço seja por inclusão de um parâmetro ou alteração no formato, também será necessária uma redefinição no código de acesso de todas as ferramentas que já possuem suporte a este serviço.

Este efeito pode ser minimizado com a reutilização do código de acesso aos serviços. Uma vez que as mensagens são padronizadas, uma ferramenta de acesso poderá reutilizar o código de tratamento das mensagens de outra. Desta forma não seria necessária a

implementação de código de tratamento de mensagens de um determinado serviço que já estaria implementado em outra ferramenta.

Assim, no momento em que uma mensagem for modificada e uma ferramenta necessitar acessar um serviço com este novo formato de mensagem, bastaria reutilizar dinamicamente o código existente em outra ferramenta para realizar este acesso.

2.2 PERFSONARUI

O perfSONARUI (JELIAZKOVA; ILIEV; JELIAZKOV, 2006) é um dos clientes que iniciaram seu desenvolvimento após a criação do perfSONAR. Tinha como objetivo inicial o preenchimento de certos requisitos de acesso que as outras ferramentas não possuíam como o acesso ao RRDMA.

Atualmente é a única iniciativa dentro do perfSONAR que se preocupa com um desenvolvimento que visa a reutilização de código. Porém, não utiliza nenhum padrão ou modelo de reutilização amplamente aceito, o que dificulta a reutilização de seu código com outras ferramentas de acesso ou em outros contextos.

Neste caso, ele possui um módulo que permite o desenvolvimento de código que pode ser acoplado na sua interface gráfica. Desta forma permite o reuso de partes da sua interface gráfica e do código de requisição e resposta já produzidos, podendo assim ser desenvolvido um novo código para acessar um outro serviço a partir dos que já existem. Porém, este código desenvolvido não pode ser facilmente reutilizado em outras ferramentas, já que não segue nenhum padrão pré-definido.

É possível concluir neste caso que a reutilização no perfSONARUI se dá com o objetivo de facilitar o desenvolvimento de acesso para os diversos serviços oferecidos pelo perfSONAR, porém não tem a intenção de facilitar a difusão, facilidade de acesso e adaptação em outras possíveis ferramentas que possam vir a necessitar de acesso às medidas de desempenho oferecidas pelo perfSONAR.

2.3 CACTI

A ferramenta Cacti (CACTI, 2007) é uma interface gráfica para o RRDTool que utiliza PHP e base de dados MySQL para criar gráficos. Ela trata a recuperação de dados e apresentação além de possuir suporte a consultas de informações através do protocolo SNMP.

Possibilita a expansão através de *plugins* que lhe adicionam novas funcionalidades. Alguns exemplos são o PHP *Network Weathermap* (WEATHERMAP, 2007) que mostra um mapa da rede e o estado de cada elemento da rede e o *FlowView* (VIEW, 2007) que permite visualizar relatórios de dados gerados pelo *Netflow* (CISCO, 2003).

Estes *plugins* podem ser instalados através do download do arquivo do *plugin*, extração do arquivo para um diretório temporário, cópia para o diretório de *plugins* do Cacti e alteração do arquivo de configuração do Cacti de forma que possa ser localizado o novo *plugin*. Além disso, podem ser necessárias algumas configurações específicas para cada *plugin* como, por exemplo, adicionar dados na base de dados do Cacti, sendo necessário ler a documentação do *plugin* para configuração destes requisitos.

O Cacti é um exemplo que permite a adaptação através de *plugins* que podem ser adicionados à sua estrutura. Porém, são *plugins* que funcionam como a abordagem do perfSONARUI e só funcionam para esta estrutura do Cacti, não permitindo o reuso do código por outras ferramentas.

Se um *plugin* for desenvolvido para o Cacti, somente poderá ser utilizado nele, não podendo ser utilizado em outras ferramentas, mesmo que sejam desenvolvidas com as mesmas tecnologias do Cacti. Ou seja, linguagem Web PHP, base de dados MySQL etc.

2.4 OPENVIEW

O OpenView é um produto da *Hewlett Packard* (HP) consistindo num portfólio de produtos para redes e sistemas. É mais comumente descrito como um conjunto de aplicações que permite o gerenciamento em larga escala de sistemas e redes das organizações. Ele possui diversos módulos opcionais assim como milhares de módulos de terceiros que conectam com um *framework* bem definido e podem comunicar entre si.

O Network Node Manager (NNM) é um destes módulos do OpenView que permite gerenciar redes. Ele usa SNMP para se comunicar com dispositivos de rede, permitindo a eles serem auto-descobertos, monitorados e controlados. O NNM determina e demonstra conectividade lógica e física em redes, assim como informações de protocolos executando na rede. Também permite que dados históricos possam ser coletados e visualizados.

Entretanto ele não define um padrão aberto para definição de componentes, apesar de implementar interfaces gráficas extensíveis.

2.5 MONALISA

MonALISA é um framework que fornece um serviço de monitoramento distribuído usando JINI, WSDL e SOAP. Ele é baseado na arquitetura de serviço dinâmica e distribuída (DDSA) (NEWMAN; LEGRAND; BUNN, 2001) e é composto por duas estruturas: o serviço MonALISA e o cliente MonALISA em que o primeiro é responsável por coletar dados e o segundo por emitir relatórios e gráficos dos dados coletados. O MonALISA permite a inserção de novas fontes de dados para visualização pelo desenvolvimento de módulos para acessar estes dados.

O cliente MonALISA contém uma visualização 3D que usa coordenadas geográficas para desenhar a localização dos fornecedores de dados. O serviço MonALISA também pode usar Serviços Web disponíveis para recuperar dados e plotar gráficos. Entretanto, ele não possui o código fonte aberto e não permite a inserção de novas funcionalidades, como por exemplo, funcionalidades para gerência de redes ou detecção de anomalias.

Deste modo, ela permite a disponibilização de dados para acesso no cliente MonALISA através de serviços Web, porém não possibilita a inserção de novas visualizações deste cliente e nem mesmo o reuso do seu código para outros fins por ser de código fonte fechado.

2.6 CONSIDERAÇÕES FINAIS

Percebe-se que as diversas aplicações apresentadas possuem algum recurso que facilite o desenvolvimento de novas funcionalidades:

a) perfSONARUI – código plugável em sua interface gráfica;

b) Cacti – plugins adicionáveis à sua estrutura;

c) OpenView – módulos conectáveis ao seu framework;

d) MonALISA – utilização de seus serviços para publicação de dados e visualização em seu cliente.

Todos estes recursos têm o objetivo de facilitar o desenvolvimento de novas funcionalidades, ou seja, possibilitar de certa forma a adaptabilidade da aplicação. Porém elas tanto não utilizam uma forma padronizada de reutilização de software que permita o desenvolvimento colaborativo de novas funcionalidades, quanto não possibilitam o compartilhamento de suas funcionalidades com outras ferramentas.

O próximo capítulo descreve os modelos componentes de software distribuídos que possibilitam o desenvolvimento reusável e flexível de software. A utilização deste modelo de desenvolvimento facilita a criação, reutilização e distribuição de software entre aplicações distintas.

3 COMPONENTES

Neste capítulo é feita uma introdução a componentes de software e componentes de software distribuídos bem como algumas tecnologias de componentes de software: EJB, CORBA, COM+ e OSGi.

Também é introduzido o conceito de *framework* a ser utilizado neste trabalho e as suas principais características.

3.1 COMPONENTES DE SOFTWARE

A abordagem de desenvolvimento de componentes é anterior à utilização da mesma em computação. As engenharias já faziam uso da padronização na forma de componentes passíveis de serem encaixadas para a construção de produtos finais como aparelhos elétricos, automóveis, edifícios, etc.

A concepção de desenvolvimento de componentes de software também não é nova (CRNKOVIC, 2001). Esta concepção tem sido usada por muitos anos. Apesar disto, não há uma definição amplamente aceita, provavelmente pelo fato do termo ser usado de forma diversa para descrever várias coisas diferentes. Entretanto, na maior parte dos casos as definições são bem parecidas com diferenças de ênfase nas interfaces ou empacotamento físico (HOPKINS, 2000).

D'Souza descreve que componentes são pacotes que contêm artefatos de *software* que podem ser desenvolvidos de forma independente e entregues como uma unidade que pode ser unida, sem modificação, a outros componentes para construir algo maior (D'SOUZA; WILLS, 1999). Já Szyperski diz que um componente de *software* é uma unidade de composição com interfaces especificadas contratualmente e com dependências de contexto explícitas, podendo ser implantado independentemente e estar sujeito a composição por terceiros (SZYPERSKI; GRUNTZ; MURER, 2002).

Componentes, portanto, são peças de software que possuem as seguintes características (OLSEN, 2006):

a) múltiplo uso;

b) sem contexto específico;

c) possibilidade de composição com outros componentes;

d) encapsulado, ou seja, entidades externas não possuem acesso aos seus detalhes internos ou de implementação;

e) unidade de implantação e versionamento independente.

Assim como as peças pré-fabricadas utilizadas pela engenharia, os componentes de software possuem características de reusabilidade, facilidade de distribuição, facilidade de configuração e combinação.

A possibilidade de reuso de um componente está diretamente relacionada com a amplitude de difusão do modelo de componentes adotado em sua implementação. Existe um grande número de modelos de componentes mas os dois mais amplamente adotados são: o *Component Object Model*, da Microsoft e o Java *Beans*, da Sun (EMMERICH, 2002).

Estes modelos têm em comum o fato de suas execuções estarem confinadas ao escopo de uma única máquina, não provendo mecanismos para implantação e comunicação com outros componentes que estejam em máquinas distintas.

3.2 COMPONENTES DE SOFTWARE DISTRIBUÍDOS

A programação orientada a objetos criou uma distinção clara entre interface e implementação, o que permite que objetos possam se comunicar utilizando uma interface padronizada, independente da implementação da mesma. Isso permitiu que objetos pudessem se comunicar mesmo que sejam executados em máquinas diferentes, com sistemas operacionais diferentes e implementados em linguagens de programação diferentes.

Percebendo a limitação de objetos locais confinados em uma única máquina, a *Object Management Group* (OMG) (OMG, 2007) definiu o CORBA que viabiliza a comunicação distribuída de objetos heterogêneos (EMMERICH, 2002).

O CORBA fornece uma infra-estrutura de objetos distribuídos que é independente de linguagem e com serviços de localização de objetos, gerenciamento do estado dos objetos num armazenamento persistente, transação distribuída, controle de segurança e comunicação entre objetos.

Apesar disto, no contexto dos objetos distribuídos, as dificuldades de reutilização de software são ainda maiores, não apenas pela heterogeneidade de linguagens e sistemas, mas principalmente pelas dependências dos serviços proporcionados pelo CORBA ou plataforma equivalente, o que dificulta a realocação de um subconjunto de objetos para um contexto distinto. Isto motivou a elaboração de um modelo para a produção de componentes que operam em um ambiente distribuído, que combina características dos componentes de software e dos objetos distribuídos.

Este modelo fornece uma unidade de programação onde o projetista do componente se preocupa somente com as regras de negócio da aplicação, sendo o resto da implementação como localização de componentes, persistência, transação e segurança fornecidos pelos *containers* que são disponibilizados em servidores de aplicações que controlam o ciclo de vida destes componentes.

Existem atualmente quatro principais modelos de componentes distribuídos: CORBA *Component Model* (CCM), da OMG, *Enterprise Java Beans* (EJB), da Sun, *Component Object Model* (COM+), da Microsoft e *Open Services Gateway Initiative* (OSGi), da OSGi Alliance.

3.2.1 Enterprise Java Beans (EJB)

Enterprise Java Beans (DEMICHIEL, 2003) define uma arquitetura para objetos distribuídos e transacionais baseado em componentes, denominados de *beans* (JGURU, 2007). Esta arquitetura hospeda estes *beans* em *containers* padronizados que oferecem serviços remotos para clientes distribuídos na rede, como pode ser visto na Figura 6.

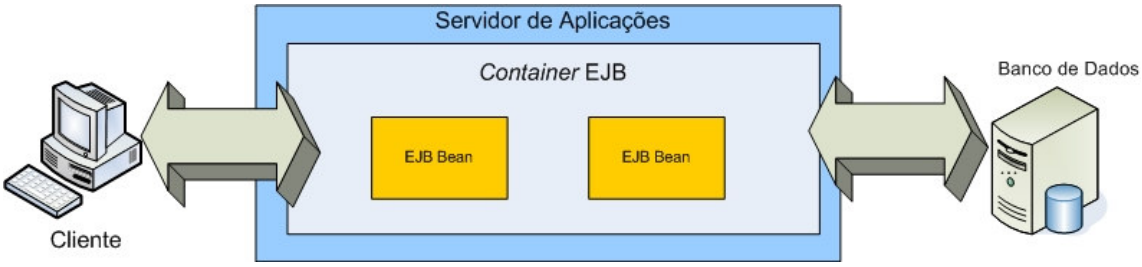

Figura 6 - Visão Geral da arquitetura EJB

Esta arquitetura determina um modelo de programação, através de uma API que é formada por classes, interfaces, protocolos e convenções. Isto fornece aos desenvolvedores e

fornecedores de servidores EJB uma série de contratos que devem ser seguidos para garantir a portabilidade entre as diversas implementações de servidores existentes, o que permite uma escolha de componentes de software e servidores de acordo com os interesses estratégicos do projeto.

Desta forma as organizações podem implementar seus componentes, comprar de terceiros ou reutilizar aqueles já existentes em suas bases de código. O desenvolvedor também pode se beneficiar desta arquitetura, no momento em que o aprendizado desta tecnologia poderá ser aplicado a uma diversidade de soluções disponíveis no mercado.

3.2.2 CORBA Component Model

O modelo de objetos distribuídos do CORBA resolveu diversos problemas no que se refere à comunicação remota de objetos, porém não resolveu problemas como empacotamento de serviços, facilidade de uso e aprendizado, além da facilidade de distribuição.

O CORBA Component Model é uma das partes chave da especificação 3.0 do CORBA, que passou a suportar o modelo de componentes e foi projetado para contornar as limitações das versões anteriores do CORBA como:

a) ausência de um mecanismo padronizado de implantação para promover o efetivo reuso de código, o que tornava o reuso uma atividade muitas vezes impraticável. Um objeto candidato a reuso muitas vezes dependia de outros, dificultando a confecção de um pacote autônomo - característica fundamental dos componentes. Adicionalmente, a heterogeneidade de plataformas e linguagens aumentava as dificuldades para o reuso;

b) necessidade de uma infra-estrutura que inclua os serviços de segurança, notificação de eventos, persistência e transações, que seja fácil de usar e de distribuir, ou seja, algumas das características das plataformas de componentes distribuídos.

O CCM é um modelo de componentes do lado do servidor para construção e implantação de aplicações CORBA que se assemelha ao EJB pelo uso de padrões de projeto e pelo uso da infra-estrutura de *containers*, como visto na Figura 7. Ele amplia o modelo de objetos distribuídos do CORBA e define recursos e serviços em um ambiente padrão que permite o desenvolvimento de aplicações, gerência, configuração e implantação de componentes que se integram com os serviços do CORBA. Estes serviços do lado do servidor

incluem transação, segurança, persistência e notificação de eventos. Como no EJB, a operação destes serviços pode ser feita automaticamente pelo *container*, conforme uma configuração dos mesmos, definida de forma declarativa, ou pode ser controlada pelo componente com intermediação do *container*, que será sempre responsável por se comunicar com os objetos que oferecem os referidos serviços.

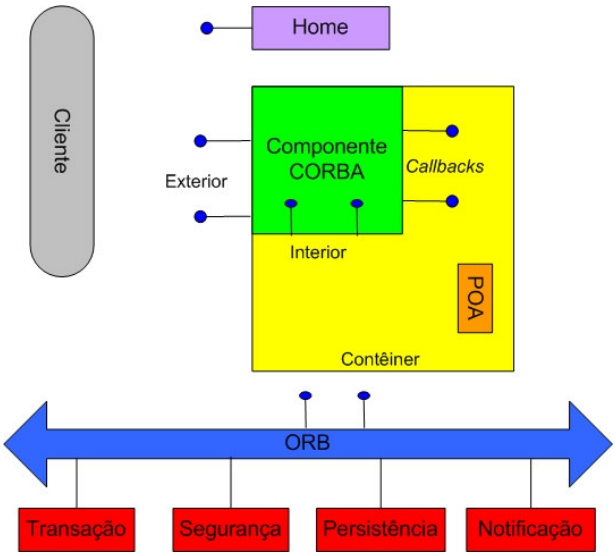

Figura 7 - Infra-estrutura do CCM.
Fonte: OMG (2003)

Em conjunto com o *container* atua o Portable Object Adapter (POA) que é responsável por operações importantes, como a associação de requisições do usuário com implementações de objeto e por ativação de objetos sob demanda.

A seguir são definidas as terminologias utilizadas pelo CCM (figura 8):

a) *facets*: são as interfaces do componente, ou seja, onde as funcionalidades do componente são expostas;

b) *receptacles*: permite ao componente declarar sua dependência a outra referência de objeto que ele precisa usar. Fornece mecanismos para especificar interfaces necessárias para que um componente funcione corretamente;

c) *event source/sinks*: permite componentes trabalharem em conjunto sendo capazes de produzir e receber eventos. O CCM define um suporte à notificação de eventos, por

este motivo, cada componente declara quais os eventos que ele pode produzir/emitir e quais os que pode receber.

d) *attributes*: permitem a configuração do componente. O CCM permite acessar e modificar os atributos do componente sem a necessidade de modificação de seu código.

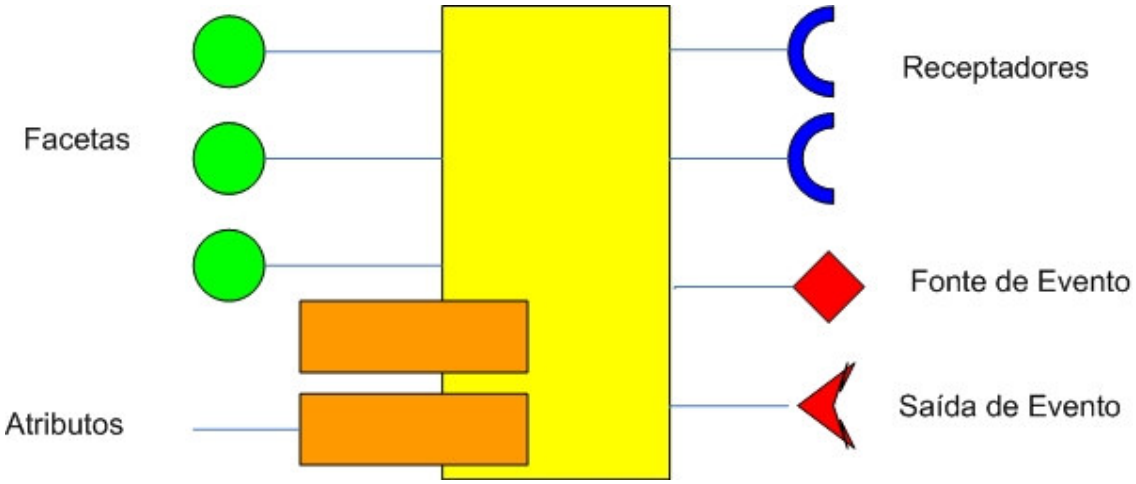

Figura 8 - Terminologia do CCM.
Fonte: OMG (2003)

O CCM enfatiza a separação de serviços funcionais (diretamente ligados ao objetivo da aplicação) e não funcionais (essenciais e de infra-estrutura da aplicação) como gerência do ciclo de vida, persistência e ativação. Os aspectos funcionais são fornecidos pelo componente enquanto que os aspectos não funcionais são gerenciados pelos *containers*.

3.2.3 COM+

A partir do COM, modelo de componentes locais da Microsoft, foi criado o Distributed COM (DCOM) que é um modelo que fornece as funcionalidades de componentes distribuídos. Apesar do nome, o DCOM não foi concebido como algo complementar ao COM. Na verdade ele é um novo modelo que fornece tanto acesso local (igual ao COM) quanto acesso remoto aos componentes. Através do DCOM, componentes que estão operando em diferentes máquinas podem interagir, desde que a plataforma dê suporte a esta tecnologia.

Em cima do COM e do DCOM estão apoiadas tecnologias de mais alto nível, como OLE, ActiveX e MTS. Dentre estas, a última é a mais significativa para se entender a evolução do COM. O *Microsoft Transaction Server* (MTS) provê serviços --- como controle

transacional e segurança --- para que aplicações distribuídas possam ser construídas utilizando-se a tecnologia DCOM.

COM+ é a evolução destes modelos e além de integrar os serviços do COM/DCOM, também possui os serviços do MTS e de fila de mensagens. Ele simplifica a programação dos componentes que usam tais serviços, através de uma melhor integração com as linguagens de programação da Microsoft (como Visual Basic, Visual C++ e J++) com estas tecnologias.

Diversas questões devem ser consideradas antes de se optar por usar a tecnologia da Microsoft:

a) diversidade de plataformas: a tecnologia é primariamente suportada nos sistemas Windows, com pouco suporte às demais plataformas;

b) portabilidade: uma vez que a base da comunicação entre os componentes é um formato binário, os componentes não são independentes de plataforma;

c) segurança: como os componentes possuem acesso à API do Windows, códigos não confiáveis poderiam deteriorar o ambiente do sistema do usuário;

d) infra-estrutura distribuída limitada: a infra-estrutura para aplicações distribuídas é básica, não contemplando situações onde, por exemplo, faz-se necessário o processamento em tempo real ou a confiabilidade.

3.2.4 OSGi – Open Services Gateway Initiative

Estabelecida em 1999, a OSGi Alliance é uma corporação sem fins lucrativos, trabalhando para definir e promover especificações abertas para a entrega de serviços gerenciáveis para ambientes em rede.

As especificações do OSGi definem um ambiente padronizado e orientado a componentes que fornece um *framework* Java que suporta a implantação de componentes extensíveis conhecidos como *bundles* (MARPLES; KRIENS, 2001). Estes *bundles* são arquivos compactados em formato ZIP, chamados de JAR (Java Archive), que contêm classes Java e outros recursos que fornecem funções para o usuário final e, opcionalmente, para outros *bundles*. Eles contêm os recursos necessários para implementar serviços que são as funcionalidades exportadas dos *bundles*. Eles contêm um arquivo *manifest* que descreve o conteúdo do jar e fornece informações sobre o *bundle*, suas dependências sobre outros

recursos, como pacotes Java que devem estar disponíveis para o *bundle* antes que ele possa executar, e designa uma classe especial que ativa e desativa o *bundle* denominada *bundle activator*.

O framework OSGi, como mostrado na Figura 9, situa-se acima do *Java runtime environment* e permite que dispositivos compatíveis com o OSGi façam *download*, instalem, atualizem e removam *bundles* quando eles não são mais necessários. Estes *bundles* podem acessar as funcionalidades do *framework*, da máquina virtual (VM) a seguir, e do sistema operacional se necessário utilizando a tecnologia *Java Native Interface* (JNI) (SUN, 2007). Os *bundles* instalados podem compartilhar recursos com outros *bundles* e também importar e exportar pacotes Java sob controle estrito do *framework*.

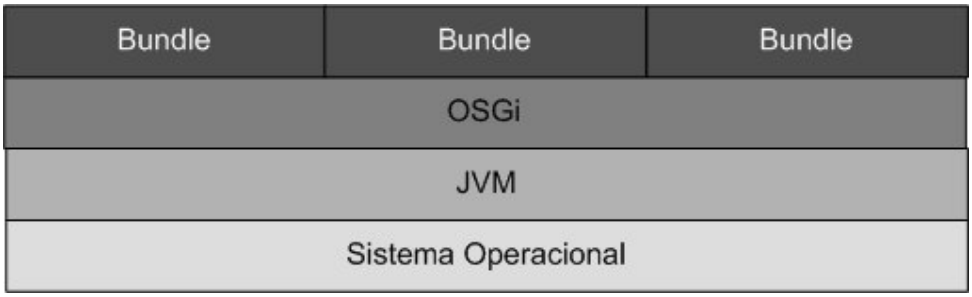

Figura 9 - Relacionamento dos componentes da plataforma de serviços OSGi

O ciclo de vida dos *bundles* OSGi é mostrado na Figura 10 e demonstra que quando o *bundle* é instalado ele vai para um estado resolvido onde o *framework* checa o arquivo *manifest* para verificar as dependências de pacotes Java externos. Assim que ele é resolvido, o *framework* está livre para usar os pacotes que este *bundle* fornece e, se necessário, também utilizá-los para resolver a dependência de outros *bundles*. Depois disso, o *bundle* resolvido pode ser ativado.

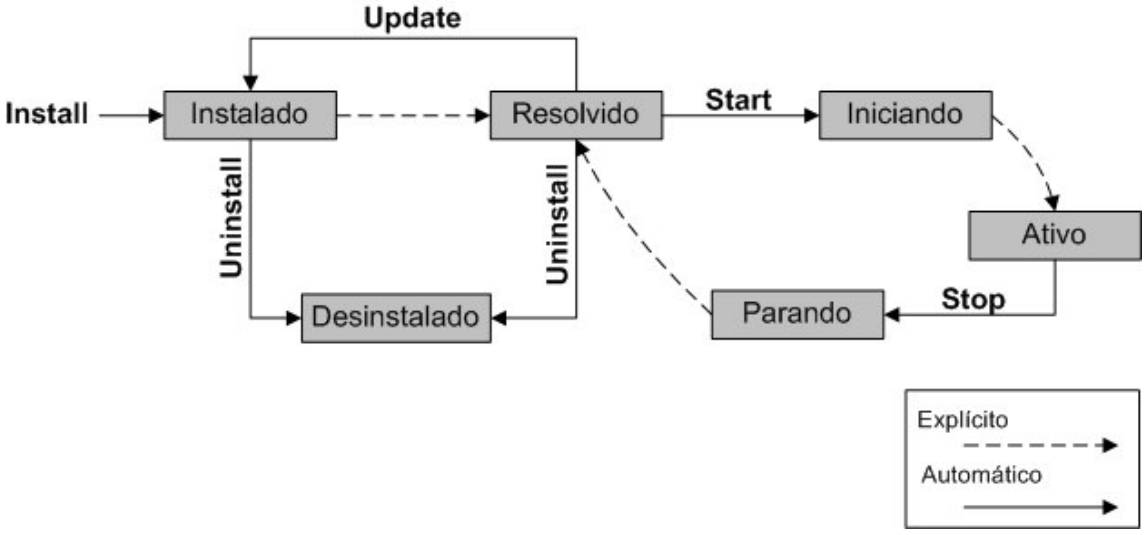

Figura 10 - Ciclo de vida dos Bundles OSGi.
Fonte: Hall e Cervantes (2004a)

A ativação resulta na criação de uma instância da classe *activator* referenciada no *manifest.* A classe *activator* implementa uma interface com dois métodos: um é chamado quando o *bundle* é ativado, e o outro quando o *bundle* é desativado. Esses métodos recebem um objeto de contexto que dá ao *bundle* o acesso às funcionalidades do *framework*, como o acesso a outros *bundles*, para desempenhar operações de gerência de *bundles*, registrar serviços, procurar por outros serviços, e registrar ele mesmo como um ouvinte para diferente tipos de eventos (HALL; CERVANTES, 2004b).

A tecnologia OSGi fornece primitivas padronizadas que permitem às aplicações serem construídas através de componentes pequenos, reusáveis e colaborativos. Esse *framework* gerencia a instalação e atualização dos *bundles* de uma forma escalável e dinâmica, e gerencia a dependência entre os *bundles* e serviços. Ela fornece ao desenvolvedor os recursos necessários para tirar vantagem da independência e carregamento de código dinâmico da plataforma Java de uma maneira fácil de desenvolver e implantar em larga escala serviços para dispositivos de pouca memória (MARPLES; KRIENS, 2001).

3.3 AVALIAÇÃO DAS TECNOLOGIAS

A plataforma COM+ apesar de possuir diversos recursos para a elaboração de componentes distribuídos não possui ampla difusão já que é uma tecnologia proprietária e restrita a plataforma Microsoft. Desta forma ela se tornou inadequada para a utilização dentro do contexto apresentado, ou seja, na existência de ambientes heterogêneos com a ampla diversidade de tecnologias e plataformas.

As plataformas CCM e EJB se concentram em componentes que serão executados em um servidor de aplicações, conforme pode ser visto na Figura 11. Isto lhes dá a característica de “componentes de servidor”. Os aspectos de distribuição, instalação, segurança, entre outros, estão diretamente associados a servidores de aplicações. Neste contexto, o serviço e suporte proporcionados pelo CCM e EJB não foram projetados para clientes e dispositivos com pouca memória.

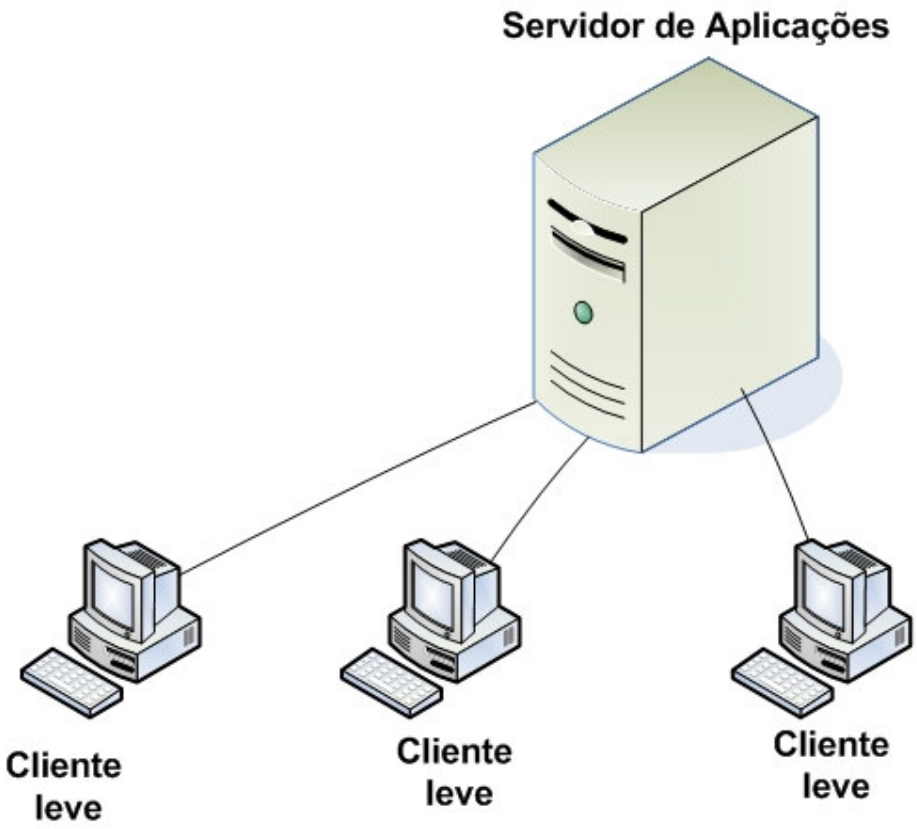

Figura 11 - Componentes de servidor: EJB e CCM

O OSGi, por outro lado, tem um enfoque bem mais distribuído, como mostrado na Figura 12. A infra-estrutura do OSGi se preocupa com a distribuição e instalação de componentes em todas as plataformas envolvidas em uma aplicação, sejam elas, clientes ou servidores, plataformas “pesadas” ou “leves”, que vão de um grande espectro desde servidores de aplicação, até pequenos dispositivos com pouca memória. Por este motivo OSGi é considerada uma infra-estrutura *lightweight* (CRNKOVIC; LARSSON, 2002).

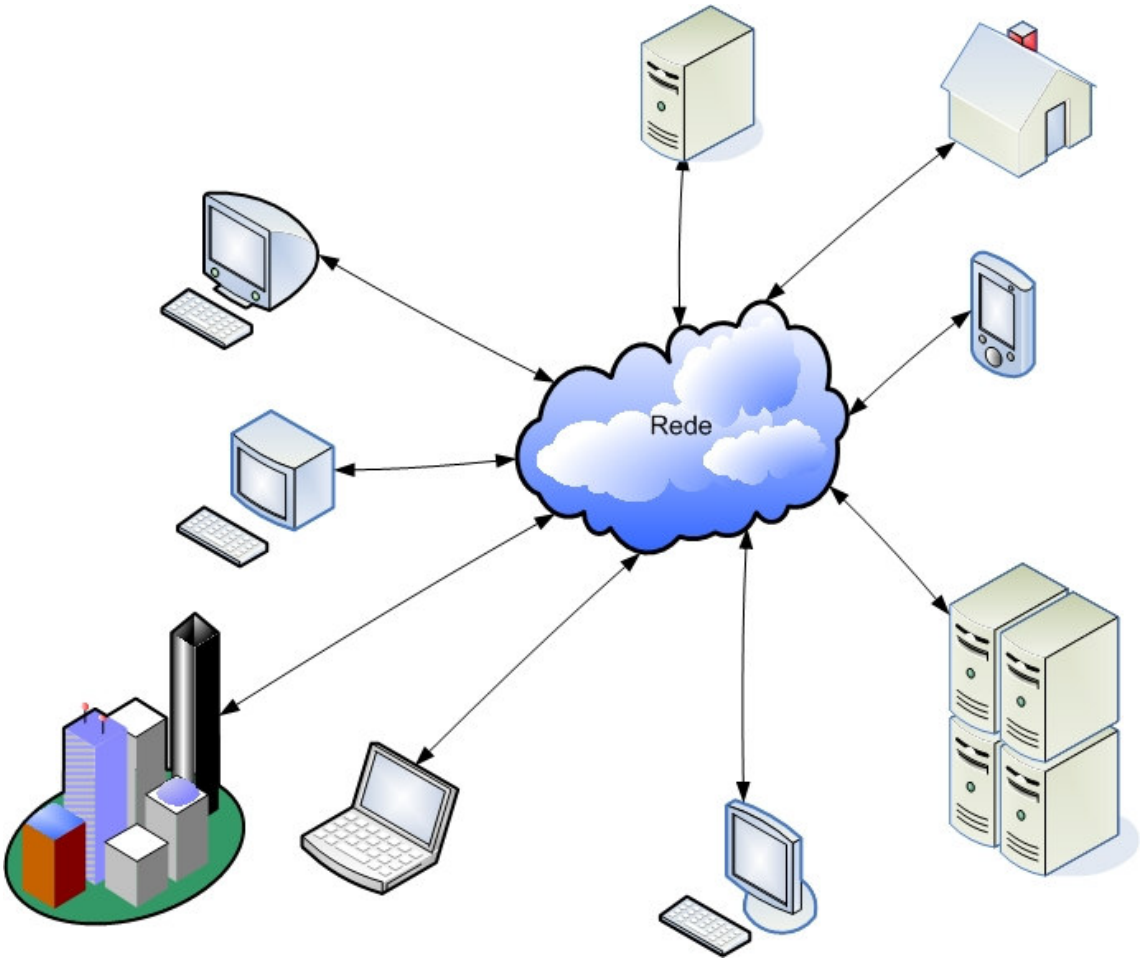

Figura 12 - Componentes distribuídos OSGi

Diferente dos outros padrões, o OSGi não vincula necessariamente seu padrão de distribuição a componentes que atuam de forma distribuída. Deste modo, seus benefícios de distribuição e instalação podem ser igualmente aplicados a componentes que atuam localmente. Isto será de vital importância para a infra-estrutura do FLAVOR, cujos componentes visuais atuam localmente, mas dispõem dos mesmos serviços OSGi que os componentes de acesso que atuam de forma distribuída.

A infra-estrutura OSGi foi escolhida para a implantação do FLAVOR uma vez que seu modelo reflete exatamente a demanda verificada no contexto dos ambientes de monitoração de redes, no qual torna-se necessário o gerenciamento da distribuição de componentes que atuarão em diversos contextos: possibilitando o acesso a dados de monitoramento de redes fornecidos pelos serviços, apresentando resultados nos clientes e em dispositivos móveis.

4 FLAVOR – UM ARCABOUÇO PARA APLICAÇÕES DE ACESSO A SERVIÇOS DE MONITORAMENTO MULTI-DOMÍNIO

Para superar as dificuldades destacadas na seção 2, foi iniciado o desenvolvimento de um *framework* que utiliza a tecnologia de componentes de software distribuídos. Ele permite o reuso no desenvolvimento de ferramentas de acesso às infra-estruturas de monitoramento de redes como a infra-estrutura do perfSONAR. O *framework* é denominado FLAVOR – *Framework Layer for Access and Visualization Of network resources*.

Não existe uma definição clara do que seja um *framework*, porém afirma-se que é uma arquitetura de reuso orientada a objetos que cria a estrutura para a definição de novos componentes através de abstração de classes, além de definir também a maneira como as instâncias destas classes irão cooperar para funcionar (JOHNSON, 1997). Outra definição é que um "*framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma determinada categoria de software" (GAMMA e outros, 2005).

Desta maneira um *framework* fornece uma estrutura reutilizável para que uma aplicação de um domínio específico possa ser criada e customizada através do reuso das partes integrantes desta estrutura.

O nosso *framework*, que denominamos de FLAVOR, provê uma camada de infra-estrutura sobre o OSGi, adaptado para serviços de monitoramento do perfSONAR.

4.1 INTRODUÇÃO

O FLAVOR é um *framework* voltado para o desenvolvimento reutilizável de aplicações de acesso ao perfSONAR formado por interfaces Java e componentes OSGi que promovem a padronização necessária para a reutilização deste tipo de *software*.

Com a utilização do FLAVOR é possível que qualquer aplicação desenvolvida em Java e que possua suporte ao OSGi faça uso das medições do perfSONAR de forma fácil e transparente. Seus componentes podem ser instalados através de arquivos localizados na máquina do usuário ou através de uma URL que aponta para um arquivo em um repositório de componentes remoto na Internet. Além disso, também é possível a combinação, atualização e remoção sem a preocupação de implementação destas funcionalidades na aplicação que fará o uso das medições, já que estas funcionalidade já são fornecidas pelo OSGi.

A Figura 13 mostra este contexto de interação, onde o FLAVOR pode ser instalado e utilizado em diversas plataformas suportadas pelo Java e pelo OSGi: desde aplicações localizadas em servidores e máquinas desktop até dispositivos móveis conectados na Internet. Desta forma, os serviços disponibilizados pelo perfSONAR (SQLMA, RRDMA, CLMP, LS, entre outros) podem ser acessados diretamente por qualquer aplicativo de forma imediata após instalar um componente do FLAVOR específico de acesso a este serviço.

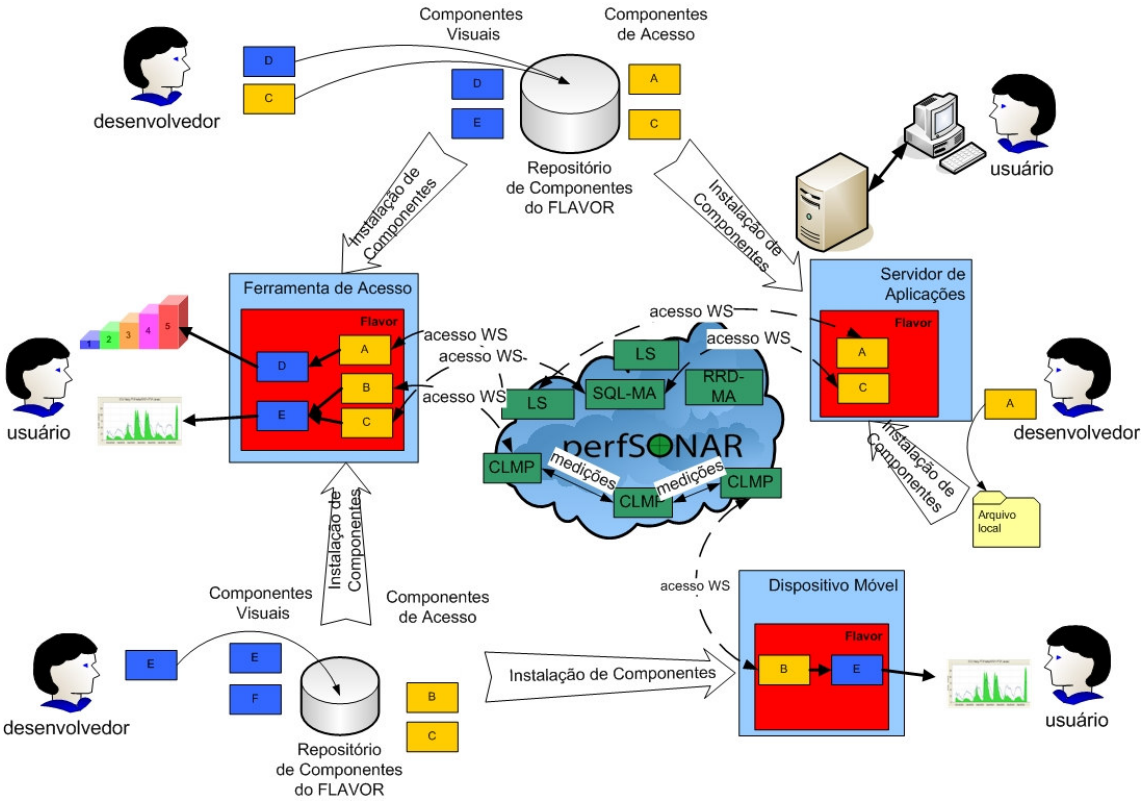

Figura 13 - Contextos de utilização do FLAVOR

Esta abordagem permite a atualização da aplicação sem a necessidade de recompilação ou reinicialização. Ou seja, no momento em que um novo serviço do perfSONAR se tornar disponível, basta o desenvolvimento de um novo componente do FLAVOR para que todas as aplicações que façam uso deste *framework* possam utilizar este novo serviço. A atualização também de um serviço existente se torna mais facilitada, já que não é necessário que se faça uma modificação em cada uma das aplicações de acesso aos serviços para que esta modificação seja suportada. Basta a realização da atualização do que foi modificado no componente que faz acesso ao serviço e disponibilização deste para os usuários. Estes usuários devem realizar a atualização dos seus componentes em suas aplicações.

Através do uso do modelo *Model-View-Controller* (MVC) (KRASNER; POPE, 1988) o FLAVOR define duas categorias distintas de componentes: acesso e visualização. O primeiro possibilita o acesso a um determinado serviço e o segundo faz uso do primeiro para disponibilizar visualizações para os usuários. Cada uma das categorias possui regras específicas para a sua construção.

No MVC uma aplicação é dividida em três partes distintas (GAMMA e outros, 2005):

a) Model – modelo da aplicação representada como estruturas de dados ou de classes;

b) View – lida com a parte de apresentação visual (gráfica) na tela do usuário;

c) Controller – interface entre Model e View e com os dispositivos de entrada. Define a maneira como a interface reagirá às entradas do usuário.

O MVC separa esses conceitos para aumentar a flexibilidade e reutilização. O FLAVOR adota este modelo e define um padrão para a construção de componentes e um protocolo para comunicação entre eles, conforme o MVC, como mostra a Figura 14 a seguir.

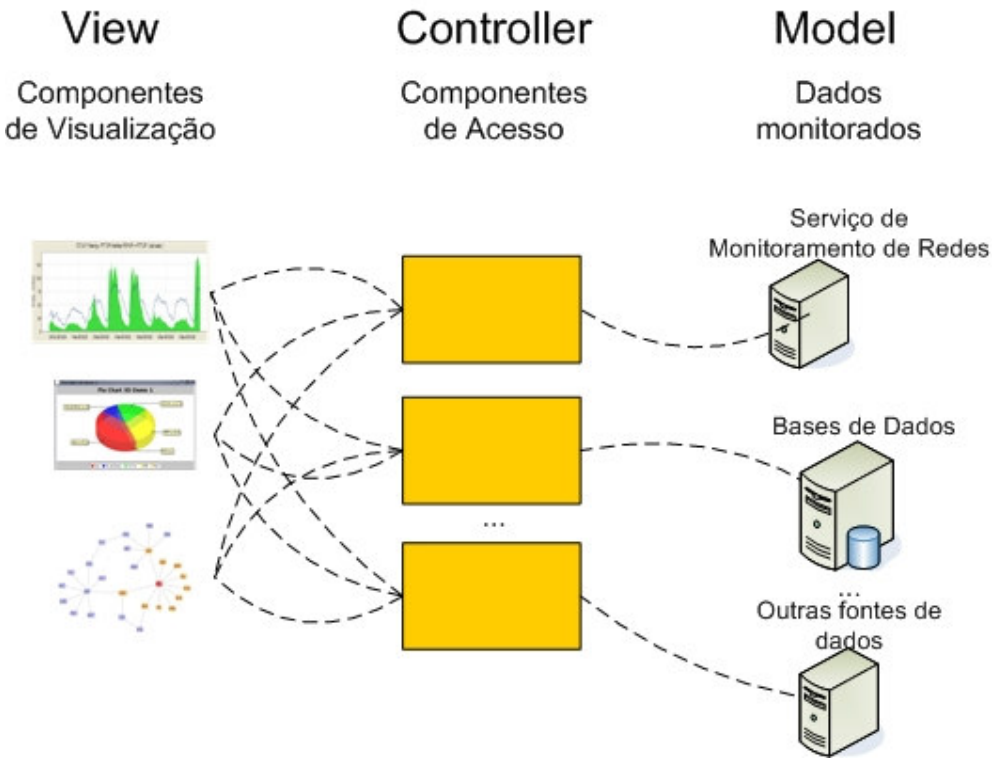

Figura 14 - Implementação do MVC no FLAVOR

Esta divisão é necessária para possibilitar o uso independente entre acesso e visualização pelas aplicações do FLAVOR. Ou seja, permite que determinada aplicação faça uso de um ou mais componentes de acesso ou de um ou mais componentes de visualização de forma independente. Isto permite, por exemplo, que um componente de acesso possa ser utilizado por dois componentes de visualização ou que um componente de visualização faça uso de vários componentes de acesso para fornecer ao usuário uma visualização composta por várias medidas disponibilizadas pelos serviços do perfSONAR.

Além disto, o FLAVOR também possui interfaces que permitem a adição de novos componentes de forma que estes possam interagir com os componentes existentes. Um desenvolvedor pode implementar estes novos componentes e os disponibilizar para o público em um repositório de componentes ou apenas utilizá-lo em sua aplicação fazendo a implantação através de um arquivo local.

4.2 ENTENDENDO AS INTERFACES

Para o desenvolvimento do FLAVOR, foi necessária a padronização de suas funcionalidades em interfaces Java, já que as funcionalidades dos componentes devem ser expostas através delas. A Figura 15 mostra a hierarquia de interfaces do FLAVOR bem como os métodos de cada uma delas. A interface *FlavorPlugin* estende a interface *BundleActivator* do OSGi (que define um meio padrão para iniciar e parar um componente OSGi) adicionando outros métodos para um componente padrão do FLAVOR, que permite recuperar o nome, versão, informações do componente, informações de direitos e um método para ajuda. Abaixo desta, outras três interfaces herdeiras do *FlavorPlugin* são definidas: *FlavorDataPlugin*, *FlavorRemoteDataPlugin* e *FlavorViewPlugin*.

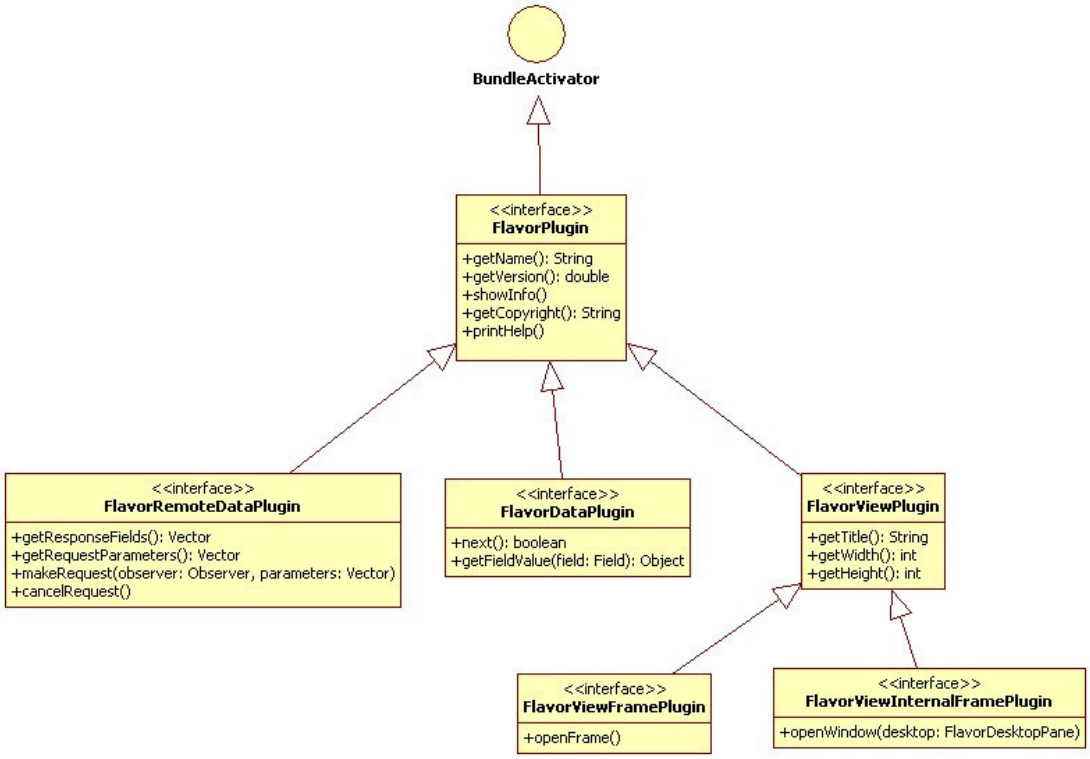

Figura 15 - Interfaces do FLAVOR

A interface *FlavorDataPlugin* é adequada para fontes de dados locais como um arquivo de texto, banco de dados local, ou qualquer outra fonte de dados local, estende de *FlavorPlugin* e define o método (**next**) para recuperar a próxima linha de um arquivo, banco de dados ou tabela de dados locais e um método (**getFieldValue**) para recuperar o valor de um campo de dados, representado em um objeto do tipo da interface *Field*. A interface *Field* representa um campo de dados de qualquer tipo representado pelo Java, nele você define o nome e seu tipo (valueClass), além de uma descrição opcional.

Já a interface *FlavorRemoteDataPlugin* é adequada para acesso a uma fonte de dados remota e estende *FlavorPlugin* e a interface *Observer* de modo a possibilitar a utilização do padrão de projeto *Observer* (GAMMA e outros, 2005), cuja intenção é definir uma dependência entre os objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente. Desta forma, o componente que implementar esta interface além de requisitar os parâmetros de requisição e resposta através dos métodos **getRequestParameters()** e **getResponseFields()**, respectivamente, poderá requisitar medidas e notificar o objeto que o requisitou.

O método **makeRequest** recebe como parâmetro um observador, que irá observá-lo e um objeto Vector Java com os parâmetros de requisição preenchidos, os métodos

getResponseFields e getRequestParameters retornam um Vector com os parâmetros de resposta e de requisição, respectivamente.

Desta forma, pode-se perceber que a depender da natureza de atualização dos dados existem duas estratégias distintas para monitorá-los. No primeiro grupo estão aqueles dados que precisam ser atualizados no visualizador a cada vez que sofrem modificação; neste caso, é mais conveniente que o objeto monitorado notifique o monitor a cada vez que exista uma modificação (utilizando a interface *FlavorRemoteDataPlugin*). No segundo grupo estão aqueles dados que são requisitados sob demanda; neste caso, o monitor solicita ao objeto seu estado quando necessário, desta forma é utilizada a interface *FlavorDataPlugin*.

É possível resumir sob as seguintes perspectivas esta divisão de monitoramento e atualização:

a) sob-demanda: A solicitação é feita, o componente que solicitou fica esperando até receber a resposta. Exemplo de aplicação: um cliente que tem acesso direto e imediato a um serviço com resposta com um tempo curto e/ou determinado.

b) padrão Observer: O componente solicitante se registra para receber uma atualização do componente observado, o componente solicitante realiza suas operações até ser notificado do recebimento da resposta. Exemplo de aplicação: Um cliente com acesso a um serviço que exija processamento e demorará um tempo aleatório para dar a resposta.

Além das interfaces de acesso a dados, temos a interface FlavorViewPlugin que lida com as funcionalidades básicas de um componente de visualização. Ela define métodos que permitem recuperar o título e o tamanho de uma janela de visualização. A partir dela duas interfaces herdeiras da mesma foram criadas chamadas de FlavorViewFramePlugin e FlavorViewInternalFramePlugin. A primeira define um componente de visualização que pode ser usado como um JFrame do Java e a segunda define um componente que poderá ser utilizado como um JInternalFrame também do Java.

A interface FlavorViewInternalFramePlugin define apenas um método chamado openWindow que recebe como parâmetro um FlavorDesktopPane que é uma especialização da classe JDesktopPane do Java e que fornece além da funcionalidade de desktop outros métodos para organização visual dos frames internos.

A interface *FlavorViewFramePlugin* define também apenas um método para abrir Frames chamada openFrame, a mesma não possui parâmetros. A chamada desta deve gerar a abertura de um Jframe.

4.3 COMPONENTES DO FLAVOR

A partir da padronização das interfaces mencionadas anteriormente, foram desenvolvidos alguns componentes para acesso a serviços do perfSONAR:

a) CLMP Ping Access Bundle – fornece acesso à funcionalidade de Ping por linha de comando do CLMP do perfSONAR;

b) CLMP Traceroute Access Bundle – fornece acesso à funcionalidade de acompanhamento de rota *(traceroute)* por linha de comando do CLMP do perfSONAR;

c) CLMP BWCTL Access Bundle – fornece acesso à funcionalidade do BWCTL por linha de comando do CLMP do perfSONAR;

d) CLMP OWAMP Access Bundle – fornece acesso à funcionalidade do OWAMP por linha de comando do CLMP do perfSONAR;

e) RRDMA Access Bundle – fornece acesso à funcionalidade de arquivo de medições RRD do perfSONAR;

f) LS XML Access Bundle – fornece acesso à funcionalidade do Lookup Service através da base de dados XML eXist do perfSONAR.

E também alguns componentes de visualização:

a) CLMP-Ping View Bundle – permite visualizar dados do componente de acesso do CLMP Ping;

b) RRDMA View Bundle – permite visualizar dados do componente de acesso RRDMA Access;

c) Tabular View Bundle – permite visualizar qualquer tipo de dado provindo de qualquer tipo de componente de acesso. Ele lista todos os componentes de acesso do FLAVOR em um determinado contexto do OSGi e cria uma interface que permite realizar o acesso de forma genérica e retornar os dados em forma tabular.

Todos estes componentes desenvolvidos podem ser utilizados em qualquer plataforma que suporte o OSGi como pode ser visto na Figura 16. Conseqüentemente, o FLAVOR usa as funcionalidades do OSGi, padronizando interfaces específicas para o monitoramento fornecendo ainda componentes pré-definidos para acesso aos serviços de monitoramento disponíveis do perfSONAR.

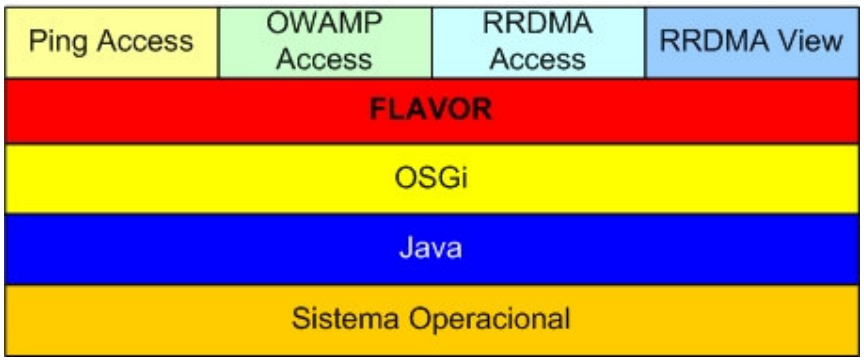

Figura 16 - Componentes do FLAVOR

A cada desenvolvimento de um novo componente que seja definido pelas interfaces do FLAVOR, a amplitude de uso deste se multiplica entre todas as ferramentas que possuam suporte ao FLAVOR.

4.4 AMPLIANDO O FRAMEWORK

Os desenvolvedores terão a possibilidade de estender o FLAVOR pelo desenvolvimento de componentes OSGi que fazem uso das interfaces mostradas anteriormente. Estes poderão ser usados em qualquer ferramenta que forneça as funcionalidades de componentes do FLAVOR. Além disso, caso achem necessário, os desenvolvedores podem modificar as interfaces do *framework* e anexá-las ao mesmo usando o seu código aberto. Mas se as interfaces forem modificadas, elas só serão visíveis para todos os

usuários se estas novas interfaces estiverem disponíveis e forem utilizados por todos os outros, pois o padrão foi modificado.

A Figura 17 ilustra os passos necessários para o desenvolvimento assim como os benefícios do framework. No passo um, o desenvolvedor 1 desenvolve um componente de acesso para seu uso e o torna disponível. Outro desenvolvedor pode desenvolver um componente de visualização, como mostrado no passo 2. Este segundo desenvolvedor torna este componente de visualização disponível, possibilitando a todos os usuários do FLAVOR se beneficiarem deste desenvolvimento com a reutilização dos componentes desenvolvidos nos passos 1 e 2, bastando instalar os componentes desenvolvidos e usá-los, sem a necessidade de se realizar qualquer desenvolvimento ou configuração das suas aplicações.

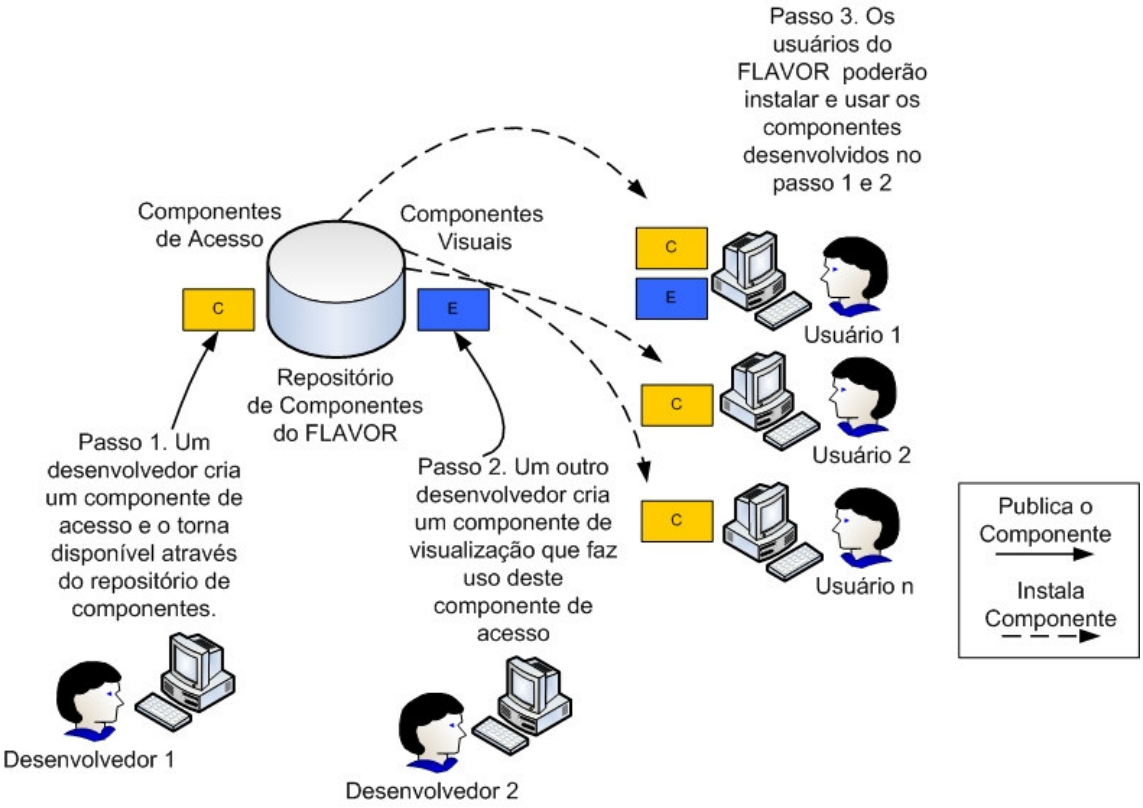

Figura 17 - Ciclo de desenvolvimento do FLAVOR

4.5 USO DO FLAVOR

Após a definição das interfaces, implementação e empacotamento dos componentes de monitoramento OSGi, o FLAVOR pode ser usado e reconhecido por qualquer ferramenta que tenha suporte ao OSGi. Isto pode ser conseguido quando o desenvolvedor de aplicações

instala uma das implementações do OSGi disponíveis e integra o FLAVOR dentro da sua aplicação.

Inicialmente o OSGi irá fornecer funções básicas para gerenciamento do ciclo de vida dos componentes como a instalação, busca, atualização e remoção, sem nenhum esforço de desenvolvimento como foi demonstrado na seção 3.2.4. O FLAVOR então irá fornecer a padronização das interfaces e componentes para o uso no desenvolvimento de aplicações OSGi para acesso e visualização dos serviços do perfSONAR.

Essa interação é mostrada na Figura 18 onde uma ferramenta de visualização de monitoração de redes instala alguns componentes no lado esquerdo, e outras aplicações possivelmente de outros contextos também podem se beneficiar deste *framework* pelo uso dos componentes desenvolvidos, o que é representado do lado direito da figura. Assim, os componentes padronizados podem ser usados por qualquer aplicação que faça uso do FLAVOR, pela simples incorporação de uma implementação do OSGi, incorporação das interfaces padronizadas e instalação dos componentes desenvolvidos.

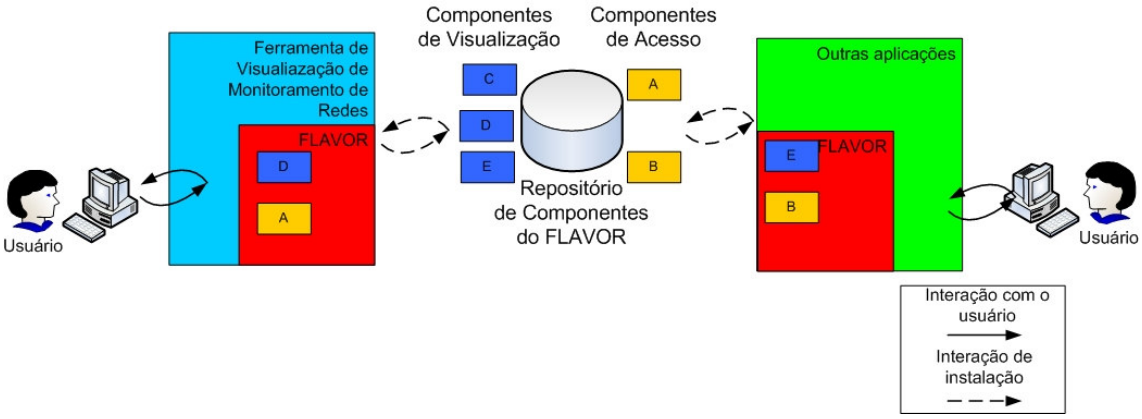

Figura 18 - Interação de uso do FLAVOR

Isso foi demonstrado na adoção do FLAVOR em duas iniciativas que necessitavam de acesso aos serviços do perfSONAR. A primeira se deu em um cliente de visualização de medidas de desempenho de redes denominado ICE e a segunda em um ambiente de Gerência de vídeo, ou seja, respectivamente em uma ferramenta de visualização de monitoração de redes e em uma aplicação de outro contexto, mas que necessita das medições para fornecer melhores serviços aos seus usuários. Essas adoções, descritas no capítulo a seguir, além de fornecer todos os recursos do FLAVOR descritos neste capítulo serviu também como teste e validação das funcionalidades deste *framework*.

5 APLICAÇÕES DO FLAVOR

O FLAVOR pôde ser testado e utilizado em duas iniciativas distintas que necessitam dos serviços do perfSONAR. A primeira a ser descrita neste capítulo foi a incorporação do FLAVOR no ICE (KOGA e outros, 2007a) o que deu flexibilidade a este cliente de visualização do GT-Medições. A segunda foi realizada através da utilização do FLAVOR pelo monitor do gerente de vídeo, em uma iniciativa de utilização das medidas coletadas na RNP pelo ambiente de gerência de vídeo desenvolvido pelo Grupo de Trabalho de Gerência de Vídeo (GTGV) da RNP.

5.1 ICE – INTERNET COMPUTER NETWORK EYE

Os usuários do piPEs-BR inicialmente utilizavam o RRDTool (RRDTOOL, 2007), Gnuplot (GNUPLOT, 2007), scripts PHP e Perl para gerar gráficos de dados recuperados das ferramentas de medição. Depois disso, foi usado o *framework* MonALISA (LEGRAND e outros, 2004), que acessava uma base de dados centralizada através de Serviços Web para fazer a visualização de determinados dados de medição usando a interface gráfica do usuário do próprio MonALISA. Entretanto, todas estas ferramentas não estavam integradas. Em alguns casos para fazer visualizações de diferentes dados de medição era necessário executar diferentes aplicações e não era possível desenvolver visualizações personalizadas e flexíveis para os usuários.

O ICE (*Internet Computer network Eye*) é um cliente de visualização que tem sido desenvolvido desde 2005. Ele tinha o objetivo inicial de integrar diversas visualizações no ambiente do usuário e acessar as diversas funcionalidades do ambiente de monitoração de redes piPEs-BR. Ele tem sido desenvolvido usando Java, JNI, Apache Axis (APACHE, 2007a) e uma biblioteca de gráficos chamada JFreeChart (JFREE, 2007) permitindo aos usuários consultar de uma forma integrada os serviços de monitoração de redes.

Durante o desenvolvimento do ICE foi percebido que as funcionalidades de outras aplicações não podiam ser reutilizadadas imediatamente, ou seja, era necessário uma análise do código destas aplicações para que a funcionalidade pudesse ser implementada/incorporada. Desta forma, foi percebida a necessidade de superação deste problema com um desenvolvimento de código mais flexível e reusável por parte de todas as aplicações que faziam acesso às medidas de desempenho do perfSONAR.

A partir disto foi idealizado o uso do FLAVOR em seu desenvolvimento para superar estas dificuldades já que o mesmo permite a adaptação dinâmica e reuso de componentes

OSGi de acesso ao perfSONAR. Além disto a separação entre componentes visuais e de acesso, conforme é feita no FLAVOR, permite combinações diversas daquelas para que eles foram projetados originalmente, o que não é possível ou muito difícil de implementar em aplicações monolíticas, ou seja, do modo como o ICE estava sendo desenvolvido.

Os componentes de acesso do FLAVOR podem ser combinados com outros componentes que não sejam aqueles de visualização, por exemplo, componentes que realizem algum ajuste na rede de acordo com o tráfego observado.

Para utilizar o FLAVOR, o ICE foi modificado para adquirir as suas funcionalidades, o que é descrito a seguir.

5.1.1 Reconstruindo o ICE

O ICE começou a ser desenvolvido com a intenção de integrar as diversas funcionalidades de visualização do piPEs-BR, conforme dito anteriormente. Na medida em que eram demandadas novas visualização para acessar um determinado serviço, uma nova construção era feita dentro do ICE, ou seja, novo código era produzido para este cliente sem a intenção do reuso, conforme pode ser visto na Figura 19. Era desenvolvido código específico para cada acesso aos diversos serviços que surgiam e isto estava amarrado à aplicação, ou seja, sem possibilidade (de forma facilitada) de alteração, remoção ou reutilização.

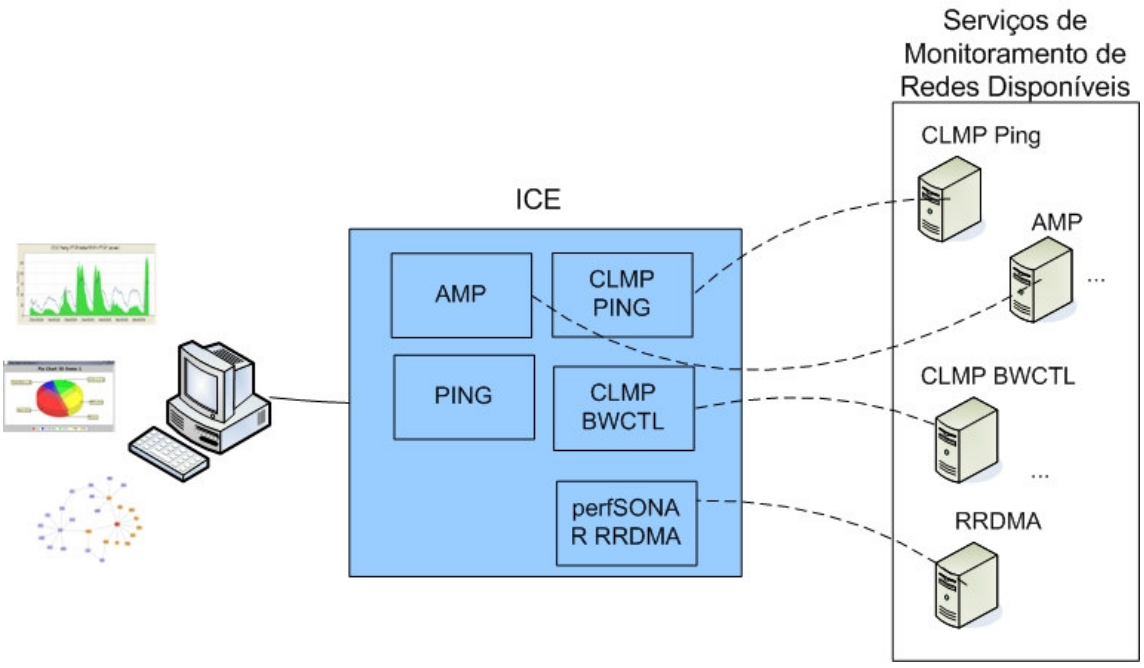

Figura 19 - ICE em sua concepção original

Percebendo que esta não era a melhor abordagem, o ICE foi reestruturado para incorporar as funcionalidades do FLAVOR. Isto é mostrado no diagrama da Figura 20, que mostra os passos necessários para atingir este objetivo:

a) no passo 1 o ICE incorporou uma implementação do OSGi chamado Felix (APACHE, 2007b);

b) no passo 2 a biblioteca do *framework* FLAVOR foi colocada no *classpath* do ICE para tornar disponíveis as funcionalidades do FLAVOR para o ICE;

c) no passo 3, o ICE usou algumas funcionalidades já desenvolvidas para o FLAVOR como o *desktop* e o gerenciador de *plugins* que torna mais fácil a interação do usuário com o FLAVOR;

d) finalmente, no passo 4, o ICE instala alguns componentes do FLAVOR para seu uso.

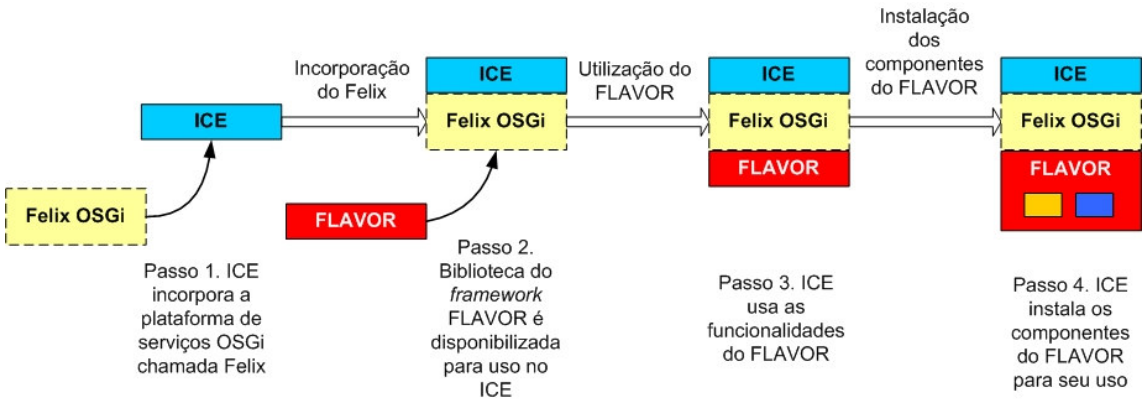

Figura 20 - Reconstruindo o ICE

Com esta nova abordagem, a interação com o ICE foi modificada como pode ser vista na Figura 21. Inicialmente, no passo 1 o ICE é inicializado e todas as variáveis do OSGi são iniciadas. No passo 2 o usuário recupera alguns componentes disponíveis e os instala no ICE, sendo que a solicitação feita pelo usuário (passo 2.1) é realizada de forma automática pelo Felix (passo 2.2). Finalmente, no passo 3, o usuário interage com os componentes instalados requisitando dados através dos componentes de acesso e visualizando os dados através dos componentes de visualização. Na próxima vez que o ICE inicializar, o OSGi procura pelos componentes instalados e já os inicializa de forma automática sem qualquer intervenção do usuário.

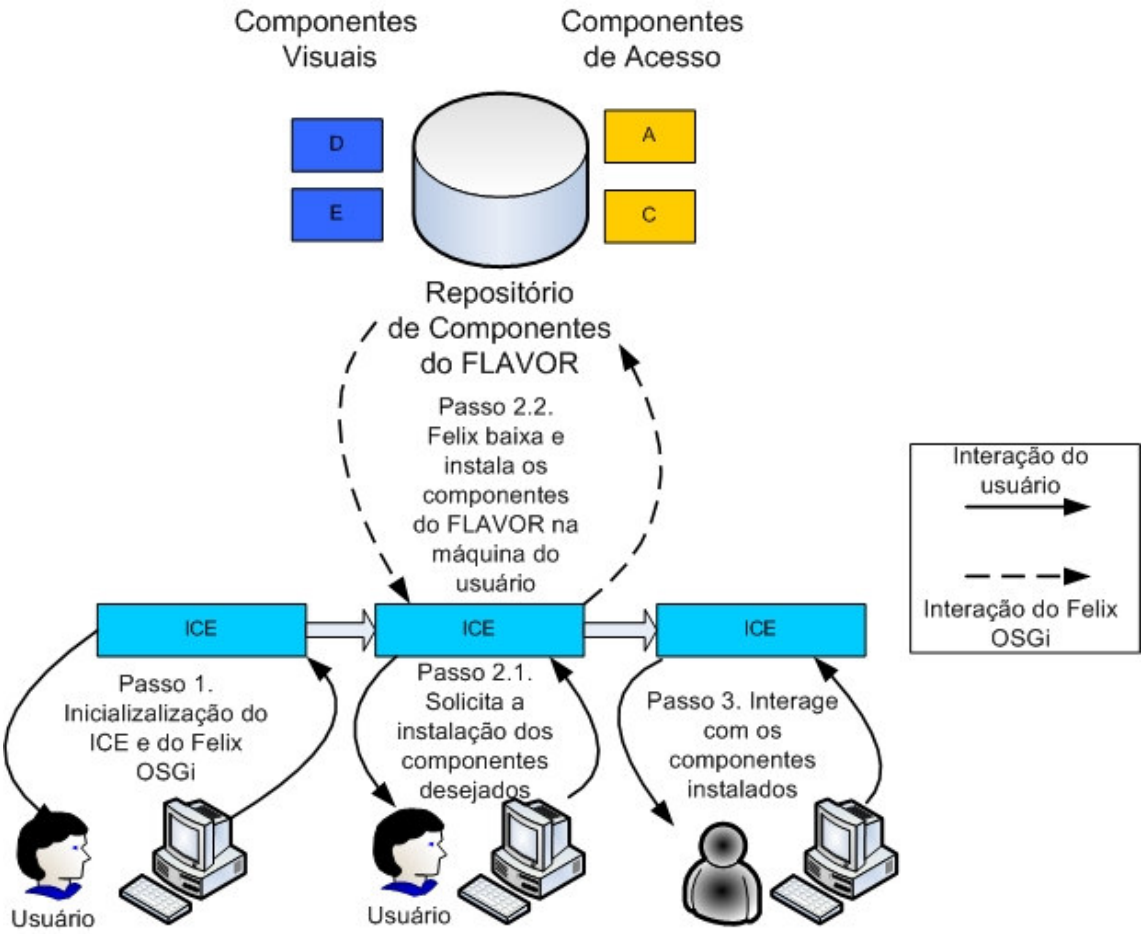

Figura 21 - ICE e o Framework FLAVOR em uso

Agora toda a ação de requisição e visualização de dados de medição pode ser feita pelos componentes instalados do FLAVOR. A Figura 22 mostra a arquitetura atual do ICE onde é possível instalar, atualizar ou remover componentes de acesso e visualização dinamicamente, sem necessidade de reinicializar a aplicação. É possível verificar o container e os componentes do FLAVOR.

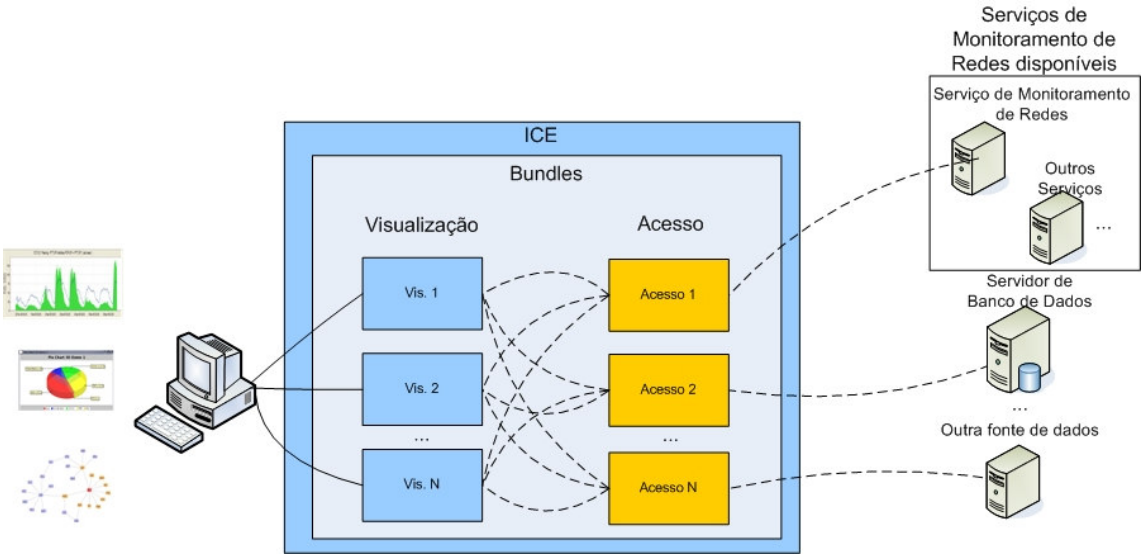

Figura 22 - Arquitetura do ICE

Esta arquitetura poderá ser visualizada em todas as ferramentas que utilizarem o FLAVOR. Com isto, o usuário pode unir componentes e usá-los quando necessário. Isto possibilita compor um ou vários componentes da maneira que for mais conveniente.

5.1.2 Cenário de uso do ICE

Inicialmente o ICE foi usado para acessar a infra-estrutura de medição do *backbone* da RNP que é mostrada na Figura 23. Essa infra-estrutura tem quatro pontos de medição (MPs) implantados nas cidades de São Paulo, Rio de Janeiro, Florianópolis e Salvador onde inicialmente foram implantados os serviços do CLMP e a ferramenta *NLANR Active Measurement Project* AMP (PROJECT, 2007). As métricas disponibilizadas por estes serviços de medição são: atraso em um sentido, atraso de ida e volta e acompanhamento de rota (*traceroute*).

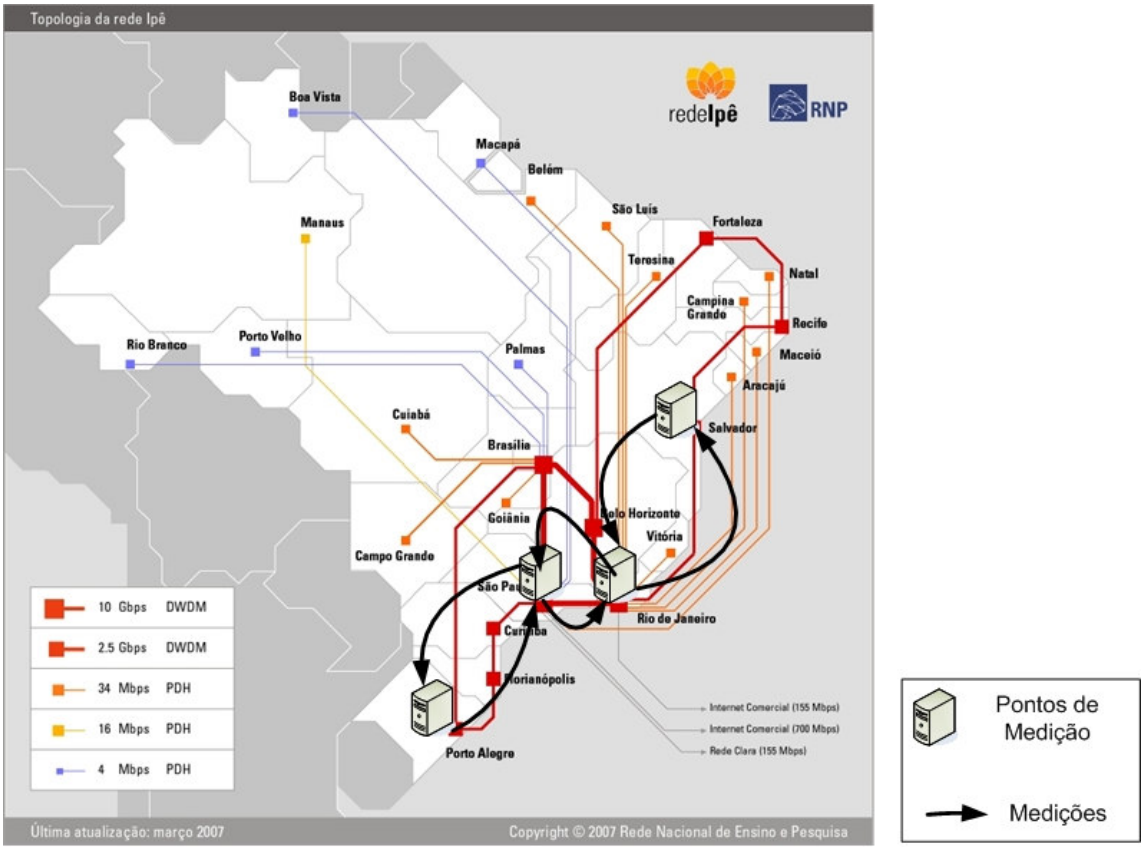

Figura 23 - Cenário do uso do ICE

Com a disponibilização destas métricas, o ICE pôde requisitar os serviços implantados nestes MPs e fornecer visualização em alguns gráficos para estes dados. Depois do desenvolvimento do FLAVOR e seu uso no ICE, alguns componentes foram desenvolvidos e instalados no ICE para fornecer acesso e visualização das métricas disponíveis. Estes componentes forneceram flexibilidade no desenvolvimento de novas visualizações, já que foi possível desenvolver várias visualizações usando o mesmo componente de acesso a dados, pelo seu baixo nível de acoplamento.

A Figura 24 mostra o uso do FLAVOR onde no lado esquerdo é mostrado o componente *Tabular View Ping Plugin* que é um componente de visualização que pode fazer visualização de medidas Ping de forma tabular do componente de acesso ao Ping. Na janela do lado direito aparece o *Plugin Manager* do FLAVOR que controla os componente instalados pelo fornecimento de controles para interação com o *framework* OSGi. O *Plugin Manager* mostra os componentes instalados: *Tabular View Ping Plugin*, que é um componente de visualização e o *Ping Plugin* que é um componente de acesso a dados.

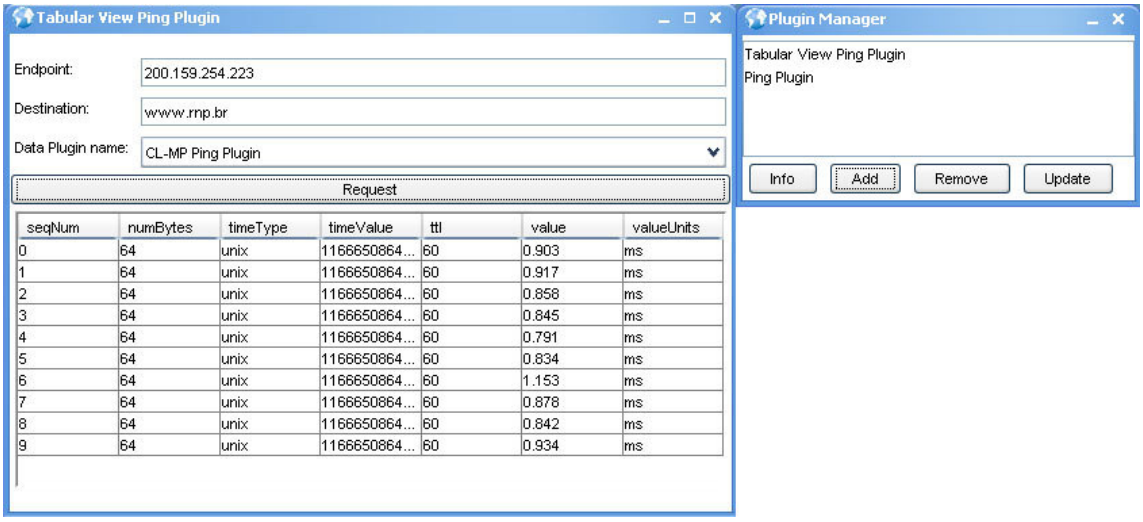

Figura 24 - Exemplo usando o plugin de visualização CLMP Tabular View Ping e o Plugin Manager

A integração com o FLAVOR e a reestruturação do ICE forneceu a flexibilidade necessária para implementar os componentes que podem ser reusados por qualquer aplicação e mostrou maneiras mais fáceis de desenvolver novas funcionalidades pelo reuso do código desenvolvido dos componentes já implementados. O ICE tem sido reestruturado para seguir esta nova abordagem e especificação e outras ferramentas e aplicações podem fazer o mesmo para se beneficiarem do uso do FLAVOR.

5.2 UTILIZAÇÃO DO FLAVOR EM UM SERVIDOR DE APLICAÇÕES DE GERÊNCIA DE VÍDEO

O Grupo de Trabalho de Gerência de Vídeo da RNP (GTGV) é um grupo de trabalho criado dentro do escopo da RNP e que tem como proposta a investigação e especificação de uma plataforma de gerência de serviços multimídia baseada na infra-estrutura e serviços da rede RNP. Tem como base a utilização e reuso do legado dos Grupos de Trabalho anteriores apoiados pela RNP tais como o portal e a rede de vídeo digital do GT de vídeo digital, o ambiente de monitoramento do GT de Medições e serviços e esquema de diretórios dos GT de Middleware e GT de Diretórios. O trabalho do grupo foi dividido em três atividades:

a) especificação de uma plataforma de gerência de serviços multimídia;

b) desenvolvimento de um sistema para gerenciamento de distribuição de vídeo;

c) testes do sistema implementado na rede da RNP de forma integrada a serviços já existentes.

A plataforma proposta (KULESZA e outros, 2007) está sendo desenvolvida em Java tendo como base o "*Java Stream Assembly*" para prover uma interface padrão para o controle de software e hardware utilizados na construção de serviços multimídia. Esta plataforma contém diversos módulos que tratam desde os elementos de baixo nível (*sockets*, arquivos, portas, *drivers*, *buffers*, *threads* etc.) até a parte de monitoração de eventos e mensagens de sessões das transmissões abertas.

O sistema para gerenciamento de distribuição de vídeo está sendo implementado utilizando a plataforma proposta e tem como objetivo explorar os padrões MPEG-7 e MPEG-21 e fornece um mecanismo de gerenciamento altamente adaptável e que possibilita a adoção do sistema dentro de contextos e modelos de negócios diversificados.

Dentre os módulos ou componentes do sistema existe o portal que é a porta de entrada dos usuários desse sistema. Além desse, existe o controle de acesso que controla o armazenamento, recuperação das mídias, cadastro de entidades, políticas de acesso e configurações, dentre outros. Também existe o Repositório das mídias de vídeo, codificados em MPEG 1,2,4 ou Windows Media, e que mantém também a gerência de serviços e metadados com a descrição dos conteúdos e serviços multimídia. Por último existe o Gerente que gerencia os serviços multimídia além de definir modelo de eventos e mensagens entre os componentes.

Está sendo utilizada a rede de distribuição de vídeo digital da RNP que conta com servidores em 10 localidades no país (São Paulo, Rio de Janeiro, Brasília, Curitiba, Minas Gerais, João Pessoa, Santa Catarina, Porto Alegre, Fortaleza e Recife) e que deverão ser integrados a esta nova plataforma. Além disso, está sendo realizada a integração com os serviços e ferramentas da infra-estrutura de medições do GT-Medições. Esta integração se deu no Gerente GTGV, mostrado na figura 25 que possui dois subsistemas principais:

a) coordenador: configura a topologia da rede de distribuição de conteúdo, configura a monitoração, realiza persistência, processamento e notificações de informações/alarmes dos barramentos de gerência e calcula rotas para transmissão de vídeos baseado nas informações que as funcionalidades anteriores provêem;

b) monitor: realiza *polling* através de monitores do nível de rede IP, servidores Linux e serviços de transmissão de vídeo.

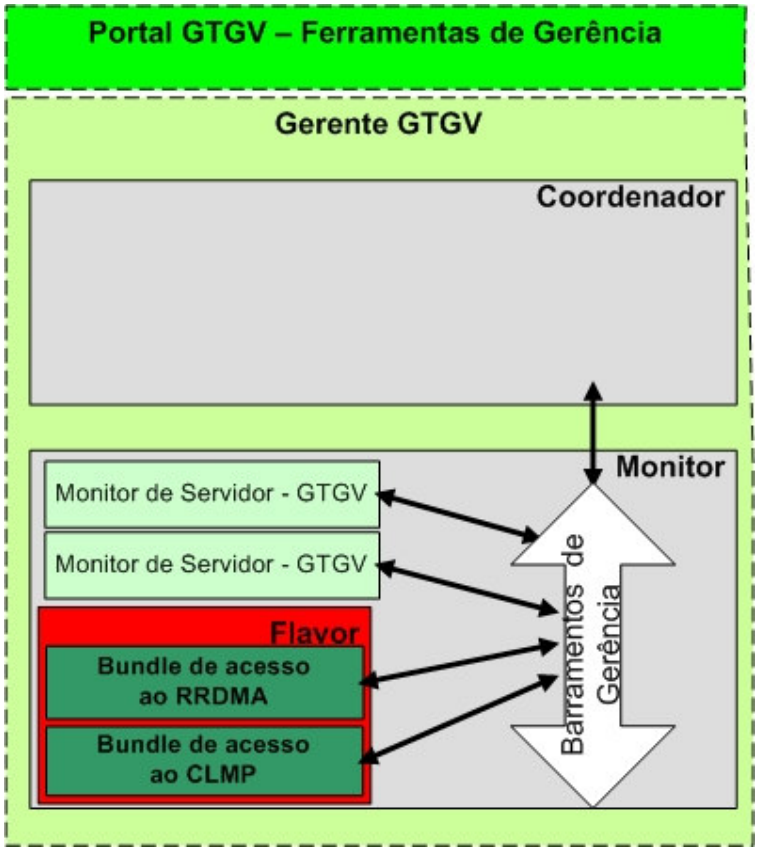

Figura 25 - Portal de Gerência de Vídeo e seus componentes

Na arquitetura do Gerente GTGV foi implantado o OSGi e o FLAVOR, especificamente no subsistema Monitor, conforme pode ser visto na Figura 5.7. Além de realizar esta integração, foi realizada a instalação de dois componentes do FLAVOR que são de interesse do GTGV: os bundles de acesso ao RRDMA e CLMP. O primeiro fornece informações dos estados dos enlaces do backbone da RNP e o segundo permite realizar medições sob demanda a partir da escolha de pontos de medição onde também existem serviços de transmissão de vídeo. Dessa forma é possível recuperar algumas medidas do estado de um determinado caminho onde passa um fluxo de vídeo (por exemplo, banda disponível, atraso, variação de atraso e perda).

Para a comunicação destes componentes com os barramentos de Gerência do GTGV, foram desenvolvidos *proxies* que implementam uma interface de produtor de informações de gerência do barramento. São realizadas requisições aos componentes e esses respondem para os *proxies* enviarem informações de gerência para os barramentos e, após isso, os dados são armazenados ou processados no Coordenador.

Essas informações recuperadas pelos *proxies* são úteis para melhorar o algoritmo de cálculo de rota de transmissões de vídeo e também serve para a Gerência de Contabilidade e

Desempenho da infra-estrutura IP que fornece os serviços de distribuição de vídeo na rede da RNP.

Caso não fosse utilizado o FLAVOR, seria necessário o desenvolvimento de código de acesso no monitor do Gerente GTGV ou cópia do código de acesso desenvolvido em outra ferramenta como no ICE. Para esta migração de código seria necessária a análise e adaptação para ajuste às necessidades específicas do Monitor. Ao contrário, o uso das medições disponíveis foi possível apenas com a instalação de um framework OSGi, implantação do FLAVOR e uso dos 2 bundles úteis ao GTGV.

Se houver disponibilidade de um novo serviço do perfSONAR e o GTGV tiver interesse em acessá-lo, bastará desenvolver um novo bundle de acesso e instalá-lo no monitor GTGV.

5.3 CONCLUSÃO

Os dois casos de uso do FLAVOR permitiram observar a facilidade do reuso de código e adaptação dinâmica além de compartilhamento de código nestas duas aplicações, conforme visto na Figura 26 onde as duas aplicações podem instalar componentes visuais e de acesso de um repositório e prontamente acessar os serviços do perfSONAR. Caso o FLAVOR não fosse utilizado, a reutilização de código seria mais dificultada e não seria possível a adaptação dinâmica das aplicações. Tanto o ICE quanto no servidor de aplicações de gerência de vídeo o desenvolvimento seria de forma monolítico e a migração de uma funcionalidade existente em uma aplicação para outra seria mais difícil.

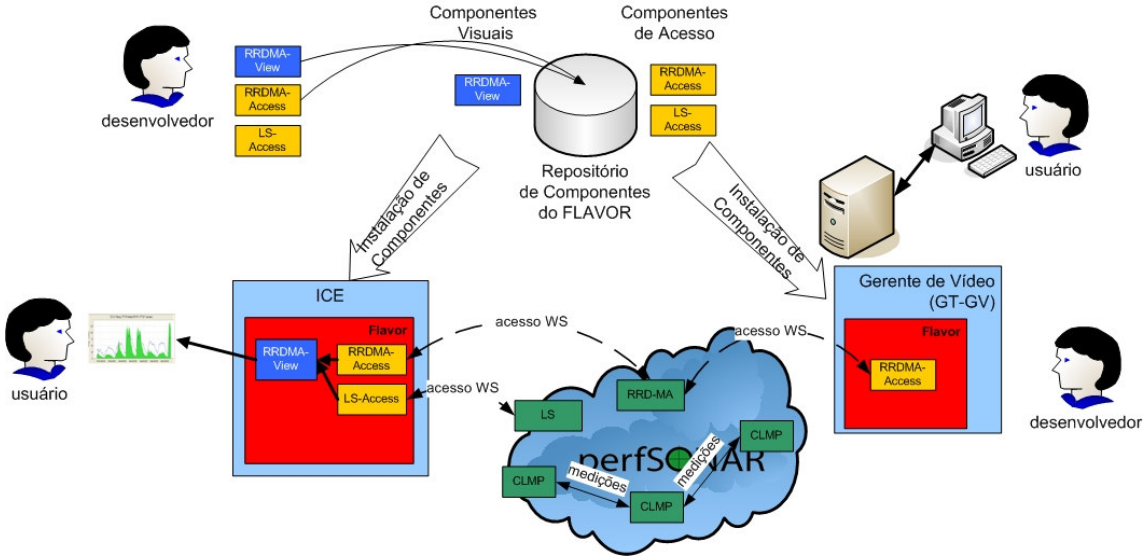

Figura 26 - Uso do FLAVOR no ICE e na ferramenta de Gerência de Vídeo da RNP

O ICE se tornou mais flexível e seu código menos acoplado enquanto o servidor de aplicações de Gerência de Vídeo pôde também desfrutar do acesso às medidas de desempenho do perfSONAR facilmente e sem o desenvolvimento de código específico para isto através do uso dos bundles já desenvolvidos. Ambas agora possuem o recurso de adaptação dinâmica, ou seja, podem instalar, atualizar ou remover funcionalidades em tempo de execução e sem a necessidade de reinicialização da aplicação seguindo os padrões do OSGi e FLAVOR.

6 CONCLUSÕES

As medidas de desempenho fim-a-fim que atravessam diferentes domínios administrativos que eram difíceis ou impossíveis de serem recuperadas, pelas dificuldades técnicas, políticas e de segurança, se tornaram acessíveis pela iniciativa de disponibilização dos serviços do perfSONAR em diferentes redes espalhadas pelo mundo. Desta forma, o acesso ao perfSONAR é de fundamental importância para os usuários de redes, já que estes são os interessados nestes dados.

As diversas ferramentas que atualmente possibilitam o acesso ao perfSONAR sofreram adaptações ou foram desenvolvidas para acessar os seus serviços. Este desenvolvimento estava sendo realizado de forma independente e com código altamente acoplado. Outras ferramentas que desejassem acessar o perfSONAR teriam que desenvolver seu código para que fosse possível o acesso, sem qualquer forma facilitada de reutilização do código já desenvolvido. Porém, com o desenvolvimento do FLAVOR, basta a implementação de componentes com a sua padronização para que todas as ferramentas possam utilizá-los. Esta é a principal vantagem com relação aos recursos de reutilização que estão sendo adotados na perfSONARUI, por exemplo, dado que ela não permite a reutilização dos seus componentes em outras ferramentas, mas somente na sua interface gráfica.

No FLAVOR, apesar do esforço inicial de instalação e uso da plataforma OSGi e do próprio FLAVOR, os desenvolvedores das ferramentas terão facilidades pelo desenvolvimento de componentes que poderão ser compartilhados e utilizados em outras ferramentas.

Este trabalho explorou o desenvolvimento do FLAVOR no fornecimento de funcionalidades mais facilitadas para o desenvolvimento reusável de ferramentas de acesso ao perfSONAR. Com o uso do FLAVOR, desenvolvedores das ferramentas de acesso às medições ou qualquer outro tipo de aplicação que possua acesso a redes podem usar a abordagem de componentes para obter flexibilidade, adaptabilidade e reuso, dentre outros benefícios que ele fornece.

Os usuários das medições (os usuários interessados em dados, análise e visualização de dados de medição de redes) podem agora, com o uso de componentes, escolher suas funcionalidades favoritas para usar dentro de sua aplicação. Os componentes desenvolvidos podem ser montados e compostos nas suas aplicações usando os benefícios da gerência do ciclo de vida do OSGi para implantar componentes dinamicamente e, conseqüentemente, configurando sua ferramenta de forma transparente e flexível.

Esforços futuros deverão se concentrar na disseminação do desenvolvimento de novos componentes para o FLAVOR, fornecendo componentes reusáveis para toda a comunidade de redes de computadores. Existe também a intenção do uso desta abordagem em outras infra-estruturas multi-domínio e em outros domínios de aplicação como na área de gerência de redes. Isto pode beneficiar muitos outros usuários e fornecer benefícios para seu uso.

Com a disseminação do desenvolvimento de componentes e com a experiência de construção do FLAVOR e o ICE, o desenvolvimento de ferramentas de monitoramento de redes devem necessitar de um repositório de componentes que forneça recursos mais avançados para procura, publicação e gerência de dependência de componentes. Isto pode minimizar ainda mais o tempo para publicação e descoberta de componentes, tornando mais fácil o desenvolvimento destas ferramentas de acesso ao perfSONAR.

REFERÊNCIAS

APACHE. **Web services axis.** Disponível em: <http://ws.apache.org/axis/>. Acesso em: 05 dez. 2007.

APACHE. **Apache Felix.** Disponível em: <http://felix.apache.org/.>. Acesso em: 01 out. 2007.

BOOTE, J. W. et al. Towards Multi-Domain Monitoring for the European Research Networks. **Terena: Networking Conference**. 01 jun. 2005.

CACTI. **Cacti:** the complete RRDTool-based graphing solution. Disponível em: <http://cacti.net/>. Acesso em: 06 out. 2007.

CISCO. **Cisco:** Cisco netflow collection Engine. Disponível em: <http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/netflsol/nfwhite.htm>. Acesso em: 08 out. 2003.

CNM. **Customer network management.** Disponível em: <http://www.cnm.dfn.de/>. Acesso em: 08 jun. 2006.

CRNKOVIC, Ivica. Component-based software engineering: new challenges in software development. **Software Focus.** Dez. 2001.

CRNKOVIC, Ivica; LARSSON, Magnus. **Building reliable component-based software systems**. [S.l.]: Artech House Publishers, 2002.

DEMICHIEL, Linda G. **Enterprise javabeans specification version 2.1**. **Sun Microsystems. N**ov. 2003.

DFN. **Deutsche Forschungsnetz.** Disponível em: <http://www.dfn.de/>. Acesso em: 08 jun. 2006.

D'SOUZA, Desmond Francis; WILLS, Alan Cameron. **Objects, components,and frameworks with (UML):** the catalysis approach. New York: Addison Wesley, 1999.

EMMERICH, Wolfgang. **Distributed component technologies and their software engineering implications.** New York: ACM Press, 2002. 537--546 p.

GAMMA, Erich et al. **Padrões de projeto.** New York: Bookman, 2005.

GÉANT2. **Deliverable DJ.1.2.1:** GEANT2 general monitoring framework design. Disponível em: <https://wiki.man.poznan.pl/perfsonar-mdm/images/perfsonar-mdm/9/95/GN2-05-057v5.pdf>. Acesso em: 02 jun. 2005.

GÉANT2. **Performance measurement and monitoring.** Disponível em: <http://www.geant2.net/server/show/nav.754>. Acesso em: 19 out. 2007.

GGF. **GGF network measurement working group.** Disponível em: <http://nmwg.internet2.edu/.>. Acesso em: 07 mar. 2005.

GNUPLOT. **Gnuplot homepage.** Disponível em: <http://www.gnuplot.info/>. Acesso em: 07 dez. 2007.

GOOGLE. **Google maps.** Disponível em: <http://maps.google.com/>. Acesso em: 08 out. 2007.

HALL, Richard S.; CERVANTES, Humberto. Challenges in building service-oriented applications for OSGi. **IEEE Communications Magazine,** New Jersey, p. 6. Maio 2004.

HALL, Richard S.; CERVANTES, Humberto. An OSGi implementation and experience report. **IEEE Consumer Communications And Networking Conference,** Grenoble, p. 394-399. Maio 2004.

HANEMANN, A. et al. **PerfSONAR:** a service oriented architecture for multi-domain network monitoring**.** Amsterdam: Acm Sigsoft And Sigweb, 2005. 241-254 p.

HANEMANN, A. et al. **Complementary visualization of perfSONAR network performance measurements.** Cap Esterel: IARIA/IEEE, 2006.

HOPKINS, Jon. Component primer. **Communications Of The Acm,** New York, p. 27-30. Out. 2000.

INTERNET2. **E2EpiPEs:** End-to-end performance initiative performance environment system architecture. Disponível em: <http://e2epi.internet2.edu/e2epipes//e2epipe11.pdf>. Acesso em: 16 out. 2003.

INTERNET2. **Internet2.** Disponível em: <http://e2epi.internet2.edu/e2epipes//e2epipe11.pdf>. Acesso em: 20 out. 2006.

INTERNET2. **E2EPI:** internet2 end to end performance initiative. Disponível em: <http://e2epi.internet2.edu.>. Acesso em: 25 set. 2007.

INTERNET2. **Bandwidth test controller (BWCTL).** Disponível em: <http://www.internet2.edu/performance/bwctl/>. Acesso em: 11 out. 2007.

JELIAZKOVA, N.; ILIEV, L.; JELIAZKOV, V.. PerfsonarUI: A standalone graphical user interface for querying perfSONAR services. In: IEEE JOHN VINCENT ATANASOFF**:** INTERNATIONAL SYMPOSIUM ON MODERN COMPUTING, 2006, Bulgaria, **Anais...** Bulgaria, p.77-81, 3 out. 2006.

JFREE. **JFreeChart.** Disponível em: <http://www.jfree.org/jfreechart/index.php>. Acesso em: 21 dez. 2007.

JGURU. **JGuru:** enterprise javabeans fundamentals. Disponível em: <http://java.sun.com/developer/onlineTraining/EJBIntro/EJBIntro.html>. Acesso em: 20 dez. 2007.

JOHNSON, Ralph E. Frameworks = (components + patterns). **Communications Of The ACM**, New York, v. 40, n. 10, p.39-42, 01 out. 1997.

KOGA, I. K. et al. A flexible network monitoring access environment. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 25., 2007, Belém. **Anais...** Belém. 2007. p. 1181 - 1188

KOGA, I. K. et al. FLAVOR: A dynamic and open framework for the development of network measurement access and visualization tools. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 25., 2007, Belém. **Anais ....** Belém. 2007. p. 665 - 678.

KRASNER, G. E.; POPE, S. T.. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. **J. Object Oriented Program**, Denville, v. 1, n. 3, p.26-49, 01 ago. 1988.

KULESZA, R. et al. Uma ferramenta para gerência de distribuição de mídias. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 25., 2007, Belém. **Anais ....** Belém. 2007. p. 1131 - 1138.

LEGRAND, I. C. et al. MonALISA: an agent based, dynamic service system to monitor, control and optimize grid based applications. **Computing In High Energy And Nuclear Physics (Chep) Conference**, Interlaken, 27 set. 2004.

MARPLES, D.; KRIENS, P. The open services gateway initiative: an introductory overview. **IEEE Communications Magazine**, New York, v. 39, n. 12, p.5, dez. 2001.

MONTEIRO, José A. S.. **GT-Medições:** documento de avaliação dos pilotos. [S.l.]: RNP, 2005.

NEMO. **NetMonitor:** enterprise javabeans fundamentals. Disponível em: <http://drift.uninett.no/kart/nemo/>. Acesso em: 09 ago. 2006.

NEWMAN, H. B.; LEGRAND, I. C.; BUNN, J. J. A Distributed Agent-based Architecture for Dynamic Services. **Conference For Computing In High Energy And Nuclear Physics**, Beijing, set. 2001.

OLSEN, Greg. From COM to Common. **Queue**, New York, v. 4, n. 5, p.20-26, 01 jun. 2006.

OMG. **CORBA:** Components – Version 3.0 – formal/02-06-65. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/02-06-65>. Acesso em: 09 out. 2003.

OMG. **The object management group.** Disponível em: <http://www.omg.org/>. Acesso em: 01 dez. 2007.

PAPAZOGLOU, M. P. Service-oriented computing: concepts, characteristics and directions. In: KEYNOTE FOR THE 4TH INTERNATIONAL CONFERENCE ON WEB INFORMATION SYSTEMS ENGINEERING, 2003. **Anais…** IEEE Computer Society. Netherlands. 2003.

PERFSONAR. **PERFSONAR.** Disponível em: <http://www.perfsonar.net./>. Acesso em: 01 dez. 2007.

PROJECT, N. A. M.. **Active measurement project.** Disponível em: <http://amp.nlanr.net/.>. Acesso em: 14 dez. 2007.

RNP. **Rede Nacional de Ensino e Pesquisa.** Disponível em: <http://www.rnp.br/>. Acesso em: 04 ago. 2007.

RRDTOOL. **RRDtool.** Disponível em: <http://oss.oetiker.ch/rrdtool/>. Acesso em: 04 dez. 2007.

SAMPAIO, Leobino et al. PiPEs-BR: uma arquitetura para a medição de desempenho em redes IP. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 24., 2006, Curitiba. **Anais ...** Curitiba: SBRC, 2006.

SUN. **JNI.** Disponível em: <http://java.sun.com/j2se/1.4.2/docs/guide/jni/.>. Acesso em: 04 dez. 2007.

SUNET. **Swedish University Computer Network.** Disponível em: <http://www.sunet.se/>. Acesso em: 04 out. 2006.

SZYPERSKI, C.; GRUNTZ, D.; MURER, S.. **Component software:** beyond object-oriented programming. 2. ed. New York: Addison-wesley/ACM Press, 2002. 589 p.

UNINETT. **Norwegian research network.** Disponível em: <http://www.dfn.de/>. Acesso em: 05 out. 2006.

VIEW, Flow. **CactiUsers:** flow view. Disponível em: <http://cactiusers.org/wiki/FlowView>. Acesso em: 05 out. 2007.

VISUALPERFSONAR. **VisualperfSONAR web Interface.** Disponível em: <https://noc-mon.srce.hr/visual_perf>. Acesso em: 13 out. 2006.

WEATHERMAP, Network. **Network WeatherMap.** Disponível em: <http://www.network-weathermap.com/>. Acesso em: 22 out. 2007.