UNIVERSIDADE FEDERAL DE SANTA CATARINA

OBTENÇÃO DE DADOS DE QoS DOS ELEMENTOS DE UM GRID MÓVEL

Rudson Vieira de Souza

Florianópolis-SC
2006/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

OBTENÇÃO DE DADOS DE QoS DOS ELEMENTOS DE UM GRID MÓVEL

Rudson Vieira de Souza

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis – SC
2006/2

Rudson Vieira de Souza

OBTENÇÃO DE DADOS DE QoS DOS ELEMENTOS DE UM GRID MÓVEL

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Orientador(a): Carlos Becker Westphall

Banca examinadora

Carla Merkle Westphall

Fernando Augusto da Silva Cruz

Dedico este trabalho à minha família, que sempre me proveu tudo aquilo que necessitava.

Agradecimentos

Agradeço à minha família, por sempre estar ao meu lado concedendo todo o apoio necessário.

À Universidade Federal de Santa Catarina pela oportunidade a mim conferida de me graduar e obter conhecimento.

Aos professores do Curso de Sistemas de Informação, por toda a paciência e dedicação quando ministram suas aulas.

Ao meu orientador, Professor Carlos Becker Westhpall e a sua esposa e membro da banca Professora Carla Merkle Westphall, pelo apoio, direcionamento, paciência, e principalmente por acreditar em minha capacidade de desenvolver este trabalho.

Aos colegas de curso, por transformar o ambiente educacional em um ambiente familiar.

Sumário

Lista de Figuras

Lista de Reduções (abreviaturas, siglas e símbolos)

QoS - Quality of Service
CoS - Classes of Service
XML - Extensible Markup Language
XSLT - Extensible Stylesheet Language Transformation
XSL - Extensible Stylesheet Language
XSL-FO - XSL Formatting Objects
DTD - Document Type Definition
HTTP - HyperText Transfer Protocol
TCP/IP - Transmission Control Protocol / Internet Protocol
Grid-M - Middleware for Embedded and Mobile Grid Computing
HTML - HyperText Markup Language
XHTML - Extensible HyperText Markup Language
W3C - World Wide Web Consortium
API - Application Programming Interface

Resumo

O Grid-M é um projeto que visa desenvolver um middleware para aplicações do tipo Grids computacionais. Os atributos de QoS deste projeto antes da concepção deste trabalho estavam restritos aos nodos. Como alteração de tal paradigma, foi feita a criação de um sistema de coleta de atributos que incorpora os elementos de tal nodo (hooks, serviços e sensores), de modo a fomentar uma nova tendência ou metodologia na gerência de QoS em tais estruturas. Tais atributos foram dispostos em forma de código XML para complementar o modelo criado, atendendo ao padrão do projeto. Ao final do mesmo pode-se verificar a criação, mesmo que embrionária e desprovida de sofisticação, de uma nova forma de coletar atributos de QoS em Grids computacionais.

Abstract

Grid-M is a project that seeks to develop a middleware for applications of the type Grid Computing. The attributes of QoS of this project before the conception of this work were restricted to the nodes. As alteration of such paradigm, it was made the creation of a system of collection of attributes that incorporates the elements of such node (hooks, services and sensor), in way to foment a new tendency or methodology in the management of QoS in such structures. Such attributes were disposed in XML code to complement the model servant, assisting to the pattern of the project. At the end of the same can be verified the creation, even that embryonic and without sophistication, of a new form of collecting attributes of QoS in Grid Computing.

1. Introdução

Desde o início do século passado temos visto uma rápida evolução sobre os computadores tanto do hardware quanto do software. Criaram-se diversas novas funcionalidades, novos paradigmas, novas linguagens de programação, novos processadores, novas teorias.

Um dos novos paradigmas em ambientes computacionais são os Grids. Os Grids computacionais são estruturas abstratas capazes de compartilhar processamento de requisições e utilização de memória por redes de computadores. Os grids não têm fronteiras políticas vigentes e trafegam informação por todo o globo.

Entretanto, com o emprego em larga escala de redes de computadores, cresce o receio com as falhas de comunicação inerentes a estas estruturas, e é necessário o estudo de técnicas e metodologias que possam melhorar os serviços. Uma das técnicas para a melhoria dos serviços utilizada é a Qualidade de Serviço ou QoS (*Quality of Service*), que procura ao máximo atender aos requisitos de qualidade que uma aplicação deverá atingir através da definição de parâmetros aceitáveis.

Grandes projetos nas áreas de Grids aliam a reusabilidade provida por estes com o gerenciamento de QoS para obter os melhores resultados e se precaver de falhas que comprometam seu funcionamento. O projeto Grid-M não é diferente apesar de estar em fase embrionária o mesmo inspira alterações na forma de gerenciar o QoS. No intuito inicial o mesmo seria obtido como na maioria dos projetos nesta área, sobre o elemento principal - o nodo. Posteriormente foi decidido que a coleta dos atributos de QoS seria feita sobre os elementos do nodo, a fim de prover futuramente uma nova forma de gerenciamento de QoS. Esta é a motivação para o desenvolvimento deste projeto.

Este projeto tem como tema principal a obtenção de atributos de QoS dos elementos dos nodos componentes de um Grid Computacional Móvel, e posterior exibição dos mesmos para fins de gerenciamento do ambiente.

Neste trabalho será usado o Grid Computacional Móvel também denominado Grid-M (*Middleware for Embedded and Mobile Grid Computing*) desenvolvido por colaboradores do Laboratório de Redes e Gerência (LRG) desta instituição. Esta exploração será realizada de forma mais centralizada, não utilizando uma rede de computadores mais somente um único computador que proverá as simulações através de um Ambiente de Desenvolvimento, o JBuilder da Borland que vem sendo utilizado desde a concepção do projeto do Grid-M.

1.1. Objetivos

O objetivo geral deste trabalho é a obtenção dos dados de QoS de uma maneira nunca vista até o presente momento, através de cada elemento (Hook, Sensor, Serviços, etc.) pertencentes a um Nodo, ou seja será feito o gerenciamento de um Nodo através dos elementos que compõe o mesmo.

Os objetivos específicos que podem ser mencionados são:

- Compreender a construção do Grid-M;
- Auxiliar na elaboração do sistema de coleta, transmissão e gerenciamentos dos atributos de QoS dos elementos pertencentes aos nodos;
- Apresentar estes dados através de uma interface XML.

1.2. Motivação / Justificativa

Cada vez mais os computadores tangem a ruptura do seu espaço singelo e alçam conexões a redes de computadores no intuito inicial de obter informações que não estavam disponíveis nas mesmas, mas ultimamente estes vêm gozando de uma gama ínfima de serviços que surgiram nos últimos tempos, que transcendem também ao nível de hardware disponibilizando espaços fora dos computadores para armazenamento de dados e para processamento de tal com o intuito de aumentar a velocidade do processamento das informações e distribuir a carga de trabalho igualmente entre estes. Uma das propostas com este sentido são os Grids Computacionais que são estruturas distribuídas que compartilham recursos tanto de hardware quanto de software com a finalidade de distribuir as cargas de processamento entre seus componentes para agilizar e tornar mais eficiente a obtenção dos resultados esperados.

Como tais estruturas estão espalhadas por diversas localidades, até fora dos mesmos territórios, ou seja, sem fronteiras delimitadas os mesmos fazem uso de redes públicas pertencentes a provedores de serviços que estão suscetíveis assim como redes locais a falhas que ocasionam perda de informações que podem ser vitais aos processos. Com o intuito de tratar a qualidade dos dados e dos serviços das redes foi elaborado o conceito de QoS (Quality of Service – Qualidade dos Serviços) que concentra os esforços em manter um nível de serviço pré-estabelecido pelo usuário através da obtenção de métricas que embasem suas decisões sobre determinado serviço.

Sobre esta ótica a obtenção de dados para tal é parte fundamental e inicial para estipulação do que seria necessário para garantir as mesmas os patamares de funcionamento de determinado serviço em estado de excelência e condizente com os anseios de seus usuários.

1.3. Problema

O projeto do Grid-M está em fase se aprimoramento do que foi realizado até o presente momento e um dos pontos a serem revisados é a forma como este incorporará o QoS e os seus atributos ao âmago dos elementos (Hook, Sensores, Serviços, etc.) de um Nodo. Todos os elementos tenderiam a possuir um conjunto básico de atributos de QoS.

Para tal é necessária a criação de rotinas de obtenção de dados para atribuí-los aos elementos e conceber os atributos necessários que seriam coletados e armazenados em um Nodo de Gerenciamento com o intuito de disponibilizá-los a quem for cabível o gerenciamento dos Nodos componentes deste Grid.

Tais dados serão disponibilizados por uma interface em XML que tem como grande benefício a total liberdade para identificação e organização destes dados como fora concebido por seus idealizadores.

Pretende-se diante de tais fatos conceber estruturas abstratas que possam ser utilizadas de maneira singular por cada elemento para conceber os atributos de QoS destes e repassar tais ao Nodo de Gerenciamento na forma de estrutura de dados como Hashtables, e na forma de Árvores XML para posterior conversão em XML para exibição nas Interfaces.

1.4 Organização do Documento

Este documento está dividido em quatro capítulos. No primeiro capítulo foi feita a introdução do trabalho, apresentando a motivação, os objetivos, o problema e a justificativa. O segundo capítulo descreve os principais conceitos abordados neste trabalho, descrevendo e conceituando os Grids Computacionais, QoS, e por fim XML.

O que foi desenvolvido neste projeto é apresentado no terceiro capitulo, dispondo inicialmente uma explanação sobre o projeto Grid-M, os elementos que compõem os nodos (hooks, sensores e serviços), seus módulos e as funcionalidades atribuídas aos mesmos, além da apresentação do modelo de comunicação do Grid-M, e de que forma eram obtidos os atributos de QoS originalmente neste projeto. Além disto, é apresentado no restante do capitulo as classes criadas para a realização deste trabalho (classes ManagedElement e Node Manager) e como as mesmas interagem com o restante do projeto para atingir seus objetivos, descrevendo e exibindo seus métodos. Para encerrar este capitulo são descritos os métodos criados para a coleta de dados e sua exibição através de código XML, e a exibição dos testes realizados.

Por fim, no capitulo quatro são realizadas as devidas conclusões sobre o trabalho realizado, bem como, a apresentação das sugestões para futuros trabalhos acerca deste.

2. Conceitos sobre Grids Computacionais

2.1. Grids Computacionais

Um Grid Computacional é uma combinação de recursos de hardware e software com o intuito de distribuir as cargas de processamentos entre seus componentes e assim viabilizar resultados de maneira mais ágil e eficiente. O conceito de Grids possui diversas definições.

De acordo com Foster (2004, apud PY, 2005, p. 4) “O termo grid denota "uma proposta de infra-estrutura de software e hardware para a integração de recursos computacionais, dados e pessoas geograficamente dispersas de modo a formar um ambiente colaborativo de trabalho.”

Segundo Baker e Buyya (2000, apud PY, 2005, p. 5) “Considera que a população da internet a disponibilidade de computadores poderosos e de alta velocidade a baixo custo fornecem a oportunidade tecnológica de usar redes como um recurso computacional unificado.”

Krauter (2002, apud PY, 2005, p. 5) afirma que “É um sistema computacional de rede que pode escalar ambientes do tamanho da internet com máquinas distribuídas através de múltiplas organizações e domínios administrativos.”

Dentre seus objetivos estão a busca da interoperabilidade entre organizações, políticas operacionais e tipos de recursos; o acoplamento destes recursos distribuídos oferecendo acesso consistente e de baixo custo a recursos independente da sua localização física.

Foster (2002, apud PY, 2005, p. 6) afirma que “Os Grid Computacionais provêm acesso remoto, seguro e escalável a computação, dados, e outros recursos computacionais através da internet”.

Os Grids possuem uma taxonomia e classificação conforme ao que se destina o mesmo. Eles são divididos em:

- Grid Computacional – O objetivo deste é a integração de recursos computacionais dispersos para prover maior capacidade de processamento (Oportunistas - recursos ociosos);

- Grid de Dados - O objetivo deste é o acesso, pesquisa e classificação de grandes volumes de dados, potencialmente distribuídos em repositórios;

- Grid de Serviços – O objetivo deste é prover serviços disponibilizados pela integração de diversos recursos; ex: ambiente para colaboração à distância.

A figura 1 representa uma taxonomia dos Grids.

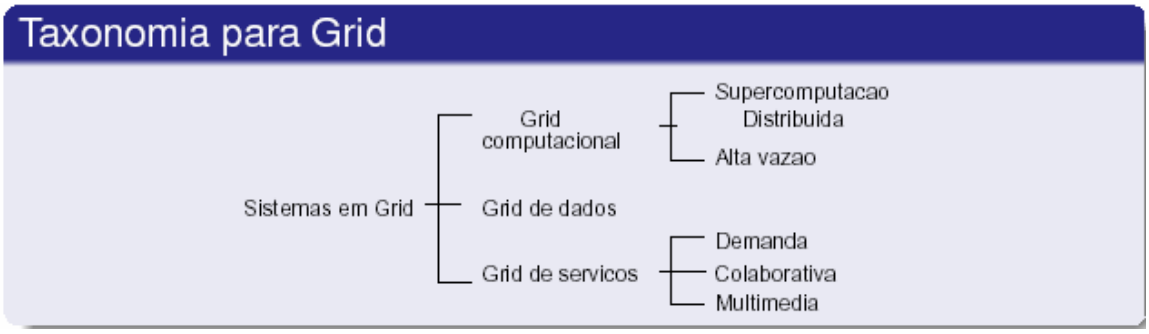

Figura 1 – Taxonomia dos Grids

Como Características dos Grids temos a heterogeneidade (multiplicidade de recursos, tratamento sobre as diferenças de processadores, velocidade e arquitetura), escalabilidade (pode crescer de poucos recursos para milhões, o problema em potencial é o desempenho a medida que o tamanho aumenta), dinamicidade ou adaptabilidade (falha de um recurso é regra, e não uma exceção, gerenciar recursos e organizar seu comportamento para extrair desempenho a partir dos recursos e serviços disponíveis), além disso possuem alta dispersão geográfica, status do sistema e tolerância a falhas, baixo custo, portabilidade das aplicações, entre outros.

A figura 2 mostra a estrutura genérica de um Grid, que pode ser dividida em:

- Fábrica - corresponde a todos os recursos geograficamente distribuídos e que podem ser acessados no Grid (computadores, estações de trabalho, clusters);

- Serviços (Middlewares) - corresponde ao fornecimento de funcionalidades básicas para a utilização do Grid (serviços de localização, alocação, escalonadores, protocolos de comunicação, autenticação, QoS). Nessa camada é que está localizado o foco deste projeto;

- Ferramentas - engloba ambientes de programação e sistemas que permitem o desenvolvimento de aplicações; funcionalidades: especificação de recursos necessários;

- Aplicações - podem ser desenvolvidas ou adaptadas através de linguagens, bibliotecas, etc.

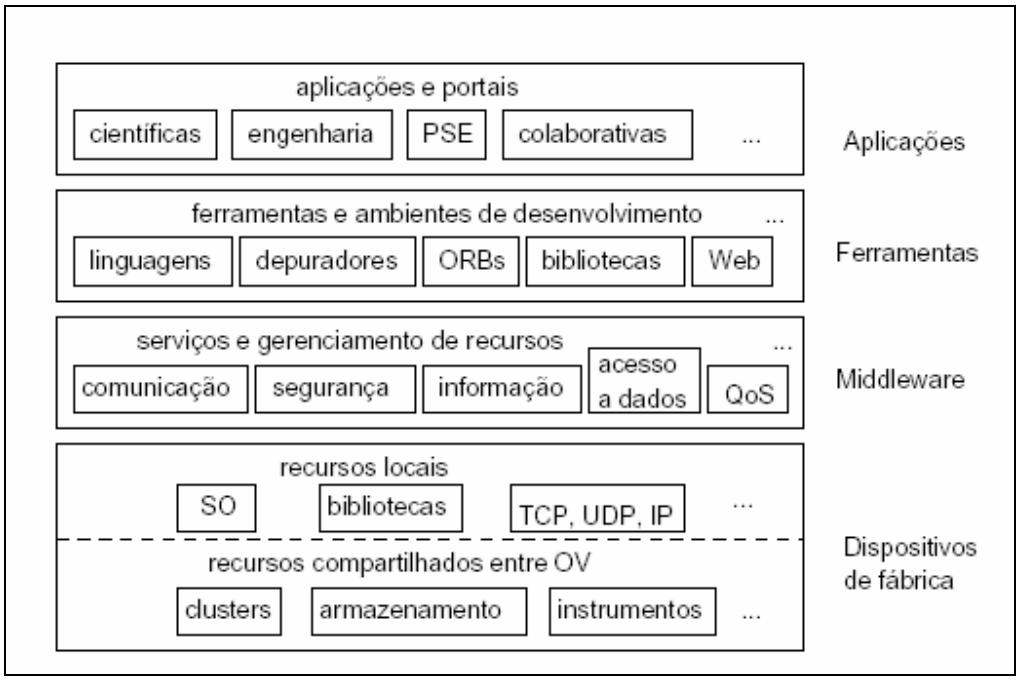

Figura 2 – Estrutura genérica de um Grid

A reusabilidade, economia, utilização eficiente dos recursos, e o compartilhamento dos recursos dedicados são algumas das vantagens da utilização de Grids. As desvantagens da utilização de tal sistema se dão com o aumento do mesmo e podem acarretar alta complexidade, elevação do custo para concepção do mesmo e dificuldade em alterar as aplicações.

Como nota de conclusão vale ressaltar o grande número de projetos nesta área, que podem ser observados na página do Grid Computing Info Centre (http://www.gridcomputing.com) que apresenta algumas de dezenas de projeto nesta área. Em âmbito nacional pode-se citar o projeto Grid Brasil (http://www.gridbrasil.com.br). Para adotar os Grids Computacionais é necessário trabalhar de forma colaborativa: usuários, sistemas e recursos podem ser integrados de modo a formar uma grande plataforma de desenvolvimento de ferramentas e resolução de problemas.

2.2. QoS

Qualidade de serviço (QoS) é um tema que possui várias definições dependendo do contexto em que é aplicado. Algumas visões diferentes sobre o conceito são identificadas na lista a seguir:

- ISO: Na visão da ISO, QoS é definida como o efeito coletivo do desempenho de um serviço, o qual determina o grau de satisfação

de um usuário do serviço. ISO/IEC (1995, apud Kamienski e Sadok, 2000, p. 4)

- Sistemas Multimídia Distribuídos: Em um sistema multimídia distribuído a qualidade de serviço pode ser definida como a representação do conjunto de características qualitativas e quantitativas de um sistema multimídia distribuído, necessário para alcançar a funcionalidade de uma aplicação. Vogel et al. (1995, apud Kamienski e Sadok, 2000, p. 4)

- Redes de Computadores: QoS é utilizado para definir o desempenho de uma rede relativa às necessidades das aplicações, como também o conjunto de tecnologias que possibilita às redes oferecer garantias de desempenho, segundo Teitelman e Hanss (1998, apud Kamienski e Sadok, 2000, p. 4). Em um ambiente compartilhado de rede, QoS necessariamente está relacionada à reserva de recursos. Essa reserva pode ser feita para um conjunto (agregação) de fluxos ou sob demanda para fluxos individuais. QoS pode ser interpretada como um método para oferecer alguma forma de tratamento preferencial para determinada quantidade de tráfego da rede. Em outra definição, QoS é vista como a capacidade de um elemento de rede ter algum nível de garantia de que seus requisitos de serviço e tráfego podem ser satisfeitos. Stardust.com (1999, apud Kamienski e Sadok, 2000, p. 4)

A junção dos termos qualidade e serviço podem dar margem a várias interpretações e definições diferentes. No entanto, existe um certo consenso, que aparece em praticamente todas as definições de QoS, que é a capacidade de diferenciar entre tráfego e tipos de serviços, para que o usuário possa tratar uma ou mais classes de tráfego diferente das demais. O modo como isso pode ser obtido e os mecanismos utilizados variam, dando origem a duas expressões freqüentemente utilizadas: Classes de Serviço (CoS) diferenciados e a mais genérica e ambígua Qualidade de Serviço (QoS). Algumas pessoas podem argumentar que os dois termos, QoS e CoS, são sinônimos, mas existem diferenças sutis. QoS tem uma conotação ampla e abrangente. CoS significa que serviços podem ser categorizados em classes, onde têm um tratamento diferenciado dos demais. O principal conceito em CoS é a diferenciação. QoS algumas vezes é utilizado em um sentido mais específico, para designar serviços que oferecem garantias estritas com relação a determinados parâmetros (como largura de banda e atraso) a seus usuários. Isso é tornar uma rede de comutação por pacotes semelhante ao sistema telefônico. Pode-se então classificar QoS de acordo com o nível de garantia oferecido:

- QoS baseado em reserva de recursos, ou rígido, que oferece garantias para cada fluxo individualmente. Esse é o tipo de QoS que seria o ideal mas é mais complexo (e caro) de implementar.

- QoS baseada em priorização, ou flexível, onde as garantias são para grupos, ou agregações de fluxos. Nesse caso, cada fluxo individual não possui garantias. CoS utiliza esse conceito, que é mais fácil de implementar, por isso mais provável de ser disponibilizado em uma rede como a Internet em um futuro próximo.

Outro componente importante para a determinação do modelo de QoS a ser fornecido aos usuários diz respeito ao tipo de tráfego que as aplicações geram e qual o comportamento esperado da rede para que elas funcionem corretamente. Com relação ao tipo de tráfego as aplicações podem ser classificadas segundo Braden, Clark e Shenker (1994, apud Kamienski e Sadok, 2000, p. 5) e Ferguson e Huston (1998, apud Kamienski e Sadok, 2000, p. 5) em :

- Aplicações de tempo real (não elásticas): Podem ser definidas como aquelas com características rígidas de reprodução (playback), ou seja, um fluxo de dados é empacotado na fonte e transportado através da rede ao seu destino, onde é desempacotado e reproduzido pela aplicação receptora. As aplicações tolerantes são aquelas que mesmo diante de variações no atraso (jitter) causadas pela rede, ainda assim produzem um sinal de qualidade quando reproduzidas. Já nas aplicações intolerantes variações no atraso produzem sinais (de áudio ou vídeo, por exemplo) com qualidade inaceitável.

- Aplicações elásticas (não tempo real, ou adaptáveis): Para esse tipo de aplicação, a recepção correta dos dados é mais importante do que a sua apresentação em uma taxa constante. Exemplos de aplicações elásticas são correio eletrônico, transferência de arquivos, consultas interativas a informações (como na Web) e aplicações cliente/servidor tradicionais.

2.2.1. Expectativas, possibilidades e limitações de QoS

A introdução de QoS está cercada de fantasias, comuns nesses casos de tecnologias inovadoras que representam mudanças de paradigmas, e que certamente irão alterar qualitativamente os serviços fornecidos pelas redes de computadores no futuro de acordo com Kamienski e Sadok (2000, p. 6). Provavelmente, a principal causa disso é a pouca experiência com QoS em redes de computadores.

- Imperfeições na rede: QoS não é uma ferramenta para compensar imperfeições na rede, como a venda maior que os recursos disponíveis (overbooking ou over-subscription), situações drásticas de congestionamento e projeto mal feito.

- Mágica na rede: QoS não é mágica, ou seja, não faz milagres. Por exemplo, QoS não altera a velocidade da luz, não cria largura de banda inexistente e não cura redes com baixo desempenho generalizado.

- A margem de QoS é pequena: Os mecanismos de QoS procuram dar preferência para classes de tráfego pré-determinadas na alocação de recursos, quando eles estão sob contenção. Em situações onde os recursos possuem capacidade ociosa, a utilização de mecanismos de QoS é irrelevante.

- Igualdade ou desigualdade: QoS é intencionalmente elitista e injusta. Alguns usuários pagam mais caro e precisam sentir claramente que têm um serviço melhor que a maioria. Em um ambiente com recursos limitados, o aumento dos recursos destinados a uma classe de serviços certamente diminui os recursos para as demais classes.

- Tipos de tráfego: QoS não funciona para todo tipo de tráfego, independente do tipo de QoS e do contrato estabelecido entre provedor e usuário.

2.2.2 Modelo de Classificação de QoS

Um modelo de QoS é útil para auxiliar a compreensão da sua abrangência e definir com precisão os requisitos específicos da aplicação desejada. Um modelo, definido em Bradner et al. (1997, apud Kamienski e Sadok, 2000, p. 6) e Teitelman e Hanss (1999, apud Kamienski e Sadok, 2000, p. 6) divide conceitualmente as abordagens para QoS em três dimensões:

- Escopo: define os limites do serviço de QoS. Um escopo fim a fim é acessível para as aplicações nos sistemas finais. Já em um escopo intermediário, os sistemas finais não requisitam diretamente a QoS que necessitam, mas são atendidos por algum elemento de rede habilitado para tal tarefa.

- Modelo de controle: descreve características relacionadas ao gerenciamento das requisições (pedidos) de QoS. Essas características são:

Granularidade: um pedido pode ser para um único fluxo entre sistemas finais, ou então para uma agregação de fluxos de uma rede inteira.

Duração: as requisições de QoS podem variar muito com relação à duração dos níveis de QoS solicitados (minutos, horas, dias, semanas, meses).

Local de controle: independente do escopo de QoS, um pedido pode ser controlado pelo sistema final, ou por algum sistema intermediário (proxy).

- Garantia de transmissão: pode ser expressa como a combinação de algumas das seguintes métricas:

  Atraso: É o tempo necessário para um pacote percorrer a rede, medido do momento em que é transmitido pelo emissor até ser recebido pelo receptor.

  Variação do atraso (jitter): É a variação no atraso fim-a-fim.

  Largura de banda: É a taxa de transmissão de dados máxima que pode ser sustentada entre dois pontos. Geralmente é vista como uma característica do enlace físico. O termo vazão é utilizado para designar a taxa máxima que alguma aplicação ou protocolo consegue manter em uma rede.

  Confiabilidade: está relacionada à perda de pacotes pela rede.

Através dos trechos retirados do texto de Kamienski e Sadok (2000) podemos ter uma noção da grande complexidade e indefinições sobre QoS além do que ela não passa de um artefato de software sem nenhuma mágica aparente. Cada qual pode ter parâmetros relevantes para determinadas aplicações. Todas as referências citadas acima são mais condizentes com aplicações para Internet, não que estes parâmetros não sirvam para Grids, mas para este tipo de aplicação os principais parâmetros são: o tempo de resposta; throughput (dados transferidos); disponibilidade; e o escalonamento das tarefas (distribuição de processamento).

2.3 XML

XML (EXtensible Markup Language) é uma linguagem para formatação que é considerada uma grande evolução por estudiosos. Esta linguagem foi desenvolvida pela W3C (World Wide Web Consortium) entidade internacional responsável pela adequação de padrões para a Internet, com o intuito de se tornar um padrão válido e largamente empregado na internet, visto as vantagens que a mesma possui sobre o HTML, superando as limitações deste.

Esta linguagem também é definida como formato universal para dados estruturados (meta-dados) na Web, pois trata de definir regras que permitem escrever estes documentos de forma que sua visualização seja possível.

Na figura 3 é exposto um exemplo de código de um documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dvd>
    <titulo>Vertigo</titulo>
    <artista>U2</artista>
    <ano>2005</ano>
</dvd>
<dvd>
    <titulo>Monkey Business</titulo>
    <artista>Black Eyed Peas</artista>
    <ano>2006</ano>
</dvd>
```

Figura 3 - Código de um documento XML

Através da figura 3 podemos observar e descrever alguns dos componentes presentes em um documento XML, que são:

- Declaração XML (em azul): mesmo não sendo obrigatória, ela ao ser inserida no inicio do documento é a melhor forma de dizer a um processador que o documento é um documento XML.

- Elementos (em verde e vermelho): são os itens básicos de um documento XML. Eles podem ser vazios não possuindo assim texto nem elementos filhos agregados, ou podem possuir conteúdo sendo organizados hierarquicamente em elementos raízes (verde) e elementos filhos (vermelho). Os elementos raízes mencionam o que será especificado, como em exemplo na figura 3 teremos informações sobre DVDs. Os elementos filhos descrevem as informações a serem obtidas do elemento raiz, no caso da figura 3 eles informam o titulo, o artista e o ano de cada DVD.

Além destes podemos enumerar outros componentes de um documento XML, como:

- DTD (Document Type Definition) ou Schema: Apesar de não estar presente na figura 3, eles especificam como um documento XML deve ser interpretado e renderizado. Ambos definem a estrutura do documento com uma lista dos elementos permitidos. O XML Schema é o sucessor do DTD, pois o mesmo é extensível para futuras adições, mais sofisticado que o DTD, são escritos em XML, além de suportar data types e namespaces. Vistos nas figuras 4 e 5, respectivamente.

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Figura 4 - Exemplo de DTD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
```

Figura 5 - Exemplo de Schema

- Atributos: São utilizados para fornecer informações adicionais sobre um elemento, devem ser sempre cercados por aspas (simples ou duplas). Não existe um consenso entre se utilizar atributos ao invés de elementos filhos, porém recomenda-se o uso de atributos somente para informações que não são relevantes para os dados. Alguns problemas que surgem com a utilização de atributos que não ocorrem quando se utiliza elementos são que estes não podem conter múltiplos valores, não podem descrever estruturas, e são mais difíceis de manipular via código. A figura 6 exibe um trecho de código XML com atributos.

```
<empregado sexo="masculino">
<nome>Fernando</nome>
</empregado>

<empregado>
<sexo>masculino</sexo>
<nome>Fernando</nome>
</empregado>
```

Figura 6 – Trecho de código XML com atributos

Cabe ressaltar também os conceitos de XSLT e XSL. XSL (EXtensible Stylesheet Language) também desenvolvido pelo W3C surgiu devido a necessidade de uma linguagem de estilo baseada em XML. Ele descreve como um documento XML deve ser exibido. Este processo segundo Freitas e Barros (2004, p. 3) consiste em dois estágios:

- Transformação Estrutural – os dados são convertidos de uma estrutura do documento XML para uma outra estrutura de saída. Envolve seleção, agregação, ordenação, conversão de dados, etc.

- Formatação – um formato requerido (por exemplo HTML ou PDF) é aplicado à árvore resultado.

Tais estágios foram reconhecidos como independentes, o que resultou na divisão de XSL em:

- XSLT (EXtensible Stylesheet Language Transformations) – para definição das transformações;

- XSL-FO: (XSL Formatting Objects) – para formatação – adiciona informações de apresentação aos elementos;

Além destes, existe mais uma parte que compõe um documento XSL que é responsável pela navegação nos documentos XML, denominado XPath. A figura 7 exibe um código de um documento XSL.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match='/'>
    <html>
    <head>
    <title>Relação de Clientes</title>
    </head>
    <body bgcolor="#FFFFF">
      <h1>Relação de Clientes</h1>
      <xsl:apply-templates select='clientes'/>
    </body>
    </html>
  </xsl:template>
...
</xsl:stylesheet>
```

Figura 7 – Código de um documento XSL

Das partes componentes do XSL, a mais importante é O XSLT que é usado para transformar um documento XML em outro documento XML, ou outro tipo de documento reconhecido por um browser, como HTML ou XHTML. Ele realiza tal transformação, transformando cada elemento XML em um elemento (X)HTML. Podemos mencionar que XSLT transforma uma árvore-fonte em XML em uma árvore-resultado em XML.

Através do XSLT pode-se adicionar ou remover elementos e atributos do arquivo de saída, re-alocar ou classificar elementos, realizar testes e tomar decisões sobre qual elemento deve ser exibido ou escondido, entre outras funcionalidades. W3Schools (www.w3schools.com)

3. Metodologia / Desenvolvimento do Trabalho

Como o projeto Grid-M dispõe de um arquivo de projeto para o JBuilder da Borland, este foi utilizado para o desenvolvimento deste projeto, pois ele contém toda estrutura necessária para a concepção das atualizações preteridas, além de fornecer ambiente para os devidos testes.

A utilização da linguagem Java e do paradigma de programação OO (Orientado a Objeto), seguem os padrões utilizados no projeto Grid-M.

Antes da apresentação sobre as alterações realizadas para alcançar o objetivo geral deste trabalho, será descrito o projeto Grid-M.

3.1. Grid-M

O projeto do Grid-M foi desenvolvido pelo Laboratório de Redes e Gerência (LRG) desta instituição de ensino. Seu conceito é o de uma plataforma para construção de aplicações para Grids Computacionais que possuem dispositivos móveis para comunicação, seja para a obtenção de dados para processamento, seja para a disposição do mesmo para consultas. Ele prove uma API para ser utilizada por aplicações Java no ambiente de Grids.

Grids Computacionais provêm os protocolos para possibilitar interoperabilidade e infra-estrutura em comum. No contexto deste projeto, os Grids são plataformas para padronizar computação distribuída.

A falta de padrões para plataformas de desenvolvimento para computação distribuída levou a criação de soluções ad hoc com custos para desenvolvimento e para reusabilidade. Os Grids possuem a vantagem neste contexto de oferecer reusabilidade considerando que o mesmo disponibiliza padrões para a construção de ferramentas para a transferência de serviços em dispositivos móveis e embutidos. Esses padrões tangem a padrões abertos, para protocolos e interfaces (para comunicação de dados), ferramentas de gerenciamento, padronização de interfaces programáveis.

O elemento principal deste projeto é o nodo, estrutura definida no contexto de Grids como sendo um ponto de união entre varias redes. Tais nodos possuem elementos que neste projeto são os:

- Hooks – Responsáveis pela requisição de serviços tem seu conceito interligado ao conceito de tarefas. Geralmente enviam as tarefas a um nodo que dispõe do serviço procurado, mas também podem ser utilizados no próprio nodo;

- Serviços – Determinam o que os nodos podem ou não oferecer aos demais. São serviços oferecidos neste projeto: o serviço de autenticação, registro, encontrar nodos, encontrar rotas, entre outros. A lista de serviços disposto por cada nodo pode ser verificado pelo Diretório de Serviços inerente ao mesmo.

- Sensores – Responsáveis pela coleta dos dados. Podem coletar tanto informações do nodo ao qual foi adicionado, bem como, de dispositivos externos acoplados ao nodo. No projeto Grid-M estes dispositivos são de comunicação móvel, como antenas, PDAs, entre outros. Além de coletar dados quando necessário, um sensor pode obter dados através do agendamento das tarefas em tempos estipulados pelo usuário.

3.1.1. Módulos e suas funções no Grid-M

O Grid-M é composto por sete módulos como podem ser vistos na figura 8, todos estes estão inseridos em um Nodo. Uma descrição de cada um deles é feita na seqüência.

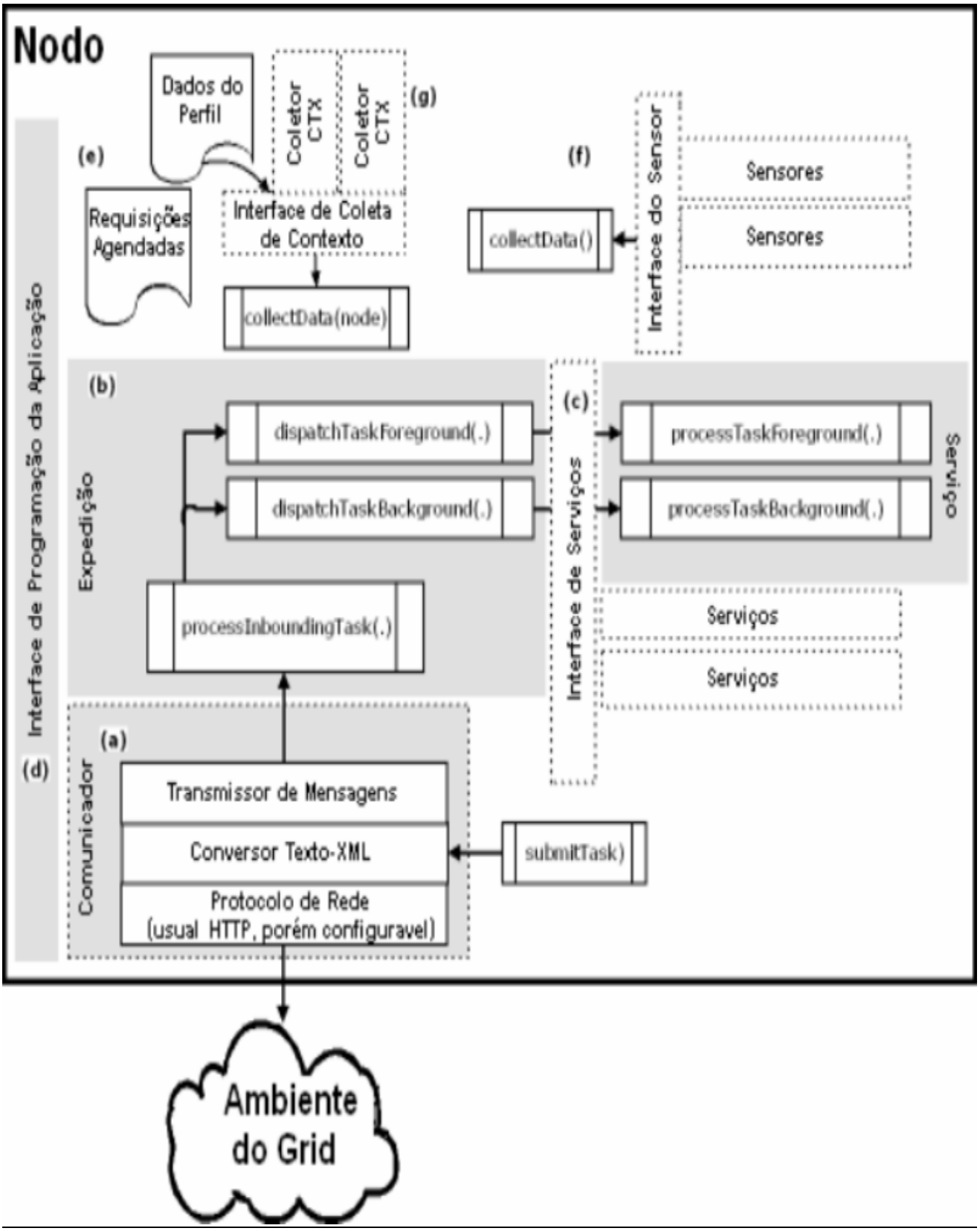

Figura 8 – Nodo e Módulos componentes

(a) Comunicação (COMMUNICATION module): Contém as sub-rotinas para comunicação entre os nodos. O protocolo escolhido foi o HTTP sobre TCP/IP. É esperado que os nodos sejam capazes de realizar requisições por clientes HTTP (ou através de um proxy para aqueles nodos não equipados com pilhas TCP/IP). Nodos podem receber pacotes através de uma porta de um servidor HTTP (que fica aguardando na porta default do Grid Middleware) ou os pacotes são enviados por um servidor HTTP que os redireciona para um outro nodo com cliente HTTP devido a uma indisponibilidade em outro nodo requerido. Este então envia tais solicitações. Este módulo ainda contém as sub-rotinas para converter XML em estruturas de dados interna e a interface para o módulo de expedição.

(b) Expedição (DISPATCHER module): Contém as sub-rotinas para expedir mensagens recebidas (que já foram transformadas em estruturas de dados interna) para os serviços disponíveis. Existe um perfil de serviços obrigatório para cada nodo. Novos serviços podem ser adicionados através da interface de programação. A expedição é baseada em parâmetros de desempenho contidos nas mensagens recebidas. Tais parâmetros são associados com novos serviços durante o processo de associação.

(c) Interface de Serviços (SERVICE INTERFACE): permite a implementação de serviços associados. Este módulo gera call-backs para os pontos de entrada (entry points) de execução do serviço.

(d) Interface de Programação da Aplicação (APPLICATION PROGRAMMING INTERFACE): permite a conexão de aplicações externas. Aplicações Java são implementadas sobre este módulo. Ele pode utilizar os Executores de Serviço para receber as chamadas feitas pelo nodo e implementar respostas para requisições.

(e) Módulo de Agendamento e Requisições Agendadas (SCHEDULED REQUESTS and SCHEDULER MODULE): é parte integrante da implementação do nodo e permite a programação de requisições geradas internamente em períodos de tempo estabelecidos. A programação de tais requisições é feita através do módulo anterior (d).

(f) Interface do Sensor (SENSOR INTERFACE): permite adicionar sensores com o intuito de coletar dados de módulos externos que eventualmente podem se conectar a hardwares externos.

(g) Interface de Coleta de Contexto (CONTEXT COLLECTOR INTERFACE): permite adicionar novos coletores de dados de contexto através dos métodos Node.addProfileContextCollector(ProfileDataCollectorInterface collector) e addProfileStaticCollector(ProfileDataCollectorInterface collector). A classe Node implementa o SensorInterface que coleta seus

próprios dados. A coleta e o envio destes para o Service Provider podem ser agendados através do método Node.scheduleContextCollect(int intervalSec), onde o parâmetro intervalSec denota os segundos entre cada coleta.

3.1.2. Modelo de Comunicação do Grid-M

Existem dois tipos de processamento de requisições no Grid-M, ambos visualizados na figura 9: IMMEDIATE e DELAYED.

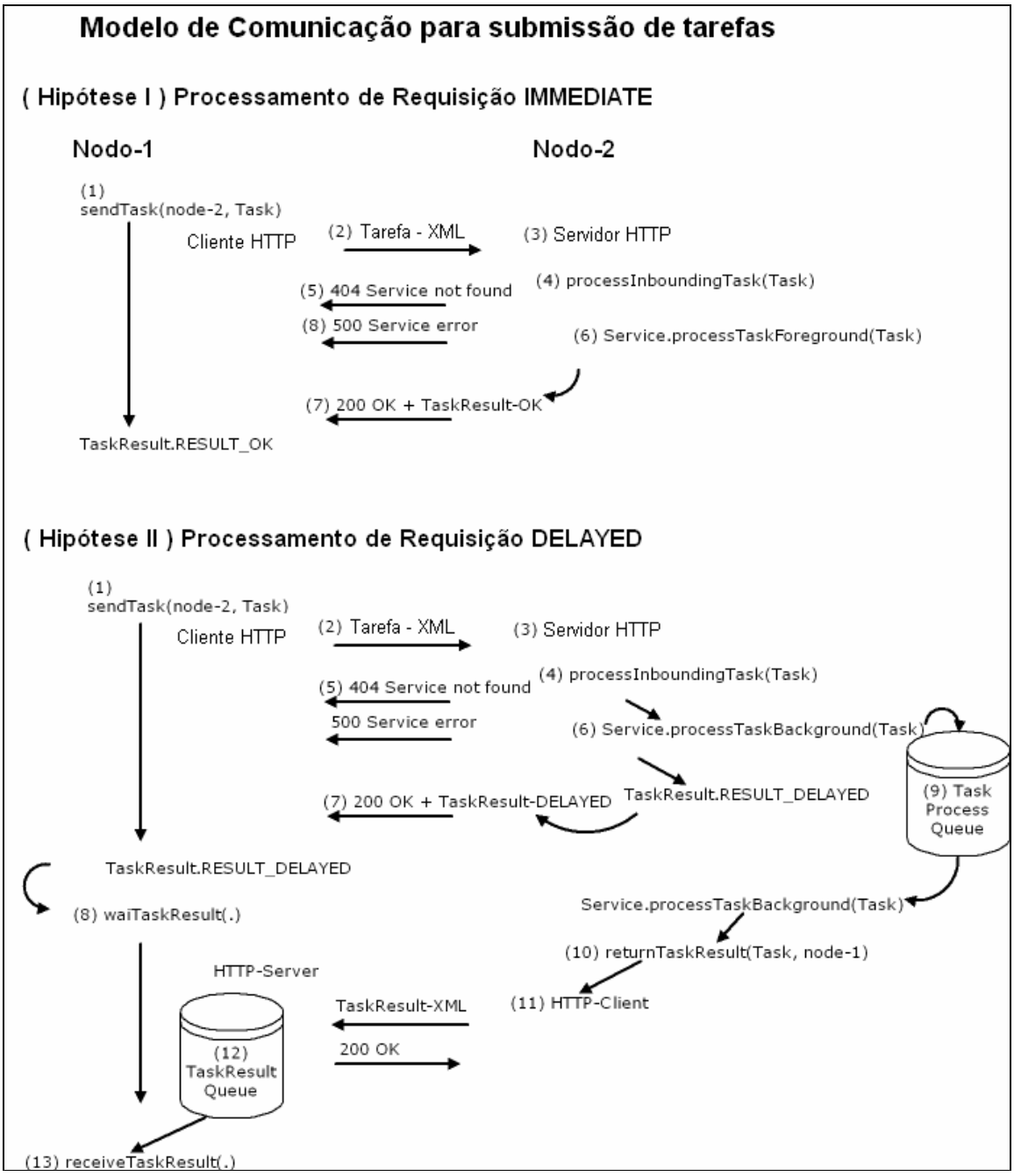

Figura 9 – Modelo de Comunicação para Requisições

O processamento é IMMEDIATE quando o serviço processa a requisição imediatamente e retorna o TaskResult (resultado da tarefa) através da mesma conexão. O processamento de requisições IMMEDIATE trabalha da seguinte forma:

1- O nodo-1 envia a tarefa para ser processada no nodo-2: método sendTask(node-2, Task).

2- A tarefa é codificada em XML e transmitida para o nodo-2 por uma requisição HTTP através da criação da conexão-1 pelo cliente HTTP.

3- O nodo-2 recebe a Tarefa-XML através do servidor HTTP, decodifica de XML para a representação da Tarefa e submete esta para processamento através do método processInboundingTask(Task).

4- Este método (3) encontra a ServiceInterface do serviço para processar a Tarefa.

5- Se o serviço não esta registrado no nodo-2, é retornado uma resposta em HTTP (404 NOT FOUND/{empty}) para a conexão-1.

6- O método processInboundingTask(Task) tenta submeter como um processo em primeiro plano (Foreground) utilizando o método Service.processTaskForeground(Task).

7 - Se o método anterior retornar TaskResult.OK então este codifica o resultado em XML e retorna através de uma resposta HTTP (200 OK / TaskResult-XML) para a conexão-1.

8 - Caso ocorra uma exceção durante o processamento, é retornado (500 SERVICE ERROR /{empty}) para a conexão-1.

A desvantagem do processamento IMMEDIATE é no caso de ocorrerem múltiplas tarefas sendo processadas simultaneamente e/ou o processamento da tarefa leva muito tempo, mantendo a conexão aberta e fazendo com que o cliente aguarde por esta por este período de tempo.

O processamento é DELAYED como o próprio nome sugere, quando este sofre atraso (Delay) para retornar a resposta. Até o passo 6 este se comporta igualmente ao processo IMMEDIATE. Os demais passos realizados por este processo são:

7- Se o método Service.processTaskForeground(Task) retorna DELAYED (mencionando que o processo não pode ser executado em primeiro plano (Foreground)), então a tarefa é adicionada a fila do TASK PROCESS QUEUE e uma resposta HTTP (200 OK, TaskResult.DELAYED) é retornada através da conexão-1 e este é fechada.

8- No nodo-1, TaskResult.DELAYED é recebido e o código é enviado ao método waitTaskResult(Task). Porém algum processamento pode ser realizado enquanto se aguarda o retorno do resultado da tarefa pendente.

9- No nodo-2, o método processInboundingTask(.) inicia uma thread (que ainda não está rodando) de processamento em segundo plano (Background) que processa aquela tarefa adicionada a fila no TASK PROCESS QUEUE utilizando o método Service.processTaskBackground(Task).

10- Quando a tarefa é processada pelo método processTaskBackground(Task), este retorna TaskResult para o nodo-1.

11- O nodo-2 abre uma conexão-2 através de um cliente HTTP com o nodo-1 e envia o resultado codificado em XML.

12- No nodo-1, TaskResult é adicionado a TASKRESULT QUEUE e o método waitTaskResult(Task) indica o recebimento do mesmo.

13- O nodo-1 recebe o resultado chamando o método receiveTaskResult(Task.id).

A vantagem do processamento DELAYED é que a conexão utilizada para submeter a tarefa para processamento permanece aberta por um pequeno período de tempo, somente até a transferência da tarefa.

3.2. Como eram obtidos originalmente os atributos de QoS

Antes da realização deste trabalho os atributos de QoS obtidos informavam somente a situação do nodo em geral. Tais atributos contidos na classe Node levantavam os parâmetros a respeito da execução do mesmo, distinguindo os parâmetros em relação ao tipo de processamento (Foreground ou Background), quantas vezes os mesmos foram executados em cada tipo (tasks_processed_fg, tasks_processed_bg), o tempo gasto com o processamento destes (tasks_processed_fg_time, tasks_processed_bg_time), quantas vezes os mesmos falharam (tasks_failed), e quantas vezes fora acionada a thread para processamento de tarefas na fila de processamento em Background (task_processor_counter).

Durante a revisão desta primeira implementação surgiu a necessidade de coletar os atributos não do nodo em geral, mas sim dos elementos que compõe o mesmo (Hooks, Sensores e Serviços), com o intuito de se criar uma plataforma de gerenciamento de QoS em Grids computacionais, já que a literatura não possui referência a outro ambiente que considere estas questões. Tais atributos deveriam utilizar rotinas e valores básicos fornecidos por uma classe abstrata, de quem estenderiam estes valores e rotinas.

3.3. A classe ManagedElement

A classe abstrata ManagedElement foi criada para prover aos elementos do nodo (classe Node) os valores e rotinas para obtenção dos atributos de QoS. Como tais elementos estão inseridos na classe Node, esta passou a estender a classe ManagedElement.

Para cada elemento foi criado uma série de valores condizentes com a operação de cada um no nodo. Para os hooks (tarefas encaminhadas a outro nodo) foram criados valores que informem quantas vezes o mesmo foi executado (hook_executed), quantas vezes o mesmo conseguiu obter resultado para determinada solicitação (hook_processed), o tempo médio de resposta para as requisições (hook_average_answer_time), quantas vezes o mesmo falhou (hook_failed), e quais erros foram retornados (hook_failed_id).

Como os sensores têm até então somente o intuito de coletar as informações do perfil do nodo, foram criados valores para o mesmo que informam quantas vezes ele foi acionado (sensor_executed), quantas vezes essa execução não se deu de forma agendada (sensor_executed_normal) e quantas vezes esta coleta foi obtida através de agendamento (sensor_executed_scheduled).

Os serviços são os elementos que possuem a implementação mais completa neste projeto. Para os serviços foram criados valores que remetem ao processo original de obtenção de atributos que possuía como parâmetro divisor a forma de processamento. Assim sendo os valores dos serviços informam o numero de vezes que o serviço foi executado (service_executed), quantas vezes ele fora executado IMMEDIATE (processed_immediate), quantas vezes este foi processado DELAYED (processed_delayed), o tempo médio gasto com cada processamento (avg_time_processed_immediate e avg_time_processed_delayed), o tempo médio de execução de cada serviço (service_avg_time) e quantas falhas ocorreram para cada serviço (failed).

Com estes valores foi possível construir as rotinas de criação, atualização e obtenção dos atributos de QoS dos elementos de um nodo. Os métodos createHookAttribute(String nodeId, String hookId), createSensorAttribute(String nodeId, String sensorId), e createServiceAttribute(String nodeId, String serviceId) como os nomes já preconizam, inicializam a coleta dos atributos, instanciando os valores e os armazenando em um Vetor que por sua vez é armazenado dentro de um Hashtable. Como parâmetros passados aos mesmos estão a identificação do nodo e do elemento requerido. Na figura 10 é mostrado como exemplo o método createHookAttribute.

```
public void createHookAttribute(String nodeId, String hookId){

    String name = nodeId + "_" + hookId;

    Vector ahook = new Vector();
    hook_executed = 0;
    hook_processed = 0;
    hook_average_answer_time = 0L;
    hook_failed = 0;
    hook_failed_id = null ;

    ahook.add(0, Integer.toString(hook_executed));
    ahook.add(1, Integer.toString(hook_processed));
    ahook.add(2, Long.toString(hook_average_answer_time));
    ahook.add(3, Integer.toString(hook_failed));
    ahook.add(4, hook_failed_id);

    hook.put(name, ahook);

}
```

Figura 10 – Método createHookAttribute

Quando um dos elementos é utilizado os métodos updateHookAttribute(String nodeId, String hookId, boolean failed, long time, String failure), updateSensorAttribute(String nodeId, String sensorId, boolean scheduled), e updateServiceAttribute(String nodeId, String serviceId, String processed_id, long time) são chamados para atualizar os seus valores. Observa-se que os parâmetros repassados ao mesmo condizem com seus valores. São repassados a identificação do nodo e do serviço, além dos seguintes dados: aos hooks são repassados ainda um valor lógico (booleano) para verificar se a requisição foi enviada com sucesso (failed), um tipo inteiro longo (long) para repassar o tempo de resposta (time) e um String que repassa o erro encontrado (failure); aos sensores um valor lógico (booleano) que afere se a coleta do mesmo foi agendada (scheduled); aos serviços um String que menciona a forma de processamento (foreground ou background), e um tipo inteiro longo (long) para o tempo de processamento do serviço (time).

Como os métodos de criação, os métodos de atualização incrementam os devidos valores retornando-os ao Vetor para serem armazenados e, por conseguinte armazenados ao Hashtable. Como exemplo é exibido na figura 11 o método updateSensorAttribute.

```
public void updateSensorAttribute(String nodeId, String sensorId, boolean scheduled){

    String name = nodeId + "_" + sensorId;

    if (sensor.containsKey(name)) {
        Vector asensor = (Vector) sensor.get(name);
        if(!scheduled){
            Integer a = new Integer((String) asensor.get(0));
            int b = a.intValue() + 1;
            Integer c = new Integer((String) asensor.get(2));
            int d = c.intValue();

            asensor.set(0, Integer.toString(b));
            asensor.set(1, Integer.toString(b - d));
        }
        else if(scheduled){

            Integer a = new Integer((String) asensor.get(2));
            int b = a.intValue() + 1;

            asensor.set(2, Integer.toString(b));
        }
    }
}
```

Figura 11 – Método updateSensorAttribute.

Como rotina de retorno dos atributos de QoS foi implementado o método getQoSAttributes(String nodeId, String serviceId, String sensorId, String hookId) da Interface QoSAttributes. Como visto os parâmetros passados para este são as identificações do nodo e dos elementos. O elemento utilizado no momento corrente passa sua identificação para este método anulando a entrada dos demais elementos, ao executar o método os valores contidos no Hashtable são repassados a um vetor, e os elementos deste são transformados em uma XMLTree (uma árvore XML concebida através da extensão que a classe ManagedElement faz da classe XMLTree que possui todos os métodos e mecanismos necessários para gerar tal árvore) retornando tal estrutura ao objeto da classe solicitante. A figura 12 apresenta trecho do código do método getQosAttributes.

```
public XMLTree getQoSAttributes(String nodeId, String serviceId, String sensorId, String hookId){

    if (serviceId != null){

        String name = nodeId + "_" + serviceId;

        if (service.containsKey(name)){
            Vector aservice = (Vector) service.get(name);

            XMLTree aserviceTree = new XMLTree(serviceId);

            aserviceTree.put("Service_Executed",String.valueOf(aservice.get(0)));
            aserviceTree.put("Service_Executed_Immediate",String.valueOf(aservice.get(1)));
            aserviceTree.put("Service_Executed_Delayed",String.valueOf(aservice.get(2)));
      aserviceTree.put("Service_Average_Time_Executed_Immediate",String.valueOf(aservice.get(3)));
      aserviceTree.put("Service_Average_Time_Executed_Delayed",String.valueOf(aservice.get(4)));
            aserviceTree.put("Service_Average_Time_Executed",String.valueOf(aservice.get(5)));
            aserviceTree.put("Service_Executed_Failure",String.valueOf(aservice.get(6)));

            return aserviceTree;

        }
```

Figura 12 – Método getQoSAttributes

Estas XMLTrees são armazenadas em uma classe denominada NodeManager que além de armazenar, posteriormente exibirá o código XML pertinente aos atributos de QoS dos elementos do nodo.

3.4. A classe NodeManager

Com o intuito de armazenar as XMLTrees retornadas pelo método getQoSAttributes e posteriormente exibir o código XML dos atributos de QoS dos elementos do nodo, foi criada a classe NodeManager que realiza extensão a classe XMLTree e implementa a interface XMLInterface. Tal extensão permite que a mesma possa incorporar os atributos e executar os métodos para construção de árvores XML que por sua vez serão transformadas em código XML pelo método generateXML() pertencente a interface XMLInterface.

Para esta classe foram criados métodos para adicionar a XMLTree dos elementos e organizá-las em árvores maiores que compõe todos os serviços, sensores e hooks para cada nodo, tendo como elemento pai a tag Node_Manager. Tais métodos são addServiceSubTree(String nodeId, String serviceId,XMLTree service), addSensorSubTree(String nodeId, String sensorId,XMLTree sensor) e addHookSubTree(String nodeId, String hookId,XMLTree hook). Assim como na classe ManagedElement os parâmetros são a identificação do nodo e do elemento bem como a XMLTree pertencente a este elemento. Estas XMLTrees são adicionadas ao Hashtable pertence a cada elemento que por sua vez é armazenado no vetor no local pré-disposto para tal e finalmente inseridos em um Hashtable que armazena todas as informações do nodo vigente. Como exemplo é exibido na figura 13 o método addServiceSubTree.

```
public void addServiceSubTree(String nodeId, String serviceId,XMLTree service)
                                                    throws NullPointerException{

      if (nodes.containsKey(nodeId)){

          if (services.containsKey(serviceId)){
              services.remove(serviceId);
              services.put(serviceId, service);
          }
          else{
              services.put(serviceId, service);
          }

          nodeInfo.set(2, services);

      }
      else{
          try{
             nodes.put(nodeId, nodeInfo);
             this.addServiceSubTree(nodeId, serviceId, service);

         }catch (NullPointerException ex) { }

      }
```

Figura 13 – Método addServiceSubTree

Para a geração das árvores XML, são retornadas as árvores de cada elemento pertencente ao nodo. Estas árvores são adicionadas ao elemento pai do nodo para a exibição do código XML do mesmo.

Os métodos que retornam as árvores dos elementos são: getHooksTree(String nodeId), getServicesTree(String nodeId), e getSensorsTree(String nodeId), que possuem como único parâmetro a identificação do nodo. Nestes métodos são retornadas todas as XMLTrees armazenadas nos Hashtables de cada elemento através do método XMLTree.hashToXMLTree("Sensors", sensors); que monta uma XMLTree com a tag inserida no primeiro parâmetro (e. e. “Sensors”) como nó pai e as XMLTrees dos elementos como nós filhas. O método getSensorsTree é exibido na figura 14 como exemplo.

```
public XMLTree getSensorsTree(String nodeId){

    if (nodes.containsKey(nodeId)){

        sensorTree = new XMLTree();
        sensorTree = XMLTree.hashToXMLTree("Sensors", sensors);

    }
    else{
        try{


       }catch (NullPointerException ex) { }

    }


    return sensorTree;
}
```

Figura 14 – Método getSensorsTree

Para a construção da XMLTree completa que será usada para a exibição do código XML será invocado o método generateXML() que adiciona aos nós Node_Manager e node criados pelo construtor da classe as demais árvores geradas pelos métodos explicitados anteriormente. Tal invocação será realizada pelo método addXSLTInterface implementado na classe HTTPServerNode.

3.5. Coleta dos Dados e Exibição de Código XML

Para realizar a coleta proposta neste trabalho seria necessário obter os atributos de cada elemento, porém como exposto anteriormente nenhum dos elementos pertencentes a um nodo está em uma classe separada, todos os elementos estão concentrados na classe Node, que implementa os métodos e os valores para estes elementos. Sendo assim a classe Node realiza extensão da classe ManagedElement como mencionado anteriormente com o intuito de obter as rotinas e valores para a finalidade desejada.

Para a criação dos valores dos elementos foram adicionadas entradas dos métodos de criação (e. e. createSensorAttribute) nos métodos criadores dos elementos: addService, addSensor e sendTask. Como exemplo, a figura 15 apresenta o método addService.

```
public final void addService(ServiceInterface service) {
    if (shouldLog("node", Logger.BASIC)) {
        log("node", Logger.BASIC,
            "Node.addService(.) " + name + ": service=" + service);
    }

      XMLTree servicesTree = new XMLTree("services");

    // add to local directory of services
    String id[] = service.getServiceIds();
    if (id != null) {
        for (int i = 0; i < id.length; i++) {

            // tracing
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "Node.addingService=>" + id[i]);
            }

            this.services.put(id[i], service);

            super.createServiceAttribute(name, id[i]);
        }
```

Figura 15 – Método addService invocando (em azul) a criação dos atributos de um serviço

Para a atualização destes valores os métodos de atualização (e. e. updateSensorAttribute) foram inseridos nos métodos que executam as ações pertinentes aos elementos. São eles: collectSensor, scheduleSensorCollect, sendTask, dispatchTaskForeground, dispatchTaskBackground,e a therad run (que processa as requisições DELAYED inseridas na fila). Na figura 16 pode-se observar como exemplo o método sendTask.

```
public final TaskResult sendTask(Task task) throws IOException {
    TaskResult result;

    super.createHookAttribute(this.name, task.service);

    try {
        // tracing
        if (shouldLog("node", Logger.BASIC)) {
            log("node", Logger.BASIC, "Node.sendTask(.) started: " + task);
        }

        // hook out to Communicator
        long start_time = System.currentTimeMillis();
        result = this.communicator.sendTask(task);
        long end_time = System.currentTimeMillis();

        super.updateHookAttribute(this.name, task.service,
                                  false, end_time - start_time, null);

        this.nodeManager.addHookSubTree(this.name,task.service,
                                        super.getQoSAttributes(this

    }
```

Figura 16 – Método sendTask atualizando (em azul) os atributos de um Hook.

Após a criação e atualização dos atributos fica faltando somente a criação das XMLTrees correspondentes. Como visto em tópicos anteriores tais estruturas são criadas pelo método getQoSAttributes implementado na classe ManagedElement e armazenados na classe NodeManager para futura exibição

em código XML. Mas para que ocorra tal interação foram inseridos nos métodos que realizam a atualização dos atributos chamadas ao método de obtenção das XMLTrees da classe NodeManager que possui como parâmetro uma estrutura na forma XMLTree, repassada a este através do método getQoSAttributes visto anteriormente. Tal interação pode ser vista na figura 17.

```
// hook out to Communicator
long start_time = System.currentTimeMillis();
result = this.communicator.sendTask(task);
long end_time = System.currentTimeMillis();

super.updateHookAttribute(this.name, task.service,
                          false, end_time - start_time, null);

this.nodeManager.addHookSubTree(this.name,task.service,
        super.getQoSAttributes(this.name, null, null, task.service));
```

Figura 17 – Interação entre os métodos para obtenção das XMLTrees

Visualizar o conteúdo dos atributos através de código XML será realizado pela classe HTTPServerNode que através de uma instância da classe NodeManager e da chamada ao método addXSLTInterface que retorna o conteúdo das XMLInterface solicitadas. Este último método possui como parâmetros o cabeçalho do endereço, o elemento pai, o caminho para um arquivo XSL e um array com as diversas XMLInterface. Sendo assim o mesmo dispõe no endereço descrito no seu parâmetro o arquivo em código XML das XMLInterfaces requeridas que foram implementadas nas classes Profiler, DirectoryService e NodeManager. Para separar e somente realizar a exibição do código XML da classe NodeManager foi criado um novo endereço “/manager/” e solicitado somente a exibição da XMLInterface pertencente a esta classe através de uma nova chamada ao método assXSLTInterface (“/manager/”, “Manager”, “manager.xsl”, XMLInterface[] {nodeManager}). O código XML dos atributos de QoS dos elementos do nodo será exibido no tópico sobre testes.

3.6. Testes

Para aferir sobre o funcionamento das implementações realizadas bem como visualizar o código XML esperado com os atributos de QoS foram realizados testes através de classes de exemplos inseridas no projeto. Não foi possível realizar testes com outras implementações devido ao projeto Grid-M ainda estar em construção não disponibilizando todas as funcionalidades previstas.

As classes usadas foram ExampleNodeRegistering e ExampleSensorNode. Na primeira é solicitado o registro de um novo nodo (nodo-1) ao nodo provider que executa tal registro. Além disto, no primeiro teste o nodo-1 agenda a coleta de informações do sensor node e tenta lançar o sinal de HEARTBEAT para informar o Directory Service que o nodo está ativo. Na segunda além do registro do novo nodo (nodo-1) ao nodo provider, são agendadas coletas nos sensores node criado pelo próprio nodo em sua inicialização e no novo sensor denominado collector,

existe uma diferença de 5 segundos entre as coletas dos sensores. Neste teste também há a criação do serviço store no nodo-1 que terá seus dados capturados pelo sensor collector. Os resultados destes testes são disponibilizados na figura 18, 19 e 20.

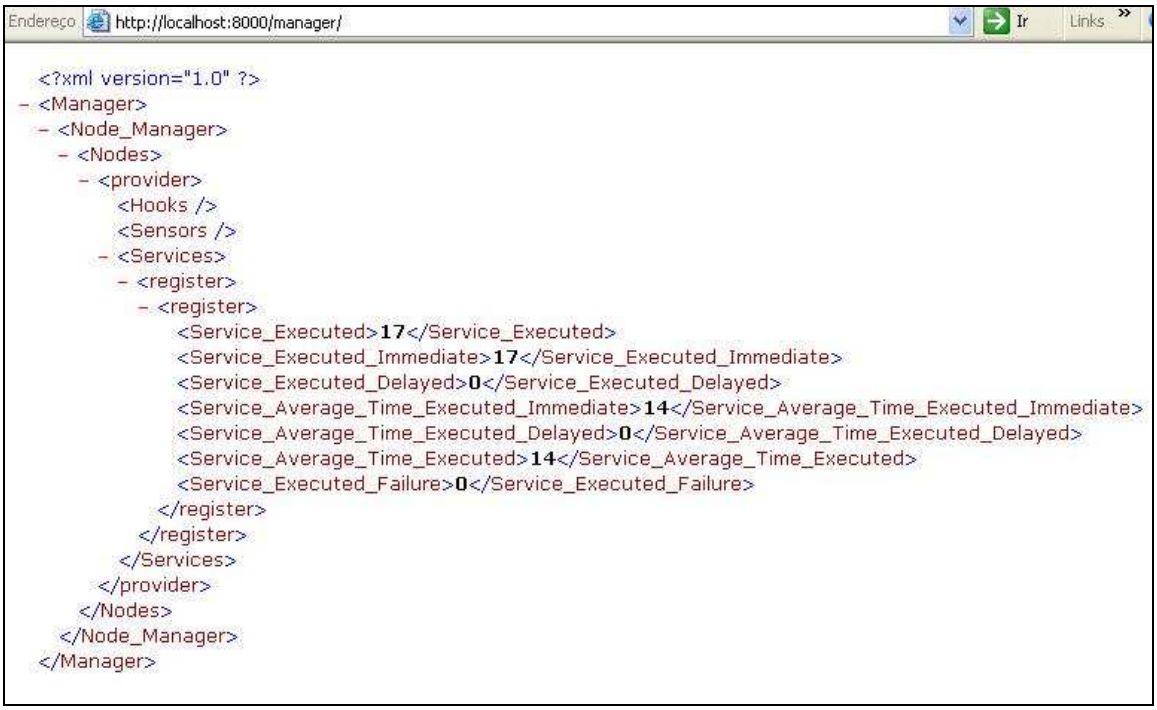

Figura 18 – Código XML do nodo provider

Como o nodo provider somente processa o serviço de registro (register), em ambos os testes aparecem somente a árvore para este serviço. Pode-se observar que os elementos hooks e sensores estão somente como folhas sem nenhum elemento agregado.

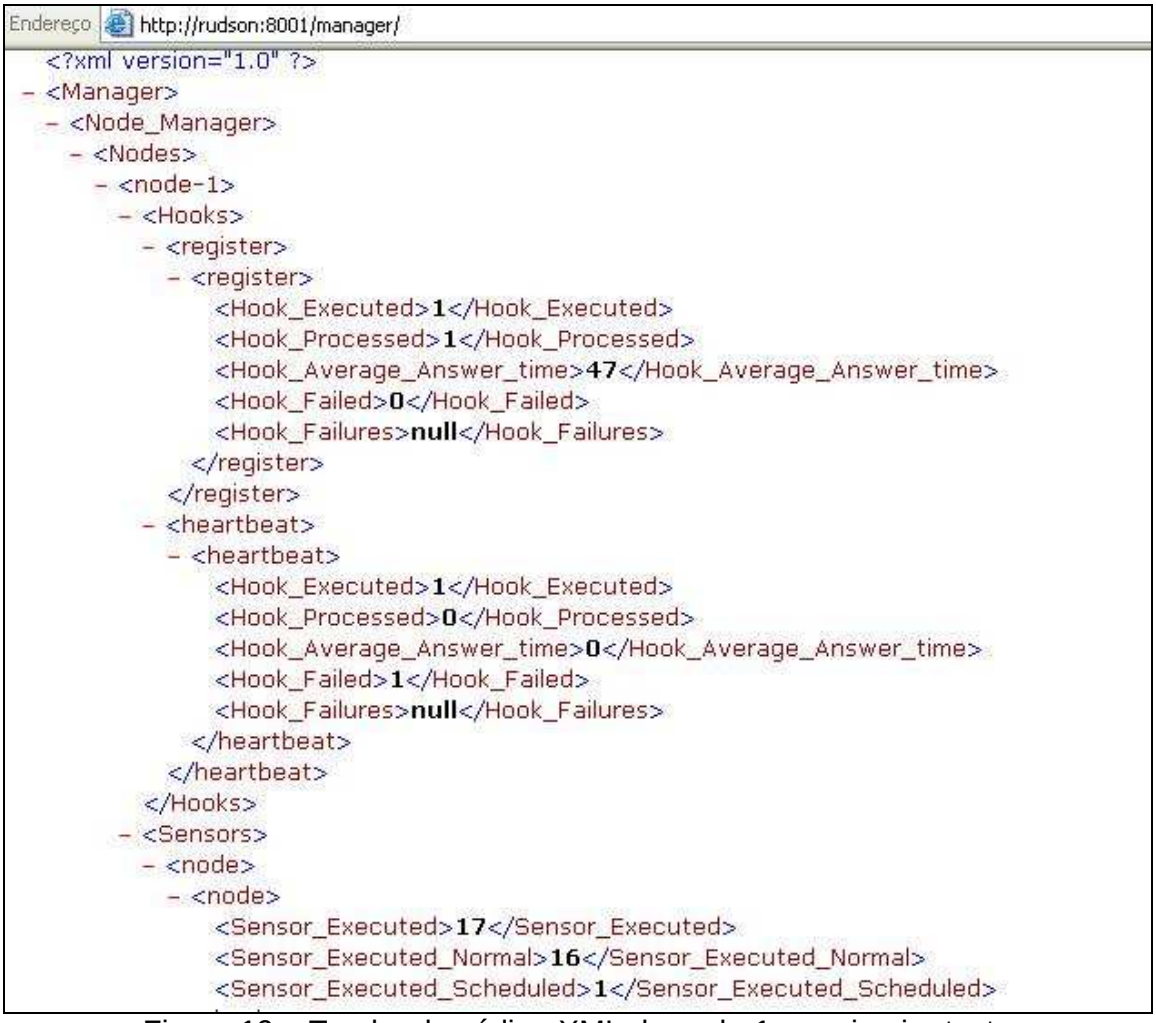

Figura 19 – Trecho do código XML do nodo-1 no primeiro teste

Neste trecho do código XML do nodo-1 no primeiro teste, na figura 19, pode-se observar claramente os dois hooks solicitados e a contabilização das coletas efetuadas pelo sensor node.

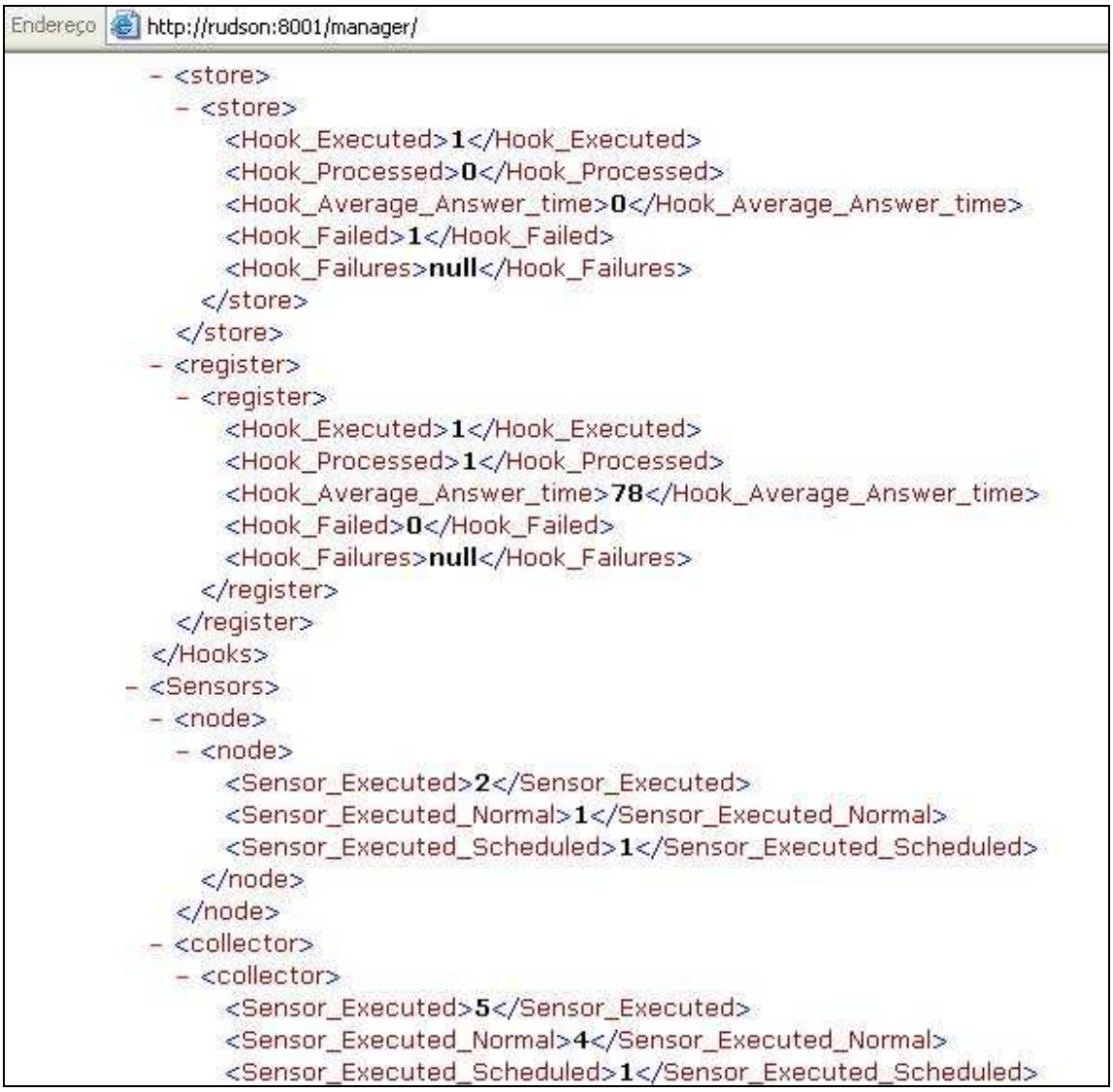

Figura 20 – Trecho do código XML do nodo-1 no segundo teste

Podemos averiguar no segundo teste - na figura 20 - a diferença no tempo das coletas dos sensores, bem como a adição do hook para solicitar o serviço store no nodo provider: como tal serviço não está implementado retornou ao hook a mensagem de falha.

Após estes testes, pode-se aferir sobre a eficácia dos métodos de coleta e atualização, que geram o código XML dos atributos de QoS dos elementos dos nodos. O único problema encontrado foi a duplicata da identificação dos elementos na árvore, fruto da inexistência de um construtor para a XMLTree que não crie nenhuma tag, perpetuando este problema.

4. Conclusão e Trabalhos Futuros

Tendo em vista a elaboração deste trabalho a principal desvantagem em se realizar um trabalho com Grids computacionais é a grande complexidade que o mesmo atinge quando está em constante crescimento. Foi necessário estudar minuciosamente um projeto muito extenso com diversas classes e muita informação.

Superada esta fase foi possível identificar facilmente os pontos que deviam ser alterados, porém tais alterações deveriam ser cuidadosamente testadas para verificar possíveis efeitos colaterais no projeto como um todo. Quando os testes retornaram os valores desejados foi possível constatar que apesar de pequeno, este trabalho pode ser um passo importante para uma nova metodologia de gerenciamento de QoS em Grids computacionais – através dos elementos dos nodos.

Este trabalho foi importante para a revisão de conhecimentos adquiridos durante o curso, bem como a atualização e aprimoramento dos mesmos. Mesmo assim, servem de alerta os problemas encontrados durante sua execução: é necessário despender de muito tempo para entender todo o projeto, e também é necessário obter o máximo de informação com os criadores dos códigos.

Como sugestão para trabalhos futuros fica a possibilidade de se criar interfaces xsl padrões para exibição dos resultados das aplicações no Grid-M. Também seria possível desenvolver um trabalho para utilizar os atributos de QoS dos elementos para fomentar uma nova metodologia de gerenciamento de QoS.

Uma última sugestão seria a criação de uma plataforma para simulação e testes em ambientes compostos por Grids computacionais, que pudesse automatizar e aumentar o potencial dos testes.

Referências Bibliográficas

KAMIENSKI, Carlos Alberto; SADOK, Djamel. **Qualidade de Serviço na Internet.** 2000. Centro de Informática, Universidade Federal de Pernambuco, Belo Horizonte, 2000.

PY, Mônica Xavier. **Grid Computacional - O que precisa-se para ter um!!.** Faculdades Rio-Grandenses, 2005.

Freitas, Bruno; Barros, Roberto S. M. **XML – XSL-FO.** Centro de Informática, Universidade Federal de Pernambuco, 2004.

Refsnes Data, **W3Schools Online Web Tutorials.** Disponível em: <http://www.w3schools.com>. Acesso em: 24 fevereiro 2007.

APÊNDICE A – Código fonte das classes criadas e alteradas

Classe ManagedElement

```
package br.ufsc.lrg.grid;

import java.util.*;
import java.lang.*;
import java.io.*;
import br.ufsc.lrg.grid.net.*;
/**
 * <p>Title: </p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: </p>
 *
 * @author not attributable
 * @version 1.0
 */
public abstract class ManagedElement extends XMLTree implements QoSAttributes {

    protected Logger logger;

    protected Hashtable service;
    protected Hashtable sensor;
    protected Hashtable hook;

    protected int hook_executed;
    protected int hook_processed;
    protected long hook_average_answer_time;
    protected int hook_failed;
    protected Vector hook_failed_id;

    protected int service_executed;
    protected int processed_immediate;
    protected int processed_delayed;
    protected long avg_time_processed_immediate;
    protected long avg_time_processed_delayed;
    protected long service_avg_time;
    protected int failed;

    protected int sensor_executed;
    protected int sensor_executed_normal;
    protected int sensor_executed_scheduled;


    public ManagedElement(){

        this.service = new Hashtable();
        this.sensor = new Hashtable();
        this.hook = new Hashtable();
        this.logger = null;

    }



    public Logger getLogger() {
        return this.logger;
    }
```

```
public void createServiceAttribute(String nodeId, String serviceId){

    String name = nodeId + "_" + serviceId;

    Vector aservice = new Vector();
    service_executed = 0;
    processed_immediate = 0;
    processed_delayed = 0;
    avg_time_processed_immediate = 0L;
    avg_time_processed_delayed = 0L;
    service_avg_time = 0L;
    failed = 0;


    aservice.add(0, Integer.toString(service_executed));
    aservice.add(1, Integer.toString(processed_immediate));
    aservice.add(2, Integer.toString(processed_delayed));
    aservice.add(3, Long.toString(avg_time_processed_immediate));
    aservice.add(4, Long.toString(avg_time_processed_delayed));
    aservice.add(5, Long.toString(service_avg_time));
    aservice.add(6, Integer.toString(failed));

    service.put(name, aservice);

}

public void createSensorAttribute(String nodeId, String sensorId){

    String name = nodeId + "_" + sensorId;

    Vector asensor = new Vector();
    sensor_executed = 0;
    sensor_executed_normal = 0;
    sensor_executed_scheduled = 0;

    asensor.add(0, Integer.toString(sensor_executed));
    asensor.add(1, Integer.toString(sensor_executed_normal));
    asensor.add(2, Integer.toString(sensor_executed_scheduled));

    sensor.put(name, asensor);

}

public void createHookAttribute(String nodeId, String hookId){

   String name = nodeId + "_" + hookId;

   Vector ahook = new Vector();
   hook_executed = 0;
   hook_processed = 0;
   hook_average_answer_time = 0L;
   hook_failed = 0;
   hook_failed_id = null ;

   ahook.add(0, Integer.toString(hook_executed));
   ahook.add(1, Integer.toString(hook_processed));
   ahook.add(2, Long.toString(hook_average_answer_time));
   ahook.add(3, Integer.toString(hook_failed));
   ahook.add(4, hook_failed_id);
```

```
        hook.put(name, ahook);

    }



    public void updateServiceAttribute(String nodeId, String serviceId, String
processed_id, long time){

        String name = nodeId + "_" + serviceId;

        if (service.containsKey(name)){
            if (shouldLog("node", Logger.DETAIL)) {
                log("node", Logger.DETAIL,
                    "Node.Managed(.) " +
                    "Passou" );
            }


            Vector aservice = (Vector) service.get(name);

            if (shouldLog("node", Logger.DETAIL)) {
                log("node", Logger.DETAIL,
                    "Node.Managed(.) " + aservice + "||" + processed_id +
                    "||" + time + "Ferrou" );
            }



            if(processed_id.equals("fg")){

                //String a = (String) aservice.get(0);
                Integer a = new Integer((String) aservice.get(0));
                int b = a.intValue() + 1;
                Integer c = new Integer((String) aservice.get(1));
                int d = c.intValue() + 1;
                Integer e = new Integer((String) aservice.get(3));
                long f = e.longValue() + time / d;
                Integer g = new Integer((String) aservice.get(5));
                long h = g.longValue() + time / b;

                aservice.set(0, Integer.toString(b));
                aservice.set(1, Integer.toString(d));
                aservice.set(3, Long.toString(f));
                aservice.set(5, Long.toString(h));


            }else if (processed_id.equals("bg")){

                Integer a = new Integer((String) aservice.get(0));
                int b = a.intValue() + 1;
                Integer c = new Integer((String) aservice.get(1));
                int d = c.intValue() + 1;
                Integer e = new Integer((String) aservice.get(3));
                long f = e.longValue() + time / d;
                Integer g = new Integer((String) aservice.get(5));
                long h = g.longValue() + time / b;


                aservice.set(0, Integer.toString(b));
```

```
                aservice.set(2, Integer.toString(d));
                aservice.set(4, Long.toString(f));
                aservice.set(5, Long.toString(h));

            }else if (processed_id.equals("failed")){

                Integer a = (Integer) aservice.get(6);
                int b = a.intValue() + 1;

                aservice.set(6, Integer.toString(b));

            }
        }

    }



    public void updateSensorAttribute(String nodeId, String sensorId, boolean
scheduled){

        String name = nodeId + "_" + sensorId;

        if (sensor.containsKey(name)) {
            Vector asensor = (Vector) sensor.get(name);
            if(!scheduled){
                Integer a = new Integer((String) asensor.get(0));
                int b = a.intValue() + 1;
                Integer c = new Integer((String) asensor.get(2));
                int d = c.intValue();

                asensor.set(0, Integer.toString(b));
                asensor.set(1, Integer.toString(b - d));
            }
            else if(scheduled){

                Integer a = new Integer((String) asensor.get(2));
                int b = a.intValue() + 1;

                asensor.set(2, Integer.toString(b));
            }
        }
    }







    public void updateHookAttribute(String nodeId, String hookId, boolean failed,
long time, String failure){

        String name = nodeId + "_" + hookId;

        if (hook.containsKey(name)) {

            Vector ahook = (Vector) hook.get(name);
            if(!failed){

                Integer a = new Integer((String) ahook.get(0));
                int b = a.intValue() + 1;
```

```
                Integer c = new Integer((String) ahook.get(1));
                int d = c.intValue() + 1;
                Integer e = new Integer((String) ahook.get(2));
                long f = e.longValue() + time / d;

                ahook.set(0, Integer.toString(b));
                ahook.set(1, Integer.toString(d));
                ahook.set(2, Long.toString(f));

            }
            else if(failed){

                Integer a = new Integer((String) ahook.get(0));
                int b = a.intValue() + 1;
                Integer c = new Integer((String) ahook.get(3));
                int d = c.intValue() + 1;
                Vector copy = (Vector)ahook.get(4);

                ahook.set(0, Integer.toString(b));
                ahook.set(3, Integer.toString(d));
                ahook.set(4, failure);


            }

        }


    }


    public XMLTree getQoSAttributes(String nodeId, String serviceId, String
sensorId, String hookId){

        if (serviceId != null){

            String name = nodeId + "_" + serviceId;

            if (service.containsKey(name)){
                Vector aservice = (Vector) service.get(name);

                XMLTree aserviceTree = new XMLTree(serviceId);


aserviceTree.put("Service_Executed",String.valueOf(aservice.get(0)));

aserviceTree.put("Service_Executed_Immediate",String.valueOf(aservice.get(1)));

aserviceTree.put("Service_Executed_Delayed",String.valueOf(aservice.get(2)));

aserviceTree.put("Service_Average_Time_Executed_Immediate",String.valueOf(aservic
e.get(3)));

aserviceTree.put("Service_Average_Time_Executed_Delayed",String.valueOf(aservice.
get(4)));

aserviceTree.put("Service_Average_Time_Executed",String.valueOf(aservice.get(5)))
;

aserviceTree.put("Service_Executed_Failure",String.valueOf(aservice.get(6)));

                return aserviceTree;
```

```
            }

            return null;
        }
        else if(sensorId != null){

            String name = nodeId + "_" + sensorId;

            if (sensor.containsKey(name)){
                Vector asensor = (Vector) sensor.get(name);

                XMLTree asensorTree = new XMLTree(sensorId);


asensorTree.put("Sensor_Executed",String.valueOf(asensor.get(0)));

asensorTree.put("Sensor_Executed_Normal",String.valueOf(asensor.get(1)));

asensorTree.put("Sensor_Executed_Scheduled",String.valueOf(asensor.get(2)));


                return asensorTree;

            }

            return null;
        }

        else if(hookId != null){

            String name = nodeId + "_" + hookId;

            if (hook.containsKey(name)){
                Vector ahook = (Vector) hook.get(name);

                XMLTree ahookTree = new XMLTree(hookId);

                ahookTree.put("Hook_Executed",String.valueOf(ahook.get(0)));
                ahookTree.put("Hook_Processed",String.valueOf(ahook.get(1)));

ahookTree.put("Hook_Average_Answer_time",String.valueOf(ahook.get(2)));
                ahookTree.put("Hook_Failed",String.valueOf(ahook.get(3)));
                ahookTree.put("Hook_Failures",String.valueOf(ahook.get(4)));

                return ahookTree;

            }

            return null;
        }


        return this;
}


}
```

Classe NodeManager

```
package br.ufsc.lrg.grid;

import java.util.*;
import java.lang.*;
import java.io.*;
/**
 * <p>Title: </p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: </p>
 *
 * @author not attributable
 * @version 1.0
 */
public class NodeManager extends XMLTree implements XMLInterface {

    protected Logger logger;
    protected Node node;
    protected XMLTree nodeTree;
    protected XMLTree serviceTree;
    protected XMLTree sensorTree;
    protected XMLTree hookTree;

    protected Hashtable nodes;
    protected Hashtable hooks;
    protected Hashtable sensors;
    protected Hashtable services;

    protected Vector nodeInfo;



    public NodeManager() {
        super("Node_Manager");


        this.nodeTree = new XMLTree("Nodes");
        add(nodeTree);

    }


    public NodeManager(Node node) {
        this();
        this.node = node;

        String nodeId = node.name;

        this.nodes = new Hashtable();
        this.services = new Hashtable();
        this.sensors = new Hashtable();
        this.hooks = new Hashtable();

        this.nodeInfo = new Vector();
        nodeInfo.add(0, hooks);
        nodeInfo.add(1, sensors);
        nodeInfo.add(2, services);
```

```
        nodes.put(nodeId, nodeInfo);


    }



    public Logger getLogger() {
       return this.logger;
   }

   public Node getNode(){
        return node;
    }



  public void addServiceSubTree(String nodeId, String serviceId,XMLTree service)
                                                        throws NullPointerException{

        if (nodes.containsKey(nodeId)){

            if (services.containsKey(serviceId)){
                services.remove(serviceId);
                services.put(serviceId, service);
            }
            else{
                services.put(serviceId, service);
            }

            nodeInfo.set(2, services);

        }
        else{
            try{
               nodes.put(nodeId, nodeInfo);
               this.addServiceSubTree(nodeId, serviceId, service);

           }catch (NullPointerException ex) { }

        }



    }

    public void addSensorSubTree (String nodeId, String sensorId, XMLTree sensor)
                                                          throws NullPointerException
{

           if (nodes.containsKey(nodeId)){

                 if (sensors.containsKey(sensorId)){
                     sensors.remove(sensorId);
                     sensors.put(sensorId, sensor);
                 }
                 else{
                 sensors.put(sensorId, sensor);
                 }

                 nodeInfo.set(1, sensors);

          }
```

```
          else{
              try{
                  nodes.put(nodeId, nodeInfo);
                  this.addSensorSubTree(nodeId, sensorId, sensor);

              }catch (NullPointerException ex) { }

           }



    }


    public void addHookSubTree (String nodeId, String hookId, XMLTree hook)
                                                          throws NullPointerException
{
            if (nodes.containsKey(nodeId)){

                  if (hooks.containsKey(hookId)){
                      hooks.remove(hookId);
                      hooks.put(hookId, hook);
                  }
                  else{
                  hooks.put(hookId, hook);
                  }

                  nodeInfo.set(0, hooks);

           }
           else{
              try{
                  nodes.put(nodeId, nodeInfo);
                  this.addHookSubTree(nodeId, hookId, hook);

              }catch (NullPointerException ex) { }

           }



    }



    public XMLTree getServicesTree(String nodeId){

        if (nodes.containsKey(nodeId)){

            serviceTree = new XMLTree();
            serviceTree = XMLTree.hashToXMLTree("Services", services);

        }
        else{
            try{


           }catch (NullPointerException ex) { }

        }
```

```
        return serviceTree;
    }


    public XMLTree getSensorsTree(String nodeId){

        if (nodes.containsKey(nodeId)){

            sensorTree = new XMLTree();
            sensorTree = XMLTree.hashToXMLTree("Sensors", sensors);

        }
        else{
            try{


           }catch (NullPointerException ex) { }

        }

        return sensorTree;
    }


    public XMLTree getHooksTree(String nodeId){

       if (nodes.containsKey(nodeId)){

            hookTree = new XMLTree();
            hookTree = XMLTree.hashToXMLTree("Hooks", hooks);

       }
       else{
            try{


          }catch (NullPointerException ex) { }

       }



       return hookTree;
   }

    public XMLTree generateXML(String URI, Hashtable parameters) {

        XMLTree xml = new XMLTree(node.name);
        xml.add(getHooksTree(node.name));
        xml.add(getSensorsTree(node.name));
        xml.add(getServicesTree(node.name));
        nodeTree.add(xml);


        /*XMLTree xml2 = new XMLTree(node.name);
        xml2.add(getServicesTree(node.name));
        nodeTree.add(xml2);*/

        return this;
    }
}
```

Interface QoSAttributes

```
package br.ufsc.lrg.grid;

/**
 * <p>Title: </p>
 *
 * <p>Description: </p>
 *
 * <p>Copyright: Copyright (c) 2006</p>
 *
 * <p>Company: </p>
 *
 * @author not attributable
 * @version 1.0
 */
public interface QoSAttributes {

        /**
        * This method returns the data collected by ManagerElement
        * @return the gathered data
        */


    public XMLTree getQoSAttributes();

}
```

Classe Node

```
package br.ufsc.lrg.grid;

import java.io.*;
import java.util.*;
import br.ufsc.lrg.grid.net.*;

/**
 * Abstract for Grid Node.
 * This abstract class contains the main functionality for a Grid Node.
 * It requires the instantiation with adequate Communicator to add
 * data communication functionality. As per original project, the two
 * extensions would be the HTTP-server node (@see HTTPServerNode) and
 * the assynchronous version (@see HTTPClientNode).
 *
 * <h3>TECHNOLOGIES: </h3>
 *
 * <pre>
 * This software may be distributed under the
 * GNU GENERAL PUBLIC LICENSE
 * Version 2, June 1991
 * {@link http://www.gnu.org/licenses/gpl.txt}
 * </pre>
 *
 * @compatible J2SE, J2ME
 * @project Grid-M Middleware for Mobile and Embedded Grid Computing
 * @author LRG-UFSC (http://grids.lrg.ufsc.br) -- Fernando Koch
 *
 * @see HTTPServerNode
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */

public abstract class Node extends ManagedElement implements SensorInterface,
Runnable {

    /** Manually assigned COMPILATION TAG. Verifier should sign and date this
signature */
    final static public String
            VERSION = "v0.7 (1011) fkoch 15-Jul-2006";

    /** Time interval for sending a heartbeat signal to the directory service */
    final static public int HEARTBEAT_INTERVAL = 30;

    // ** Attributes
    protected String name;
    protected String fullAddress;
    protected String urlAddress;
    protected String serviceProviderURL;
    protected Hashtable services;
    protected Hashtable sensors;

    // ** Data structures
    protected NodeManager nodeManager;
    protected ManagedElement managedElement;
    protected Profiler profiler;
    protected Communicator communicator;
    protected Logger logger;

    // **  Statistics
    /** @todo fkoch QOS Research on what stats are worth/nedded to monitor */
    protected long start_time;
    protected int tasks_processed_fg;
```

```
protected long tasks_processed_fg_time;
protected int tasks_processed_bg;
protected long tasks_processed_bg_time;
protected int tasks_failed;
protected int task_processor_counter;

// ** Cache for received results
protected Hashtable taskResultCache;

// ** Task Queue holder and handlers
protected ArrayList taskQueue;
protected Thread taskQueueProcessor;

// ** Scheduler
Timer scheduler = null;

//
//  CONSTRUCTORS
//

/**
 * Initialize the Grid Node.
 *
 * @param name of Grid Node
 * @param comm Communicator to be associated to node
 */
public Node(String name, Communicator comm) {
    // initialize empty values
    this.urlAddress = null;
    this.logger = null;
    this.taskQueueProcessor = null;

    // assign values
    this.name = name;
    this.fullAddress = name;
    this.communicator = comm;

    // initialize data structures
    this.services = new Hashtable();
    this.taskQueue = new ArrayList();
    this.taskResultCache = new Hashtable();

    // scheduler
    this.scheduler = new Timer();

    // reset statistics
    this.start_time = System.currentTimeMillis();
    this.tasks_processed_fg = 0;
    this.tasks_processed_fg_time = 0L;
    this.tasks_processed_bg = 0;
    this.tasks_processed_bg_time = 0L;
    this.tasks_failed = 0;
    this.task_processor_counter = 0;

    // assign service provider
    setServiceProvider(serviceProviderURL);


    // add itself as sensor
    addSensor("node", this);
}

//
```

```
//   ENCAPSULATORS
//

/**
 * Return NAME assigned to this Node.
 *
 * @return NAME assigned to this Node
 */
public String getName() {
    return this.name;
}

/**
 * Return FULL ADDRESS (i.e. name @ domain) assigned to this Node.
 *
 * @return FULL ADDRESS  assigned to this Node
 */
public String getFullAddress() {
    return this.name;
}

/**
 * Return URL ADDRESS (i.e. http://<url> for http-server enabled hosts)
 * assigned to this Node.
 *
 * @return URL ADDRESS assigned to this Node
 */
public String getURLAddress() {

    // Strip / from end
    if (this.urlAddress.endsWith("/")) {
        this.urlAddress = this.urlAddress.substring(0,
                this.urlAddress.length() - 1);
    }

    return this.urlAddress;
}


public void setNodeManager(NodeManager nodeManager){
    this.nodeManager = nodeManager;
}

public NodeManager getNodeManager() {
    return this.nodeManager;
}




public void setManagedElement(ManagedElement managed){
   this.managedElement = managed;
}

public ManagedElement getManagedElement() {
    return this.managedElement;
}



/**
```

```
 * Set the COMMUNICATOR (i.e. data communication layer object) to be
 * used by this Grid Node.
 *
 * @param comm COMMUNICATOR to be used by this Node
 */
public void setCommunicator(Communicator comm) {
    this.communicator = comm;
}

/**
 * Return COMMUNICATOR (i.e. data communication layer object) used
 * by this Grid Node.
 *
 * @return COMMUNICATOR used by this Node
 */
public Communicator getCommunicator() {
    return this.communicator;
}


/**
 * Return LOGGER assigned to this Node.
 *
 * @return LOGGER assigned to this Node
 */
public Logger getLogger() {
    return this.logger;
}

/**
 * Set the PROFILER (i.e. profile collector object) to be
 * used by this Node.
 *
 * @param profiler PROFILER to be used by this Node
 * @see Profiler
 */
public void setProfiler(Profiler profiler) {
    this.profiler = profiler;
}

/**
 * Return PROFILER (i.e. profile collector object) used
 * by this Node.
 *
 * @return profiler used by this Node
 * @see Profiler
 */
public Profiler getProfiler() {
    return this.profiler;
}

/**
 * Set a static parameter in Node's Profile
 *
 * @param parameter to be set
 * @param value to assign to this parameter
 */
public void setProfileParameter(String parameter, String value) {
    if (this.profiler != null) {
        this.profiler.setProfileParameter(parameter, value);
    }
}
```

```
/**
 * Set a dynamic parameter in Node's Profile
 *
 * @param parameter to be set
 * @param value to assign to this parameter
 */
public void setContextParameter(String parameter, String value) {
    if (this.profiler != null) {
        this.profiler.setContextParameter(parameter, value);
    }
}

/**
 * Set ServiceProvider to where this Node is connected.
 *
 * @param url to where this Grid Node is connected
 */
public void setServiceProvider(String url) {
    this.serviceProviderURL = url;

    // grid service becomes default address
    if (this.serviceProviderURL != null) {
        // adjust url
        if (this.serviceProviderURL.endsWith("/")) {
            this.serviceProviderURL =
                    this.serviceProviderURL.substring(0,
                    this.serviceProviderURL.length() - 1);
        }

        // add default route
        this.communicator.addRoute(Communicator.DEFAULT_ROUTE_LABEL,
                                   serviceProviderURL);
    }
}

//
// LOGGING METHODS
//

/**
 * Set log output stream
 *
 * @param out PritnStream to receive log output
 */
public void setLogOutput(PrintStream out) {
    this.logger = new Logger(out);
}

/**
 * Set context and level to be logged
 *
 * @param ctx to add logging
 * @param level to log in this context
 */
public void setLogLevel(String ctx, int level) {
    if (this.logger == null) {
        setLogOutput(System.err);
    }
    this.logger.set(ctx, level);
}

/**
 * Check if should log to a given context and level
```

```
 *
 * @param context to check
 * @param level to check
 * @return true if should log for this context and level
 */
public boolean shouldLog(String context, int level) {
    return (this.logger != null) ?
            this.logger.enabled(context, level) : false;
}

/**
 * Print log message to log stream
 *
 * @param context for this log message
 * @param level for this log message
 * @param message to be printed out
 */
public void log(String context, int level, String message) {
    if (this.logger != null) {
        this.logger.log(context, level, this.name, message);
    }
}

//
//  START / STOP METHODS
//

/**
 * Start Grid Node's execution.
 */
public void start() {
    if (shouldLog("node", Logger.BASIC)) {
        log("node", Logger.BASIC, "Node.start(.) " + name);
    }

    // start COMMUNICATOR
    this.communicator.start();
}

/**
 * Stop Grid Node's execution.
 */
public void stop() {
    if (shouldLog("node", Logger.BASIC)) {
        log("node", Logger.BASIC, "Node.stop(.) " + name);
    }

    // stop SCHEDULER
    this.scheduler.cancel();

    // stop COMMUNICATOR
    this.communicator.stop();
}


//
//  GRID COMPUTING RELATED METHODS
//

/**
 * Register this Node to Service provider.
 *
 * @return true if registering went OK
```

```
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     *
     * @todo fkoch SECURITY missing authentication code
     */
    public boolean register() {
        if (shouldLog("node", Logger.BASIC)) {
            log("node", Logger.BASIC, "Node.register(.) started");
        }

        // force profiler to re-load (specially dynamic part)
        this.profiler.refresh();

        // configure profile
        setProfileParameter("name", this.name);
        setProfileParameter("fulladdress", this.fullAddress);

        // Strip / from end
        if (this.urlAddress.endsWith("/")) {
            this.urlAddress = this.urlAddress.substring(0,
                    this.urlAddress.length() - 1);
        }

        setProfileParameter("url", this.urlAddress);

        // convert to XML and create Task
        Task task = this.createTask("*", "register", this.profiler);

        // send out task for processing
        TaskResult result;

        try {
            result = sendTask(task);
        } catch (IOException ex) {
            result = this.createTaskResult(task, TaskResult.RESULT_FAILURE,
null);
        }

        // register is a IMMEDIATE task by default, so fetch the result
        if (result.isOK()) {
            // receive the new domain back.. change fulladdress
            String domain = (String) result.getResult("domain");
            if (domain != null) {
                this.fullAddress = this.name + "@" + domain;

                if (shouldLog("node", Logger.MEDIUM)) {
                    log("node", Logger.MEDIUM,
                        "Node.register(.) completed: provider=" +
                        result.originator + ": fullAddress=" + this.fullAddress);
                }
            }
            return true;
        } else {
            if (shouldLog("node", Logger.MEDIUM)) {
                log("node", Logger.MEDIUM, "FAILED Node.register(.)");
            }
            return false;
        }
    }

    /**
     * Un-Register this Node from Service provider.
     *
     * @return true if un-registering went OK
```

```
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public boolean unRegister() {
        if (shouldLog("node", Logger.BASIC)) {
            log("node", Logger.BASIC, "Node.unRegister(.) started");
        }
        Task task = this.createTask("*", "unregister", null);

        try {
            TaskResult result = sendTask(task);
            return result.isOK();
        } catch (IOException ex) {
            return false;
        }
    }

    /**
     * Add new service to this Grid Node.
     *
     * @param service is the Service instance to be added
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     * @see ServiceInterface
     */
    public final void addService(ServiceInterface service) {
        if (shouldLog("node", Logger.BASIC)) {
            log("node", Logger.BASIC,
                "Node.addService(.) " + name + ": service=" + service);
        }

//        XMLTree servicesTree = new XMLTree("services");

        // add to local directory of services
        String id[] = service.getServiceIds();
        if (id != null) {
            for (int i = 0; i < id.length; i++) {

                // tracing
                if (shouldLog("node", Logger.BASIC)) {
                    log("node", Logger.BASIC,
                        "Node.addingService=>" + id[i]);
                }

                this.services.put(id[i], service);

                super.createServiceAttribute(name, id[i]);
            }

            // construct XMLTree
            //servicesTree.add(XMLTree.hashToXMLTree("services",services));
//            servicesTree.add("divide");
//            this.addGlobalService(servicesTree);

        }
    }

    /**
     * Add global service
     * @param service is the service list to be published
     * @return true is services are registered ok
     *
     */
    private final boolean addGlobalService(XMLTree service) {
        if (shouldLog("node", Logger.BASIC)) {
```

```
        log("node", Logger.BASIC,
            "Node.addServiceGlobal(.) service=" + service.toString());
    }

    Task task = this.createTask("*", "registerServices", service);

    try {
        TaskResult result = sendTask(task);
        return result.isOK();
    } catch (IOException ex) {
        return false;
    }
}


/**
 * Remove service from this Grid Node.
 *
 * @param id list of Service-IDs to be removed
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public void removeService(String id[]) {
    if (id != null) {
        for (int i = 0; i < id.length; i++) {
            this.services.remove(id[i]);
        }
    }
}

/**
 * Find a registered Service in this node based on its id.
 *
 * @param id to look at
 * @return ServiceInterface found in service table
 * @see ServiceInterface
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public ServiceInterface findService(String id) {
    return (ServiceInterface) services.get(id);
}


//
//  PROFILER COLLECTOR INTERFACE METHODS
//

/**
 * Add Profiler STATIC data collector interface
 *
 * @param collector ProfileDataCollectorInterface
 */
public void addProfileStaticCollector(ProfileDataCollectorInterface
                                      collector) {
    this.profiler.addProfileStaticCollector(collector);
}

/**
 * Add Profiler CONTEXT data collector interface
 *
 * @param collector ProfileDataCollectorInterface
 */
public void addProfileContextCollector(ProfileDataCollectorInterface
                                       collector) {
```

```
    this.profiler.addProfileContextCollector(collector);
}

//
//  SENSOR NETWORKING RELATED METHODS
//

/**
 * Add Sensor to this Node.
 *
 * @param sensorID is the identification for this sensor
 * @param sensor is the SensorInterface implementation
 * @see SensorInterface
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public final void addSensor(String sensorID, SensorInterface sensor) {
    if (shouldLog("node", Logger.BASIC)) {
        log("node", Logger.BASIC,
            "Node.addSensor(.) " + name + ": sensor=" + sensor);
    }

    // initialize sensor table
    if (this.sensors == null) {
        this.sensors = new Hashtable(5);
    }

    // add to local directory of sensors
    this.sensors.put(sensorID, sensor);

    super.createSensorAttribute(name, sensorID);
}


/**
 * Remove Sensor from this Node.
 *
 * @param sensorID is the sensor identification to be removed
 *
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public final void removeSensor(String sensorID) {
    if (shouldLog("node", Logger.BASIC)) {
        log("node", Logger.BASIC,
            "Node.removeSensor(.) " + sensorID);
    }

    // find sensor interface and remove
    if (this.sensors.containsKey(sensorID)) {
        this.sensors.remove(sensorID);
    }
}


/**
 * Force data collection from Sensor in this Node.
 *
 * @param sensorID is the sensor identification to be collected
 * @return collected data
 * @see XMLTree
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public final XMLTree collectSensor(String sensorID) {
    if (shouldLog("node", Logger.BASIC)) {
```

```
            log("node", Logger.BASIC,
                "Node.collectSensor(.) " + sensorID);
        }

        // find sensor interface to use and collectData(.)
        SensorInterface sensor = (SensorInterface)this.sensors.get(sensorID);
        //return sensor != null ? sensor.collectData() : null;

        if (sensor != null){

            super.updateSensorAttribute(this.name , sensorID, false);
            this.nodeManager.addSensorSubTree(this.name, sensorID,
                                              super.getQoSAttributes(name, null,
sensorID, null));

            return sensor.collectData();
        }
        else{
            return null;
        }

    }

    //
    //   SCHEDULER MANIPULATION
    //

    /**
     * Scheduler a regular data collection from Sensor and send data to
     * destination invoking a service.
     *
     * @param sensorId to collect data from
     * @param intervalSec to collect and send data (in seconds)
     * @param destination to submit Task
     * @param service to be invoked by Task
     * @param priority of task to be submitted
     * @param inference filter data should be submitted before sending up to
destination
     * @see InferenceFilterInterface
     */
    public void scheduleSensorCollect(String sensorId, int intervalSec,
                                      String destination, String service,
                                      int priority,
                                      InferenceFilterInterface inference) {

        // stat scheduler if not there
        if (this.scheduler == null) {
            this.scheduler = new Timer();
        }

        // create SensorCollectTask
        TimerTask timedTask = new SensorCollectTask(this, sensorId,
                destination, service, priority, inference);

        // add SensorCollectTask to SCHEDULER
        this.scheduler.schedule(timedTask, intervalSec * 1000 + 1,
                                intervalSec * 1000 + 1);

        super.updateSensorAttribute(this.name , sensorId, true);

    }

    /**
```

```
 * Schedule regular collection and send of self CONTEXT data
 *
 * @param intervalSec to collect and send data (in seconds)
 */
public void scheduleContextCollect(int intervalSec) {
    scheduleSensorCollect("node", intervalSec, "*", "register",
                          Task.PRIORITY_NORMAL, null);
}


/**
 * Schedule a regular heartbeat to inform the directory service that
 * the node is active
 *
 * @param intervalSec to send the update (in seconds)
 */
public void scheduleHeartbeat(int intervalSec) {
    scheduleSensorCollect(null, intervalSec, "*", "heartbeat",
                          Task.PRIORITY_NORMAL, null);
}


/**
 * Schedule regular Garbagge Collection
 *
 * @param intervalSec to collect and send data (in seconds)
 */
public void scheduleGarbaggeCollect(int intervalSec) {

    // add garbagge collection to SCHEDULER
    this.scheduler.schedule(new TimerTask() {
        public void run() {
            System.gc();
        }
    },
            intervalSec * 1000 + 1, intervalSec * 1000 + 1);
}

/**
 * Schedule a regular internal task carried out by the node
 * For example, if the node is the directory service, the activity
 * can be to look for nodes that have not updated their status and
 * remove them
 * @param activity is the task to be carried out
 * @param intervalSec to collect and send data (in seconds)
 */
public void scheduleContinuousInternalActivity(TimerTask activity,
        int intervalSec) {
    // add the activity to SCHEDULER
    this.scheduler.schedule(activity, intervalSec * 1000 + 1,
                            intervalSec * 1000 + 1);
}

//
// SENSOR INTERFACE: COLLECT CONTEXT INFORMATION
//

/**
 * Implement SensorInterface. Collect CONTEXT information about this node
 * through Profiler.
 *
 * @return XMLTree contains the CONTEXT information about this node
 *
```

```
     */
    public XMLTree collectData() {
        return this.profiler != null ?
                this.profiler.getUpdatedData() : null;
    }

    //
    //   TASK MANIPULATION
    //

    /**
     * Create new Task for submission.
     *
     * @param destination is the node to process this task (or "*" if to be
     * distributed by GridService)
     * @param service is the Service-ID
     * @param parameters is the array of parameters
     * @return Task is the created object
     *
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public final Task createTask(String destination, String service,
                                 XMLTree parameters) {
        return new Task(this.name, destination, service, parameters);
    }

    /**
     * Create new TaskResult.
     *
     * @param task being responded
     * @param resultCode for result
     * @param results are the resulting values
     * @return TaskResult is the created object
     *
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public final TaskResult createTaskResult(Task task, String resultCode,
                                             XMLTree results) {
        return new TaskResult(task, this.name, resultCode, results);
    }


    /**
     * Send Task to execute in other Grid Node (direct submission).
     *
     * @param task to be executed
     * @return TaskResult where:
     *         TaskResult.getResultCode() return submissin status (W3C standard)
     *         if TaskResult == TaskResult.DELAYED_TASK means a request to a
delayed response service
     *         if TaskResult == TaskResult.ERROR_TASK means error in task
submission
     * @throws IOException for communication errors
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public final TaskResult sendTask(Task task) throws IOException {
        TaskResult result;

        super.createHookAttribute(this.name, task.service);

        try {
            // tracing
            if (shouldLog("node", Logger.BASIC)) {
```

```
                log("node", Logger.BASIC, "Node.sendTask(.) started: " + task);
            }

            // hook out to Communicator
            long start_time = System.currentTimeMillis();
            result = this.communicator.sendTask(task);
            long end_time = System.currentTimeMillis();

            super.updateHookAttribute(this.name, task.service,
                                      false, end_time - start_time, null);

            this.nodeManager.addHookSubTree(this.name,task.service,

super.getQoSAttributes(this.name, null, null, task.service));

        }

        catch (IOException ex) {

            // tracing for IOException
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "EXCEPTION: Node.sendTask() failed " + task + ": " +
                    ex.getMessage());
            }

            result = this.createTaskResult(task, TaskResult.RESULT_FAILURE,
null);

            super.updateHookAttribute(this.name, task.service,
                                      true, 0, ex.getMessage());

            this.nodeManager.addHookSubTree(this.name,task.service,

super.getQoSAttributes(this.name, null, null, task.service));



        }

        // tracing
        if (shouldLog("node", Logger.DETAIL)) {
            log("node", Logger.DETAIL,
                "Node.sendTask(.) completed: " + task + ": " +
                result);
        }

        return result;
    }

    /**
     * Check whether the result for a Task submission with DELAYED response
     * is now available. First, check in local cache and if not there
     * check through Communicator
     *
     * @param taskId to be received
     * @return true if result for that task submission is available
     * @throws IOException for communication errors
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public final boolean hasTaskResult(String taskId) throws IOException {
        return this.taskResultCache.contains(taskId) ||
                this.communicator.hasTaskResult(taskId);
```

```
}

/**
 * Fetch the result for a Task submission with DELAYED response.
 * First, check in local cache and if not there
 * check through Communicator
 *
 * @param taskId to be received
 * @return true if result for that task submission is available
 * @throws IOException for communication errors
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public final TaskResult fetchTaskResult(String taskId) throws IOException {

    // first go to cache and then through Communicator
    TaskResult result = hasTaskResult(taskId) ?
                        (TaskResult)this.taskResultCache.get(taskId) :
                        this.communicator.fetchTaskResult(taskId);

    // tracing
    if (shouldLog("node", Logger.MEDIUM)) {
        log("node", Logger.MEDIUM,
            "Node.fetchTaskResult(.) completed: " + taskId + ": result=" +
            result);
    }

    // return TaskResult
    return result;
}

/**
 * Wait for TaskResult from Task submission with DELAYED response
 * or until timeout.
 *
 * @param taskId is the Task-ID (get with Task.getTaskId())
 * @param timeoutMS is the time-out in Ms (-1 means no time-out)
 * @return TaskResult if available
 * @throws IOException in case of i/o problems
 *
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 */
public final TaskResult waitTaskResult(String taskId, int timeoutMS) throws
        IOException {
    long start_time = System.currentTimeMillis();
    TaskResult result = null; ;

    // while cannot find result in queue and has not timed out yet
    while ((result == null) &&
           ((timeoutMS == -1) ||
            (System.currentTimeMillis() - start_time < timeoutMS))) {

        // try to fetch result
        result = fetchTaskResult(taskId);

        // if not there yet, give interval and try again
        if (result == null) {
            try {
                Thread.sleep(100);
                Thread.yield();
            } catch (InterruptedException ex) {
            }
        } else {
            if (shouldLog("node", Logger.MEDIUM)) {
```

```
                    log("node", Logger.MEDIUM,
                        "Node.waitTask() succeeded: " + result);

                }
            }
        }

        // tracing
        if (result == null) {
            if (shouldLog("node", Logger.MEDIUM)) {
                log("node", Logger.MEDIUM,
                    "Node.waitTask() timeout " + taskId);
            }
        }

        return result;
    }

    /**
     * Dispatch task to be processed in foreground by this Node's Service.
     *
     * @param task to be processed
     * @return TaskResult result from processing
     */
    public final TaskResult dispatchTaskForeground(Task task) {

        // find Service to process this task
        ServiceInterface service = findService(task.service);

        // if there is a Service associated to the service invoked
        if (service != null) {
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "Node.dispatchTaskForeground(.) " +
                    task.toString() + ": service=" + service);

            }

            // process Service Task
            long start_time = System.currentTimeMillis();
            TaskResult result = service.processTaskForeground(this, task);
            long end_time = System.currentTimeMillis();

            // add statistics
            this.tasks_processed_fg++;
            this.tasks_processed_fg_time +=
                    (end_time - start_time);

            if (shouldLog("node", Logger.DETAIL)) {
                log("node", Logger.DETAIL,
                    "Node.dispatchTaskForeground(.) processed " +
                    task + ": " + result + " :time=" + (end_time - start_time));

            }


            super.updateServiceAttribute(name, String.valueOf(task.service),
                                         "fg", tasks_processed_fg_time);

            this.nodeManager.addServiceSubTree(name,String.valueOf(task.service),
                                                   super.getQoSAttributes(name,
String.valueOf(task.service), null, null));
```

```
            // return result
            return result;
        }

        // otherwise, fail to process
        else {
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "EXCEPTION: Node.dispatchTaskForeground(.) service not found
" +
                    task.toString());
            }

            super.updateServiceAttribute(name, String.valueOf(service), "failed",
0);
            // add statistics
            this.tasks_failed++;

            this.nodeManager.addServiceSubTree(name,
String.valueOf(task.service),
                                                    super.getQoSAttributes(name,
String.valueOf(task.service), null, null));


        }

        return null;
    }

    /**
     * Dispatch task to be processed in background by this Node's Service.
     *
     * @param task to be processed
     */
    public final void dispatchTaskBackground(Task task) {

        // find Service to process this task
        ServiceInterface service = findService(task.service);

        // if there is a Service associated to the service invoked
        if (service != null) {
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "Node.dispatchTaskBackground(.) " +
                    task.toString() + ": service=" + service);

            }

            /** @todo fkoch QOS Tasks with higher priority should be queued at
the top.
             */
            // enqueue Task in the list of tasks to process
            this.taskQueue.add(task);

            /**
             * @todo fkoch INCOMPLETE Future implementation taskProcessor
should/could become a
             * configurable POLL OF THREADS. It is important to keep the number
of
             * allocated threads under control to be able to estimate Node's
resource
             * usage, performance and capability. */
```

```
            // start task processor, if not yet running
            if (this.taskQueueProcessor == null) {
                this.taskQueueProcessor = new Thread(this, "Node-" +
                        this.name + "-" +
                        this.task_processor_counter++);
                this.taskQueueProcessor.start();
            }
        }

        // otherwise, fail to process
        else {
            if (shouldLog("node", Logger.BASIC)) {
                log("node", Logger.BASIC,
                    "EXCEPTION: Node.dispatchTaskBackground(.) service not found
" +
                    task.toString());
            }

            super.updateServiceAttribute(name, String.valueOf(service), "failed",
0);
            // add statistics
            this.tasks_failed++;

            this.nodeManager.addServiceSubTree(name,
String.valueOf(task.service),
                                                    super.getQoSAttributes(name,
String.valueOf(task.service), null, null));
        }
    }

    /**
     * Process inbounding Task.
     *
     * If this is a TaskResult, cache it (TaskResults come from HTTP connections
     * returning delayed results for specific services).
     *
     * @param task to be processed
     * @return submission result code (following W3C standard)
     * @throws IOException in case of i/o problems
     *
     * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
     */
    public TaskResult processInboundingTask(Task task) throws IOException {

        // If returning TaskResult (which also process through here): queue it up
        if (task instanceof TaskResult) {

            /** @todo fkoch SECURITY can't accept ANY TaskResult otherwise
             * it will the susceptible to DoS attacks! Must be selective here!
             *
             * if(!isValid(task)){
             *    return TaskResult(.. TaskResult.RESULT_REFUSE, ...)
             */
            if (shouldLog("node", Logger.MEDIUM)) {
                log("node", Logger.MEDIUM,
                    "Node.processInboundingTask(.) enqueued task-result " +
                    task.toString());
            }
            this.taskResultCache.put(task.taskId, task);
            return (TaskResult) task;
        }

        // otherwise, process as Task
```

```
        else {

            /** @todo fkoch SECURITY It can't process message like this!
             * Must first check whether isValid(Task) to process, based on
authentication,
             * service bein requested, reputation, packet contents and others.
             *
             * if(!isValid(task)){
             *    return TaskResult(.. TaskResult.RESULT_REFUSE, ...)
             */

            // (1) check if Tasks is addressed to this Node otherwise re-route it
            if (this.communicator.isLocalAddress(task.destination)) {

                // (2) if task is addressed to this node
                // (3) retrieve associated ServiceInterface
                ServiceInterface service = (ServiceInterface) services.get(task.
                        service);

                if (service != null) {

                    // (4) dispatch task for processing in foreground
                    TaskResult result = this.dispatchTaskForeground(task);

                    // tracing
                    if (shouldLog("node", Logger.BASIC)) {
                        log("node", Logger.BASIC,
                            "Node.processInboundingTask(.) processed as " +
                            (result.isDelayed() ? "DELAYED " : "IMMEDIATE ") +
                            task.toString() + ": " + result);
                    }

                    // (5) if result isDelayed(), dispatch to process in
Background
                    if ((result == null) || result.isDelayed()) {
                        dispatchTaskBackground(task);
                    }

                    /** @todo fkoch QOS must treat RESULT_TOO_BUSY in case
                     * Service refuses to process in the given amount of time
                     * (Task.max_processing_time)
                     */

                    // return result (which can be foreground result or DELAYED)
                    return result;
                }
            }

            // (6) If Task is NOT ADDRESSED to this node, re-route it
            else {
                /** @todo fkoch INCOMPLETE is it possible to re-route just
                 * by forwarding the task through sendTask(.)? What if the task
is
                 * immediate? Maybe should it be discarded as a routable Task?
                 */

                // tracing
                if (shouldLog("node", Logger.MEDIUM)) {
                    log("node", Logger.MEDIUM,
                        "Node.processInboundingTask(.) re-routed " +
                        task.toString() + ": destination=" +
                        task.destination);
                }
```

```
            // send Task through communicator
            return sendTask(task);
        }
    }

    // exception cases: return SERVICE FAILED
    return this.createTaskResult(task, TaskResult.RESULT_FAILURE, null);
}


/**
 * Run processing queue.
 * Tasks are enqueued by processTask(.) when requested services are not of
 * immediate processing.
 *
 * <pre>
 * <b>
 * As per this implementation, the Grid Node is a single-thread
 * task processor. That is, there is one pre-allocated thread (the
 * taskProcessor) which is initialize by the first task
 * submitted to this Node and stays idle after completing the processing.
 * For the second task submitted and on this thread is re-activated to
 * process that task. In case multiple tasks arrive before the completion
 * of the one currently processing, these go to the taskQueue to wait
 * for their turn.
 * </b>
 * </pre>
 *
 */
public void run() {

    if (shouldLog("node", Logger.DETAIL)) {
        log("node", Logger.DETAIL,
            "Node.run(.) started taskProcessor: " +
            Thread.currentThread().getName() +
            " :queueSize=" + this.taskQueue.size());
    }

    // executes in loop until queue gets empty
    while (!this.taskQueue.isEmpty()) {

        // process Task from processing queue
        Task task = (Task)this.taskQueue.remove(0);

        // tracing
        if (shouldLog("node", Logger.MEDIUM)) {
            log("node", Logger.MEDIUM,
                "Node.run() processing " + task);
        }

        // load service interface to be used
        ServiceInterface service = (ServiceInterface)
                                   this.services.get(task.service);
        if (service != null) {
            if (shouldLog("node", Logger.DETAIL)) {
                log("node", Logger.DETAIL,
                    "Node.dispatchTaskBackground(.) " +
                    task.toString() + ": service=" + service);

            }
        }
```

```
            // process Service Task
            long start_time = System.currentTimeMillis();
            TaskResult result = service.processTaskBackground(this, task);
            long end_time = System.currentTimeMillis();

            // add statistics
            this.tasks_processed_bg++;
            this.tasks_processed_bg_time += (end_time - start_time);

            super.updateServiceAttribute(name, String.valueOf(service), "bg",
tasks_processed_bg_time);
            this.nodeManager.addServiceSubTree(name,String.valueOf(task.service),
                                                    super.getQoSAttributes(name,
String.valueOf(task.service), null, null));

            try {
                // tracing
                if (shouldLog("node", Logger.MEDIUM)) {
                    log("node", Logger.MEDIUM,
                        "Node.run() returning " + result + ": task=" + task);
                }

                // returning result
                this.communicator.sendTask(result);

            } catch (IOException ex1) {
                if (shouldLog("node", Logger.DETAIL)) {
                    log("node", Logger.DETAIL,
                        "EXCEPTION: Node.run() failed returning " +
                        result + ": task=" + task);
                }
            }
        }

        // queue processed, released taskQueueProcessor
        this.taskQueueProcessor = null;
    }


    /**
     * DEBUG CODE.
     * This code should be commented out for final compilation!
     *
     * @param args String[] arguments passed-in by command line
     * @throws java.lang.Exception
     */
   /* public static void main(String[] args) {
        System.out.println("Hello");
    }*/

}


/**
 * AUXILIARY CLASS.
 * TimerTask class to collect data from given sensor and send Task
 * to provided destination.
 */
class SensorCollectTask extends TimerTask {
    Node node;
    String destination;
    String service;
    String sensorId;
```

```
int priority;
InferenceFilterInterface inference;

/**
 * Create Task to collect data from given Sensor and send as Service request
 * to Destination.
 *
 * @param node is the parent Node
 * @param sensorId is the Sensor to collect data
 * @param destination to send Task
 * @param service to be executed by periodical Task
 * @param priority to proces this service (list of priorities in Task)
 * @param inference is the Filter to process collected information
 */

public SensorCollectTask(Node node, String sensorId,
                         String destination, String service,
                         int priority,
                         InferenceFilterInterface inference) {

    this.node = node;
    this.sensorId = sensorId;
    this.destination = destination;
    this.service = service;
    this.priority = priority;
    this.inference = inference;
}

/**
 * Entry point to run Timed Task.
 */
public void run() {
    XMLTree data = null;
    // collect data from Sensor if sensor is not null
    // Otherwise, it will send just a heartbeat signal
    if (this.sensorId != null) {



        data = this.node.collectSensor(this.sensorId);

        if (this.node.shouldLog("node", Logger.DETAIL)) {
            this.node.log("node", Logger.DETAIL,
                          "NodeTimedTask.run(.) " + this.sensorId +
                          ": collected=" + data);
        }

        // if filter is configured, process it before upload
        if (this.inference != null) {
            data = this.inference.processCollectedData(data);

            if (this.node.shouldLog("node", Logger.DETAIL)) {
                this.node.log("node", Logger.DETAIL,
                              "NodeTimedTask.run(.) " + this.sensorId +
                              ": filtered=" + data);
            }
        }
    }

    // if there is data to upload, send it to destination
    if ((this.destination != null) && (this.service != null)) {
        Task task = this.node.createTask(this.destination, this.service,
                                         data);
```

```
            try {
                TaskResult result = this.node.sendTask(task);

                if (this.node.shouldLog("node", Logger.BASIC)) {
                    this.node.log("node", Logger.BASIC,
                                  "NodeTimedTask.run(.) " + this.sensorId +
                                  ": task=" +
                                  task + ": result=" + result);
                }
            } catch (IOException ex) {
                if (this.node.shouldLog("node", Logger.BASIC)) {
                    this.node.log("node", Logger.BASIC,
                                  "EXCEPTION: NodeTimedTask.run(.) " +
                                  this.sensorId +
                                  ": msg=" + ex.getMessage());
                }
            }
        }
    }
}
```

Classe HTTPServerNode

```
package br.ufsc.lrg.grid;

import br.ufsc.lrg.grid.net.*;
import minihttp.*;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * HTTP-Server enabled Grid Node.
 * This class implements the Grid Node with HTTP-server communication
 * channel. This class is J2SE compatible only and intended to be
 * executed in relatively resourceful hardware.
 *
 * <h3>TECHNOLOGIES: </h3>
 *
 * <h3>This class contains a COMMAND-LINE ENTRY POINT.</h3>
 * <pre>
 * Command-line:
 *
 * $ java -cp  mGrid-j2se.jar br.ufsc.lrg.grid.HTTPServerNode
 * -name s                          :process name (mandatory)
 * [-port i]                        :server port (default 8000)
 * -provider s                      :service provider
 * [-services class[;class]*]       :list of services to add
 * [-snoop]                         :snoop mode
 * <br/>
 * Defautl XSLT interface attached to '/':
 *
 * // add XML interface for provider
 * addXSLTInterface("/", "node", "node.xsl", new XMLInterface[] {this});
 *
 * Access the INFO page in: http://HOST:PORT/
 * </pre>
 *
 * <pre>
 * This software may be distributed under the
 * GNU GENERAL PUBLIC LICENSE
 * Version 2, June 1991
 * {@link http://www.gnu.org/licenses/gpl.txt}
 * </pre>
 *
 * @compatible J2SE
 * @project Grid-M Middleware for Mobile and Embedded Grid Computing
 * @author LRG-UFSC (http://grids.lrg.ufsc.br) -- Fernando Koch
 *
 * @see Node
 * @see HTTPServerCommunicator
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 *
 */

public class HTTPServerNode extends Node implements XMLInterface {

    protected HTTPServerCommunicator httpServerCommunicator;
    protected int port;

    /**
     * Initialize a HTTP-server node.
     *
```

```
     * @param name of this node
     * @param port to attach HTTP listener to
     */
    public HTTPServerNode(String name, int port) {
        super(name, null);
        this.port = port;
        this.profiler = new Profiler(this);
        this.nodeManager = new NodeManager(this);

        // initialize and plug-in HTTPServerCommunicator
        this.httpServerCommunicator = new HTTPServerCommunicator(this, port);
        super.setCommunicator(this.httpServerCommunicator);

        // set localhost address
        try {
            String hostName = java.net.InetAddress.getLocalHost().getHostName();
            this.urlAddress = new URL("http", hostName, port, "/").toString();
        } catch (MalformedURLException ex) {
        } catch (UnknownHostException ex) {
        }

        // add XML interface for provider
        addXSLTInterface("/", "node", "node.xsl", new XMLInterface[] {this});

        // add XML interface for NodeManager
        addXSLTInterface("/manager/",      "Manager",      "manager.xsl",      new
XMLInterface[] {nodeManager});

    }

    /**
     * Return attached minihttp.HTTPServer. Check the minihttp
     * documentation on how to configure the associated HTTPServer.
     *
     * <pre>
     * Quick How-to:
     *
     * Set up log level and output stream:
     * getHTTPServer().setLog(level, logStream);
     *
     * Set up snoop mode
     * getHTTPServer().setSnoopMode(snoopMode);
     *
     * @return minihttp.HTTPServer is the HTTP-Server for this Node
     * @see http://grid.lrg.ufsc.br/javadoc/miniHTTP/index.html
     */
    public final HTTPServer getHTTPServer() {
        return this.httpServerCommunicator.getHTTPServer();
    }

    /**
     * Enables the Server-based XSLT transformer. Depends on auxiliary
     * JAR file accessible in the CLASSPATH containing a $ClassName$ class
     * that implements HTTPServerXSLTTransformerInterface.
     *
     * @param XSLTTransformerClassName is the class name to be used as XSLT
     * Transformer that implments HTTPServerXSLTTransformerInterface.
     */
    public void setXSLTServerBasedTransformer(String XSLTTransformerClassName){

this.httpServerCommunicator.setXSLTServerBasedTransformer(XSLTTransformerClassNam
e);
    }
```

```
    /**
     * Set the base directory that holds XSLT template files.
     *
     * @param baseXSLTDirectory is the base directory that holds XSLT templates.
     */
    public void setXSLTFilesBaseDirectory(String baseXSLTDirectory) {
        this.httpServerCommunicator.setXSLTBaseDirectory(baseXSLTDirectory);
    }

    /**
     * Add XLST interface Browser path mapping. When a browser User-Agent
     * matches to one of the mapped paths the alternative directory is used to
     * load the template XSLT. For example:
     *
     * IE6: user-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
     * and
     * setXSLTBaseDirectory("./myXSLT")
     * addXSLTBrowserMapping("*MSIE 6*","/msie6")
     *
     * Means that if a Ie6 browser access the XSLT interface the HTTPServerNode
     * will first try to find the xsltTemplate in ./myXSLT/msie6/xsltTemplate.xsl
     * and if not found then try ./myXSLT/xsltTemplate.xsl.
     *
     * @param regExp is the regular expression for Browser's user-agent: header
field
     * @param xsltSubDir is the sub-directory to look for XSLT template file to
use
     * @param tryServerBasedXSLT if true means it shoudl try server-based XSLT
transformation
     */
    public final void addXSLTBrowserMapping(String regExp, String xsltSubDir,
                                            boolean tryServerBasedXSLT) {
        this.httpServerCommunicator.addXSLTBrowserMapping(regExp, xsltSubDir,
                tryServerBasedXSLT);
    }

    /**
     * Add Composed XLST interface handler.
     *
     * @param address to be accessible (Servlet address)
     * @param topElement is the top XML element
     * @param templateXSLT to be used for manipulation (local filename or URL)
     * @param xmlGenerators is the list of XMLInterface to generate the content
     */
    public final void addXSLTInterface(String address, String topElement,
                                       String templateXSLT,
                                       XMLInterface[] xmlGenerators) {
        this.httpServerCommunicator.addXSLTInterface(address, topElement,
                templateXSLT,
                xmlGenerators);
    }

    /**
     * Add Servlet interface handler.
     *
     * @param address to be accessible (Servlet address)
     * @param servletClassName is the class name of Servlet to be instantiate
     * and link to Servlet container. Notice: the class should implement
     * ServletInterface to be able to retrive this Node's information.
     *
     * @see ServletInterface
     */
```

```
    public final void addServletInterface(String address,
                                           String servletClassName) {
        this.httpServerCommunicator.addServletInterface(address,
                servletClassName);
    }

    /**
     * Generate XML output to be used for XSLT interface integration.
     *
     * @param URI is the requesting URI
     * @param parameters to be taken in consideration to generate the XMLTree.
     * @return XMLTree to be interfaced out
     */
    public XMLTree generateXML(String URI, Hashtable parameters) {
        XMLTree xml = new XMLTree("myself");

        // profile
        Profiler profiler = getProfiler();
        profiler.refresh();
        xml.add(profiler);


        NodeManager nodeManager = getNodeManager();
        xml.add(nodeManager);

        // http-server
//        Hashtable httpServerStats =
//                this.httpServerCommunicator.getHTTPServer().getServerStatus();
//        xml.add(XMLTree.hashToXMLTree("http-server", httpServerStats));

        return xml;
    }




    /**
     * COMMAND-LINE ENTRY POINT.
     * Check this class' header for instrucions on how to execute this class
     * from command-line.
     *
     * @param args String[] arguments passed-in by command line
     */
    public static void main(String[] args) {

        // read command-line
        CommandLine cmd = new CommandLine(
                new String[][] { {"name", "s+", "", "process name (mandatory)"}
                , {"port", "i", "8000", "server port (default 8000)"}
                , {"provider", "s+", "http://localhost:8000/",
                "service provider"}
                , {"services", "s", "", "list of services to add",
                "class[;class]*"}
                , {"snoop", "b", "", "snoop mode"}
        });

        // Process the incoming command line arguments
        // Generate the help page in case of error
        if (!cmd.getOptions(args, false)) {
            cmd.printHelp();
            return;
        }
```

```
        // Collect the parsed values with getArgument*() methods
        String name = cmd.getArgument("name");
        int port = cmd.getArgumentInt("port");
        String serviceProvider = cmd.getArgument("provider");
        String serviceListStr = cmd.getArgument("services");
        boolean snoopMode = cmd.getArgumentBoolean("snoop");

        // initialize and start Service Provider
        HTTPServerNode node = new HTTPServerNode(name, port);
        node.setServiceProvider(serviceProvider);
        node.getHTTPServer().setSnoopMode(snoopMode);

        // set logger
        node.setLogOutput(System.err);
        node.setLogLevel("node", Logger.DETAIL);
        node.setLogLevel("net", Logger.DETAIL);
        node.setLogLevel("provider", Logger.DETAIL);

        node.log("node", 1, "----------------------------------------");
        node.log("node", 1, "STARTING DUMMY GRID NODE");
        node.log("node", 1, "Compilation: " + Node.VERSION);
        node.log("node", 1, "----------------------------------------");

        // add command-line services to provider
        if (serviceListStr != null) {
            StringTokenizer st = new StringTokenizer(serviceListStr, ";");
            while (st.hasMoreElements()) {
                String serviceClassName = st.nextToken().trim();
                try {
                    ServiceInterface service =
                            (ServiceInterface) Class.forName(serviceClassName).
                            newInstance();
                    node.addService(service);
                } catch (Exception ex) {
                    System.err.println("EXCEPTION: " + ex);
                }
            }
        }

        // start provider
        node.start();

        // register provider
        node.register();
    }

    public XMLTree getQoSAttributes() {
        return null;
    }


}
```

APÊNDICE B – Artigo

OBTENÇÃO DE DADOS DE QoS DOS ELEMENTOS DE UM GRID MÓVEL

Rudson Vieira de Souza

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brazil

rudroofs@inf.ufsc.br

Resumo*: O Grid-M é um projeto que visa desenvolver um middleware para aplicações do tipo Grids computacionais. Os atributos de QoS deste projeto antes da concepção deste trabalho estavam restritos aos nodos. Como alteração de tal paradigma, foi feita a criação de um sistema de coleta de atributos que incorpora os elementos de tal nodo (hooks, serviços e sensores), de modo a fomentar uma nova tendência ou metodologia na gerência de QoS em tais estruturas. Tais atributos foram dispostos em forma de código XML para complementar o modelo criado, atendendo ao padrão do projeto. Ao final do mesmo pode-se verificar a criação, mesmo que embrionária e desprovida de sofisticação, de uma nova forma de coletar atributos de QoS em Grids computacionais.*

Abstract*: Grid-M is a project that seeks to develop a middleware for applications of the type Grid Computing. The attributes of QoS of this project before the conception of this work were restricted to the nodes. As alteration of such paradigm, it was made the creation of a system of collection of attributes that incorporates the elements of such node (hooks, services and sensor), in way to foment a new tendency or methodology in the management of QoS in such structures. Such attributes were disposed in XML code to complement the model servant, assisting to the pattern of the project. At the end of the same can be verified the creation, even that embryonic and without sophistication, of a new form of collecting attributes of QoS in Grid Computing.*

1. Introdução

Desde o início do século passado temos visto uma rápida evolução sobre os computadores tanto do hardware quanto do software. Criaram-se diversas novas funcionalidades, novos paradigmas, novas linguagens de programação, novos processadores, novas teorias.

Um dos novos paradigmas em ambientes computacionais são os Grids. Os Grids computacionais são estruturas abstratas capazes de compartilhar processamento de requisições e utilização de memória por redes de computadores. Os grids não têm fronteiras políticas vigentes e trafegam informação por todo o globo.

Entretanto, com o emprego em larga escala de redes de computadores, cresce o receio com as falhas de comunicação inerentes a estas estruturas, e é necessário o estudo de

técnicas e metodologias que possam melhorar os serviços. Uma das técnicas para a melhoria dos serviços utilizada é a Qualidade de Serviço ou QoS (*Quality of Service*), que procura ao máximo atender aos requisitos de qualidade que uma aplicação deverá atingir através da definição de parâmetros aceitáveis.

Grandes projetos nas áreas de Grids aliam a reusabilidade provida por estes com o gerenciamento de QoS para obter os melhores resultados e se precaver de falhas que comprometam seu funcionamento. O projeto Grid-M não é diferente apesar de estar em fase embrionária o mesmo inspira alterações na forma de gerenciar o QoS. No intuito inicial o mesmo seria obtido como na maioria dos projetos nesta área, sobre o elemento principal - o nodo. Posteriormente foi decidido que a coleta dos atributos de QoS seria feita sobre os elementos do nodo, a fim de prover futuramente uma nova forma de gerenciamento de QoS. Esta é a motivação para o desenvolvimento deste projeto.

Este projeto tem como tema principal a obtenção de atributos de QoS dos elementos dos nodos componentes de um Grid Computacional Móvel, e posterior exibição dos mesmos para fins de gerenciamento do ambiente.

Neste trabalho será usado o Grid Computacional Móvel também denominado Grid-M (*Middleware for Embedded and Mobile Grid Computing*) desenvolvido por colaboradores do Laboratório de Redes e Gerência (LRG) desta instituição. Esta exploração será realizada de forma mais centralizada, não utilizando uma rede de computadores mais somente um único computador que proverá as simulações através de um Ambiente de Desenvolvimento, o JBuilder da Borland que vem sendo utilizado desde a concepção do projeto do Grid-M.

Este artigo está organizado da seguinte forma: a seção 2 apresenta um estudo teórico sobre Grids Computacionais, a seção 3 sobre QoS, a seção 4 aborda os conceitos do Projeto Grid-M, base deste projeto, a seção 5 apresenta como o modelo de obtenção de dados de QoS foi elaborado, a seção 6 apresenta os resultados obtidos e na seção 7 são comentadas as conclusões.

2. Grids Computacionais

Um Grid Computacional é uma combinação de recursos de hardware e software com o intuito de distribuir as cargas de processamentos entre seus componentes e assim viabilizar resultados de maneira mais ágil e eficiente.

Dentre seus objetivos estão a busca da interoperabilidade entre organizações, políticas operacionais e tipos de recursos; o acoplamento destes recursos distribuídos oferecendo acesso consistente e de baixo custo a recursos independente da sua localização física.

Como Características dos Grids temos a heterogeneidade (multiplicidade de recursos, tratamento sobre as diferenças de processadores, velocidade e arquitetura), escalabilidade (pode crescer de poucos recursos para milhões, o problema em potencial é o desempenho a medida que o tamanho aumenta), dinamicidade ou adaptabilidade (falha de um recurso é regra, e não uma exceção, gerenciar recursos e organizar seu comportamento para extrair desempenho a partir dos recursos e serviços disponíveis), além disso, possuem alta dispersão geográfica, status do sistema e tolerância a falhas, baixo custo, portabilidade das aplicações, entre outros.

A reusabilidade, economia, utilização eficiente dos recursos, e o compartilhamento dos recursos dedicados são algumas das vantagens da utilização de Grids. As desvantagens da utilização de tal sistema se dão com o aumento do mesmo e podem acarretar alta complexidade, elevação do custo para concepção do mesmo e dificuldade em alterar as aplicações.

Como nota de conclusão vale ressaltar o grande número de projetos nesta área, que podem ser observados na página do Grid Computing Info Centre (http://www.gridcomputing.com) que apresenta algumas de dezenas de projeto nesta área. Em âmbito nacional pode-se citar o projeto Grid Brasil (http://www.gridbrasil.com.br). Para adotar os Grids Computacionais é necessário trabalhar de forma colaborativa: usuários, sistemas e recursos podem ser integrados de modo a formar uma grande plataforma de desenvolvimento de ferramentas e resolução de problemas.

3. QoS

Qualidade de serviço (QoS) é um tema que possui várias definições dependendo do contexto em que é aplicado. Algumas visões diferentes sobre o conceito são identificadas na lista a seguir:

- ISO: Na visão da ISO, QoS é definida como o efeito coletivo do desempenho de um serviço, o qual determina o grau de satisfação de um usuário do serviço. ISO/IEC (1995, apud Kamienski e Sadok, 2000, p. 4)

- Sistemas Multimídia Distribuídos: Em um sistema multimídia distribuído a qualidade de serviço pode ser definida como a representação do conjunto de características qualitativas e quantitativas de um sistema multimídia distribuído, necessário para alcançar a funcionalidade de uma aplicação. Vogel et al. (1995, apud Kamienski e Sadok, 2000, p. 4)

- Redes de Computadores: QoS é utilizado para definir o desempenho de uma rede relativa às necessidades das aplicações, como também o conjunto de tecnologias que possibilita às redes oferecer garantias de desempenho, segundo Teitelman e Hanss (1998, apud Kamienski e Sadok, 2000, p. 4). Em um ambiente compartilhado de rede, QoS necessariamente está relacionada à reserva de recursos. Essa reserva pode ser feita para um conjunto (agregação) de fluxos ou sob demanda para fluxos individuais. QoS pode ser interpretada como um método para oferecer alguma forma de tratamento preferencial para determinada quantidade de tráfego da rede. Em outra definição, QoS é vista como a capacidade de um elemento de rede ter algum nível de garantia de que seus requisitos de serviço e tráfego podem ser satisfeitos. Stardust.com (1999, apud Kamienski e Sadok, 2000, p. 4)

A junção dos termos qualidade e serviço podem dar margem a várias interpretações e definições diferentes. No entanto, existe um certo consenso, que aparece em praticamente todas as definições de QoS, que é a capacidade de diferenciar entre tráfego e tipos de

serviços, para que o usuário possa tratar uma ou mais classes de tráfego diferente das demais. Pode-se então classificar QoS de acordo com o nível de garantia oferecido:

- QoS baseado em reserva de recursos, ou rígido, que oferece garantias para cada fluxo individualmente. Esse é o tipo de QoS que seria o ideal mas é mais complexo (e caro) de implementar.

- QoS baseada em priorização, ou flexível, onde as garantias são para grupos, ou agregações de fluxos. Nesse caso, cada fluxo individual não possui garantias. CoS utiliza esse conceito, que é mais fácil de implementar, por isso mais provável de ser disponibilizado em uma rede como a Internet em um futuro próximo.

Outro componente importante para a determinação do modelo de QoS a ser fornecido aos usuários diz respeito ao tipo de tráfego que as aplicações geram e qual o comportamento esperado da rede para que elas funcionem corretamente. Com relação ao tipo de tráfego as aplicações podem ser classificadas segundo Braden, Clark e Shenker (1994, apud Kamienski e Sadok, 2000, p. 5) e Ferguson e Huston (1998, apud Kamienski e Sadok, 2000, p. 5) em :

- Aplicações de tempo real (não elásticas): Podem ser definidas como aquelas com características rígidas de reprodução (play-back), ou seja, um fluxo de dados é empacotado na fonte e transportado através da rede ao seu destino, onde é desempacotado e reproduzido pela aplicação receptora. As aplicações tolerantes são aquelas que mesmo diante de variações no atraso (jitter) causadas pela rede, ainda assim produzem um sinal de qualidade quando reproduzidas. Já nas aplicações intolerantes variações no atraso produzem sinais (de áudio ou vídeo, por exemplo) com qualidade inaceitável.

- Aplicações elásticas (não tempo real, ou adaptáveis): Para esse tipo de aplicação, a recepção correta dos dados é mais importante do que a sua apresentação em uma taxa constante. Exemplos de aplicações elásticas são correio eletrônico, transferência de arquivos, consultas interativas a informações (como na Web) e aplicações cliente/servidor tradicionais.

O conceito de QoS apresenta ainda grande complexidade e indefinições. Para aplicações em Grids, os principais parâmetros de QoS são: o tempo de resposta; throughput (dados transferidos); disponibilidade; e o escalonamento das tarefas (distribuição de processamento).

4. Projeto Grid-M

O projeto do Grid-M foi desenvolvido pelo Laboratório de Redes e Gerência (LRG) desta instituição de ensino. Seu conceito é o de uma plataforma para construção de aplicações para Grids Computacionais que possuem dispositivos móveis para comunicação,

seja para a obtenção de dados para processamento, seja para a disposição do mesmo para consultas. Ele prove uma API para ser utilizada por aplicações Java no ambiente de Grids.

Grids Computacionais provêm os protocolos para possibilitar interoperabilidade e infra-estrutura em comum. No contexto deste projeto, os Grids são plataformas para padronizar computação distribuída.

A falta de padrões para plataformas de desenvolvimento para computação distribuída levou a criação de soluções ad hoc com custos para desenvolvimento e para reusabilidade. Os Grids possuem a vantagem neste contexto de oferecer reusabilidade considerando que o mesmo disponibiliza padrões para a construção de ferramentas para a transferência de serviços em dispositivos móveis e embutidos. Esses padrões tangem a padrões abertos, para protocolos e interfaces (para comunicação de dados), ferramentas de gerenciamento, padronização de interfaces programáveis.

O elemento principal deste projeto é o nodo, estrutura definida no contexto de Grids como sendo um ponto de união entre varias redes. Tais nodos possuem elementos que neste projeto são os:

- Hooks – Responsáveis pela requisição de serviços tem seu conceito interligado ao conceito de tarefas. Geralmente enviam as tarefas a um nodo que dispõe do serviço procurado, mas também podem ser utilizados no próprio nodo;

- Serviços – Determinam o que os nodos podem ou não oferecer aos demais. São serviços oferecidos neste projeto: o serviço de autenticação, registro, encontrar nodos, encontrar rotas, entre outros. A lista de serviços disposto por cada nodo pode ser verificado pelo Diretório de Serviços inerente ao mesmo.

- Sensores – Responsáveis pela coleta dos dados. Podem coletar tanto informações do nodo ao qual foi adicionado, bem como, de dispositivos externos acoplados ao nodo. No projeto Grid-M estes dispositivos são de comunicação móvel, como antenas, PDAs, entre outros. Além de coletar dados quando necessário, um sensor pode obter dados através do agendamento das tarefas em tempos estipulados pelo usuário.

Para maiores informações sobre este projeto favor visitar a pagina do mesmo em http://grid.lrg.ufsc.br/.

5. Obtenção de Dados de QoS dos Elementos do Grid-M

Os atributos de QoS obtidos informam somente a situação do nodo em geral. Tais atributos contidos na classe Node levantavam os parâmetros a respeito da execução do mesmo, e como este estava se comportando

Como proposta para melhoria destas informações, surgiu a necessidade de coletar os atributos não do nodo em geral, mas sim dos elementos que compõe o mesmo (Hooks, Sensores e Serviços), com o intuito de se criar uma plataforma de gerenciamento de QoS em Grids computacionais, já que a literatura não possui referência a outro ambiente que considere estas questões. Tais atributos deveriam utilizar rotinas e valores básicos fornecidos por uma classe abstrata, de quem estenderiam estes valores e rotinas.

5.1 A classe ManagedElement

A classe abstrata ManagedElement foi criada para prover aos elementos do nodo (classe Node) os valores e rotinas para obtenção dos atributos de QoS. Como tais elementos estão inseridos na classe Node, esta passou a estender a classe ManagedElement.

Para cada elemento foi criado uma série de valores condizentes com a operação de cada um no nodo. Para os hooks (tarefas encaminhadas a outro nodo) foram criados valores que informem quantas vezes o mesmo foi executado, quantas vezes o mesmo conseguiu obter resultado para determinada solicitação, o tempo médio de resposta para as requisições, quantas vezes o mesmo falhou , e quais erros foram retornados .

Como os sensores têm até então somente o intuito de coletar as informações do perfil do nodo, foram criados valores para o mesmo que informam quantas vezes ele foi acionado, quantas vezes essa execução não se deu de forma agendada e quantas vezes esta coleta foi obtida através de agendamento .

Os serviços são os elementos que possuem a implementação mais completa neste projeto. Para os serviços foram criados valores que remetem ao processo original de obtenção de atributos que possuía como parâmetro divisor a forma de processamento (Foreground e Background). Assim sendo os valores dos serviços informam o numero de vezes que o serviço foi executado, quantas vezes ele fora executado em Foreground, quantas vezes este foi processado em Background, o tempo médio gasto com cada processamento, o tempo médio de execução de cada serviço e quantas falhas ocorreram para cada serviço.

Com estes valores foi possível construir as rotinas de criação, atualização e obtenção dos atributos de QoS dos elementos de um nodo. Tanto as rotinas de criação e de atualização, inicializam e incrementam, respectivamente, os valores dos parâmetros retornando-os a um Vetor para serem armazenados. Este, por conseguinte é armazenados em um Hashtable que identifica o elemento.

Na rotina de obtenção de dados, os valores contidos no Hashtable são repassados a um vetor, e os elementos deste são transformados em uma XMLTree (uma árvore XML concebida através da extensão que a classe ManagedElement faz da classe XMLTree que possui todos os métodos e mecanismos necessários para gerar tal árvore) retornando tal estrutura ao objeto da classe solicitante. Estas XMLTrees são armazenadas em uma classe denominada NodeManager que além de armazenar, posteriormente exibirá o código XML pertinente aos atributos de QoS dos elementos do nodo.

5.2 A classe NodeManager

Com o intuito de armazenar as XMLTrees retornadas pela classe ManagedElement e posteriormente exibir o código XML dos atributos de QoS dos elementos do nodo, foi criada a classe NodeManager que realiza extensão a classe XMLTree e implementa a interface XMLInterface. Tal extensão permite que a mesma possa incorporar os atributos e executar os métodos para construção de árvores XML que por sua vez serão transformadas em código XML pelo método generateXML() pertencente a interface XMLInterface.

Para esta classe foram criados métodos para adicionar a XMLTree dos elementos e organizá-las em árvores maiores que compõe todos os serviços, sensores e hooks para cada nodo, tendo como elemento pai a tag Node_Manager. Estas XMLTrees são adicionadas a

um Hashtable pertence a cada elemento que por sua vez é armazenado em um vetor no local pré-disposto para tal e finalmente inseridos em uma Hashtable que armazena todas as informações do nodo vigente. Para a geração das árvores XML, são retornadas as árvores de cada elemento pertencente ao nodo. Estas árvores são adicionadas ao elemento pai do nodo para a exibição do código XML do mesmo.

Para a construção da XMLTree completa que será usada para a exibição do código XML será invocado o método generateXML() que adiciona aos nós Node_Manager e node criados pelo construtor da classe as demais árvores geradas pelos métodos explicitados anteriormente. Tal invocação será realizada pelo método addXSLTInterface implementado na classe HTTPServerNode.

5.3 Coleta dos Dados e Exibição de Código XML

Para realizar a coleta proposta neste trabalho seria necessário obter os atributos de cada elemento, porém os elementos pertencentes a um nodo não estão em uma classe separada, todos eles estão concentrados na classe Node, que implementa os métodos e os valores para estes elementos. Sendo assim a classe Node realiza extensão da classe ManagedElement como mencionado anteriormente com o intuito de obter as rotinas e valores para a finalidade desejada.

Para a criação dos valores dos elementos foram adicionadas entradas dos métodos de criação nos métodos criadores dos elementos. Para a atualização destes valores os métodos de atualização foram inseridos nos métodos que executam as ações pertinentes aos elementos. Após a criação e atualização dos atributos fica faltando somente a criação das XMLTrees correspondentes.

Como visto, tais estruturas são criadas na classe ManagedElement e armazenados na classe NodeManager para futura exibição em código XML. Mas para que ocorra tal interação foram inseridos nos métodos que realizam a atualização dos atributos chamadas ao método de obtenção das XMLTrees da classe NodeManager que possui como parâmetro uma estrutura na forma XMLTree,

Visualizar o conteúdo dos atributos através de código XML será realizado pela classe HTTPServerNode que através de uma instância da classe NodeManager e da chamada ao método addXSLTInterface que retorna o conteúdo das XMLInterface solicitadas. Este último método possui como parâmetros o cabeçalho do endereço, o elemento pai, o caminho para um arquivo XSL e um array com as diversas XMLInterface. Sendo assim o mesmo dispõe no endereço descrito no seu parâmetro o arquivo em código XML das XMLInterfaces requeridas que foram implementadas nas classes Profiler, DirectoryService e NodeManager. Para separar e somente realizar a exibição do código XML da classe NodeManager foi criado um novo endereço “/manager/” e solicitado somente a exibição da XMLInterface pertencente a esta classe através de uma nova chamada ao método assXSLTInterface (“/manager/”, “Manager”, “manager.xsl”, XMLInterface[] {nodeManager}). O código XML dos atributos de QoS dos elementos do nodo será exibido no próximo tópico.

6. Resultados Obtidos

Para aferir sobre o funcionamento das implementações realizadas bem como visualizar o código XML esperado com os atributos de QoS foram realizados testes através de classes de exemplos inseridas no projeto. Não foi possível realizar testes com outras implementações devido ao projeto Grid-M ainda estar em construção não disponibilizando todas as funcionalidades previstas.

As classes usadas foram ExampleNodeRegistering e ExampleSensorNode. Na primeira é solicitado o registro de um novo nodo (nodo-1) ao nodo provider que executa tal registro. Além disto, no primeiro teste o nodo-1 agenda a coleta de informações do sensor node e tenta lançar o sinal de HEARTBEAT para informar o Directory Service que o nodo está ativo. Na segunda além do registro do novo nodo (nodo-1) ao nodo provider, são agendadas coletas nos sensores node criado pelo próprio nodo em sua inicialização e no novo sensor denominado collector, existe uma diferença de 5 segundos entre as coletas dos sensores. Neste teste também há a criação do serviço store no nodo-1 que terá seus dados capturados pelo sensor collector. Os resultados destes testes são disponibilizados na figura 1, 2 e 3.

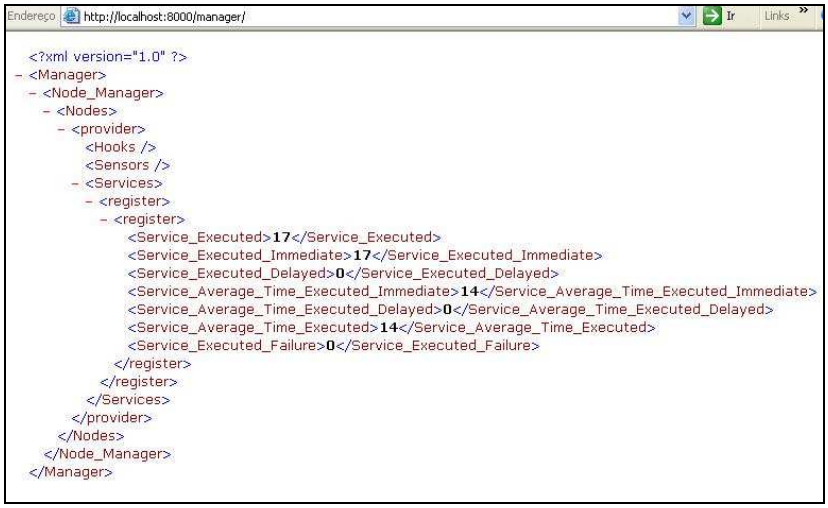

Figura 1 – Código XML do nodo provider

Como o nodo provider somente processa o serviço de registro (register), em ambos os testes aparecem somente a árvore para este serviço. Pode-se observar que os elementos hooks e sensores estão somente como folhas sem nenhum elemento agregado.

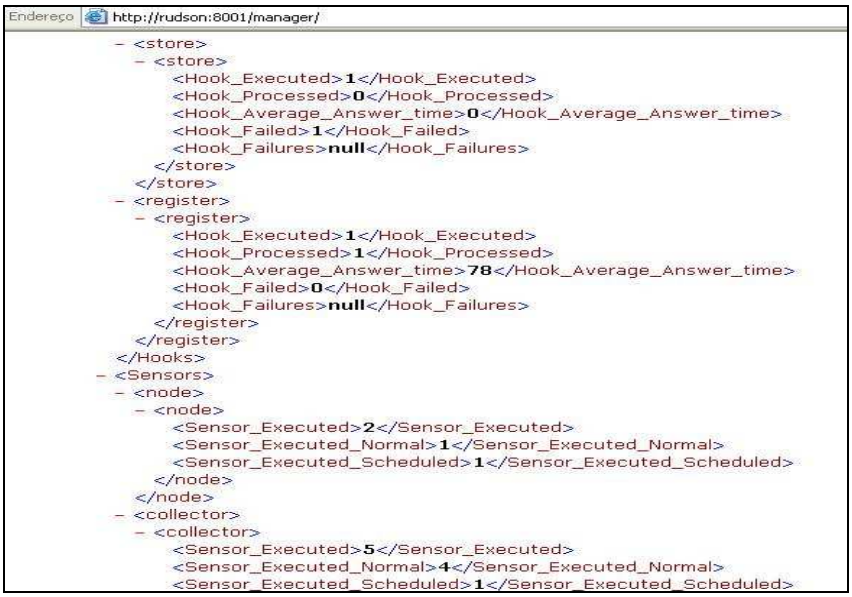

Figura 2 – Trecho do código XML do nodo-1 no primeiro teste

Neste trecho do código XML do nodo-1 no primeiro teste, na figura 19, pode-se observar claramente os dois hooks solicitados e a contabilização das coletas efetuadas pelo sensor node.

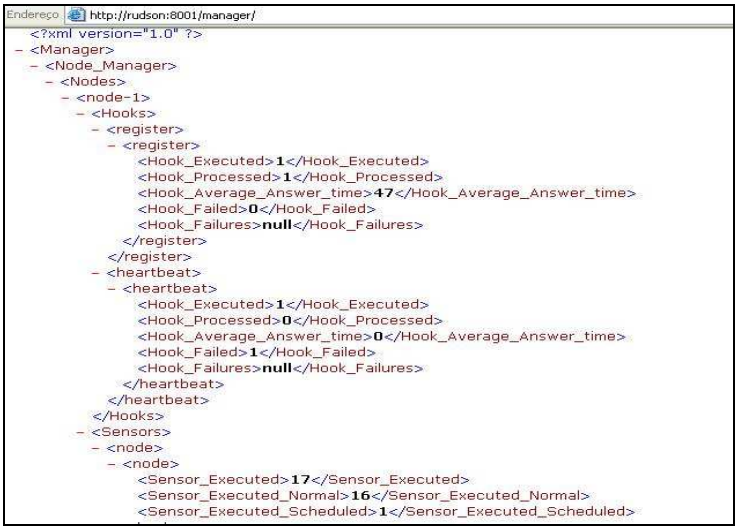

Figura 3 – Trecho do código XML do nodo-1 no segundo teste

Podemos averiguar no segundo teste - na figura 20 - a diferença no tempo das coletas dos sensores, bem como a adição do hook para solicitar o serviço store no nodo provider: como tal serviço não está implementado retornou ao hook a mensagem de falha.

Após estes testes, pode-se aferir sobre a eficácia dos métodos de coleta e atualização, que geram o código XML dos atributos de QoS dos elementos dos nodos. O único problema encontrado foi a duplicata da identificação dos elementos na árvore, fruto da inexistência de um construtor para a XMLTree que não crie nenhuma tag, perpetuando este problema.

7. Conclusão e Trabalhos Futuros

Tendo em vista a elaboração deste trabalho a principal desvantagem em se realizar um trabalho com Grids computacionais é a grande complexidade que o mesmo atinge quando está em constante crescimento. Foi necessário estudar minuciosamente um projeto muito extenso com diversas classes e muita informação.

Superada esta fase foi possível identificar facilmente os pontos que deviam ser alterados, porém tais alterações deveriam ser cuidadosamente testadas para verificar possíveis efeitos colaterais no projeto como um todo. Quando os testes retornaram os valores desejados foi possível constatar que apesar de pequeno, este trabalho pode ser um passo importante para uma nova metodologia de gerenciamento de QoS em Grids computacionais – através dos elementos dos nodos.

Este trabalho foi importante para a revisão de conhecimentos adquiridos durante o curso, bem como a atualização e aprimoramento dos mesmos. Mesmo assim, servem de alerta os problemas encontrados durante sua execução: é necessário despender de muito tempo para entender todo o projeto, e também é necessário obter o máximo de informação com os criadores dos códigos.

Como sugestão para trabalhos futuros fica a possibilidade de se criar interfaces xsl padrões para exibição dos resultados das aplicações no Grid-M. Também seria possível desenvolver um trabalho para utilizar os atributos de QoS dos elementos para fomentar uma nova metodologia de gerenciamento de QoS.

Uma última sugestão seria a criação de uma plataforma para simulação e testes em ambientes compostos por Grids computacionais, que pudesse automatizar e aumentar o potencial dos testes.

Referências

KAMIENSKI, Carlos Alberto; SADOK, Djamel. **Qualidade de Serviço na Internet.** 2000. Centro de Informática, Universidade Federal de Pernambuco, Belo Horizonte, 2000.

PY, Mônica Xavier. **Grid Computacional - O que precisa-se para ter um!!.** Faculdades Rio-Grandenses, 2005.

Freitas, Bruno; Barros, Roberto S. M. **XML – XSL-FO.** Centro de Informática, Universidade Federal de Pernambuco, 2004.

Refsnes Data, **W3Schools Online Web Tutorials.** Disponível em: <http://www.w3schools.com>. Acesso em: 24 fevereiro 2007.

ANEXO A – Código das classes usadas nos testes

Classe ExampleNodeRegistering

```
package br.ufsc.lrg.grid.examples;

import br.ufsc.lrg.grid.*;
import java.util.*;

/**
 * ExampleNodeRegistering.
 * In this example, a node registers into a ServiceProvider (also started
 * by this code) and keep sending context information in intervals of time.
 * This software may be distributed under the
 * GNU GENERAL PUBLIC LICENSE
 * Version 2, June 1991
 * {@link http://www.gnu.org/licenses/gpl.txt}
 * @compatible J2SE
 * @project Grid-M Middleware for Mobile and Embedded Grid Computing
 * @author LRG-UFSC (http://grids.lrg.ufsc.br) -- Fernando Koch
 *
 * @see Node
 * @see ServiceProviderNode
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
*/
public class ExampleNodeRegistering {

    /**
     * COMMAND-LINE ENTRY POINT.
     * Check this class' header for instrucions on how to execute this class
     * from command-line.
     *
     * @param args String[] arguments passed-in by command line
     */
    public static void main(String[] args) {

        // initialize and start Service Provider
        ServiceProviderNode.main(new String[] {"-name", "provider", "-port",
"8000",
                                "-domain", "myGrid", "-snoop"});

        // initialize and start Node
        HTTPServerNode node1 = new HTTPServerNode("node-1", 8001);
        node1.setServiceProvider("http://localhost:8000/");
        node1.setLogOutput(System.err);
        node1.setLogLevel("node", Logger.DETAIL);
        node1.setLogLevel("net", Logger.DETAIL);
        node1.start();

        // register and program to send context information each 30 secs
        node1.register();
        node1.scheduleContextCollect(3);
        node1.scheduleHeartbeat(Node.HEARTBEAT_INTERVAL);

    }
}
```

Classe ExampleSensorNode

```
package br.ufsc.lrg.grid.examples;

import br.ufsc.lrg.grid.*;
import java.util.*;

/**
 * ExampleSensorNode.
 * In this example, a node has a very simple Sensor and schedule to collect
 * data and send to Service Provider.
 *
 * <pre>
 * This software may be distributed under the
 * GNU GENERAL PUBLIC LICENSE
 * Version 2, June 1991
 * {@link http://www.gnu.org/licenses/gpl.txt}
 * </pre>
 *
 * @compatible J2SE
 * @project Grid-M Middleware for Mobile and Embedded Grid Computing
 * @author LRG-UFSC (http://grids.lrg.ufsc.br) -- Fernando Koch
 *
 * @see Node
 * @see ServiceProviderNode
 * @see http://grid.lrg.ufsc.br/docs/GridMProgramming.pdf
 *
 */

public class ExampleSensorNode {

    /**
     * COMMAND-LINE ENTRY POINT.
     * Check this class' header for instrucions on how to execute this class
     * from command-line.
     *
     * @param args String[] arguments passed-in by command line
     */
    public static void main(String[] args) {

        // initialize and start Service Provider
        ServiceProviderNode providerNode = new ServiceProviderNode("provider",
"myGrid", 8000);

        // set logger
        providerNode.setLogOutput(System.err);
        providerNode.setLogLevel("node", Logger.DETAIL);
        providerNode.setLogLevel("provider", Logger.DETAIL);
        providerNode.getHTTPServer().setSnoopMode(true);

        providerNode.log("node", 1, "----------------------------------------
--------");
        providerNode.log("node", 1, "STARTING EXAMPLE SENSOR NETWORK SERVICE
PROVIDER");
        providerNode.log("node", 1, "Compilation: "+Node.VERSION);
        providerNode.log("node", 1, "----------------------------------------
--------");

        // add STORE service
        ExampleStoreServices storeService = new ExampleStoreServices();
        providerNode.getHTTPServer().setLog(10, System.err);
        providerNode.addService(storeService);
```

```
        providerNode.addXSLTInterface("/store", null, "store.xsl", new
XMLInterface[] {storeService});

        // start provider
        providerNode.start();

        // initialize a Sensor Node
        HTTPServerNode node1 = new HTTPServerNode("node-1", 8001);
        node1.setServiceProvider("http://localhost:8000/");
        node1.setLogOutput(System.err);
        node1.setLogLevel("node", Logger.DETAIL);
        node1.setLogLevel("net", Logger.DETAIL);

        // initialize sensor
        SensorInterface sensor = new ExampleSensor();
        node1.addSensor("collector", sensor);
        node1.scheduleSensorCollect("collector" , 5, "*", "store",
Task.PRIORITY_NORMAL, null);
        node1.scheduleContextCollect(10);

        // start and register
        node1.start();
        node1.register();
    }
}
```