Um Modelo de Gerenciamento de Redes de Telecomunicações Utilizando a Plataforma CORBA

Junior Toshiharu Saito, Edmundo Madeira
Instituto de Computação – UNICAMP
{junior.saito, edmundo}@ic.unicamp.br

Abstract

The network management has been a very important task, mainly in telecommunication networks. This fact is caused by increasing of size and complexity of the networks which raises difficulties to detect faults and low performance. To help in this task, a framework was defined and many tools were developed. In this paper, an architecture is presented, which shows how a telecommunication network can be managed using the CORBA Notification Service.

Resumo

O gerenciamento de rede tem se tornado uma tarefa muito importante, principalmente nas redes de telecomunicações. A causa disto é o aumento do tamanho e da complexidade que dificulta a detecção de falhas e de baixo desempenho. Para ajudar nesta tarefa, uma arquitetura foi definida e várias ferramentas foram desenvolvidas. Neste trabalho, uma arquitetura será apresentada, mostrando como uma rede de telecomunicações pode ser gerenciada usando o Serviço de Notificação CORBA.

Palavras-chave: Gerenciamento de redes de telecomunicações, sistemas distribuídos, CORBA, Serviço de Notificação e correlação de eventos.

1. Introdução

A gerência de redes é uma atividade vital para o funcionamento das redes de comunicações e dos sistemas distribuídos. Com o crescimento da rede, tanto em tamanho quanto em complexidade, e com a sua utilização em serviços importantes, surgiu a necessidade de monitorar a rede para que os serviços e os equipamentos existentes não parem de funcionar. A partir da gerência é possível detectar quais elementos da rede estão prejudicando o desempenho.

Devido ao tamanho e à complexidade das redes de telecomunicações, surgiu a necessidade de ferramentas e protocolos para auxiliar na tarefa de gerência, tanto dos dispositivos quanto dos serviços. Basicamente, as ferramentas são compostas de gerentes e agentes e a comunicação entre eles é feita através de notificações de eventos. Estes eventos são produzidos pelos elementos gerenciados podendo indicar a ocorrência de uma falha.

Com o aumento de usuários das redes, novos equipamentos e serviços são instalados para atendê-los. Com isso, as ferramentas devem possuir atributos como flexibilidade, escalabilidade, confiabilidade e interoperabilidade.

O OMG apresentou uma solução utilizando a arquitetura CORBA inicialmente voltada para a gerência de redes de telecomunicações, entretanto sua utilização pode abranger outros tipos de redes. Esta solução veio através dos Serviços de Eventos [14] e de Notificação [15], e do Domínio de Eventos [18] que permite a comunicação através de notificações de eventos. O Serviço de Notificação veio para corrigir algumas deficiências existentes no Serviço de Eventos, como ausência de filtros, eventos estruturados e qualidade de serviço. Já o Domínio de Eventos veio como uma extensão do Serviço de Notificação, fornecendo novas funcionalidades tais como redes de canais de eventos e conexões entre os canais de diferentes desenvolvedores.

A proposta deste trabalho é apresentar uma arquitetura que utiliza o Serviço de Notificação e o Domínio de Eventos para o gerenciamento de redes de telecomunicações, formando uma rede de transmissão de eventos e integrando módulos com tarefas específicas como a correlação de eventos e log à arquitetura.

A seguir na Seção 2 será apresentada uma descrição dos conceitos utilizados neste trabalho. Na Seção 3 será apresentada a arquitetura proposta. A implementação será descrita na Seção 4, e a Seção 5 apresenta a conclusão e trabalhos futuros.

2. Conceitos

Para o desenvolvimento deste trabalho, foram utilizados conceitos e tecnologias que englobam áreas como CORBA e gerenciamento de redes de telecomunicações. Estes conceitos e estas tecnologias serão descritos a seguir.

2.1. CORBA

CORBA (*Common Object Request Broker Architecture*) é a plataforma definida pelo OMG (*Object Management Group*) para a interoperabilidade entre as várias aplicações desenvolvidas com diferentes linguagens de programação, utilizadas em diferentes sistemas operacionais. A estrutura desta plataforma é descrita a seguir.

2.1.1. Estrutura

O OMG publicou a OMA (*Object Management Architecture*), que define os principais componentes presentes na arquitetura CORBA: o ORB, os serviços CORBA, as facilidades CORBA, as interfaces de domínio e os objetos da aplicação.

O OMG define como ORB (*Object Request Broker*), um *middleware*, o mecanismo que permite a comunicação transparente entre os clientes e os servidores. Através deste ORB, um cliente pode invocar um método existente no servidor, sem se preocupar com a localização, linguagem de programação utilizada ou sistema operacional, uma vez que isso é tarefa do ORB.

Os Objetos de Aplicação representam os objetos que realizam tarefas específicas para os usuários finais. As Interfaces de Domínio representam áreas verticais que fornecem funcionalidades de interesse direto de usuários finais em aplicações específicas. Como exemplos de domínios têm-se: Transporte, Financeiro, Comércio Eletrônico, Saúde e Telecomunicações. O Serviço de Notificação e o Domínio de Eventos se encontram no domínio das Telecomunicações.

As Facilidades Comuns, também conhecidas como *CORBAFacilities*, fornecem um conjunto de funções genéricas que podem ser configuradas para atender determinados requisitos. Estas facilidades incluem funções para impressão, gerenciamento de documentos, bases de dados, facilidades para correio eletrônico, entre outras.

Os Serviços de Objetos são blocos para aplicações distribuídas que fornecem serviços de uso geral para facilitar o desenvolvimento de aplicações CORBA. Estes blocos podem ser utilizados e combinados de diferentes formas.

2.2. Gerenciamento de Redes

O gerenciamento de redes é de grande importância para a manutenção e o funcionamento da rede. O gerenciamento de rede pode ser visto como um conjunto de mecanismos operacionais e administrativos necessários para controlar os recursos da rede, manter os recursos da rede operacionais, facilitar o aumento da rede, gerenciar os recursos e controlar o acesso à rede.

Com o objetivo de padronizar as redes de telecomunicações, a ITU definiu o TMN (*Telecommunications Management Network*) [24] como padronização para a gerência destas redes. O TMN apresenta uma arquitetura para permitir a interconectividade e a comunicação entre os diferentes sistemas que podem existir na rede. A seguir, o modelo TMN será descrito.

2.2.1. O Modelo TMN

O modelo fornecido pelo TMN foi desenvolvido para permitir flexibilidade, confiabilidade, interoperabilidade, escalabilidade e facilidade para implementá-lo. Esta arquitetura é incorporada nas redes de telecomunicações, permitindo receber e mandar informações e gerenciar os recursos existentes na rede. O modelo TMN é definido na série de especificações M.3000 da ITU, onde são apresentadas as várias arquiteturas do modelo TMN, para este trabalho será descrita apenas a arquitetura que apresenta os componentes de maior interesse: a arquitetura física.

A principal tarefa da arquitetura funcional é fornecer a interconectividade e a comunicação entre os sistemas operacionais e as redes de telecomunicações. Isso é possível através das interfaces que são vistas pelos recursos gerenciados como objetos. Esta arquitetura é composta por blocos que possuem determinadas funções. Os blocos são apresentados na Figura 1.

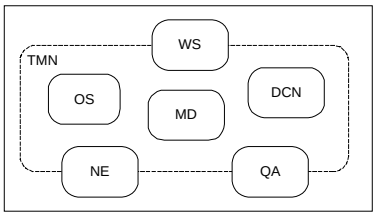

Figura 1: blocos do TMN.

Os blocos e as respectivas descrições são apresentados a seguir:

- **WS** (*WorkStation*): executa as funções de uma estação de trabalho. Traduz as informações entre o formato TMN e o formato apresentado para o usuário.
- **DCN** (*Data Communication Network*): é a rede de comunicação dentro do TMN.
- **OS** (*Operation System*): executa as funções dos sistemas de suporte à operação, incluindo funções de monitoramento e controle de funções de gerenciamento de telecomunicações.
- **MD** (*Mediation Device*): executa a mediação entre as interfaces locais TMN e o modelo OSI, a função de mediação pode ser necessária para assegurar que as informações, o escopo e a funcionalidade sejam representados de forma exata com o que o sistema operacional espera.
- **NE** (*Network Element*): possuem a informação gerenciada que é monitorada e controlada por um OS. Para que isso possa ocorrer, o NE deve possuir uma interface TMN padrão ou pode ser feito através de um QA.
- **QA** (*Q-adapters*): permite ao TMN gerenciar os NE's que não possuem as interfaces TMN. O QA realiza a tradução entre as interfaces TMN e não TMN.

O TMN define uma função de comunicação de mensagens (MCF) que todos os blocos com interfaces físicas necessitam. Esta função fornece camadas de protocolos necessárias para a comunicação de um bloco com um DCN (camadas 4 a 7). Um MCF pode fornecer todas as sete camadas OSI.

Os blocos podem atuar como gerentes ou agentes. O gerente processa suas diretivas e recebe as notificações, e um agente processa as diretivas, envia respostas, e emite eventos e alarmes.

2.3. Gerenciamento de Redes utilizando CORBA

Serviço de Eventos

Para permitir a gerencia de uma rede de telecomunicações, o OMG desenvolveu um serviço que permitisse a troca de mensagens utilizando eventos, com isso surgiu o Serviço de Eventos [14].

O Serviço de Eventos foi desenvolvido para permitir a comunicação assíncrona entre cliente e servidor. Mas o serviço apresentou algumas deficiências, como ausência de filtros, suporte a qualidade de serviço, habilidade dos clientes especificarem os eventos que desejam receber, habilidade dos produtores publicarem os eventos que produzem e a ausência de uma estrutura bem definida para a transmissão de eventos. Para corrigir essas deficiências, foi desenvolvida uma extensão, o Serviço de Notificação [15].

Notificação de Eventos

Tanto no Serviço de Eventos quanto no Serviço de Notificação, a comunicação é feita de forma assíncrona. Para utilizar estes serviços, existem dois tipos de aplicações: os produtores – que produzem os eventos – e os consumidores – que processam os eventos. Entre eles, para permitir a operação assíncrona, existe um canal de eventos que tem como papel fazer a comunicação entre os consumidores e os produtores. Este canal recebe os eventos dos produtores e a partir dele, o consumidor recebe os eventos. Desta forma, os produtores não precisam ter conhecimento dos seus consumidores e os consumidores, dos produtores. Esse tipo de visão permite que o produtor não precise gerar eventos para todos os consumidores, bastando apenas gerar um evento que será replicado e distribuído entre seus consumidores.

Para que o canal de evento/notificação permita o envio e o recebimento de eventos, existem dentro do canal de eventos, os *proxies*, responsáveis pelas trocas de mensagens. Um *proxy* está ligado a um consumidor – *Proxy Supplier* – ou a um produtor – *Proxy Consumer*, como mostrado na Figura 2. Para criar os *proxies* existem os objetos de administração (*Admin*), eles são responsáveis em gerenciar, criar e remover os *proxies*. Existem dois tipos de objetos de administração, o *Supplier Admin* – responsável pelos *proxies* dos produtores – e o *Consumer Admin* – responsável pelos *proxies* dos consumidores. Na Figura 3 pode ser notado que o *Supplier Admin* e o *Consumer Admin* geram *Proxy Consumer* para o produtor e *Proxy Supplier* para o consumidor, respectivamente. Essa inversão de nomes se deve aos papéis de cada objeto. Assim, para o produtor, o *Proxy Consumer* faz o papel do consumidor, e o *Proxy Supplier* o papel de produtor para o consumidor.

Outro mecanismo importante existente no Serviço de Notificação é a filtragem, que permite fazer com que os clientes recebam ou os produtores enviem apenas os eventos de interesse. Isso é possível através dos filtros que são conectados aos *Admin* ou aos *proxies*, como apresentado na Figura 2.

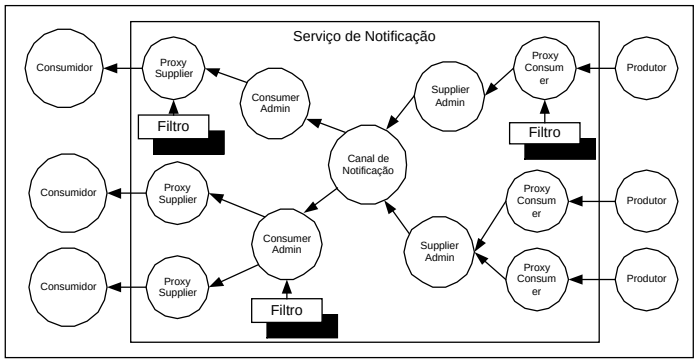

Figura 2: Produtor, consumidor e seus proxies.

O canal de eventos/notificação suporta dois modelos de comunicação: o modelo *push* e o modelo *pull*. No modelo *push*, o produtor tem a iniciativa de enviar os eventos para o consumidor, e no modelo *pull*, é o consumidor que tem a iniciativa e pergunta ao produtor sobre os eventos. Estes modelos podem ser combinados gerando quatro formas possíveis de comunicação entre produtor e canal, e entre canal e consumidor, mostrados na Tabela 1.

Produtor/Canal de Eventos	Canal de Eventos/Consumidor
pull	*Pull*
pull	*Push*
push	*Push*
push	*Pull*

Tabela 1: Possibilidades de comunicação entre produtor, canal de eventos e consumidor

Quanto aos tipos de eventos, o Serviço de Notificação utiliza três tipos. O primeiro tipo é o *Any*, que envolve a transmissão de eventos genéricos. É de fácil utilização, mas muitas aplicações requerem a troca de eventos tipados. Para satisfazer isso, o OMG definiu o evento *Typed*. Estes eventos possuem tipos (inteiro, booleano, ponto flutuante), mas sua definição é de difícil compreensão e implementação, pois os tipos não são definidos explicitamente, dificultando a sua interpretação.

O terceiro evento é o evento estruturado, composto de duas partes: o cabeçalho do evento e o corpo do evento. O cabeçalho do evento, por sua vez, é dividido em outras duas partes: o cabeçalho fixo que possui os campos domínio (*domain_type*), tipo (*type_name*) e evento (*event_name*); e o cabeçalho variável composto por tuplas do tipo nome-valor que definem atributos como prioridade e confiabilidade do evento.

O corpo do evento é composto de duas partes também: o corpo de campos filtráveis, onde as informações são armazenadas em tuplas do tipo nome-valor e o resto do corpo que pode ser utilizado para armazenar qualquer tipo de informação adicional.

O Serviço de Notificação ainda permite que os produtores em um canal descubram os tipos de eventos requisitados por todos os consumidores do canal. Assim os produtores produzem apenas os eventos necessários no canal. Da mesma forma, o consumidor é capaz de descobrir os eventos produzidos no canal e pode subscrever novos tipos de eventos para que sejam produzidos.

Outro recurso deste serviço é a possibilidade de configurar várias propriedades de qualidade de serviço (a confiabilidade do canal e a prioridade dos eventos) e administrativas para o canal de eventos (o número máximo de eventos que o canal irá armazenar em *buffer* a qualquer momento e o número máximo de produtores e consumidores que podem conectar ao canal). Opcionalmente, o Serviço de Notificação pode contar com um repositório de tipos de eventos, facilitando a formação de filtros de restrições pelos usuários finais.

Domínio de Eventos

Novamente, um novo serviço foi desenvolvido tendo como base o Serviço de Notificação, o Domínio de Eventos [17, 18]. Este serviço adiciona algumas funcionalidades não presentes no Serviço de Notificação, citando algumas:

- Permitir a conexão entre os canais de notificação, formando uma rede de canais.
- Fornecer uma interface para o gerenciamento das conexões.
- Permitir a conexão de canais de diferentes desenvolvedores.
- Configuração da qualidade de serviço de todos os canais com uma operação.

A conexão entre os canais é possível através dos *proxies*, onde os canais se comportam como produtores e consumidores de eventos, e sem a necessidade de novas interfaces, facilitando a utilização dos Serviços de Notificação existentes.

3. Descrição da Arquitetura

A arquitetura de gerenciamento de redes de telecomunicações desenvolvida utiliza-se de uma pequena rede composta por várias centrais locais interligadas, onde cada central possui seus equipamentos necessários para seu funcionamento.

Cada equipamento representa um elemento de rede TMN e possui seu agente, responsável em gerar alarmes no caso de ocorrer alguma falha no equipamento. O gerenciamento da rede segue a camada de gerenciamento de rede do modelo TMN, ou seja, todos os elementos da rede enviam os eventos para o centro de gerenciamento. Para gerenciar estes equipamentos e a comunicação entre as centrais, existem vários gerentes que representam os blocos WS e OS, com funções específicas, como supervisionar equipamentos ou a comunicação entre as centrais. A rede de comunicação de dados (DCN) é construída utilizando uma rede de canais de Notificação CORBA. A arquitetura conta com um sistema de log que permite armazenar todos os eventos gerados pelos equipamentos. Como vários eventos podem ser originados por uma mesma falha, a arquitetura possui um correlacionador capaz de realizar esta tarefa.

A seguir, será apresentada a descrição de cada componente implementado, como a rede de telecomunicações e suas centrais, os módulos desenvolvidos para a gerencia da rede, como os gerentes, o correlacionador, o *MultipleFilter* e o *EventChannelServer*.

3.1. Rede de Telecomunicações

Uma rede de telecomunicação permitiu inicialmente a transmissão da voz através de longas distâncias. Hoje, a utilização de uma rede de telecomunicação não se limita apenas a transmissão de voz, podendo transmitir dados e vídeos, por exemplo. Uma rede de telecomunicações é composta por telefones ligados em centrais que, por sua vez, estão interligadas, permitindo que quaisquer dois telefones possam se conectar. Um exemplo de rede formada por centrais é mostrado na Figura 3.

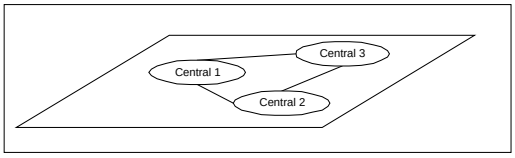

Figura 3: representação de uma rede de telecomunicações.

A ligação entre as centrais pode ser feita por vários tipos de meios: fios de cobre, fibras ópticas, ondas de rádios e microondas. A ligação entre duas centrais é chamada de tronco. Estas centrais possuem vários tipos de equipamentos entre eles: *multiplex*, central telefônica, ar condicionado, desumidificador e gerador. Além destes equipamentos que são de interesse para a gerência, outros dispositivos podem ser gerenciados como as portas que dão acesso para o equipamento.

A central possui seu próprio canal de notificação pela qual os agentes dos equipamentos enviam os alarmes que geram, como mostrado na Figura 4. Também é através dos alarmes emitidos que se pode determinar a ocorrência de alguma falha em um dos troncos que interligam as centrais.

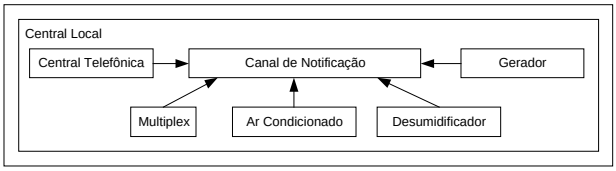

Figura 4: central telefônica.

3.2. Gerente

É o objeto responsável em receber os alarmes correlacionados ou não. Após obtidas as informações, os gerentes determinam as ações a serem tomadas.

Os gerentes desenvolvidos para esta arquitetura são os seguintes:

- **Gerente de Infraestrutura**: responsável pela infraestrutura das centrais locais (portas, umidade, temperatura). Ele utiliza informações ligadas ao ambiente em que se encontra a central telefônica. Os eventos podem informar o aumento da temperatura ou da umidade, por exemplo.
- **Gerente de Energia**: responsável pelos geradores e baterias das centrais. Os eventos podem indicar a ocorrência de alguma falha neste tipo de equipamento.
- **Gerente de Comunicação**: responsável pelos troncos e pela comunicação entre as centrais. É o gerente mais importante, pois deve detectar a ocorrência de falhas envolvendo os equipamentos de comunicação, como a central telefônica ou o multiplexador.

3.3. MultipleFilter

O *MultipleFilter* é o objeto pelo qual as centrais se conectam para transmitir os eventos. Isto é possível porque o *MultipleFilter* também possui um canal de notificação.

Os canais de notificação das centrais se conectam diretamente com o canal do *MultipleFilter* através dos *proxies* consumidores e produtores. Isto é possível porque os *proxies* possuem herança dos consumidores e produtores de eventos. Isto é mostrado na Figura 6.

Desta forma, o canal da central (canal produtor) cria um *proxy* produtor para enviar os eventos e o canal do *MultipleFilter* (canal consumidor) cria um *proxy* consumidor para receber os eventos.

3.4. EventChannelServer

Segundo a especificação que descreve o Domínio de Eventos, existe a necessidade de um gerenciador capaz de controlar as conexões entre os canais e manter um mapa da topologia da rede resultante destas conexões. Esse gerenciador tem como funções realizar as conexões entre os canais e determinar a ocorrência de diamantes e ciclos nesta rede. Um diamante é formado quando existem dois ou mais caminhos diferentes entre dois nós da rede. Já um ciclo é um caminho onde um nó é percorrido mais de uma vez.

Para a arquitetura, o gerenciador é o *EventChannelServer* que recebe os pedidos de criação e remoção de canais e de conexões entre os canais. Quanto às conexões, o *EventChannelServer* pode criar tanto conexões do tipo *pull* quanto do tipo *push*.

3.5. Network Status

Para construir a rede e manter informações relacionadas à topologia da mesma e ao funcionamento dos equipamentos presentes na rede foi desenvolvido o *Network Status*. As informações mantidas pelo *Network Status* são utilizadas para a construção da rede e pelo correlacionador para determinar a ocorrência da falha.

O *Network Status* é um servidor que contém informações como: o número de centrais, o estado de cada equipamento de cada central e as conexões entre as centrais.

3.6. Correlacionador

Quando ocorre uma falha em algum dos equipamentos existentes numa rede, vários eventos são gerados, alguns provenientes de equipamentos interligados ao que falhou. Devido

ao grande número de eventos gerados pelos vários equipamentos, a localização da falha torna-se uma tarefa difícil de ser realizada. Para diminuir o número de eventos, existe a técnica de correlação que consiste na interpretação conceitual de múltiplos alarmes, levando à atribuição de um novo significado aos alarmes originais, gerando um novo alarme. O objetivo da correlação é diminuir a quantidade de alarmes transferidos dentro do sistema de gerência de rede, aumentando o conteúdo semântico dos alarmes resultantes.

Para esta tarefa, existem várias técnicas [5, 6, 10, 11, 12, 13] para correlacionar o evento: compressão, supressão seletiva, filtragem, contagem, escalação, generalização, especialização e relacionamento temporal. A técnica adotada neste trabalho é apresentada em [23], é a correlação por codificação. Esta técnica consiste em analisar os sintomas para determinar a causa. Os sintomas e a causa são armazenados em uma matriz de codificação, o *codebook*, onde cada linha representa a causa com seus respectivos sintomas. Assim, ao receber os sintomas, o correlacionador gera o código e busca no *codebook*, descobrindo ou não a ocorrência de um evento correlacionado.

O correlacionador possui um módulo responsável em receber e correlacionar os eventos, o *correlator*. O correlacionador utiliza o canal de notificação para propagar os alarmes correlacionados entre vários gerentes, para isso basta o gerente se conectar ao canal existente no correlacionador. Para evitar os eventos que não são de interesse, são utilizados filtros do serviço de Notificação. A estrutura do correlacionador é apresentada na Figura 5.

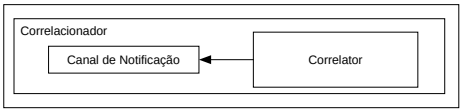

Figura 5: correlacionador.

Assim, a arquitetura final é apresentada na Figura 6, onde pode ser visto como todos os componentes são interligados.

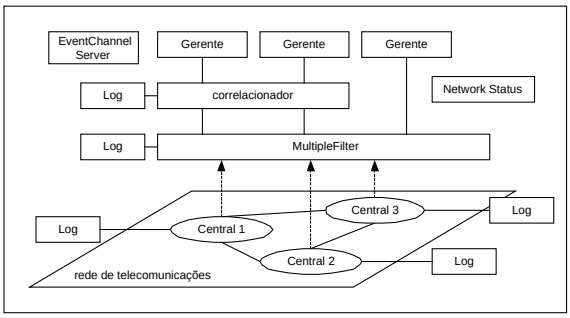

Figura 6: arquitetura proposta.

3.7. Eventos

Os alarmes são propagados na arquitetura através dos eventos estruturados. A construção dos eventos segue a especificação ITU-T X.733 [2], onde são determinados os campos e valores utilizados pela arquitetura.

Quanto ao tipo, são definidos os seguintes valores:

Tipo de alarme	type_name
Alarme de comunicação	communicationsAlarm
Alarme de Qualidade de Serviço	qualityofServiceAlarm
Alarme de Erro de Processamento	processingErrorAlarm
Alarme de Equipamento	equipmentAlarm
Alarme Ambiental	environmentalAlarm

Tabela 2: tipos de alarmes.

Para este trabalho os atributos escolhidos para a utilização são os seguintes:

Atributo	Nome do Atributo
Classe do objeto gerenciado	managedObjectClass
Instância do objeto gerenciado	managedObjectInstance
Instante do evento	eventTime
Causa provável	probableCause
Nível de Severidade	PerceivedSeverity
Texto adicional	AdditionalText

Tabela 3: atributos do alarme.

Estes atributos foram escolhidos pelo fato de serem os de maior relevância para a arquitetura, e por serem os utilizados em algumas ferramentas de gerenciamento de redes [7].

Para a arquitetura foi incluído mais um atributo: local, que indica o local de origem do evento. Este atributo é inserido como uma informação adicional. Na recomendação, uma informação adicional é composta de três itens: um identificador, um indicador e a informação. Mas devido ao fato do evento estruturado permitir apenas tipos primitivos, a informação adicional fica subentendida como o local de origem do evento, sendo um identificador do tipo inteiro. Os atributos são armazenados nos eventos estruturados como campos filtráveis.

No caso de eventos correlacionados mais um atributo é inserido, o *correlatedNotifications*, composto pelos identificadores dos eventos que foram correlacionados para originar o evento final.

O evento, após ser gerado por um dos equipamentos, é enviado para o canal da central. Em seguida, o evento é transmitido para o canal do *MultipleFilter*. Este, por sua vez, transmite o evento para o canal do correlacionador e para o gerente correspondente ao evento. Caso o evento seja correlacionado, um novo evento é gerado e transmitido ao gerente.

3.8. Logs

A principal tarefa do sistema de log é armazenar os eventos que trafegam nos canais de notificação do modelo e permitir o acesso a eles. O log segue a especificação ITU-T X.735 [3] que define várias funcionalidades e características do log. Nesta especificação, o log é composto por um pacote de atributos obrigatórios e um pacote de atributos condicionais, como descrito nas Tabelas 4 e 5.

Pacote de Atributos Obrigatórios	
Identificador	Identificador de um registro do log.
Discriminador	Filtra as informações que serão armazenadas.
Estado Administrativo	Define o funcionamento do log através de dois estados: UNLOCK – o log está disponível para armazenar, recuperar e remover registros, e LOCK – o log está disponível apenas para a leitura e remoção dos registros.
Estado Operacional	Define a capacidade operacional. Isto é feito através de dois estados: ENABLE (o log está criado e apto para o uso) e DISABLE (o log não está disponível para o uso).
Ação de Log Cheio	Define a ação a ser tomada no caso de log cheio. Possui duas formas de tratar: WRAP (os registros mais antigos são removidos) e HALT (os registros mais novos não são armazenados).
Estado de avaliabilidade	Define a disponibilidade do log. Pode indicar uma condição de log cheio, impedindo o armazenamento de novos registros.

Tabela 4: pacotes de atributos obrigatórios.

Pacote condicional de log finito	
Tamanho Máximo	Define o tamanho máximo do log em número de bytes.
Tamanho Corrente	Tamanho corrente do log em bytes
Número de Registros	Número corrente de registros no log

Tabela 5: pacote condicional de log finito.

Como o interesse deste trabalho é mostrar a viabilidade de incluir um sistema de log à arquitetura, apenas alguns atributos foram escolhidos como Identificador, Discriminador e Número de Registros.

No caso dos atributos dos registros do log, além dos atributos dos eventos estruturados foram incluídos os seguintes atributos:

- *Log Record Id*: identificador do registro armazenado.
- *Logging Time*: momento em que o evento foi armazenado no log.

Estes atributos foram escolhidos por serem capazes de determinar um registro armazenado no sistema de log.

3.9. Visão Detalhada da Arquitetura

Após a descrição dos elementos presentes na arquitetura, uma visão mais detalhada da arquitetura é apresentada na Figura 7. Nesta visão podem ser vistos os canais de notificação e os outros objetos descritos.

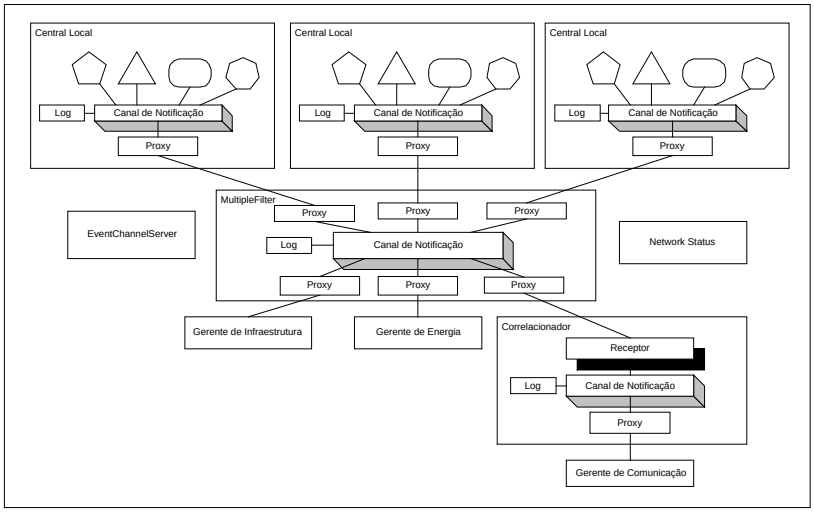

Figura 7: Visão detalhada da arquitetura.

Inicialmente, a topologia da rede de telecomunicações define as configurações dos componentes apresentados. Esta topologia é definida através do *Network Status* que irá fornecer todas as informações necessárias para a construção da rede e quanto aos equipamentos existentes em cada central local.

Após iniciar a rede, cada central conecta seu canal ao canal do *MultipleFilter*. O estado funcional de cada equipamento nas centrais também é fornecido pelo *Network Status*. O *MultipleFilter*, por sua vez, se conecta ao Correlacionador. O divisor do Correlacionador tem então a missão de dividir os eventos recebidos, segundo a topologia da rede. Isto é necessário, pois existem várias instâncias dos equipamentos na rede. Assim que os eventos são divididos, o analisador lê os eventos e decide se deve gerar um evento correlacionando os eventos recebidos. Caso sim, o evento gerado é enviado para o canal de eventos do próprio correlacionador, por onde os gerentes se conectam para receber os eventos. Todos os eventos recebidos pelo correlacionador são analisados e correlacionados, assim o gerente recebe apenas o resultado da correlação.

Os gerentes podem se conectar ao *MultipleFilter* ou ao Correlacionador, dependendo se o gerente precisa receber eventos correlacionados ou não. Para evitar os eventos que não deseja receber, filtros são utilizados. Caso o gerente necessite receber eventos correlacionados e não-correlacionados, é possível a construção de uma conexão direta entre os canais de eventos do *MultipleFilter* e do Correlacionador, não esquecendo de configurar os filtros para

os tipos de eventos que serão recebidos pelo gerente. É interessante notar que ao utilizar essa solução para os gerentes não há formação de um diamante entre os canais. Isso se deve ao fato que todos os eventos recebidos pelo correlacionador são utilizados para gerar um novo evento que será transmitido ou simplesmente serão descartados, como mostrado na Figura 8.

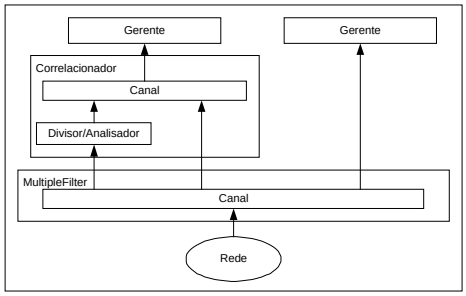

Figura 8: Gerente recebendo eventos correlacionados e não correlacionados.

4. Implementação

Para a implementação foi utilizado o ORB OrbixWeb 3.1c da IONA, o pacote de serviços Openfusion versão 1.1.0 da Prismtech e o JDK 1.2.1 da Sun. Quanto ao hardware, foi utilizada uma estação de trabalho SUN com o sistema operacional Solaris 2.7.

Algumas classes implementadas herdam interfaces, sendo necessário implementar alguns métodos existentes na interface. É o caso das classes que herdam as interfaces de consumidores (*StructuredPushConsumer* e *StructuredPullConsumer*) e produtores (*StructuredPushSupplier* e *StructuredPullSupplier*). A seguir, será feita uma breve descrição quanto à implementação de cada componente do protótipo.

4.1. EventChannelServer

É o objeto responsável em receber as chamadas para a criação de canais e conexões entre os canais, assim como a remoção dos mesmos. O pedido de criação de um canal retorna uma classe composta do canal e seu identificador (Cid), sua definição IDL é mostrada na Figura 9.

```
struct Cid{
        long id;
        CosNotifyChannelAdmin::EventChannel channel;
};
```

Figura 9: definição da classe Cid.

Além da tarefa de criação de canais e conexões entre os canais, esta interface possui métodos para obter os canais consumidores ou canais produtores referentes a um dado canal e determinar o identificador de um canal. A interface de *EventChannelServer* é apresentada na Figura 10.

```
Interface EventChannelServer{
  Sequencelong get_all_channels_supplier(in long column);
  sequencelong get_all_channels_consumer(in long line);
  long get_id(in CosNotifyChannelAdmin::EventChannel ec) raises (CosNotifyChannelAdmin::ChannelNotFound);
  CosNotifyChannelAdmin::EventChannel get_channel(in long who) raises
(CosNotifyChannelAdmin::ChannelNotFound);
  Cid create_channel(in seqProperty qos, in seqProperty adm);
  Void remove_channel(in long who) raises (CosNotifyChannelAdmin::ChannelNotFound);
  long create_connection(in long supplier, in long consumer, in ProxyType type, in seqEventType intypes)
raises (CosNotifyChannelAdmin::ChannelNotFound);
  boolean remove_connection(in long conn);
  seqconnection return_all_connections();
};
```

Figura 10: interface EventChannelServer.

4.2. Network Status

Network Status é um servidor que mantém as informações da rede, tais como número de centrais e o estado de cada equipamento de cada central. É implementado como um simples servidor de informações. É utilizado para fornecer as informações para os outros programas.

4.3. Central

A central é formada por *threads* que simulam o funcionamento dos equipamentos. O estado de cada equipamento é mantido pelo *Network Status*. Cada *thread* é um produtor de eventos que obtém a informação de seu estado do *Network Status*, gerando eventos ou não.

4.4. MultipleFilter

Cria um canal através do *EventChannelServer*. O seu canal se conecta com os canais das centrais. O canal criado por este objeto é a ponte entre os canais existentes e os gerentes.

4.5. Correlacionador

O correlacionador é formado por um canal, por onde os gerentes se conectam, e o *correlator*, responsável em receber os eventos e correlacioná-los para enviar para o canal do correlacionador. O *correlator* é composto por duas partes: o consumidor de eventos do tipo *pull* que se conecta ao canal do *MultipleFilter*; e o produtor de eventos do tipo *push*, que se conecta ao canal do correlacionador.

4.6. Gerente

Responsáveis em receber e apresentar os eventos, são implementados como consumidores de eventos. Podem se conectar ao canal do *MultipleFilter* – caso desejem receber os eventos sem correlação – ou do Correlacionador – no caso de eventos correlacionados.

4.7. Log

O log para a arquitetura é um consumidor de eventos do tipo *push* que se conecta ao canal e captura todos os eventos armazenando em seu log. Caso seja necessário, pode se anexar ao log um filtro para selecionar apenas eventos de interesse ao sistema. Da mesma forma que vários logs podem ser colocados em um mesmo canal. Os registros dos logs são armazenados em base de dados para facilitar a consulta e a recuperação dos dados.

4.8. Testes

O objetivo dos testes foi de analisar o desempenho do sistema, para isso foram desenvolvidos alguns testes, avaliando o tempo necessário para processar uma determinada quantidade de eventos e o tempo de tráfego, ou seja, o tempo que o evento permanece nos canais e *proxies* do Serviço de Notificação.

Para o primeiro teste foram utilizadas duas arquiteturas. A primeira é a Arquitetura Proposta (Figura 6) com doze canais conectados, onde dez canais se conectam com os produtores, e a outra é a Arquitetura Simplificada, onde todos os equipamentos estão ligados ao canal de Notificação do *MultipleFilter*, sem a utilização de canais locais nas centrais. Os esquemas destas arquiteturas são apresentados na Figura 11.

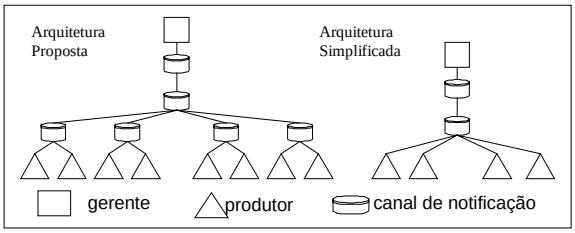

Figura 11: representação das arquiteturas para os testes.

O primeiro teste consiste em analisar o comportamento com relação ao aumento de produtores de eventos no sistema. Para isso foi utilizado um gerente e vários conjuntos de produtores (de 30 a 150 produtores). A Figura 12 apresenta alguns resultados quanto ao tempo necessário para o gerente receber os eventos.

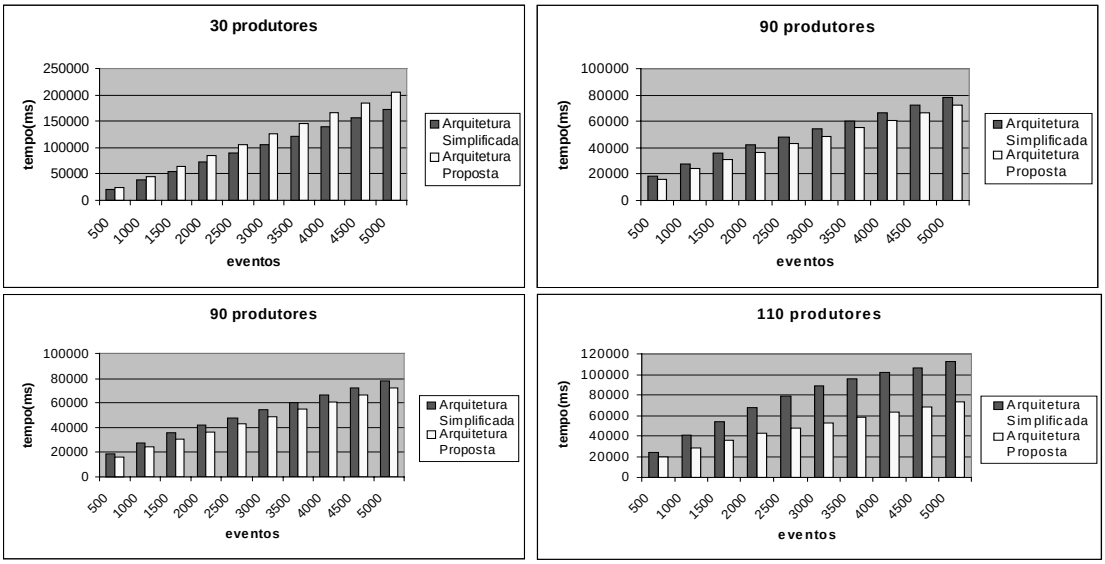

Figura 12: resultados para 30, 50, 90 e 110 produtores.

Utilizando poucos produtores (de 30 a 40 produtores), o desempenho da Arquitetura Proposta ficou um pouco abaixo da Arquitetura Simplificada. O melhor desempenho foi obtido com o intervalo entre 50 a 110 produtores. Acima disso, o desempenho não difere muito do desempenho da Arquitetura Simplificada, como apresentado na Figura 13.

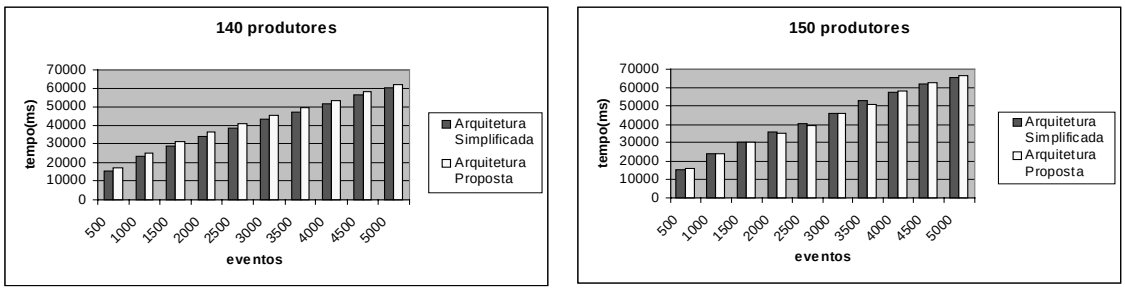

Figura 13: resultados com 140 e 150 produtores.

A Figura 14 apresenta o tempo necessário para que a Arquitetura proposta com doze canais (ou seja, dez canais produtores) receba uma determinada quantidade de eventos com o aumento do número de produtores. Como pode ser observado, com o aumento do número de produtores de eventos, o gerente processa os eventos em um menor tempo.

Por outro lado, o evento tem um aumento no tempo de tráfego, permanecendo um tempo maior aguardando para ser tratado. A Figura 15 apresenta o resultado com relação ao tempo de tráfego para o mesmo teste.

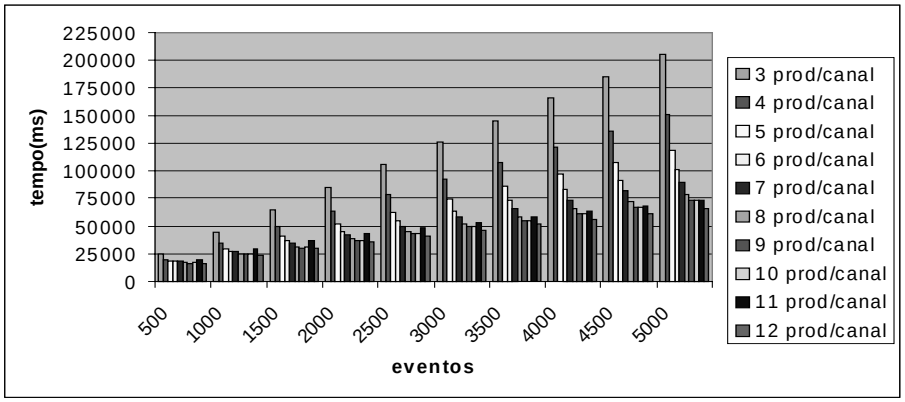

Figura 14: tempo total para o gerente receber os eventos, variando o número de produtores por canal.

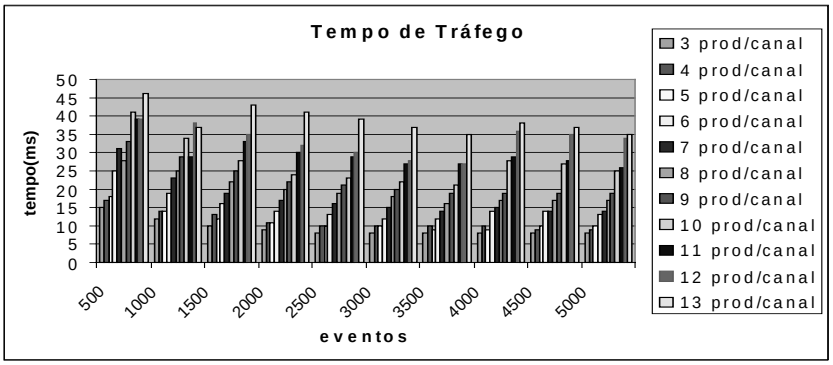

Figura 15: tempo de tráfego.

O segundo teste analisa o comportamento da Arquitetura Proposta com o aumento no número de canais ligados aos produtores, mantendo fixo o número total de produtores. A Tabela 6 apresenta os valores utilizados para os testes.

Caso	No. de canais produtores	No. de produtores por canal
1	1	12
2	2	6
3	3	4
4	4	3
5	6	2
6	12	1

Tabela 6: os casos utilizados no segundo teste.

Os resultados obtidos para o segundo teste estão nas Figuras 16 e 17. Os gráficos mostram que o tempo total necessário para receber uma determinada quantidade de eventos decai com o aumento de número de canais. O mesmo ocorre com o tempo de tráfego dos eventos.

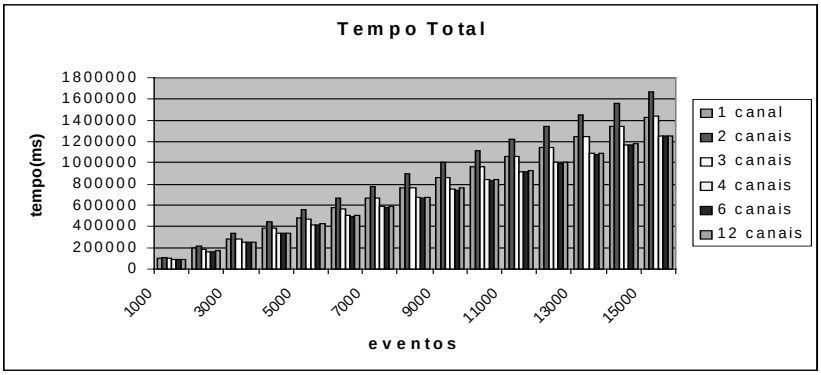

Figura 16: tempo total para o segundo teste.

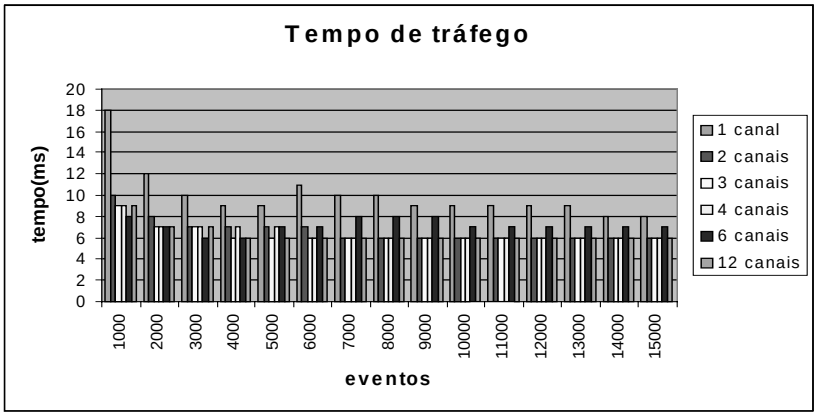

Figura 17: tempo de tráfego para o segundo teste.

A partir destes testes foi possível obter os seguintes resultados:

- A Arquitetura Proposta é escalável, pois os tempos não aumentam significativamente com o aumento do número de eventos e/ou produtores.
- Existe um intervalo de número de produtores por canal (entre 50 e 110), onde o desempenho do Modelo Proposto é melhor.
- A utilização de poucos produtores (menos de 30) faz o gerente ficar ocioso, aguardando a chegada dos eventos.
- No caso de muitos produtores (acima de 120), o gerente não consegue processar rapidamente todos os eventos, por isso, o tempo de tráfego é maior.
- A Arquitetura Proposta mostrou-se escalável quanto à inserção de novos canais, sem a necessidade de alterações na estrutura.

5. Conclusão

Este trabalho apresenta a utilização da arquitetura CORBA para o gerenciamento de redes de telecomunicações. Esta associação apresenta vários pontos em comum, desejáveis nas duas tecnologias: interoperabilidade, flexibilidade e escalabilidade.

A utilização dos recursos do serviço de Notificação CORBA, para a construção da infra-estrutura de transmissão de eventos e dos módulos com tarefas determinadas como a correlação de eventos e o gerenciamento da rede, mostra que a arquitetura atende aos requisitos necessários para a construção de um sistema de gerenciamento de redes de telecomunicações.

O trabalho desenvolvido em [7] utiliza também a arquitetura CORBA para criar uma plataforma de Supervisão de Alarmes, baseando-se na criação de um Adaptador de Objetos, responsável em receber eventos de um sistema de gerenciamento já existente e mapeá-los para os eventos estruturados. A plataforma utiliza o canal de evento para transmitir os eventos. Já o nosso trabalho mostra a possibilidade da construção de um sistema de gerenciamento totalmente baseado em CORBA, dos equipamentos aos gerenciadores, utilizando eventos estruturados para a transmissão das informações e canais de Notificação para fazer a comunicação entre os módulos.

A arquitetura desenvolvida apresenta um exemplo de integração dos módulos TMN com os módulos do Serviço de Notificação CORBA, para a realizar a tarefa de gerenciamento de redes de telecomunicações. A implementação apresenta a viabilidade de conexão entre os módulos a partir dos canais de Notificação para fazer a rede de comunicação TMN e a transmissão dos alarmes através dos eventos estruturados. Os testes mostram que o Modelo Proposto é escalável e que existe um intervalo de número de produtores onde o desempenho do modelo proposto é melhor.

Um trabalho futuro proposto é o desenvolvimento de um sistema de correlação, que possa atender não apenas a correlação de eventos voltados para a área de Telecomunicações, como os desenvolvidos neste trabalho, mas que possa operar com quaisquer outras áreas de interesse, podendo obter as informações necessárias do sistema de log. Outro trabalho que pode ser desenvolvido é a integração deste sistema com os sistemas já existentes, que utilizam os protocolos SNMP e CMIP.

Agradecimentos:

Agradecemos ao CNPq, CAPES e PRONEX pelo apoio financeiro na realização deste trabalho.

6. Referências

[1] **Cisco System**. "Cisco Service Management System". 1999. http://www.cisco.com.
[2] **CCITT**. "Information Technology – Open Systems Interconnection – Systems Management: Alarm Report Function". Recommendation X.733. 1992.
[3] **CCITT**. "Information Technology – Open Systems Interconnection – Systems Management: Log Control Function". Recommendation X.735. 1993.
[4] **DSTC Pty Ltd**. "COSNotification Service – User Guide". 1999. http://www.dstc.edu.au.
[5] **Edwards**, A. V. e **Whitaker**, R. J.. "Fault Management: A Functional View of Root Cause Analysis and Correlation". TAVVE Software Company. http://www.tavve.com.
[6] **Hewlett Packard Company**. "HP Introduces New OpenView Event-Correlation Solution And Platform Enhancements For The Telecommunications Industry". Maio, 1996. http://www.hp.com.
[7] **Higa, F. S., Wandresen, R. R. e Penna, M.C..** "Plataforma de Supervisão de Alarmes Baseada em CORBA". 18° Simpósio Brasileiro de Redes de Computadores. Maio, 2000. Belo Horizonte, Minas Gerais. pp 3-18.
[8] **IONA Technologies PCL**. "OrbixNotification Programmer's Guide and Reference". Dezembro, 1998. http://www.iona.com.
[9] **Mansouri-Samani, M. e Sloman, M..** "An Event Service for Open Distributed Systems." Open Distributed Processing and Distributed Platforms. Chapman & Hall. Maio, 1997. pp. 183-194.
[10] **Meira**, D. M. "Um Modelo para Correlação de Alarmes em Redes de Telecomunicações". Belo Horizonte, Minas Gerais. Novembro, 1997.
[11] **Meira**, D. M.. "Um Survey Sobre Correlação de Alarmes". Technical Report DCC 017/97. Belo Horizonte, Minas Gerais. Julho, 1997.
[12] **Meira**, D. M. e Nogueira, J. M. S.. "Métodos e Algoritmos para Correlação de Alarmes em Redes de Telecomunicações". XV Simpósio Brasileiro de Redes de Computadores. Maio, 1997. São Carlos, São Paulo. pp. 79-98.
[13] **Munich Network Management Team**. "Event Correlation". Novembro, 1998. http://wwwmnmteam.informatik.uni-muenchen.de/projects/evcorr/
[14] **Object Management Group**. "Event Service". Junho, 2000. http://www.omg.org.
[15] **Object Management Group**. "Notification Service – Joint Revised Submission". Novembro, 1998. http://www.omg.org.
[16] **Object Management Group**. "What is CORBA?". http://www.omg.org.
[17] **Object Management Group**. "Management of Event Networks – Request For Proposal". Novembro, 1999. http://www.omg.org.
[18] **Object Management Group**. "Management of Event Domains – Revised Submission". Janeiro, 2000. http://www.omg.org.
[19] **Object Management Group**. "Telecom Log Service – Joint Initial Submission". Julho, 1998. http://www.omg.org.
[20] **PrismTech Limited**. "OpenFusion Developer's Guide". Junho, 1999. http://www.prismtech.com.
[21] **Queiroz, J. A. G. e Madeira, E. R. M..** "Facilidade de Monitorização Assíncrona em um Ambiente de gerência CORBA". 2° Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos – SFBSID'97. Fortaleza, Ceará. Novembro, 1997. pp. 135-146.
[22] **Rasmussen**, B.. "OpenFusion Notification Service". http://www.prismtech.com.
[23] **Yemini, S. A., Kliger, S., Mozes, E., Yemini, Y. e Ohsie, D**.. "High Speed and Robust Event Correlation". IEEE Communications Magazine. Maio, 1996. pp. 82-90.
[24] **WebProForum Tutorials**. "Telecommunication Management Network(TMN)". http://www.webproforum.com/tmn/index.html